Università degli Studi di Messina

UNIVERSITÀ DEGLI STUDI DI MESSINA
DEPARTMENT OF ENGINEERING
DOCTORAL PROGRAMME IN
"CYBER PHYSICAL SYSTEMS"

STUDY AND EVALUATION OF SERVICE-ORIENTED APPROACHES AND TECHNIQUES TO MANAGE AND FEDERATE CYBER-PHYSICAL SYSTEMS

Doctoral Dissertation by:
Eng. Giuseppe Tricomi

Advisor:
Prof. Antonio Puliafito

Co-Advisor:
Dr. Giovanni Merlino

Chair of the Doctoral Programme:
Prof. Antonio Puliafito

XXXIII Cycle
Academic Year 2019-2020

Abstract

IN recent years, the world in which we live has been deeply modified by the advent of the Internet of Things, filling the environment with devices able to interact through TCP/IP. These devices are commonly connected to sensors and actuators, enabling the remote management and monitoring of physical environments. This way, it is possible to manage the physical processes via software; the environments enhanced by IoTs are called Cyber-Physical Systems.

Cyber-Physical Systems are able to produce huge volumes of data, used by applications running on a CPS as input for any applications' duties. CPSs features can be shared with other CPSs through cooperation, enabling complex workflows to manage the environments better than it would be possible without any such cooperation. Typical examples of

cooperating CPSs are Smart Buildings and Smart Cities. The former are environments hosting one or more CPSs, supporting their residents, and the latter are aggregations of environments that host IoT devices to create an especially pervasive instance of a CPS (in some cases, multiple CPSs) that support citizens across their daily routines.

At the same time, the devices composing a CPS are capable to provide some computational power, that is typically not used to the fullest, and that can be exploited to "disseminate" the computation across the environment, possibly placing most computation near where the request originates. This way, a double face benefit is achieved: on the one hand, service times get smaller, because this approach avoids, or minimizes, network latencies and, on the other hand, it optimizes both power consumption and network bandwidth overall.

This dissertation aims to provides clues about the recent and ongoing investigations about cooperation among CPSs, with the overarching goal to exploit a number of established and emerging computing paradigms to enhance the services provided to citizens living in the environments under the coverage of such cooperating systems.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Cyber Physical Systems (CPS) are complex, heterogeneous distributed systems where the cooperation among cyber components (e.g., sensors, actuators, and control centers) and physical processes (e.g., temperature, fire) is deeply intertwined. A CPS is defined as a system where computation, networking, and physical processes are integrated to monitor and control physical environments [4]. The adoption of CPSs is due to the advent of the Internet of Things (IoT). IoT devices are systems with limited computational capabilities that are able to expose their services to the Internet, as long as a TCP/IP stack [11] is available. Sundmaeker et al. [12] state that IoT devices were born in the 1999 in the MIT Auto-ID Lab as technologies including bar codes, smart cards, sensors, voice recognition, and biometrics. In 2005 Srivastava [13] identifies the trend pushing technologies in general towards a pervasive dimension, and in particular moving "things" in that direction as well. Sundmaeker again, in [12], deeply analyzes the IoT concepts and perspectives from several points of views, providing an interesting categorization. IoT devices may be equipped with MCU and/or MPU (see 4.3) exploiting their facilities to manage the physical devices (sensors: smoke, gas, fire, presence, camera, and so on; actuators: light, valve, traffic lights, motors, and so on) during their life-cycle; at the same time, they may run programs that pre-process the physical signal to produce data useful for several purposes. For example, a single smoke sensor is not enough to identify fire (a cigarette could deceive it). A traditional fire system delivers the signal perceived to a central processing system that correlates the signals and decides if it has to activate the alarms, also informing firefighters or surveillance. In this scenario, the need for a unified scheme enabling CPS interactions with IoT devices without resorting to ad-hoc infrastructure, is obvious. Nevertheless, CPSs are not isles surrounded by "*plain*" [1] environments, but they are placed side by side with a multitude of other CPS (e.g., Vehicles, Factories, Buildings, Hospital, Street, and more). The pursue to provide a unified solution to manage any kind of CPS is not actually achievable, for several reasons, as detailed in the following.

- Administrative constraints: environments belong to several owners, Private or Public, that are free to make choices, in relation to exclusive usage and/or sharing of resources, according to various factors: financial, legal, etc.
- Technology advancements: CPSs realized in different moments may adopt significantly diverse technologies.
- Incompatibility with previously deployed (e.g., possibly legacy) technologies.

As a consequence, research activities were focused on identifying a methodology able to manage such environments (see section 2.3), and according to the literature, the most common solution used is represented by systems able to manage, coordinate, and organize sensors and actuators, host resources and provide support to the development and maintenance of high-level services [14], in a nutshell, a Cloud-like experience. Earliest examples of CPS are Smart Buildings (SB), historically defined in

[1]In this context, when referring to "plain" environments, we are talking about systems that do not leverage IoT device capabilities beyond mere sensing data collection, in summary, that cannot be categorized as Smart environments.

1981 with the term Intelligent Building, coined by the United Technology Building Systems Corporation, then implemented into the City Place Building in Hartford, Connecticut [15]. These "*Smart*" buildings are mostly customized controlling systems able to provide basic automatic management facilities of the installed devices (e.g., smoke and fire sensors, ventilation peripherals, heating systems, etc.). The advent of IoT pushes towards Smart Environments and specifically Smart Buildings. As highlighted in [16], it is hard to construct a unique view of a Smart Building with a commonly accepted definition. In 2009 the European Commission's Information Society provides a long and complex definition of Smart Building [17] (see section 2.3.1); in summary it may be explained as in the following: a Smart Building is an integrated system based on IoT and Ubiquitous Computing facilities able to take advantage of a range of computational and communications infrastructure and techniques. The Smart Building concept is easily adaptable in accordance with several scenarios, modifying system behaviour to achieve a range of results. For example, an SB specialization is related to the industrial context, where the physical processes commonly monitored by an SB (e.g., HVAC, fire, and intrusion control systems) are added to the controls related to the production processes. This way, the system will be able to monitor and quickly react to emergencies coming from the safety systems, but at the same time, it can face production chain issues that are continuously analyzed by the Factory. This is usually indicated as a

Smart Factory [18–20].

Until now, we have discussed of CPSs involving private environments, but another category of CPSs very relevant and interesting is that of public environments. Streets, public buildings, subways, and so on are commonly considered part of the Smart City [14]. Several entities spent energies in this direction, both at the institutional level (as the EU community) and at the academic and industrial levels as well, with models useful to provide new services, such as: optimizing vehicular traffic flows, enhancing the safety of citizens outdoors, monitoring air pollution levels, enrich public transportation systems, make rescue operations faster and safer, and so on. The CPSs discussed until now are self-consistent; they are able to complete their duties, simply by interacting with IoT devices available within their infrastructure. Furthermore, it is common to find SB composed of several IoT-based systems that are not integrated with others available in the same environment, as shown in figure 1.1. These systems can be identified in a wider environment where other CPSs (buildings or in general *Smart* environments) are available; if they are not physically isolated, it will be possible to interconnect them realizing a new CPS. This way, we obtain a kind of puzzle game composed of several tiles where each one is represented by a Smart Building. As the tiles of a puzzle game, each element has to be interconnected with the other exchanging data, raw or pre-processed, enabling workflows involving the city to support everyday citizens' lives. With regards to the

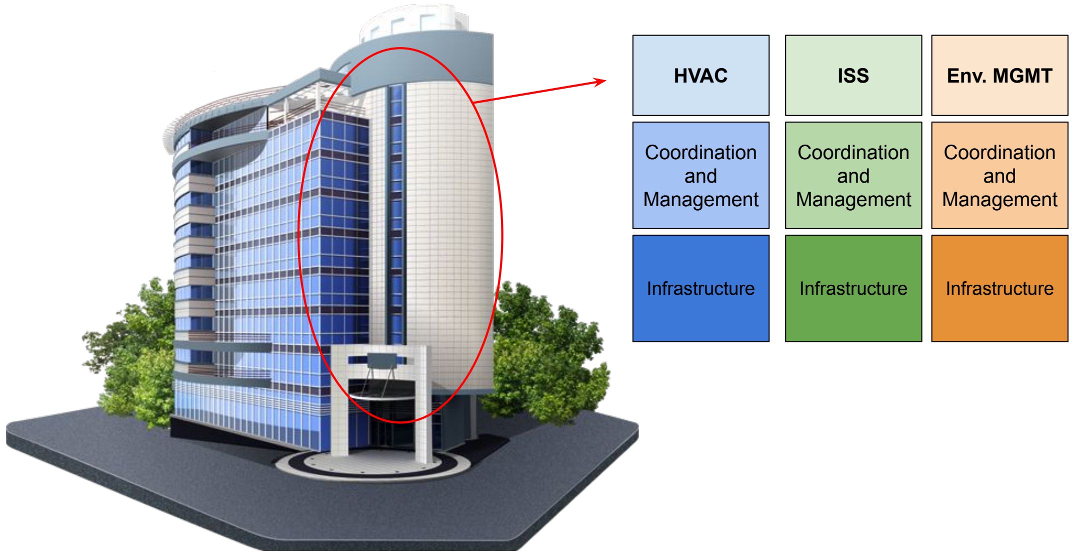

HVAC
ISS
Env. MGMT
Coordination and Management
Coordination and Management
Coordination and Management
Infrastructure
Infrastructure
Infrastructure

Figure 1.1: *An example of coexisting CPSs in a classical Smart Building.*

realization of such interconnections, the following question rises: *Are the CPSs belonging to the same administrative domain?* The answer guides us towards one of the following two solutions:

- If the answer is affirmative, we can model our interconnection in a tightly coupled way; adopting the Software-Defined Building approach (see section 2.3.1.1, and chapters 5,8, and 9 even if SDB approach is not the main focus).

- If negative we are bound to consider a loosely coupled interaction, where not all the capabilities of the systems are shareable, adopting a federated cooperative approach (see section 2.2.4)

In the following, we briefly introduce these two approaches.

A Software-Defined Building (see figure 1.2) can be defined as a building where, in line with the Software-Defined principles, the infrastructure

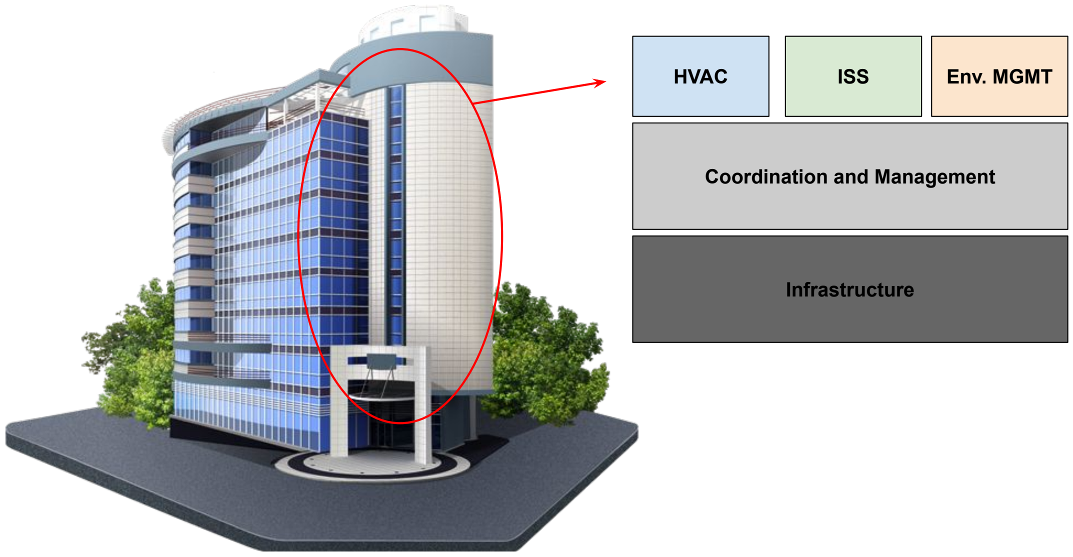


Figure 1.2: *An example of coexisting CPSs in a Software-Defined Building.*

and its composing devices are managed in a common way, offering to the upper layer the functionalities to be managed. The upper layer is the management layer, where several facilities are available:

- Control and Manage devices of the lower layer,
- Orchestrate, aggregate, filter, and preprocess data coming from the infrastructure layer,
- Provide facilities, exploitable by applications, where the former expose abstractions of the IoT devices available in the infrastructure layer.

With regards to the federated cooperative approach, we are referring to a complex CPS composed of several smaller CPSs belonging to different administrative domains (such as different private owners, or a mix of private and public owners). In this scenario, a crucial topic is how the shared CPSs facilities are exploited [9,21]. During the federating process, the domains involved have to sign a sort of agreement that defines the facilities being shared and a typical Service Level Agreement (SLA) used for the cooperation [22–25]. The cooperation system has to avoid SLA's overwhelming limitations defined and agreed by the involved entities. For this reason, coordination and cooperation patterns for service selection are also evaluated, having a look at the approaches adopted in literature, both in the cases of brokered and decentralized ones (see chapter 3). In particular, a complex CPS representing a further step lies in the Smart City research domain. This aggregation of CPSs, as shown in Figure 1.3, represents a new dimension for Smart Cities, that is anyway applicable to an even wider range of environments. As an example, it can represent a Smart Metro Area (composed by an aggregation of Smart Cities), or Smart Country, and so on. A federated cooperation among CPSs enables several advantages:

- increase the amount and diversity of data available for applications running on CPSs,
- enables the sharing of computation resources among CPSs,
- creates an infrastructure enabling the exploitation of Cloud, Fog, Edge, Cloud Continuum approaches (see sections 2.1.1, 2.1.3, 2.1.4, and 2.1.5) without increasing costs for the CPS owner.

All the advantages discussed till now about the cooperation among CPSs make the realization of platforms and applications possible, enabling simplified cross-pollination between service providers, simplifying new developments and reducing the time-to-market for services overall, that translate into advantages for the end user as well. The applications exploitable in a similar scenario are countless. They range from advanced traffic monitoring, management, and driving utilities (see chapter 6, to the realization of enhanced Intrusion Surveillance System based on neighborhood surveillance systems (see chapter 8), cooperative emergency management that supports rescue activities (such as the firefighters' activities as described in chapter 9).

Another interesting aspect related to CPSs and, in particular, to cooperating ones, is the distribution of computation among the available computing elements. Let us make an analogy among a human and a CPS: we can assume that the eyes and the hands of a CPS are represented by the IoTs, while the body and the brain are equivalent to the Cloud. In this way, the CPS becomes a perfect infrastructure where it is possible to apply the Cloud Continuum[2] principles [26]; in particular, we must refer to Fog/Edge and Cloud computing technologies to complete the analogy mentioned above.

Fog and Edge computing [27, 28] are paradigms of computing that

[2]Cloud Continuum represents the paradigms in which the computation is distributed on the whole CPS exploiting Cloud, Fog, and Edge computing facilities (see section 2.1.5).

operate near the peripheral of a system. Indeed, they differ for where (on earth) and in which (part of the) system the computation occurs. The latter makes its elaboration into or near the Edge devices (commonly IoTs, but it is exploited also on gateways, or similar). Instead, the former moves the computation to processors connected in the same area network or into the networking gear itself (router, access point, repeater, and so on).

These techniques, supported by new emergent computing paradigms [29] such as Serverless computing (see section 2.1.6), enable the CPS to be easily exploited by the applications previously under discussion. In this sense, I have made some preliminary investigations [8] with some prototypes to evaluate some assumptions of how the application of the serverless technique simplifies the setup and the re-configuration of IoT devices through simple function invocations. During my Ph.D. activities, I have organized my main research line to create “a new dimension” for an environment made of cooperating CPSs. This ambitious endeavour took the best part of the time I devoted to research activities during the PhD programme, thus, to better organize my studies, I have structured my research activity as follows:

- *Literature review.*
- *Hands-on experience about CPSs.*
- *Methods to distribute computation across CPSs.*

- *Models and applications for cooperation among CPSs.*
- *Definition of a Template framework to quickly transfer the "smarts" to new environments.*

To better describe my research path, in the following each step will be discussed.

In this research trajectory, cooperation schemes and techniques for the selection of resource providers are important issues to be addressed. A literature review to understand the differences among brokered and decentralized Federated *Cloud Service Providers* is available in [30].

Thanks to the European project BEACON[3] [31], experience was matured with typical Cyber-Physical Systems, such as *Smart Buildings* and *Smart Cities*.

Another step of my research path has been devoted to face real problems generated in the interaction with a real CPS (small, bigger, or even born by aggregation of other CPSs), taking care of the computing facilities hosted by the CPS itself and other externally available facilities. In chapters 5 and 6, several use cases have been thoroughly analyzed [6], [32], and [7]. The former presents a system to check the level of people occupancy in a room or, more generally, in a closed environment, exploiting IoT devices for air quality monitoring. The other two works are related to Smart vehicles enhancements, through a constant

[3]In BEACON the federation techniques for networking resources were implemented to distribute resource computation among Cloud Service providers.

interaction with other CPSs available across the Smart City.

Another relevant aspect studied during my research activities is related to the management of Fog/Edge computing systems that absolve complex tasks by creating pipelines. To explore this specific scenario, I first realized a system able to support the computation distribution upon a cloud-based environment [33]. Then, I used the previous studied "pipeline-based compute distribution techniques" to create workflows running on top of the IoT infrastructure [8]. So, Serverless techniques are applied on a Fog/Edge computing scenario to distribute the application's workload on the whole CPS and not only on the Cloud.

Finally, I focused my research activities on *complex CPSs*, by exploring applications and systems operating upon multiple CPSs [9] and [10]. This part of the research is related to the use of the Software-Defined Building approach, respectively, in a neighborhood for security purposes and in an industrial district to enhance fire-fighting systems and to support the firemen. I also focused my attention working on projects where the aggregation of multiple CPSs was extremely important [34, 35], i.e. the #SmartMe and the TOO(L)SMART projects (see chapter 10).

To cover in detail my Ph.D. research activities, the remainder of the thesis is structured as follows. Chapter 2 presents the state of the Art, introducing the technologies, the definitions, and the paradigms useful for my research about cooperative CPSs. Chapter 3 briefly presents a view of the literature review made on cooperation among Cloud Environment.

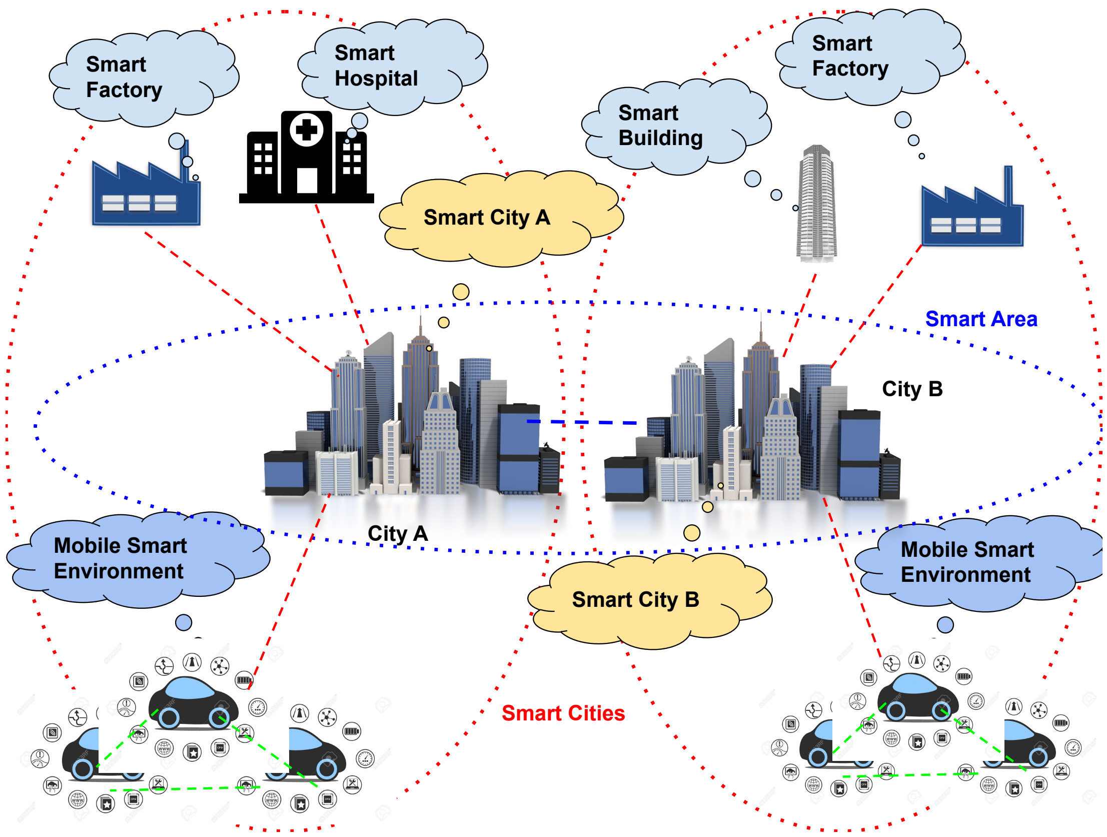


Figure 1.3: *An example of a Smart Area.*

Chapter 4, introduces the tools and the devices most used during my research. Chapters 5 and 6 describe the applications implemented in the context of a single CPS; then Chapter 7 follows, that presents the investigation on serverless paradigms applied on CPS environments. Chapters 8 and 9 present applications on cooperative CPSs. Finally, Chapter 10 describes the template defined for the Smart Cities useful to enable cooperation among CPSs. The last Chapter recaps the work made and outlines some future directions for my research plan going forward.

CHAPTER 2

State of the Art

2.1 Computing Technologies and Paradigms

OVER the last thirty years the IT world has been significantly reshaped by the advent of the Internet. The opportunities to easily interconnect systems have pushed forward both the technologies and the techniques available to the system designers and software designers, but at the same time even the problems to be faced have grown with them. This way, new business opportunities and new computing paradigms has emerged and has changed the face of telecommunications and business infrastructures and at the same time, it has modified deeply the culture of those peoples who have got in touch with it. In this chapter the most relevant paradigms and techniques are presented.

2.1.1 Cloud Computing

IN recent years, IT researchers and developers focused their attention on developing a new computing paradigm, the so-called *Cloud Computing*. The technological progress and the IT services evolution allows to offer to the end-user more and more efficient large scale services leading inevitably to an increase in management costs for the providers of such services. For this reason, IT managers has adopted more effective strategies to meet different needs. Indeed, there is a continuously increasing request of QoS (Quality of Service), and the need of lower costs to manage a growing user basin. This is the context where the Cloud Computing is placed.

With Cloud Computing, users and companies accede to computing resources, storage and software applications without the need to manage and to maintain the physical resources where them are hosted. These resources, in fact, do not reside locally into a user PC or office cluster anymore, but they are allocated within the *cloud*, dynamically virtualized and mapped to physical hosts distributed throughout the network in a totally transparent manner to the end-user. This technology takes the name *Cloud* to highlight the absence of information about where the services are instantiated.

So, when someone refers to the expression "cloud computing", he is talking about technologies that allows storing and / or processing data, using virtualization technologies to offer hardware or software resources through the network following the user's demand. The resource are offered in form of services that are modelled typically with a client-server model. These services are accessible to requesting users through interface (e.g., REST API, GUI, and so on) that serves both to provide Access Control facilities to offered services, and also to hide the underlying hardware and software architecture or features used in the execution. In a nutshell, the Cloud Computing paradigm:

1. it provides an abstraction of the hardware and software technologies that the user has requested, guaranteeing a certain level of reliability and system availability, enabling enterprises to concentrate on business without having the need to assign capital, and human resources, in the purchase and maintenance of hardware and software resources representing the cornerstone of their applications;

2. it provides a reduction of costs to the provider (and thus to the users), because the resource management is carried out in a dis-

tributed manner by algorithms allowing to optimally manage the computational capabilities of the machines, and then to minimize the costs of energy consumption, maintenance and so on;

3. the large companies exploit it in order to create their own private cloud computing system to be used inside company administrative domain. So, it can benefit for all the reasons of reliability, availability, energy saving and maintenance of the above mentioned machines, without facing the risk to fall into the data security and privacy issues.

The *virtualization* is the key concept on which the cloud computing is based. Virtualization consists in a technology that enables running a virtual instance of a resource (e.g., computer hardware, storage devices, computer networks, etc.) in a layer abstracted from the actual hardware, thus enabling features like compatibility, portability and migration of applications for administrators, and security, reliability and performance to the end user.

Commonly, the resources virtualized are entire systems (called Virtual Machine, VM) that runs on a software "layer", called Virtual Machine Monitor, VMM (also known as Hypervisor) that separates physical resources from the virtual environments that require them. The Hypervisor can be executed on the physical systems exploiting the resources of the *host* to produce virtual entities offered to the end-user. Thus, a virtual machine (VM) can be understood as a logical representation of a physical machine (PM) consisting of hardware and firmware. VMMs assign physical resources dynamically so that the virtualized environments can use them, and at the same time, they provide other advantages of virtualization, those are the ability to make backups and status updates, migrations of virtual machines from a host or server to another that, by saving the VM state, will be able to continue its running as if nothing happened.

2.1.1.1 Key Features

IN the 2011th the NIST has released a definition for the *cloud computing* in the article [36] listing its five essential characteristics that are resumed in the list below:

- *On-demand self-service*. The user can request and use autonomously the services offered by the cloud, this means that any human interaction between the user and service provider is required.
- *Broad network access*. Services have to be available through the Network and they have to be accessible through a standard mechanism, thus they can be used on different platforms.
- *Resource pooling*. The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model. Moreover, the users have no vision of how them resources are allocated, they have

only to concentrate on a logical abstraction of the services without having care of how physical and virtual resources are dynamically allocated by the provider.

- *Rapid elasticity*. In order to allow easy system scalability, the virtual and physical resources MUST be provided as fast and dynamically as possible; this could appear to the users' eyes as the providers have unlimited resources.

- *Measured service*. The resource usage is measured to apply the fees for the services used. The resources provided by the Cloud have to be adapted to the typology of services acquired.

2.1.1.2 Categories of Cloud Services

CLOUD computing is still one of the major research topics in the IT world and, even if the research has bring it towards several evolution it is still one of the most used and useful computation paradigms. Therefore a standard that represents its architecture doesn't exist it is possible to classify the main offered services as *IaaS*, *SaaS* and *PaaS*. The Figure 2.1 shows the general vision of the cloud computing with respect the NIST definitions.

Instead, in Figure 2.2.a are shown how the three service-model are connected as function of the control on the system owned by the end-

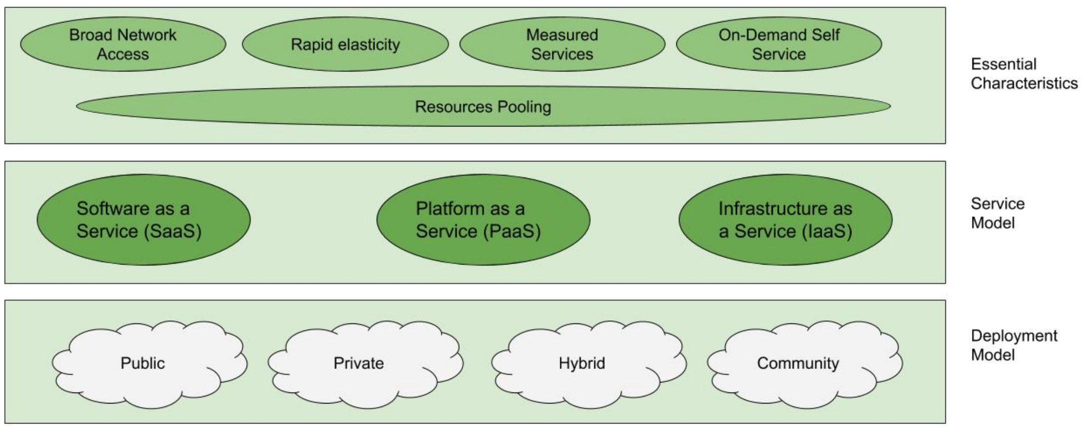
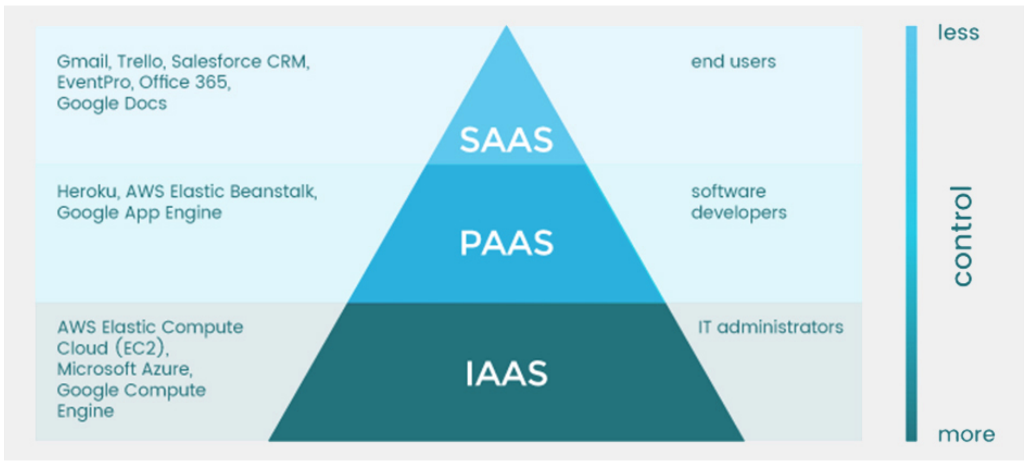


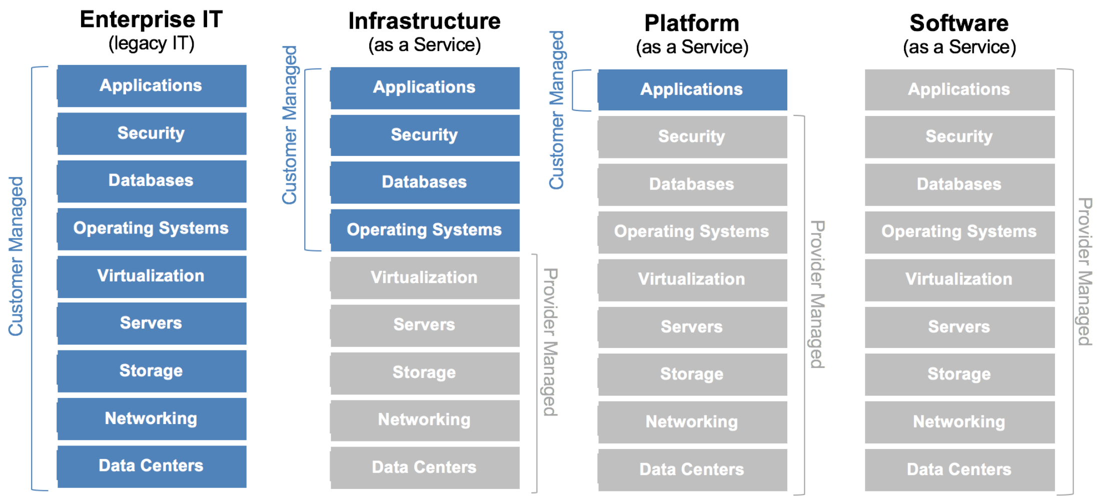
Figure 2.1: *Cloud Computing Overview: Visual Model of NIST Working Definition of Cloud Computing.*

user, in Figure 2.2.b are compared the functionalities are managed by customers or to the providers. At the base of the pyramid there is the *IaaS* (Infrastructure as a Service). The IaaS is the lowest level of abstraction and it provides to the users a virtual computing infrastructure for the execution of users' system (e.g., entire platforms or specific application). According to Cloud Computing *Rapid Elasticity* feature the virtual infrastructure could dynamically grow or decrease as a function of the actual load and of the requests to be served. Moreover the pay-per-use infrastructure includes all of the hardware needed on the network, for example servers, firewalls, switches with certain characteristics.

On the virtual infrastructure provided, the users will install, configure, manage, and use remotely their frameworks or applications, while the Cloud Provider must ensure the service provided by means the infrastructure configuration, in order to allow its use to the customer. The Cloud


Gmail, Trello, Salesforce CRM, EventPro, Office 365, Google Docs
SAAS
end users
Heroku, AWS Elastic Beanstalk, Google App Engine
PAAS
software developers
AWS Elastic Compute Cloud (EC2), Microsoft Azure, Google Compute Engine
IAAS
IT administrators
less
control
more

(a)



(b)

Figure 2.2: *Cloud Computing service Model Overview: a) Service Model and user control in relation [1]. b) Services managed by the customer and by the provider in the service model [2]*

Provider, moreover, had to guarantee the system maintenance and its replacement in case of damage to the machines; the latter is the representation of the characteristic of the IaaS layer called scalability. This way, computing power and storage resources could be added without the users' need to reconfigure everything; so the user could manage its business without the need of estimate the infrastructure costs and jump in the market without the risk of the loss of big capital for the infrastructure that may be unused and thus unnecessary and avoids the user to replace the machines in the future. The second layer of the service model is the PaaS (Platform as a Service), according to this service model the companies offer to the customers hardware and software infrastructures for running their applications without the need to configure them, it is a kind of intermediate service between *IaaS* and *SaaS*. A PaaS provider offers to its customers an useful environment for develops, tests and maintains their applications, oblige them only to accept some restrictions on the available tools, APIs and platforms that are balanced by great scalability and lack of the infrastructure management. The higher layer of the cloud stack is the *SaaS* (Software as a Service). The purpose is to allow remote access, typically via Web, to the services and functions offered by software, the use of which, also in this case, is subjected to the pay-per-use paradigm. The users of this service model, don't need to install anything or to use particular hardware resources on his local machine, to exploit the most of the *SaaS* potential.

2.1.2 Cloud and Fog computing in IoT

IN recent years, big efforts have been put in promoting the Cloud paradigm as a suitable solution for managing IoT environments. Indeed, several methods and techniques have been introduced to deal with the management of a remote and resources' constrained infrastructure. In this context, issues have been addressed in the literature, such as scalability, device accessibility, and personalization of services. To have an extensive insight into the challenges facing the integration of the Cloud and IoT, readers may refer to [37, 38]. In the same perspective, several platforms were introduced to merge IoT deployments within the Cloud management scope [39].

Despite the wide range of benefits the Cloud paradigm provides (e.g., in terms of storage and computing resources), novel constraints in terms of Quality of Service (QoS), dictated by current and forthcoming applications, make the Cloud unfit to meet the corresponding requirements.

To work around these intrinsic limitations, the Fog/Edge computing paradigms have been introduced to push resources, such as storage and compute, to the network edge, to be as close as possible to data producers. The fact that Fog computing nodes are bound to be close to data sources is a key enabler of advanced applications [40] that were not feasible when relying only on the faraway Cloud infrastructure.

2.1.3 Fog Computing

CISCO made the first introduction of the term Fog Computing in 2012 [27]. The basic idea behind the Fog Computing is to place light-weight facilities (that instead have to be executed inside the Cloud) at the mobile users' proximity. This way, a preliminary advantage is provided to the system from the characteristics of the connection implemented: with Fog, the connection towards the mobile users is shorter and with less congestion respect the case of the Cloud where the connection path is longer and potentially heavily congested [41]. Other advantages owned by Fog computing, but according to [42], some of the others are:

- Low latency and real-time interactions,
- Save bandwidth,
- Support for mobility,
- Geographical distribution.

Thanks to the Fog computing model, the data elaboration could be done with a local area network breath, reducing latency into the data and the response delivery; at the same time, it reduces the consumption of bandwidth from the Fog node to Cloud. The Fog model supports, both the mobile than the static devices (even in case of geographically spread devices), the shifting of the computation in a Fog node near the device position. Into the survey [42], the authors state that most Fog computing architectures are derived by the three-layer structure extending the Cloud computing with the introduction of a Fog layer between Cloud and IoT devices. This way, the Fog server could be a generic virtualized equipment with onboard computing, communication, and storage capabilities to provide services to its users [41].

2.1.4 Edge Computing

EDGE computing refers to a technology that moves the computation (and the storage in some cases) from a centralized element, such as a cloud computing resource, to the network edge near the data source. This way, not only reduces the latency suffered by the data transmission from the data source to the place in which the decisions are taken, but it increases the data source owner's privacy avoiding that some data could be delivered in the Cloud maintaining them near the devices. An edge device in this model has the double-role of *data producer* and *data consumer*, so they become not only an appendix of the Cloud but even active elements that could request and offer services to the cloud [43]; this is identifiable with a fine mesh of computational resource abilities [44]. The concept of Edge device has a wide scope that can range from an IoT gateway to a campus network or to an edge compute node co-located with a cellular base station; this makes, as highlighted above, the Edge computing as a sort of panacea for the issue related both to network latency and management challenges posed by real-time services. Moreover, Edge computing addresses scalability challenges by exploiting the hierarchical architecture of end-device and computational resources from the Edge compute nodes to the central Cloud compute resources. This way, the system may easily avoid network bottlenecks towards the central compute location scaling with the clients' number. Often, Edge and Fog computing are confused as it is possible to see in [45]. Still, even if they have the same primary target, the main difference stays in the main focus: the Edge focuses more on IoT peripheral side, while Fog focuses on the infrastructural side [43]. Their differences are shown in Table 2.1, [42], [46].

To complete this overview about Edge Computing a mention is in order for an emerging approach pushed by Telco scientists and by the advent of 5G, which aims to exploit a particular category of edge devices: *the highly capable end-devices (e.g., mobile phones and tablets)*. ETSI promotes this activity, [1],which has standardized the Mobile Edge Computing (MEC) [47]. The ETSI MEC group *Industry Specification Group* (ISG) objective is to create an open environment across multi-vendor cloud platforms located at the edge of the Radio Access Network. This will be accessible by application/service providers and third parties aim-

[1]https://www.etsi.org/technologies/multi-access-edge-computing

Parameter	**Fog Computing**	**Edge Computing**
Resources	Limited	More Limited
Proximity to End-Device	Near End-Device	In the End-Device
Focus	Infrastructure Level	Thing Level
Multiple IoT Application	Supported	Unsupproted
Location of data	Network edge Devices	Edge Devices

Table 2.1: *Difference among Fog and Edge Computing Model*

ing to overcome the challenges related to centralized cloud computing environments, especially in terms of both latency and assurance of higher speeds [48]. The intensive data tasks are pushed towards the edge, locally processing data in proximity to the users, to reach these goals. This way, the mobile network operator can avoid or reduce the traffic bottlenecks in the core and backhaul networks while assisting in the offloading of heavy computational tasks from power-constrained User Equipment (UE) to the edge. The purpose of this initiative is to realize a decentralized cloud architecture that can constitute a technology pillar for the emerging 5G systems, transforming legacy mobile base stations by offering cloud computing abilities and an IT service environment at the edge of the network. Since September 2016, ETSI ISG has dropped the 'Mobile' out of MEC, renaming it as Multi-access Edge Computing in order to broaden its applicability to heterogeneous networks including WiFi and fixed access technologies [49].

2.1.5 Computing Continuum

THE computing technology field after the advent of IoT was pervaded by a plethora of devices and applications that has to be managed often via the same infrastructure. So Virtualization, Cloud computing and its evolution (Fog and Edge, introduced in the previous sections) are employed to orchestrate the resources involved in homogeneous way exploiting as much as possible the hierarchical structure that characterizes this computing techniques. We refer to this as Cloud Computing Continuum. According to Bittencourt et al. [50], this technology is in literature is analyzed in several works that could be categorized in three main categories:

- infrastructure,
- management,
- application.

The management, among the others is most interesting category for our study. Indeed the cooperation among CPSs, even if it is based on federation or not, involves several aspects that are hot topic in literature. Bittencourt [50] identifies a series of works related to the management that are focusing mainly on specific aspects, such as:

- resource allocation and optimization,

- serverless computing,
- data management and locality.

The first one, *resource allocation and optimization*, it is a challenging problem became more complex from the new architecture generated by this new computational paradigms. Scheduling problem (NP-Complete) and techniques proposed in literature are sensitive to the application and the infrastructure characteristics [51]. Advent of IoT and Fog computing has introduced a new way allocate the resources, indeed the Fog computing is expecting to fulfill the requirements not managed by the Cloud platforms, but relaying on them to for the others [52, 53]. About Serverless computing, we will go deeper in the section 2.1.6, we could say that it has modified the traditional cloud-based approach, moving it from the batch-oriented to real-time processing of data. Management of data is an hot topic in literature, anyway only recently the locality of data management has gain relevance being adopted in geo-distributed data centers exploiting Cloud computing continuum (Cloud-Fog-Edge computing) [54–56].

2.1.6 Serverless Techniques

SERVERLESS techniques are meant to offer at the end user the chance to run operations without have care of the server (virtualized or physically instantiated) where the operation are executed, in few word the reader could consider the serverless paradigm as one of the evolution of the Cloud computing techniques that are focused on the execution of tasks.

2.1.6.1 Serverless vs. Function-as-a-Service (FaaS)

MOST investigations around this topic, for long have treated these techniques indistinctly as *Serverless* and *Function-as-a-Service*, FaaS. This technology is referring to the resources virtualization, introduced by Cloud Computing and its evolution, but according to how virtualization activities are managed, we can specifically refer to one or the other. Until 2017, authors confuse these two techniques, referring to them without discriminating whether a full-stack environment, or the execution of a simple function, is required, as in [29], [57], and in a few other works. An interesting definition is provided by Gilkson et al. in [58], where serverless is considered as *"a software architecture where an application is decomposed into 'triggers' (events) and 'actions' (functions), and there is a platform that provides a seamless hosting and execution environment"*, this definition is interesting but it is not useful to distinguish the differences among the two approaches. Several other publications are discussing about the differences existing among the two approaches and even the respective advantages/disadvantages, as in [59], [60], and [61]. In particular, the latter two are respectively discussing about the serverless approach, and about the trends those are rewarding the FaaS approach. At this point, the researcher has started to identify the difference that characterize these two concepts but any well definition was released.

In the 2018 the Cloud Native Computing Foundation (CNCF) [62] resolve the doubts related to the serverless computing introducing the concept of Backend-as-a-Service (BaaS) in the loop. CNFN defines that Serverless techniques as a technology composed by **BaaS** and by **FaaS**, where these two techniques are defined as follow: i) FaaS provides small units of code, representing event-driven computing facilities, where the functions get instantiated and triggered from an external source, typically through commonplace HTTP requests; ii) BaaS, instead, is an approach to handle specific common backend-tasks without any customer's involvement in their management.

2.1.6.2 Fog/Edge computing and Serverless/FaaS

WITH the evolution of Cloud solutions, all major Cloud service providers nowadays have a Serverless computing platform in their offering. For instance, Amazon Web Services (AWS) has AWS Lambda[2] that makes consumers able to run their code without provisioning the infrastructure. IBM as well provides a Serverless platform named IBM Cloud

[2]https://aws.amazon.com/lambda/

Functions[3] which is built on top of Apache OpenWhisk [63]. The same for Microsoft Azure and Google that propose Cloud Serverless plans using Azure Functions[4] and Google Functions[5] respectively thus, their consumers can deploy their functions on the Cloud. With the proliferation of IoT devices, the amount of data generated at the network edge has experiences an immense growth. Examples include sensor data, events generated by IoT devices and gateways, multimedia files such as cameras' images. To make use of this data and provide new services/applications with added values, the incumbent Cloud players are showing immense interest in the Fog/Edge paradigms and promote the Serverless/FaaS approaches as suited solutions to be adopted at the network edge. In fact, Microsoft has released a Fog/Edge platform for IoT called Microsoft Azure IoT Edge [64] that extends the Cloud Serverless paradigm towards the network edge using the containerization technology. Likewise, Amazon and IBM extended their pre-existing proprietary Cloud solutions to the network edge using AWS Greengrass [65] and IBM Watson IoT [66] platforms respectively. AWS Greengrass, for instance, make users able to run AWS Lambda functions on edge devices hence, they can deploy customized applications on the IoT devices.Although opting for Serverless offerings from public Cloud service providers is a widely adopted strategy to deploy applications, one of the biggest issues related to public

[3]https://cloud.ibm.com/functions/
[4]https://azure.microsoft.com/en-us/services/functions/
[5]https://firebase.google.com/

Cloud Serverless solutions is definitely vendor lock-in. Indeed, Cloud providers can impose their own choices for strongly (user-)restrictive configuration settings, e.g., caps for execution duration of functions, or concurrent executions. Moreover, data privacy, sovereignty and, ultimately, control, on owned infrastructure as well, are, in this setting, relinquished and cannot be easily reclaimed back by the IoT owner and/or IoT-hosted service user. Such concerns can be addressed and solved if the Serverless paradigm is deployed using a private Cloud environment. It is within this context that our S4T middleware comes in, by providing an open source solution, based on industry-standard protocols and services, that can run on-premises and without relying on third-party datacenters. An administrator can deploy his/her own self-controlled private Cloud and thus, he/she can have total control over the deployment settings and configurations. In the literature, a number of works target the use of the Serverless/FaaS paradigms in IoT deployments, considering the considerable level of flexibility and efficiency they provide. For instance, authors in [67] proposed a Fog-based Serverless system that supports data-centric IoT services, in particular, they focused their work on a smart parking use case. In the same context, authors in [68] introduced a platform named Kappa that can be used to deploy functions on devices at the network edge.

2.2 Computing Cooperation: Definition & Patterns

REGARDLESS of the used computing technology involved, the computing technology may refer to one of the following definition of operating context, valid for Cloud as it is valid for its evolutions:

- *Public Cloud*: This computing infrastructure is publicly accessible on the Web. Customers pay for the services used, reducing the CapEX assigned normally to the case of *"on-premise"* services. Examples of these services are Amazon EC2, Azure IoT Edge [64], AWS Greengrass [65], IBM Watson IoT [66], and so on.
- *Private Cloud*: This computing infrastructure, in opposition to the previous, has both suppliers and consumers belonging to the same organization. Even if this approach does not produce a CapEX reduction, the advantages, in terms of reliability availability, and the possibility to use the same infrastructure for multiple projects without having the privacy issues introduced by the *Public Cloud*, justify the adoption of this kind of infrastructure.
- *Hybrid Cloud*: This infrastructure is a compromise between the previous two: it is publicly accessible even if there are several limitations, and it tries to take advantage of both typologies.The resulting system is more complex of the previous both from the

realization point of view than from the management aspects. For example, a company might decide to use their private infrastructure to store and manage critical data, and instead to exploit the public cloud services for storing and managing non-critical data and information.

- *Cloud Federation*: This solution is an evolution of the *Hybrid* one. Here, several providers may be public, private, or hybrid systems accepting to share resources and services with each other. *Cloud Federation* is what the IT world is heading, but this target opposes various technical and legal nature difficulties.

The rest of this chapter will analyze the pattern available for the designer that wants to realize a system based on cooperative computation.

2.2.1 Peer cooperation

PEER cooperation pattern is a tightly coupled architecture (see Figure 2.3) in which the Service Providers usually managed with the same technology (e.g.,OpenStack, OpenNebula, and so on), and belonging to the same (or closely coordinated) administrative domain. According to this configuration, each Cloud Manager has the full control over remote resources (e.g., placement control, full monitoring, or VM lifecycle management and migration control). This way, for each Cloud is possible to implement other advanced features as the creation and

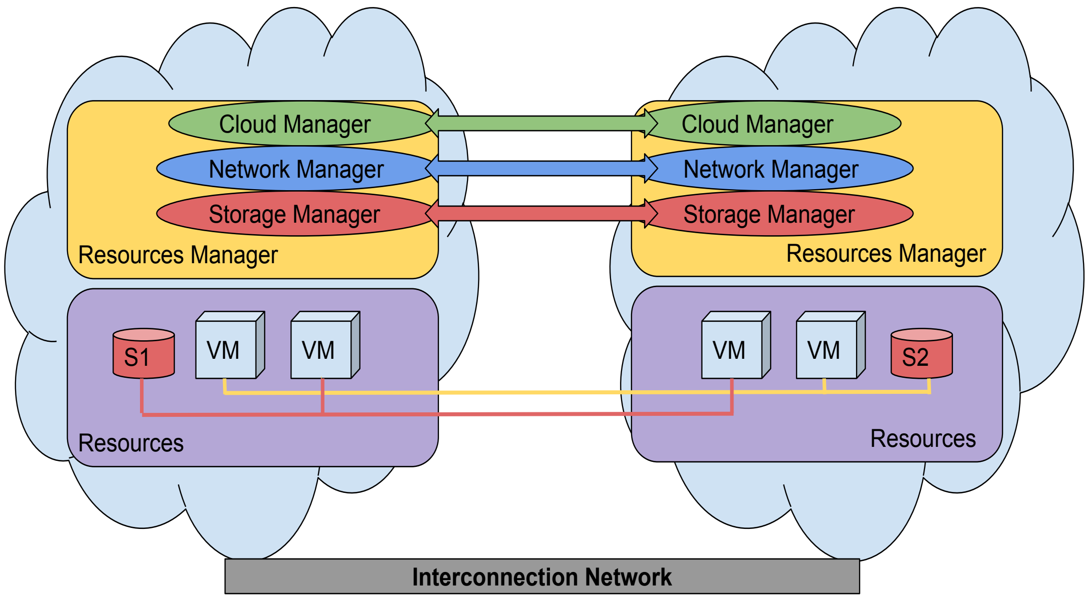


Figure 2.3: *Peer Cooperation Schema*

management of a cross-site networks, the cross-site/domain migration of VMs, High-Availability techniques among cloud instances, the creation of virtual storage systems across different Clouds, and so on.

The interaction between entities manager (as example: Cloud Manager, Network Manager, Storage Manager) is usually made through administration level API's. On top of the CM there could be a SM to simplify service definition, deployment and management.

2.2.2 Hybrid cooperation

HYBRID cooperation pattern is a more loosely coupled architecture (see Figure 2.4) than the peer one. It combines multiple independent Clouds (without have care about if they are Public or Private Clouds) [69] [70] [71].

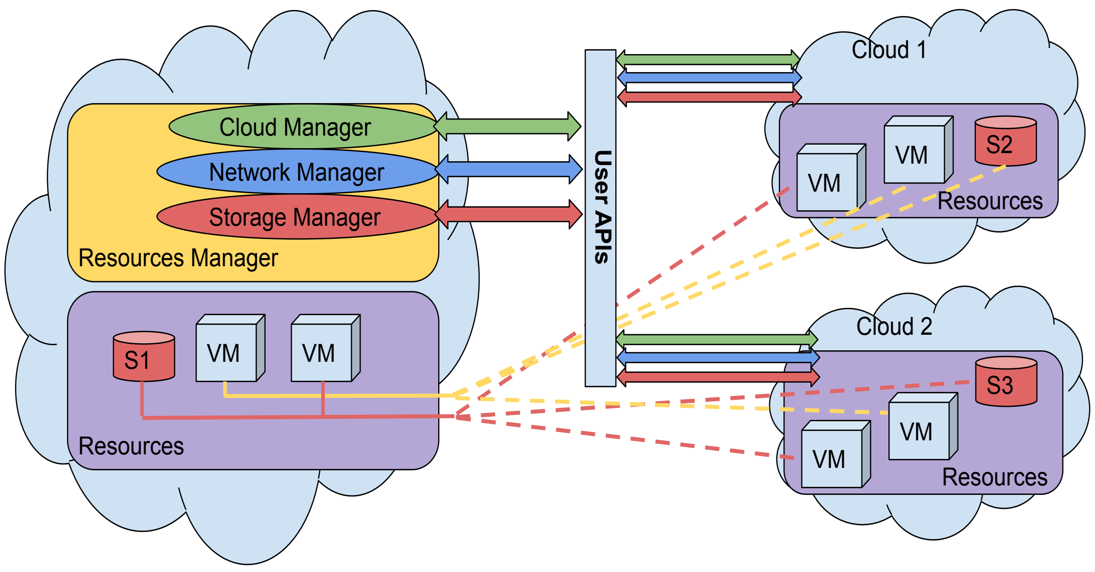


Figure 2.4: *Hybrid Cooperation Schema*

This cooperation approach could also be called the *cloud bursting* model because it combines external resources from remote clouds to extend resources to face the growth of computational/storage needs due to a burst of requests from customers. Due to the different kind of agreement requested by this approach, the management of resources is done through the user's API; this reduces the action available by the Cloud Manager requesting.

2.2.3 Brokered cooperation

THE Brokered cooperation pattern, as it is understandable from its name, is based on a coordinator element called Broker (see Figure 2.5). This element is able to orchestrate several *Public* (*Private* or both) indipendent Clouds [72], [73]. The broker can deploy virtual resources in the managed Clouds according to the criteria defined by the user that has requested the resources (e.g., location restrictions, cost restrictions, and so on), and should also provide networking capabilities to enable the interconnection of different resources deployed in geographically dispersed clouds [31]. This cooperation pattern could be realized even with decentralized brokering schemes where several brokering element interacting each other to increase the resilience of the whole system. This way, we assume that, the cloud broker is a multi-cloud Service Manager responsible for managing application and network services across clouds. Similar to the hybrid cloud federation architecture, this architecture is also loosely coupled, since the broker interacts with the different clouds using public cloud interfaces (user level API's, such as Amazon AWS EC2 API[17] or OCCI[18]) even if these interfaces usually do not allow advanced control over the virtual resources deployed.

2.2.4 Federation

THE Cloud Federation represents a specific architecture in which several Clouds cooperates to constitute a single pool of resources that, according to [3], supports three basic features resource migration, redundancy, and combination of complementary resources. This could be achieved both throughout Horizontal (one level of the Cloud stack)

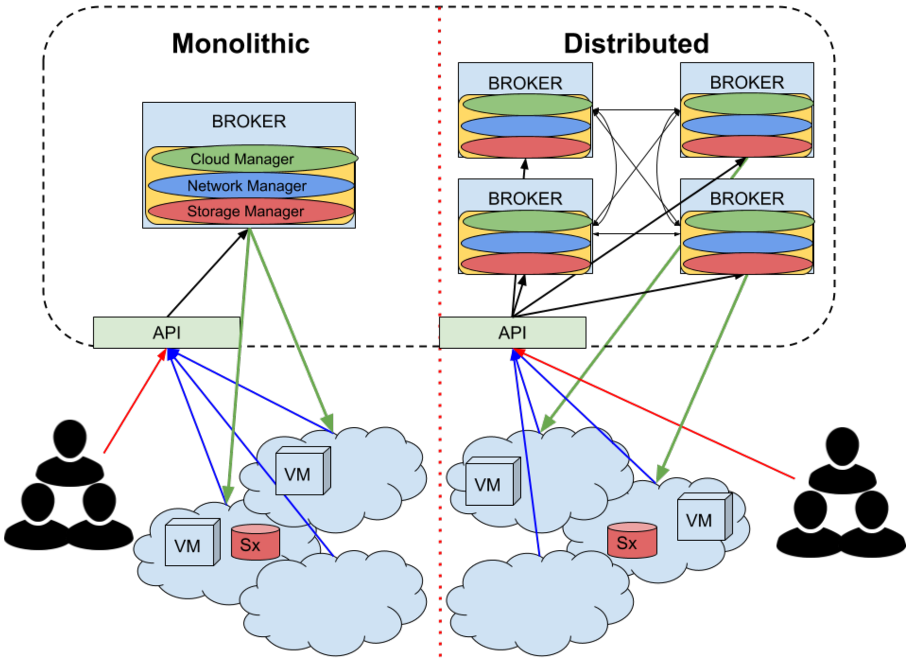


Figure 2.5: *Brokered Cooperation Schema*

and/or Vertical federation (the application stack spans on multiple levels). Anyway, a federation architecture (see Figure 2.6) is another example of loosely coupled cooperation schema combining multiple independent cloud, both *Public* and *Private* clouds. This architecture could be considered a hybrid cloud [69], [70], or a specific case of the federation called inter-cloud federation. This is linked to the cloud bursting model, which combines the existing local cloud infrastructure (e.g., a private cloud managed by a CM, such as OpenNebula or OpenStack) with external resources from public clouds (e.g., Amazon EC2, Digital Ocean, etc.), or partner clouds (managed by the same or a different CM). Similarly to the previous, this architecture is loosely coupled, since the local cloud

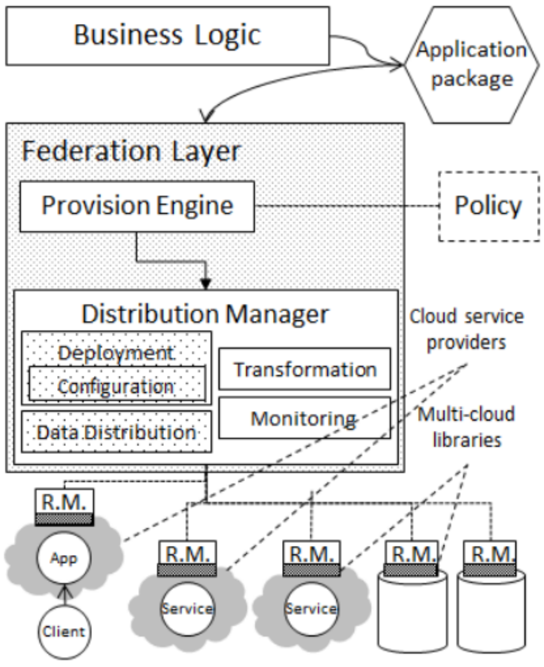


Figure 2.6: *Kurze's Federation reference architecture [3]*

has no advanced control over the virtual resources deployed in external clouds, beyond the basic operations allowed by the federated providers. The interaction between the local Cloud Manager and the various remote clouds could be made via public cloud interfaces (user-level APIâs) and data models (e.g., Amazon AWS EC2 API or OCCI). As in the previous architecture, there could be a Service Manager on top of the Cloud Manager. This architecture is the most appropriate to deploy hybrid solutions: support location aware elasticity and build and deploy highly scalable applications distributed over multiple public cloud providers.

2.2.5 Federation Versus Multi-Cloud

No universally accepted terminology has been defined to identify a Cloud computing scenario where each Cloud service provider collaborates "horizontally" or "vertically" with other Cloud service providers.

However, in our opinion, there are two terms that primarily identify scenarios where multiple Cloud providers interact each other with the aim of improving the service levels provided to users: Cloud Federation and Multi-Cloud. These terms refer to two such scenarios differ both in terms of the interaction between existing Cloud providers and in terms of operating modes .

In a Cloud Federation context, basically a Cloud service provider shares its (currently unused) own resources with other Cloud service providers participating in the same Federation. In this way a Cloud service provider is able to transparently and dynamically enlarge and optimize its own resource capabilities by instantiation of new virtual environments to keep up with incoming user requests. Thus, such a Cloud service provider does not plan to ever deny service or reject requests from their clients, thus keeping a high level of QoS. This interaction is completely transparent to the end-user, completely unaware that her Cloud provider is requesting additional resources from other Cloud providers. Moreover the user is not aware whether his service is hosted by his reference Cloud provider or across multiple federated Cloud providers.

For this reason, we can state that it is reasonable to affirm that the concept of Cloud Federation is *Cloud-oriented* and not *enduser-oriented*. In other words, from a Federation perspective, the users of the system are the federated Clouds operating within the Federation and not the end user that asked for service (IaaS, Paas or SaaS).

Cloud Federation is meant to give additional benefits and new business opportunities to Cloud Service Providers. Conversely, the end-user is actually the consumer of any service in a Multi-Cloud scenario. More in detail, we can define a Multi-Cloud as a user-centric solution where a user is aware about the presence of different Clouds, and either the user or another third party is able to make choices about the selection of the Cloud where services or resources will be instantiated. In Petcu et al. [74] a distinction between the concept of Federation and Multi-Cloud is provided.

Generally, there is a Service Provider (a Broker) which is responsible for the provisioning of services for its users. The Service Provider picks out the services from different Cloud Providers taking into consideration the users' requests. In this scenario there is no collaboration or interaction among the Cloud providers engaged by the user. The Broker performs management, negotiation, deployment, monitoring and migration operations only, in order to fulfill the users' requirements.

2.3 Cyber Physical Systems

Cyber-physical systems (CPSs) are systems born from the interaction occurring among "engineered computing and communicating systems" and the physical world. The Berkeley CPS research group provides a complete definition of a CPS:

"*Cyber-Physical Systems (CPS) are integrations of computation, networking, and physical processes. Embedded computers and networks monitor and control the physical processes, with feedback loops where physical processes affect computations and vice versa. The economic and societal potential of such systems is vastly greater than what has been realized, and major investments are being made worldwide to develop the technology. The technology builds on the older (but still very young) discipline of embedded systems, computers and software embedded in devices whose principle mission is not computation, such as cars, toys, medical devices, and scientific instruments. CPS integrates the dynamics of the physical processes with those of the software and networking, providing abstractions and modeling, design, and analysis techniques for the integrated whole.*" [4].

As it is possible to see from the taxonomy shown by figure 2.7, The diffusion of IoT devices has involved all the everyday life environments, making these scenarios fertile ground for the establishment of Cyber Physical Systems where application aiming to exploit CPS' facilities can easily run producing value for their users. The studies on CPSs began a

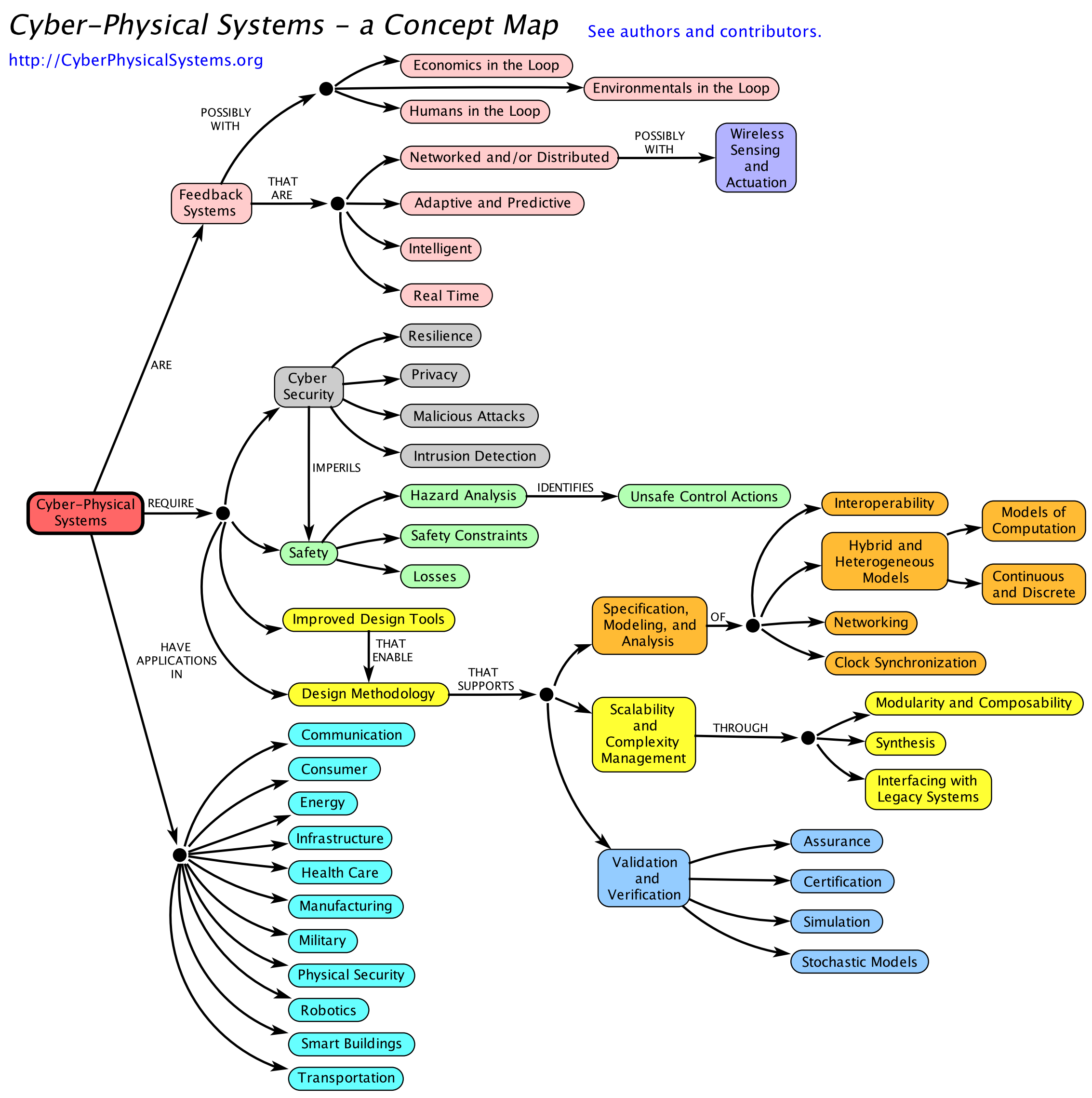


Figure 2.7: *Cyber Physical System taxonomy [4]*

long time ago, and to this day, this continuously evolving concept has not been fully explored, and it continues to be a current issue. The interest in CPS starts in 2004 when the European Union (EU) began the ARTEMIS[6]

[6]*Advanced Research Technology for Embedded Intelligence and Systems*

project, which was interested in the structural challenges pursued by the European industries [75]. Analogously, the National Science Foundation (NSF) since 2006 has founded a research project titled “Science of Integration for CPSs”. Several research entities and Universities, to mentions some “*UC Berkeley,*” and “*General Motors Research and Development Center*” have joined in this project [76, 77] Researchers have focused their studies on CPS, starting from the theoretical foundations, design and implementation, real-world applications, and education. Next, the researches spread out to energy management, network security, data transmission and management, model-based design, control technique, system resource allocation, and applications.

A complete overview of CPSs is provided by Wan et al. in 2011; in their review [78], they analyze the research made in the several fields such as Energy Management, Network Security, Data Transmission and Management, Model-based Design, Control Technique, System Resource Allocation, Applications. Nowadays, several challenges are opened and under analysis, even if uncountable researches were done in this field. Among others, Smart-Cities and more in general “Smart” Environments are the most common scenarios in relation to CPSs. Anyway, we cannot forget, among open challenges, unmanned vehicles, real-time systems, and similar research topics, connected to CPSs, that are central in people’s lives.

2.3.1 Smart Building

THE history of Smart Buildings began in 1981, with the term Intelligent Building coined by United Technology Building Systems Corporation, then implemented into the City Place Building in Hartford, Connecticut [15]. The first examples of Smart Buildings are sort of controlling systems able to manage in an automatic way the building devices (e.g., smoke and fire sensor, ventilation peripheral, heating systems, etc.). More or less around the early 1990s with the first examples of communication protocols enabling the Smart Building's autonomous control systems: BACnet and LonWorks, that respectively through direct connection with a set of certified devices or the mediation of *Neuron chip* can interconnect the devices to the controller to manage the former. In particular,in [79] is presented a comparison of those two systems. The advent of the IoT devices pushes towards Smart Environments and specifically Smart Buildings. As highlighted in [16], it is hard to construct a unique view of Smart Building with a definition commonly accepted. Anyway, citing the European Commission's Information Society is possible to define a Smart Building as a "*building empowered by ICT in the context of the merging Ubiquitous Computing and the Internet of Things: the generalisation in instrumenting buildings with sensors, actuators, micro-chips, micro- and nano-embedded systems will allow to collect, filter and produce more and more information locally, to be further con-*

solidated and managed globally according to business functions and service" [17]. Trying to simplify the previously definition is possible to say that a Smart Building is an integrated system based on IoT and Ubiquitous Computing facilities able to take advantage of a range of computational and communications infrastructure and techniques.

From the literature we have identified several approaches used to study the Smart Buildings: in [80] the authors analyze a set projects concerning Smart Home projects concerning three main topics: Comfort, Healthcare, and Security; in [81] the authors examine two kinds of works from the point of view of efficient energy consumption: Intelligent Building and Small Residential Buildings.

2.3.1.1 Software-Defined Buildings

The vast majority of the papers discussed in the literature present approaches that are tightly coupled to the building structure and didn't take into account the advantages provided by a Software-Defined approach. The research group of Berkeley [82] made some steps in this direction, In particular they define their approach as follow: "*develop software-defined buildings, to shatter existing stovepipe architectures, dramatically reduce the effort to add new functions and applications without "forklift upgrades," and expand communications and control capabilities beyond a single stand-alone building to enable groups of buildings to behave cooperatively and in cooperation with the energy grid*"; one of the most exciting works related to Berkeley's research in Software-Defined Building are related to definition of a Building Operating System as shown in [83], [84] and [85]. As stated above, the main enabling technology behind a smart building are those related to IoT.

CHAPTER 3

Literature analysis: Study on Cloud cooperation approaches and on services/providers selection techniques.

THE best way to identify the right approach to enable CPSs to cooperate among them is understand from literature how the same issues are managed in adjacent contexts, such as Cloud Service Provider (CSP) selection. Indeed, the challenges of building a cooperation-based computing environment are easily compared with those related to the construction of a cooperation-based cloud environment; so my first step is to analyze the literature about this topic. The IT sector's academics are continuously looking for new solutions, technologies, and protocols to construct a valid computing cooperative system composed of computing environments, essentially Clouds, belonging to different owners.

The first part of my literature analysis concerns the publications produced by the European Union (EU) projects and other published works about Federation or Multi-Cloud topics. The key concepts identified from the preliminary analysis of EU projects are used during the literature review made in the second step. This way, a classification of the papers reviewed used in the selection process is provided.

3.1 EU Projects Overview

In the following, a systematic overview of the most relevant EU project is provided. In the end, a summary of the lesson learned and of the concepts adaptable to CPS environments will be provided. Moreover, the knowledge acquired by this analysis is used to filter the remaining literature review, shown in section 3.2.

RESERVOIR: *Resources and Services Virtualization without Barri-*

ers is funded by the European Commission in the Seventh Framework Programme (FP7/2006-2013). The project began in November 2007, and it was concluded in January 2011 [86].

The project aims to extend, integrate, and combine the following technologies: Virtualization, GRID computing, and Business Service Management. This project's results generated much interest in high-performance scientific computing because it enhanced the job scheduling (typical in grid-computing) with the capabilities related to the virtual computing resources. In a few words, *RESERVOIR* made "virtualization-aware" the grid computing. The goals of the project *RESERVOIR* are mainly two: infrastructure and VM placement. Infrastructure was designed to enable the dynamic relocation of a VM in any grid node without having care of location, network configuration, and administrative domains. This project uses federation concepts to define where a VM has to be placed according to the best mapping criteria.

StratusLab [87] is funded by the European Commission in the Seventh Framework Programme (FP7/2006-2013). The project began in June 2010, and it was a 24-months project. StratusLab is meant to provide mechanisms to efficiently exploit computing resources for the system administrators and resource providers. It follows a Hybrid architecture model to leverage the external resources to the requesting entities.

BonFIRE [88] is funded by the European Commission in the Seventh Framework Programme (FP7/2006-2013); it ran from June 2010 to

December 2013. ***BonFIRE*** was a project aimed to design, implement, and manage multiple cloud environments supporting research in the field of the Future Internet (FI). According to [89], ***BonFIRE*** project offers a test infrastructure to the Internet of Services community useful to do experiments about the distributed applications and services. It was based on a federated approach enabling interconnection and interoperation between novel services and geo-distributed testbeds; it owned several geographically distributed testbeds across Europe, offering storage and networking resources. The coordination activities are made by a broker-based Cloud federation model where the broker was the *intermediation point* among the experimenters and the different infrastructures. The ***BonFIRE*** is a broker-based Cloud federation model where a broker component interacts among the user requests, the experimenters, and the different infrastructures. To expose all the Clouds and network features as resources to the user, it provides a common OCCI-based interface used both for interaction occurring with the users and with the Cloud [90].

CONTRAIL [91] is funded by the European Commission in the Seventh Framework Programme (FP7/2006-2013). The project started in October 2010 and ran until January 2014. It was a project focused on *Open Computing Infrastructures for Elastic Services* meant to provide federation of all types of Clouds. The main innovation was related to the introduction of a Platform-as-a-Service layer allowing easy management and deployment of applications and data storage [92]. CONTRAIL and

RESERVOIR have different affinity points. Both are exploiting Cloud Federation. They are part of the same picture in which the former was mainly related to identity management. The latter instead was focused on resource migration among federated Clouds.

VISION Cloud: *Virtualized Storage Services Foundation for the Future Internet* [93] is funded by the European Commission in the Seventh Framework Programme (FP7/2006-2013). The project began in October 2010, and it was concluded in December 2013. VISION was a project aiming to design a scalable and flexible storage cloud architecture that managed massive simultaneous users by enabling the delivery of different types of storage services. The VISION Cloud Storage environment was built using an underneath infrastructure that can exploit worldwide distributed data centers. A data center to be exploitable by the VISION's infrastructure is made by storage clusters containing physical resources with computational, storage, and network capabilities; those are included in the infrastructure for the environment management duties.

MOSAIC [94] *Multi-Modal Situation Assessment and Analytics Platform* project began in April 2011 and was successfully completed in July 2014. It was a project devoted to the Cloud-based application developers, maintainers, and users, allowing them the specifying of the service requirements in terms of Cloud ontologies. This way, the presented multi-agent brokering mechanism-based framework was used to find best-fitting Cloud services to users' actual needs and outsourcing efficiently computation tasks. ***MOASIC*** was designed not only to identify the best-fitting but even to make a service composition if no direct hit is found.

CloudWave: *Agile Service Engineering for the Future Internet* [95] is funded by the European Commission in the Seventh Framework Programme (FP7-ICT-2013-10). The project began in November 2013, and it was concluded in October 2016. *CloudWave* project aimed to provide the cornerstone for the development, deployment, and management of a new generation of Cloud-aware services. Thanks to Cloudwave results, service quality, and service optimization are obtained, the enablement of the Cloud providers to dynamically adapts the services to the environment. To reach this goal, *CloudWave* automatically selects the best adaptation method among the available such as adding more resources to the cloud application or migrating application components [96].

The French project ***CompatibleOne*** project [97] aimed to realize an open-source services broker to orchestrate cloud services into heterogeneous cloud service providers. This project exploited an object-based description model of Cloud resources called CORDS (CompatibleOne Resource Description System); it was based on the OCCI standard. CompatibleOne was meant to offer IaaS and PaaS resources from different providers that were selected according to Service Level Agreement (*SLA*); it made a matching among the SLA requested by the end-users and the one offered by the providers managed.

BEACON [31], [98], *Enabling Federated Cloud Networking*, was funded by the European Commission in the Horizon 2020(H2020) Research and Innovation Programme. It started in February 2015 and ran until October 2017. BEACON aimed to obtain a management Cloud layer based on a federated cooperation schema able to build a solution to federate Cloud network resources where federated Cloud applications can be deployed in an efficient and secure way. The project was based on a brokered architecture to support the automated deployment of applications and services across different Clouds and datacenters.

SUNFISH [99] aimed to solve the lack of infrastructure and technology to integrate computing clouds. It started in January 2015 and ran until December 2017. A secure federation of private clouds belonging to different Entities and following the "Public Sector" requirements was the result produced by SUNFISH. This way, thanks to SUNFISH, it is possible to transparently share data and services without losing security levels. As a side effect, this project improved security in federated "cross-border" clouds opening for Cloud computing market new branches characterized by a high level of privacy and control of information propagation.

SUPERCLOUD [100], *User-Centric Management of Security and Dependability in Clouds of Clouds*, is a project that was funded by the H2020 programme and it ran from February 2015 until January 2018. It proposed a security approach for infrastructure management, both user-centric or self-managed, for a"clouds of clouds" system or as commonly defined Multi-Cloud. The system uses policies running on a specific layer (SuperCloud Distribution Layer) that coordinates the appliance provided by Cloud Service Providers.

FIESTA [101], is a project called *Federated Interoperable Semantic IoT/cloud Testbeds and Applications*, which was funded by the H2020 programme and it ran from February 2015 until June 2018. The activities of FIESTA produced a federated system for the interconnection and the interoperability of different IoT testbeds. The federated architecture is used to collect and analyze semantically before to expose them via FIESTA-IoT System.

The European project analysis has highlighted that when multiple actors are involved in a cooperation activity, it is preferable to intermediate the interaction activities monitoring the operations done according to agreements and constraints defined among infrastructure providers and the cooperative framework/platform. From the analyzed project, some characteristics are recurrent, such as:

- common (or unified) interfaces exposed by the infrastructure and/or exploited by the aggregation system,
- a system to enable the feature/service discovery from the aggregation system,
- well-defined Service Level Agreement (SLA) among the involved parties.

Then in the next section, I will concentrate on publications exploiting SLA based mechanisms of selection, mostly related to the federated systems.

3.2 SLA-based Algorithm Classification for the Cloud Provider Selection

THE literature analysis made in [30] transversally analyzes different CPS selection algorithms. Moreover, I will discuss a subset of this work concerning the publications that account for the SLA as selection policies.

Commonly SLA (Service Level Agreement) represents a policy based on a priori agreements existing between cooperating parties that regulate the interactions. The agreement may involve characteristics (or parameters) or is related to a subset of resources (infrastructure or services) that owner wants to release with some constraints. Figure 3.1 shown the taxonomy related to the publications referring to SLAs, and organizing them as a function of the topic analyzed by the paper, namely: "*Frameworks and Architectures*", "*Algorithms*", and "*Optimization*". This preliminary categorization highlights the works presenting approaches and techniques useful specifically for my investigations on Cyber-Physical Systems. Indeed, the two categories, "*Frameworks and Architectures*" and "*Optimization*", contain design principles and optimization-oriented algorithms. The remaining category includes a broader range of algorithms further categorized in subsections. More in detail, the category "*Algorithms*" is split into three subcategories representing the approaches pursued by the authors; Selection, Multi-Criteria evaluation functions, and Matchmaking.

Furthermore, Figure 3.1 provides a visual categorization of papers in terms of SLA perspective analyzed: i) provider's SLA, ii) user's SLA, and iii) not specified explicitly. This further categorization is related to the perspective of the Federation and Multi-Cloud. As discussed in 2.2.5, the papers related to the provider's SLA analyze the cooperation among CSP from the federation point of view, and conversely, the paper labeled as user's SLA are exploiting the multi-cloud observation point. In more detail, the taxonomy tree contains two categorizations that are shown in Figure 3.2.a. In Figure 3.2.b, details about approaches used in the papers for each taxonomy's category are also shown.

The taxonomy tree, Figure 3.1, also highlights the percentage of an occurrence described in [30]: 65.71% for Selection, 42.86% Optimization, 25.71% Matchmaking, 11% Framework & Architecture, 17.14% Mathematical Models, 8.57% Multi-criteria evaluation; in general Algorithm branch contains 82.85% of publications.

According to the taxonomy tree (see figure 3.1), the works analyzed are presented in the three main groups:"*Frameworks and Architectures*", "*Algorithms*", and "*Optimization*".

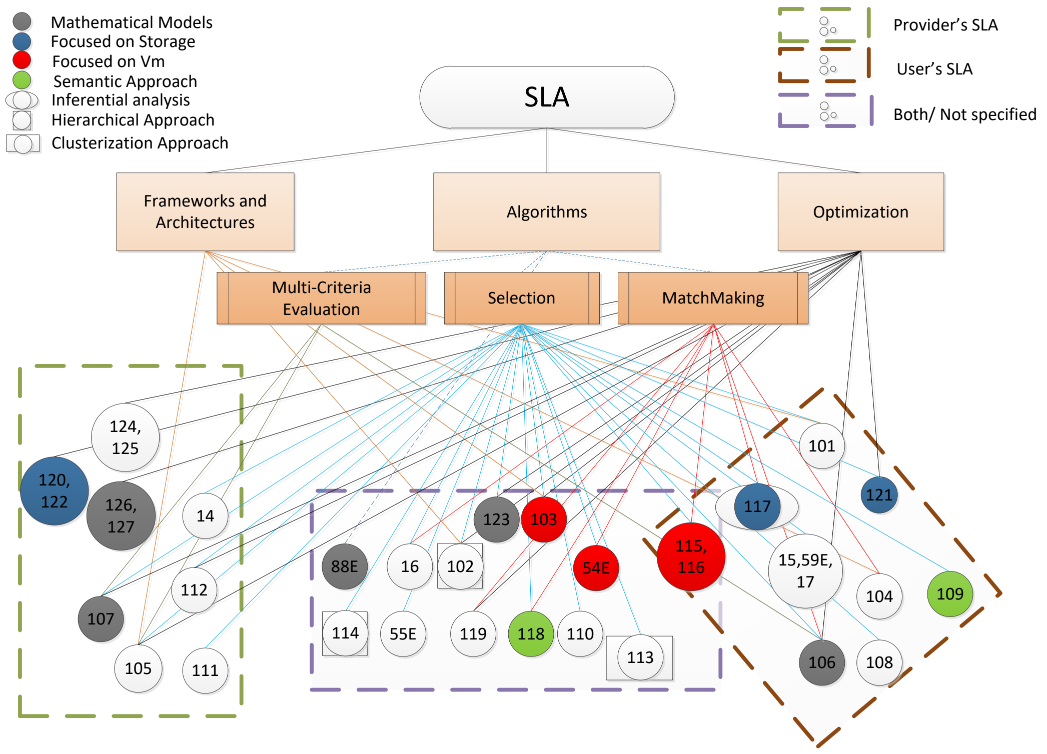
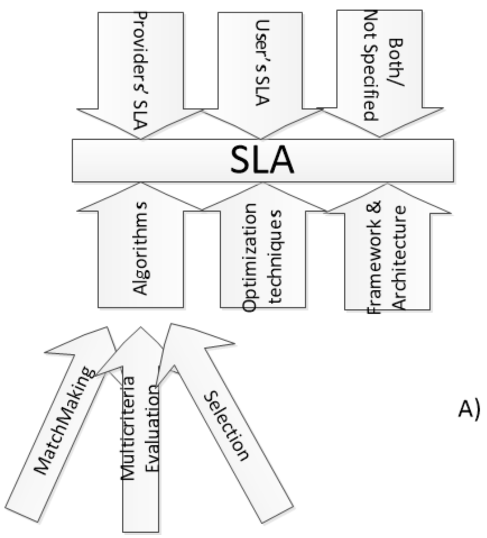
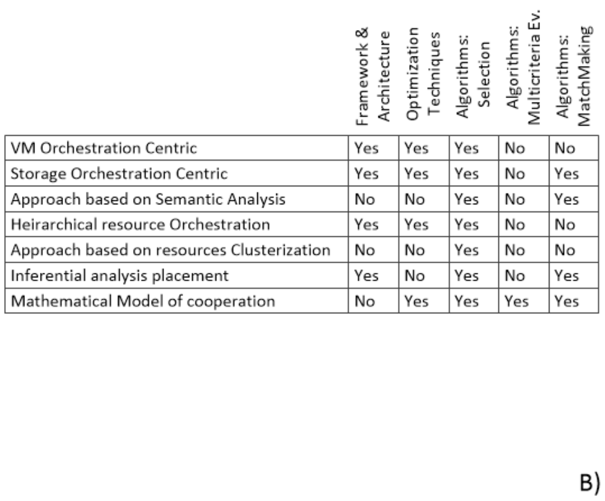


Figure 3.1: *Taxonomy Tree related to SLA-based Approaches.*

3.2.1 Frameworks and Architectures

THIS category contains 11% of the analyzed works. These publications provide as the principal contribution to the proposal of a Frameworks or an Architecture. The first framework discussed is named "*SMICloud*", and it was published by in Garg et al. [102] This is a ***Broker***-based framework that is able to support the user in the choice of Cloud providers' offering that meets the user's requirements. To support the user the framework monitoring the Cloud services, firstly, it measures the **SMI** (Service Measurement Index) attributes of a service, and after it





	Framework & Architecture	Optimization Techniques	Algorithms: Selection	Algorithms: Multicriteria Ev.	Algorithms: MatchMaking
VM Orchestration Centric	Yes	Yes	Yes	No	No
Storage Orchestration Centric	Yes	Yes	Yes	No	Yes
Approach based on Semantic Analysis	No	No	Yes	No	Yes
Heirarchical resource Orchestration	Yes	Yes	Yes	No	No
Approach based on resources Clusterization	No	No	Yes	No	No
Inferential analysis placement	Yes	No	Yes	No	Yes
Mathematical Model of cooperation	No	Yes	Yes	Yes	Yes

B)

Figure 3.2: *a) Taxonomy, b) Approaches used in the analyzed works.*

creates a ranking useful to support the users of *SMICLoud*.

Following the previous approach based on SMI, Subramanian et al. [103] present a framework based again on SMI-index, but that is en enhancement respect the previous. Architecture is based on a Broker aiming to provide an optimal virtual resource placement in a multi-cloud environment, and it also exploits an element called *Service Catalogue*. This catalog is managed by the framework that pairs each cloud service offering with its SMI. When the broker receives a request, and after a preliminary analysis, it identifies a set of cloud that is able to satisfy the user requests. In the end, an *SMI*-based evaluation of the Cloud Providers is performed, and a cost-optimized placement is developed.

Another interesting framework-based solution is presented by Kurdi et al. [104]. Their solution was able to compose services among multiple-clouds, thanks to the combinatorial optimization ability provided by a component called *cloud combiner*, and by the *service composer*. The former selects the suitable set of clouds and composes a “combination list” of the cloud starting from the previously selected set. The latter determine which services can be hosted by each element of the “combination list” of the cloud, so the best combination fulfills the user request is identified. Finally, starting from a set of service files provided by the users, it produces a service composition sequence. The selection algorithm, used by the *service composer*, aims to select a combination of services minimizing the overhead, and therefore maximizing the service performances besides a financial charge reduction.

The last framework discussed is presented by Caballer et al. [105], and it is meant to manage VMI (Virtual Machine Infrastructure). The architecture of this framework is based on a repository called *Virtual Machine Image Repository and Catalog*(VRMC), and on an *Infrastructure Manager*. The former is used to find the available VMIs that are able to satisfy the users’ requirements; the latter is the central part composed of three main components: the “*Cloud Selector*” , the “*Cloud Connector*” and the “*Configuration Manager*”. The “*Cloud Selector*” performs the selection process to selects the best combination of VMIs and Cloud Providers by querying the VRMC. A relevant aspect is the use of a specific language to define the requirements of the virtual infrastructure, the

RADL "*Resource and Application Description Language*".

As **lessons learned** by the analysis of the previous works, I have collected the properties, mechanisms, structures that may be useful in an environment composed of cooperating CPSs. According to [102, 103] a good approach to make easier the management of the services offered by a CPS infrastructure (e.g., sensing and actuation, but moreover the approach is still valid for availability, reliability, accuracy, and other parameters that qualifies the IoT device involved), is the exploitation of "*Indexes*". Similarly to "*SMI*" of [102], an index or a collection of indexes stored and listed in a service catalog [103], improves the cooperation and the quality of the services running on cooperating CPSs. This is perfectly compatible with the definition of a description language as done by [105], and with the architecture used by [104] with two specifics components: one analyzes the services available (in our case the facilities shared by the devices), and the other composes a service, offered as a combination of tasks launched in several CPSs.

3.2.2 Algorithms

3.2.2.1 Multi-Criteria Evaluation

THIS work branch is not wide, but it represents the third most populated category (8.57% of all publication in taxonomy) of the Algorithms branch. The works in these categories try to select the solution that satisfies multiple criteria (e.g., QoS, cost, network optimization, and so on). Some works are mathematical models or algorithmic solutions taking into account multiple input parameters to select the Cloud Service Provider able to better satisfy all criteria.

The first work analyzed is proposed by Duan et al. [106], in which the goal is represented by a multi-factor performance optimization. The solution proposed works upon multiple Clouds, and is structured on multiple layers. The system, called **SDCE**, provides a selection of network and Cloud services in a SDN environment, that compose the services requested inside the managed federated environment. The architecture exploits software-defined paradigm (both computing and networking) and it is composed by five layers. In particular one of these five layers, the one called "Service Layer" provides the features about the service selection and the orchestration, it composes the service having care of the users' requirements meeting the specified QoS. However in the paper the term "federated" is referred to the resulting composite service and not to a federation among several Cloud providers.

Rehman et al. [107] propose a selection methodology that considers a multi-factor performance optimization. Rehman made the Mathematical formalization of the problem through a comparison between the descriptor vectors of the service and the users' requirements. The algorithm will select the service which has the descriptor vector that best matches with the user's requirement vector. This work is relevant because the algorithm is general and can consider any kind of requirements, both functional and non-functional.

Carvalho et al. [108] starts from the *PacificClouds* architecture in which the main focus is hosting (i.e., deploying and managing) applications based on *microservices*. The authors define microservices as a set of autonomous, independent, self-contained services, where each service has a single goal, is loosely coupled, and interact to build a distributed application. This makes an analogy among microservices and IoT resources quite natural in Edge Computing. Similarly to previous works, also Carvalho proposes a selection method to compare the user and the microservice requirements (i.e., constraints) for the selection of the Cloud providers. To do this, the *Simple Additive Weighting (SAW)* method is introduced to rank the Cloud providers that are candidated to host services. The parameters that have been considerd in the service model formalization are the *application response time (execution time plus delay)*, *Cloud availability* and the *application execution cost*. All parameters are based on user-defined thresholds. Although the authors do not explicitly refer to the use of a broker, we can deduce their algorithm can be implemented at a centralized broker.

The **lessons learned** by the study made of multi-criteria algorithms are multiple. In the following, I have collected the interesting properties, mechanisms, and structures applicable to an environment composed of cooperating CPSs. Following the approach of [107], the definition of a generalized algorithm makes easier the inclusion of functional and non-functional requirements into the selection process. This characteristic, the versatility of an algorithm thus realized, makes it perfectly suitable to a CPSs scenario in which technology and devices are extremely variable. At the same time, during the design phase, it is important to look at each actor's requirements as made by [108]. Here a solution with a multi-dimensional comparison (done via "*description Vector*") is performed to accounts for all the requirements considered.

According to [106], when the selection process depends on multiple factors, a solution with multiple layers is needed; this way, the duties separation enables better management of a heterogeneous environment as it is a CPS.

3.2.2.2 Selection

THE algorithm's category *"Selection"* is the most populated (65.71% of all publication in taxonomy); for this reason I will discuss only few of the following [22,23,25] [109–115], an extensive dissertation is provided in [30].

The first algorithm that we are going to describe is proposed by Barreto et al. [23]. The algorithm is meant to work in a federated environment and analyzes the CSPs selection problem from the user's perspective. A user of the federation formalizes his requests in the form of a contract and delivery them to the "*Federation Support*" (FS). This federation's component derives the SLA from the user proposal of contract and creates a lightweight broker to negotiate with the Cloud providers. This broker publishes the users' request on the "Resource Panel", where every federated *Cloud Provider* (CP) can automatically analyze it and send a response message describing the resources offered to the requesting user. The broker fulfills the request, also, as a composition of several CPs' offers. The final decision uses optimizing criteria (e.g., costs) aiming to select the option that generates better benefits for the Cloud users.

In a work of mine, [25], is proposed a brokered solution for Cloud cooperative environments (both for federated and Multi-Cloud environments). The broker manages the application deployment according to the application descriptor. The application descriptor (called BEACON Service Manifest) contains each application component's Cloud Service ARchive (CSAR, defined in TOSCA standard), for each CSAR is defined a geographical area in which deploy it. The broker identifies all the clouds where a CSAR can be deployed (geographical constraints described in the BEACON Service Manifest) and after it selects the cloud where deploy the CSAR. The selection process aims to improve scalability, minimize cost, maximize performance.

Another approach useful for the selection of the best CSP satisfying the user's request is presented by Lin et al. [109]. The solution exploits an indexing system for the Cloud services, so when it has to place a request, it selects the Cloud searching in the indexed tree, the best candidates satisfying the parameters characterizing the request (e.g., cost, type, QoS, instance size and so on).

Farokhi et al [115] presents a solution for a hierarchical management of SLA-based service selection (**HS4MC**). The service selection happens in two moments: in a first phase the SLA are constructed from the providers' requirements (regarding QoS, the requirements are divided functional and not-functional); in the second phase, the system selects the appropriate services that satisfy the request.

This branch of the taxonomy analysis has provided several hints for my study on cooperation among CPSs. Due to the intrinsically distributed nature of the CPS's devices, according to [25] the selection or better the involvement of a device respect another is strongly depending on the device's geographical location. This means that the location of a device is a fundamental aspect to consider. It is clear that for this reason, some mechanism as the one presented in [23], where an ad-hoc negotiation system is activated for each pair of requestor and provider of some functionality. In an analogy with this in cooperation among CPS, a kind of automatism that negotiates the access to some CPS's features from another CPS can become revenue for the owner and encourage cooperation among different CPS owners. Moreover, what said in till now is extendable to a hierarchical scenario as made in [115] for the Cloud Service Providers, which is perfectly matching with figure 1.3. Another (recurring) hints are related to the feature indexing system, as described by [109].

3.2.2.3 MatchMaking

THE Algorithm's sub-category *"MatchMaking"* is the second as number of publications(25.71% of all publication in taxonomy) among the other sub-categories. It contains In this section I will discuss only the most interesting of the following [24] [116–120]; an extensive dissertation is provided in [30].

Jrad et al. in [116, 117] presents their solution based on a broker performing, among other tasks, a MatchMaking algorithm called "*Sieving*".

The *Sieving* algorithm performs a one by one comparison between the user requirement parameters and the "*SLA*" metrics of the *Cloud Provides* which are the two inputs of the algorithm.

Esposito et al. [118] presents a solution based on a fuzzy-inference-based mathematical matching algorithm to accomplish the storage service provider selection. The point of interest of this research is that this algorithm also has care of users' preferences. Another interesting aspect is that this general algorithm could be applied to a single cloud, federation, or multi-cloud scenario. The authors proved that it works both with a central broker or in the decentralized version.

Another interesting solution comes from a project named "*Cloud4SOA*", D'Andria et al. [119] presented the solution used in the project that is interesting because it used a semantic-based approach to do matchmaking among Cloud Providers offers and users' requests. Authors have used a semantic engine to help developers exploiting CPSs' PaaS facilities in an agnostic way. This way, it makes easier the work of the "*Migration*" module that has to semantically translate the application requirements in a new application descriptor compatible with the new PaaS Provider.

The last work of this category is the one presented by Kertesz et al. [24]. They propose a solution to manage a federated environment with a distributed brokered system. Each Cloud Service Provider owns a Broker able to manage its infrastructure. At the same time, they interact with them through a component called "meta-brokering", which is a kind of interface that can interconnect different Brokers together. The selection procedure is done with matchmaking by the meta-brokering layer that receives the users' service calls, checks if the service exists in a kind register called GSR, and selects a suitable Broker able to satisfy the request. The Cloud Federation is built by the GMBS (Generic Meta-Broker Service) that interconnects all the Clouds (via their broker) together.

Several insights have been provided by the analysis of the papers related to the section on matchmaking algorithms. In [116, 117] a reader can found a solution, based on the algorithm "*Sieving*", that is applicable on CPSs to identify a set IoTs useful to satisfy incoming requests. To extend the previous idea, it is possible to learn from [118]. Here, the authors address the need to select in which Cloud Storage Service Provider store the data according to the user's preferences. This is exploitable by a CPS environment to address both the owners of the IoT and users requesting CPS facilities. The work [119] presents a scenario in which a platform enables the migration of the services. For our purpose, it is interesting for the migration concept. Indeed, suppose some task for some reason needs to be migrated from a CPS to another (for example, enhanced intrusion surveillance systems, see [9]). In that case, the migration of the task becomes a central element.

3.2.3 Optimization

THE category *"Optimization"* is the second as number of publications(42.86% of all publication in taxonomy). It contains the works [121–128] but I will discuss only the most interesting, anyway an extensive dissertation is provided in [30].

The first work discussed is due to Papaioannou et al. [122]. Here is introduced *Scalia*, an architecture based on a broker aiming to continuously adapt the placement of the stored data among several Cloud Providers, according to the users' *SLA* requirements and the access pattern to the data. The solution decomposes data to be stored in chunks, and it is also able to reconstruct a complete copy of the data from an "m-subset" of Clouds. The architecture proposed is based on three layers: (i) Engine Layer, (ii) Caching Layer (iii) Database Layer. The first layer is responsible for the selection; it is made by multiple engine components able to manage the selection algorithm independently. The algorithm considers the object access history that represents useful statistics for the algorithm.

The contribution of Hadji et al. [124] is interesting because it performs the selection and placement decisions for the best allocation, taking into account several cost factors. It uses a graphical representation for the requests; the selection algorithm analyzes those. This graphical representation represents weighted vectors representing the application. This way, computational and networking constraints are analyzed to deploy VMs near to the user.

Negru et al. [127] present a mathematical algorithm to optimally select a set of Clouds where it is possible to store a large amount of data coming from different sources geographically distributed. Negru's solution can store data coming from each source in a different destination, and after data are moved in a centralized location to be computed. The algorithm is based on a matrix enabling the analysis according to the cost and latency constraints.

The papers of this taxonomy branch provide several hints and lessons. In [122] is presented “Scalia", a solution relying on multiple engine components able to manage the selection algorithm independently. This approach could be useful to scale the computational charge of CPSs' management engine. Accordingly, to the extreme variability of the application's needs insisting on a CPS when it is opened to the users' interaction (as it happens in a Smart City), the [122] becomes really interesting due to the adaptability of the system.

From [124, 127], two approaches to distributing critical application or compute tasks are provided for a federated environment. The concepts are easily mapped on a CPS in which the applications or the compute tasks are related to the IoTs facilities, both “sensing/actuation” and Edge computing.

CHAPTER 4

Enabling technologies and solutions

IN this Chapter will be presented all the main technologies and tools used during the research activities and to develop the application about CPS described in the next chapters.

4.1 Openstack: an Open Source Cloud Management Framework

The OpenStack framework provides a Cloud Infrastructure as a Service platform using the cooperation of several services, each one dedicated to the provisioning of a specific service. Most of the services are composed of agents who use different plugins to add new features or be compliant with a specific technology. Moreover, every service has its API to expose its functions. Basically, the infrastructure provides three kinds of resources: compute, network, and storage; this goal is accomplished with the projects/services shown in Figure 4.1. The most famous are:

- **Nova** (compute service): manages the Virtual Machines controlling

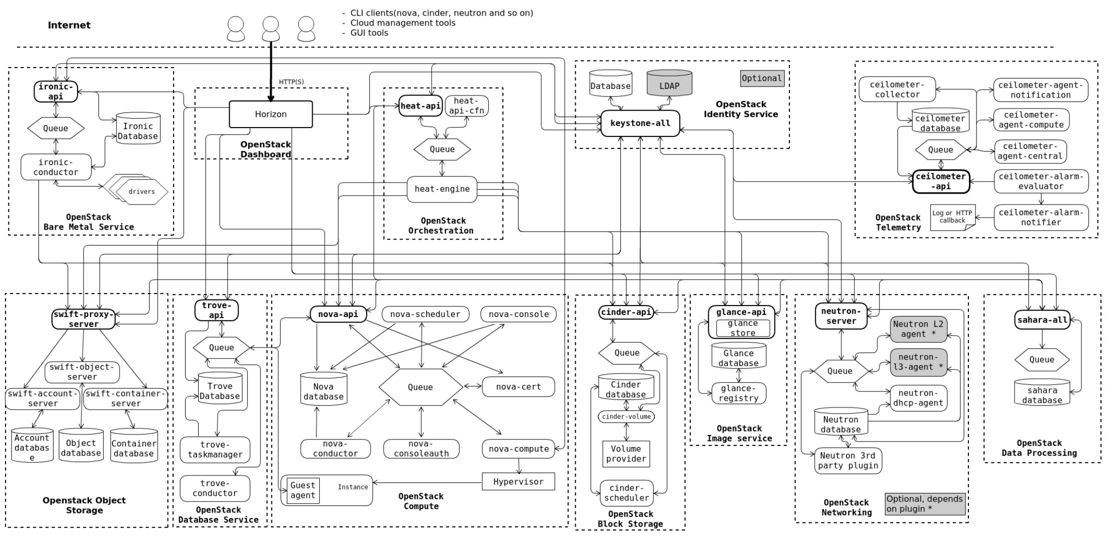


Figure 4.1: *OpenStack Logical Architecture [5].*

and supervising the hypervisors distributed in a dedicated computed node which gave the hardware computational resource.

- **Neutron** (networking service): provides an API for users to define networks and the attachments into them. The agents also provide typical network services such as routing (between VMs and between VM and external network), DHCP, firewall, load-balancing.

- **Keystone** (identity service): provides authentication and authorization service for the other OpenStack service. Every external request (the REST ones) must be validated using a token generated by Keystone according to the role of the one (service or human) who is trying to communicate with the infrastructure.

- **Glance** (image service): stores and provides the images used as a base for the VMs that are managed by Nova.

Those four services represent the main core of Openstack. It means that a minimal Openstack scenario can be risen up with just only Nova, Neutron Keystone, and Glance. A more complete and powerful infrastructure can be set up using these other services:

- **Horizon** (dashboard service): provides a web-based GUI for the administration and tenants user. It uses the REST API of each service to send commands in a more friendly way.

- **Cinder** (block storage service): provides and manages the persistent storage for the VMs using volumes that can be attached directly to the running VMs.
- **Swift** (object storage service): it is pure storage of objects that can be exported using the REST API. It provides mechanisms of redundancy on a scalable architecture.
- **Ceilometer** (telemetry service): provides the monitoring of the OpenStack resources of every service for billing, scalability, and statistical purpose.
- **Heat** (orchestration service): provides the orchestration of the resource using a file (HOT template format). With this service, different virtual scenarios and applications can be configured and monitored automatically, just writing down the file, which describes the resources and their interaction.
- **Trove** (database service): provides a Database as a service that is useful for both relational and non-relational database engine. It is the youngest service introduced into the last OpenStack releases.

The architecture is very flexible, and all the services can be spread in different machines. Basically, this architecture has a dedicated controller node in which there are all the centralized components of the services (see figure 4.2), several compute nodes that can manage the visualization,

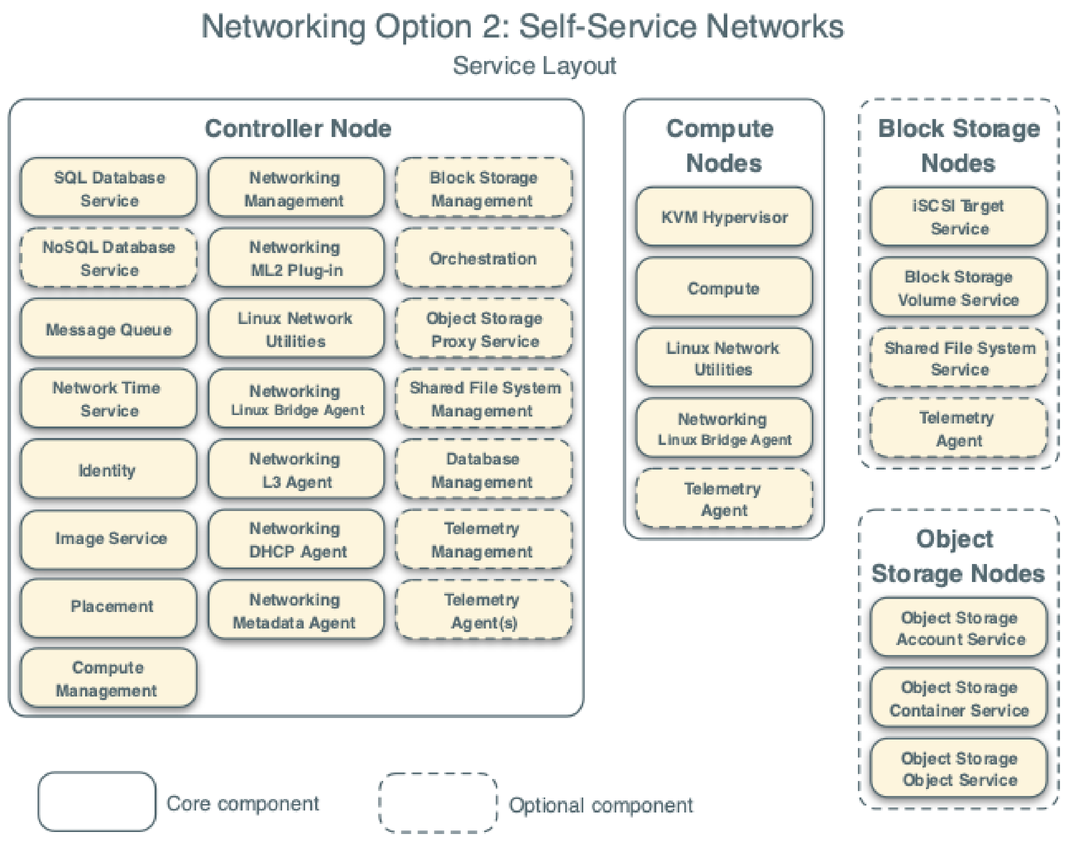


Figure 4.2: *An example of a basic OpenStack Architecture exploiting self service networking features [5].*

a dedicated network node which provides to the VMs the routing for the external network and server storage node servers for storing images and volumes guaranteeing the redundancy. The controller node is the one in which resides also the SQL database for each service and the Message Broker (generally using AMQP) that manages all the messages the services exchange to communicate each others. The compute node should be the ones with the powerful hardware because the hypervisor is installed on them. The default hypervisor is KVM, but it is able to be interfaced with other hypervisor managed by Nova using the right plugin. In relation to Cloud and Datacenter federation, OpenStack only supports identity federation, whereas it supports several mechanisms for segregating cloud resources such as cells, regions, availability zones, or host aggregates, as well as cloud bursting, through either vendor-specific offerings or DIY[1] approaches that can build on community resources, such as Chef cookbooks and Puppet configurations. Commercial vendor-agnostic multi-cloud management platforms are also available. With regards to the OpenStack networking subsystem (Neutron), that we will use underneath in several studies, it deals with all systems administration features for the Virtual Networking Infrastructure (VNI) and the entrance layer parts of the Physical Networking Infrastructure in OpenStack. Neutron allows dedicated static IP addresses or DHCP. It also allows Floating IP addresses to let traffic be dynamically rerouted. Clients can utilize software-defined networking (SDN) advancements like OpenFlow to help multi-occupancy and scale. OpenStack systems administration can send and deal with extra system administrations, for example, interruption location frameworks (IDS), load adjusting, firewalls, and virtual private systems (VPN). Users can use software-defined networking (SDN) technologies like OpenFlow to support multi-tenancy and scale. OpenStack networking can deploy and manage additional network servicesâsuch as intrusion detection systems (IDS), load balancing, firewalls, and vir-

[1]Do It by Yourself

tual private networks. The modular architecture is the main strength of Neutron: one of its most used plugins is the Modular Layer 2 (ML2) one, a framework that supports a wide range of layer two networking technologies. It provides APIs for drivers to interact with, for which two variants exist:

- **Network Type**: VLAN, VXLAN, GRE, FLAT, etc.
- **Mechanism**: Open vSwitch, Linux Bridge, etc.

4.2 Stack4Things: an OpenSource IoT Management Platform based on OpenStack

THE middleware Stack4Things (S4T) [129] is a research project that aims at extending the open-source and broadly adopted Cloud management platform, namely OpenStack, to support the management of IoT deployments as well. The S4T middleware implements several capabilities and features to make IoT deployments involved in an edge-extended IaaS/PaaS Cloud. In particular, the project goal is to provide users the ability to use the IoT devices and their I/O resources (e.g., sensors and actuators) through APIs as the case for standard Cloud resources [130].

This paradigm, called I/Ocloud [131], provides IoT virtualization features, alongside plain IaaS (computing and storage) virtualization. It uses the concept of Virtual Nodes (VNs) that can host the business logic

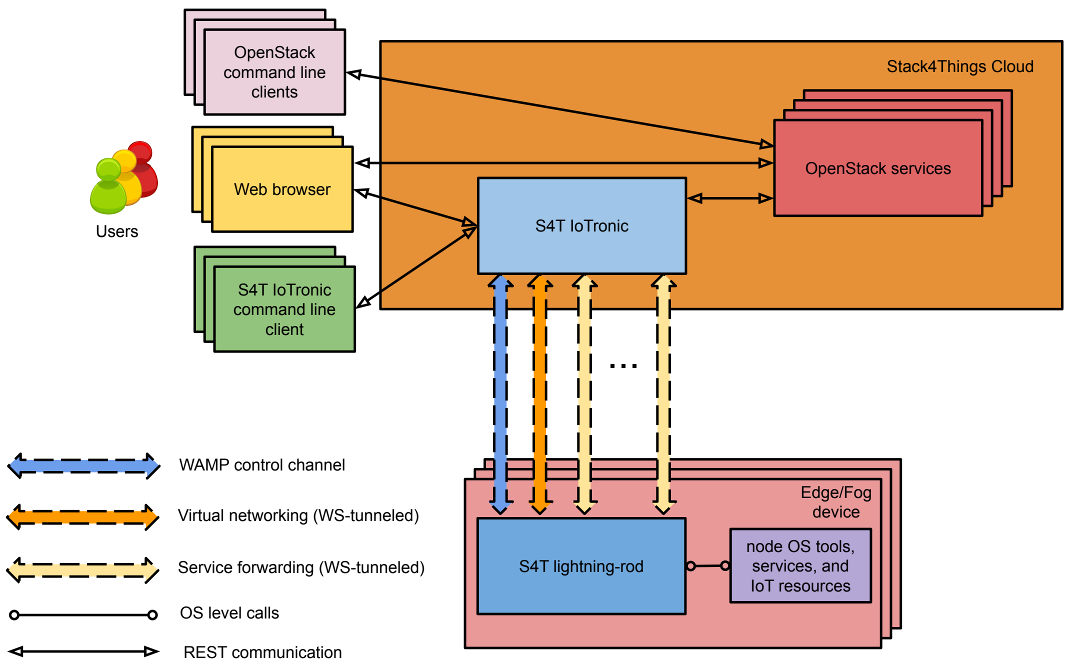

OpenStack command line clients
Stack4Things Cloud
Web browser
OpenStack services
Users
S4T IoTronic
S4T IoTronic command line client
...
WAMP control channel
Virtual networking (WS-tunneled)
Service forwarding (WS-tunneled)
OS level calls
REST communication
Edge/Fog device
S4T lightning-rod
node OS tools, services, and IoT resources

Figure 4.3: *Stack4Things core subsystems.*

and makes use of the attached I/O resources, akin to a real IoT device. The VNs can be deployed either at the datacenter-level infrastructure or on the physical IoT nodes at the edge, depending on the service context (e.g., being time-sensitive or not, computational resources needed, etc.). The S4T deployment design is split between two sides: a Cloud data center (managed by a component called IoTronic) and a set of IoT devices (managed by a component called Lightning-Rod), as shown in Figure 4.3. Regarding the IoT nodes' hardware configuration, we are making use of Single-Board Computers such as Arduino, Raspberry Pi, or Arancino. The latter is the combination of the previous two devices (introduced in section 4.3). They are suited to host a (minimal in case of a constrained device as Arduino) Linux distribution (e.g., OpenWRT, or modified

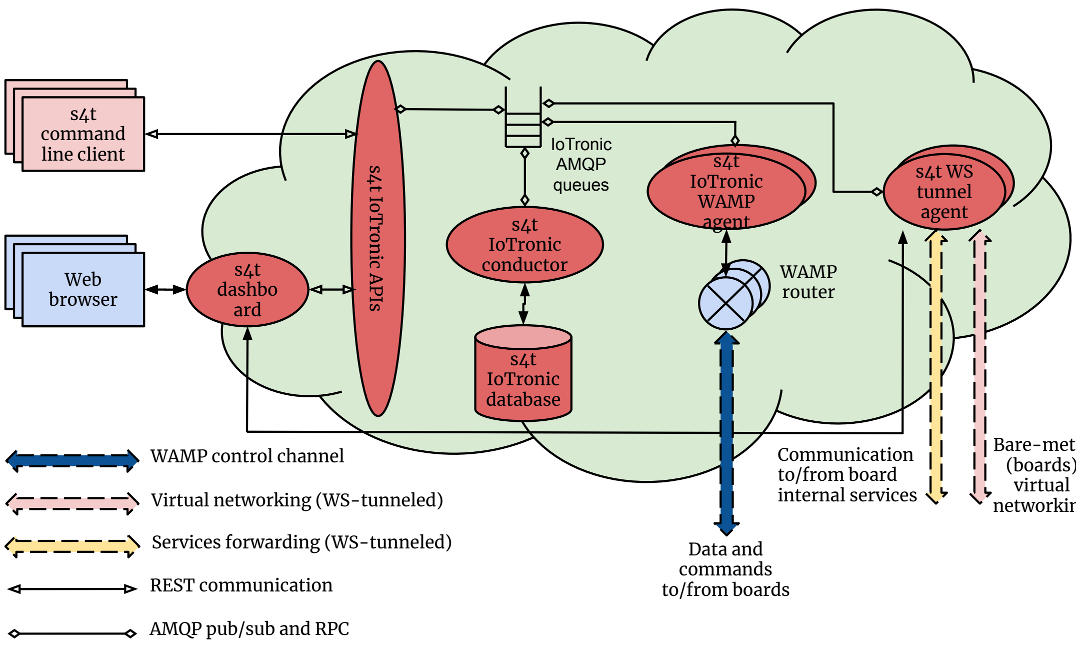


Figure 4.4: *Stack4Things FaaS Cloud-side subsystem design.*

version both of Debian and Ubuntu). Accordingly, they are able to host a set of Linux-based tools as well as different runtime environments such as Python and Node.js that are required by the node side S4T agent, Lightning-Rod(LR).

The S4T subsystem, called IoTronic, runs on the Cloud-side of the system. It is designed concerning the standard architecture of OpenStack services, as depicted in Figure 4.4 (red components) that illustrates the organization of IoTronic. This subsystem's design ensures its full compatibility with other OpenStack subsystems (e.g., Keystone, Neutron, Qinling, etc.). Regarding the node-side, the LR agent represents a key component in the S4T design that links the IoT devices, even when deployed behind NATs or strict firewalls, to the S4T infrastructure where the IoTronic is deployed. The networking middleboxes (e.g., NATs and firewalls) are bypassed thanks to the exploitation of WebSocket (WS) technology used to set up the interconnections between the Cloud and the devices. In particular, S4T exploits the Web Application Messaging Protocol (WAMP), which is a subprotocol of WS, to establish a full-duplex messaging channel used to route traffic streams (e.g., commands) between the Cloud and the distributed IoT devices. Indeed, WAMP provides two meaningful features, namely publish/subscribe (pub/sub) messages as well as Remote Procedure Calls (RPCs). The S4T middleware makes the users able to expose internal services (e.g., SSH, VNC) of an IoT device through the Cloud. For this purpose, the middleware deploy WS tunnels using a reverse mechanism (rtunnel[2]) as the IoT devices that initiate the communications to the Cloud (yellow arrows in Figures 4.3, 4.4, and 4.5). When a service request reaches the Cloud on a specific port, it is forwarded to the S4T IoTronic WS tunnel agent that is a kind of "wrapper" controlling the WS server on which the device is connected, exploiting the wstunnel libraries (Figure 4.5). The same approach based on rtunnels is used to deploy overlays between distributed IoT devices, thus making them able to reach each other and the Cloud-based instances(i.e., VMs, containers) as they were on the same LAN [132]. LR, after the establishment of the connection between a device and the Cloud, is charged with all the operations to be executed

[2]https://github.com/MDSLab/wstun

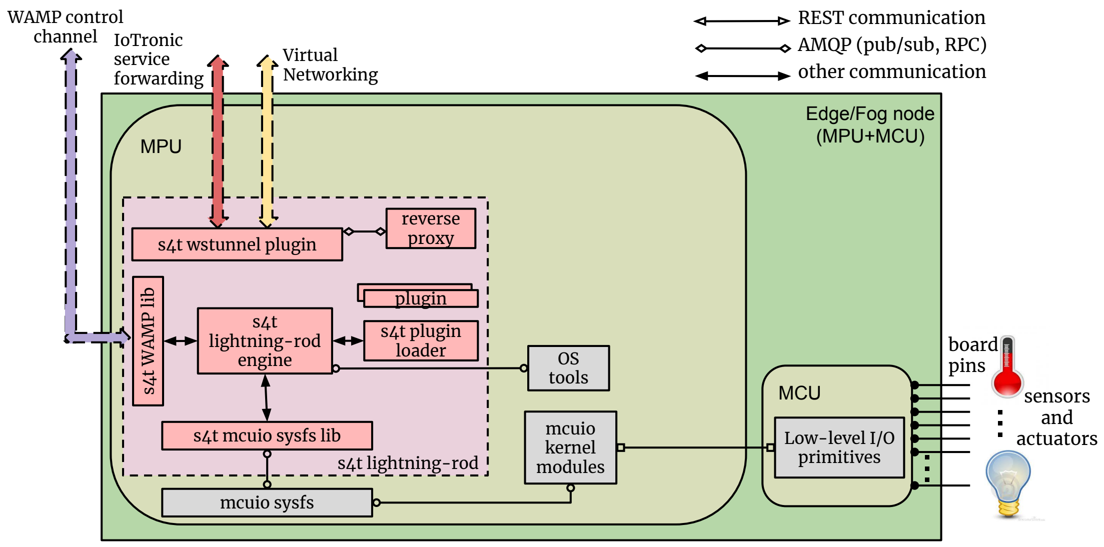


Figure 4.5: *Stack4Things FaaS Edge/Fog-side subsystem design.*

on the IoT devices, and even of management operations as interaction aoocurring among sensors and actuators.

4.3 The Arancino board: a brain for IoT microcontrollers

THE system called Arancino^{TM} is an innovative embedded system produced by an academic spin-off company (smartme.IO). This kind of device can be used in applications ranging from simple temperature sensing to applications in the automotive sector, artificial intelligence, machine learning, neural networks, cloud, big data analysis, predictive maintenance, etc. The board comprises two main parts that, in analogy to the left and right hemispheres of the human brain, can be conceptually identified in a microprocessor and a microcontroller. The connectivity between the two "hemispheres" is Arancino's "corpus callosum". To clarify the concept, while the callous body that transfers information between an Arduino board and a Raspberry board is an external physical cable, that of Arancino is implemented on-board (Figure 4.6). In this analogy, the left hemisphere is dominant for the functions of calculation, mathematical, and logical ability (Data Management, Planning); the right hemisphere is dominant for the ability to recognize faces, spatial abilities and images (Real-time, Interaction). The activity of the two hemispheres is coordinated thanks to the continuous exchange of information that takes place through the corpus callosum, the element that connects them (Shared Memory). The different specializations of the two hemispheres allow them to work together more effectively. The two parts work together and one takes the "control of the operations" according to the cases. The Arancino architecture simplifies cloud-IoT interaction and facilitates the implementation of Cyber Physical Systems. It also takes advantage of edge and fog computing and is perfectly suited to artificial intelligence and machine learning solutions.

These systems feature the joint on-board availability of one (or more) micro-processors (MPUs) alongside one (or more) microcontroller(MCUs), in order to assign workloads and peripherals (e.g.,sensors) according to the unique features of each and every module, specialized for certain duties, whilst making room for the two subsystems to work closely enough to cooperate. In particular, on the MCU side Arancino's host, in their

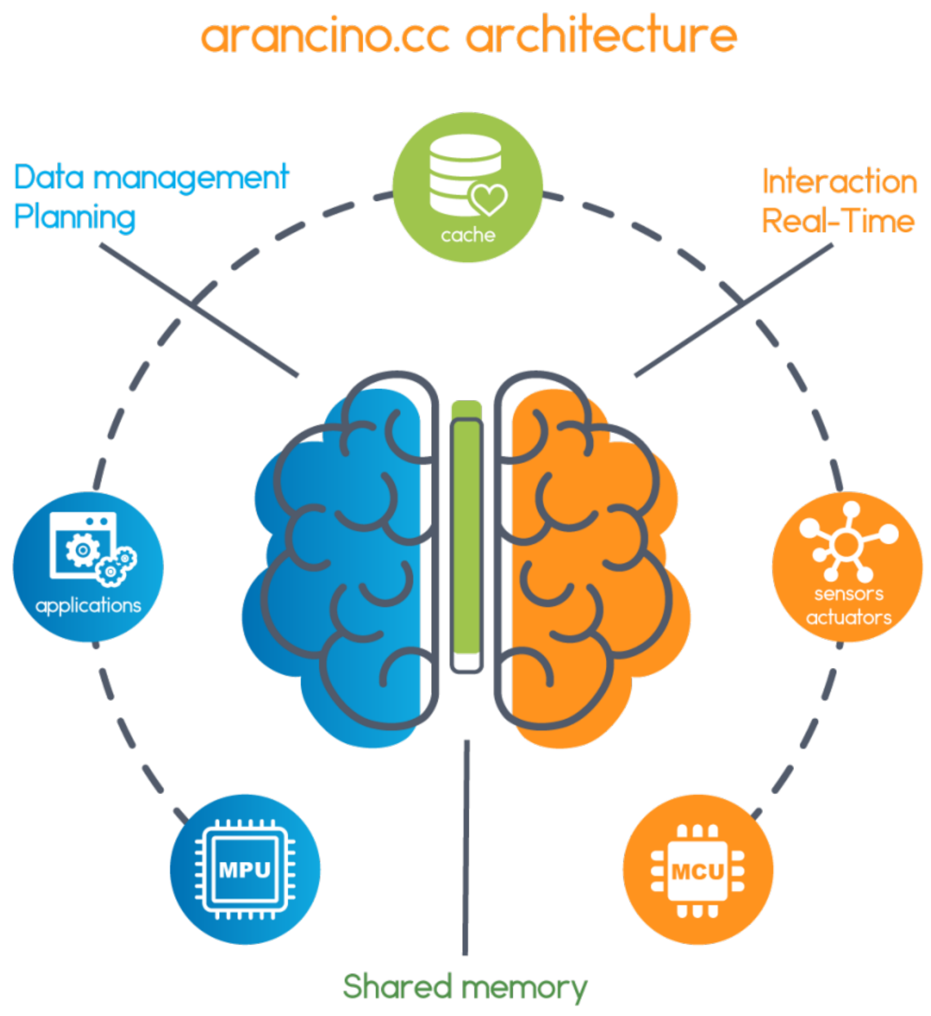


Figure 4.6: *Architecture of an Arancino.cc system.*

base configuration, an Atmel SAMD 32-bit part, which acts as a bridge between sensors and the MPU, where most logic is going to run. In terms of MPU, there is a socket able to host, e.g., a Raspberry Pi 3 Compute Module, making Arancinos, when the socket is populated, equivalent to, (and exceeding) in functionality, a Single-Board Computer, including the capability to run a minimal Linux-based environment. A simple click enables Arancino's microcontroller to communicate with the on-board Compute Module. To extend the system's flexibility, Arancinos are also equipped with a LoRa®RF technology-based transceiver operating at a sub-gigahertz frequency of 868MHz. This way, it features an embedded stack, compliant with a LoRaWAN Class A, able to provide long-range spread spectrum communication, with high interference immunity. Arancinos are equipped with one (or more) cryptographic co-processors (cryptochips), which act as tamper-proof hardware-based key storage and support crypto primitives on-chip. When battery-powered duty-cycling techniques [133] are being evaluated for future iterations.

Following some of the technical characteristics of the system:

1. Microcontroller:
   - ARM Cortex M0+ running at 48MHz
   - 256kB Flash
   - 32kB SRAM
2. Microprocessor:
   - CPU: Broadcom BCM2837 @ 1.2GHz *
   - RAM: 1 GB di RAM LPDDR2 *
   - STORAGE: 4 GB eMMC. *
3. I/O and Devices:

- Up to 32 GPIO
- Up to 6x 350ksps 12-bit ADC with programmable gain
- 1x 10-bit 350ksps DAC
- 12 Channels DMA Controller
- 12 Channels Event System
- Programmable interrupt Controller
- 32-bit Real Time Clock and calendar
- 3 x 24-bit Timer/Counter
- Watchdog Timer (WDT)
- 3x USB Full-Speed 2.0 port
- 2x I2C Interface
- 2x SPI Interface
- 1x I2S Interface
- 2x UART

4. Expansions

   - 2x Arancino.cc Connector

CHAPTER 5

Room as a CPS: Headcount through Air Quality Monitoring systems

ONE of the first step in my research aims to realize a system working in a real environment, or at least, on quasi-real ones.

A study made upon the indoor environment represents the first step in studying a concrete instance of CPS. To do this, interaction is realized with one of the most common CPS (the HVAC) of a Smart Building (see figure 1.1).

In that field, WSN and IoT appear to be the most sustainable technologies for environmental sensing and monitoring, whether it is about size-limited deployments or large-scale monitoring, thanks to the ad-hoc wireless connectivity, inherent scalability, and ease of implementation they provide, as discussed in [134]. Specifically, in [135] the authors proposed a system named *Polluino* aiming to monitor air pollution using low-cost embedded devices (i.e., Arduino boards). In [136], a similar approach is followed to realize an intelligent Indoor Environmental Monitoring System (iDEMS) developed on top of an OpenStack cloud platform. Thus, through access to the, virtually unlimited, resources (e.g., compute and storage) in a Cloud instance, managing IoT-related tasks (e.g., data management, analytics) becomes more manageable, even if the number of devices and data produced increases dramatically. However, although the Cloud approach may count on management systems with ubiquitous capabilities, relaying at all times on the remote Cloud bears several issues (e.g., latency, bandwidth consumption, data storage cost). In this context, Edge computing has emerged as a new paradigm that solves most of the aforementioned issues by pushing selected computing tasks to the network edge as outlined in [137]. Following this trend, authors in [138] exploited an edge-based WSN architecture for Indoor Air Quality monitoring. They introduced a computing data aggregator algorithm deployed at the network edge to decrease the total amount of data sent towards the Cloud platform using a data fusion model. They also proved the efficiency of the aggregation algorithm through a monitoring use case. Yet, they did not tackle the programmability or the future updates required by such a system (e.g., due to forthcoming improvements to the algorithm).

Although several works have proposed Edge-based IoT architectures for monitoring, implementing complex processing on the Edge, while addressing, at the same time, the management of the IoT nodes (programmability, business logic updates, pooling, placement, orchestration) has not been well investigated in the literature. Indeed, considering the high dynamicity of such environments, as an administrator may add/remove sensors, a system for infrastructure management is critical. Moreover, the remote (re)programmability of the devices is crucial to meet the ever-shifting demands of the users and preserve flexibility of the applications.

Trying to fill these gaps and focusing on Quality of Life (QoL) goals, the research [6] was meant to improve the Indoor Air Quality (IAQ), acting on the smaller environment shareable in a building: the "Room". The idea was to create an intelligent system able to maintain the level of IAQ automatically. The solution proposed is called SHIRS: *Smart and Healthy Intelligent Room System*. It is equipped with IoT boards driving specific sensors able to probe air quality and also actuators (even pre-installed ones) to control room-level facilities (e.g., HVACs, or even motorized shutters for windows). The boards act as an IoT data gateway, collecting, storing, and processing air quality data. This way, the sensed data are processed to decide the counteractions to take to preserve the IAQ. In more detail, SHIRS aims to supervise different environmental air parameters inside buildings that may affect people's comfort, safety, performance. In fact, poor air quality can inflict several medical issues known as "Sick Building Syndrome" (SBS) [139]. In particular, Carbon Dioxide (CO2) can potentially cause severe fatigue with nose and throat irritation; Carbon Monoxide (CO) can inflict headaches, impaired vision, nausea, and reduction of mental functioning. In this monitoring, even other indoor air pollutants have to be monitored, such as Environmental Tobacco Smoke (ETS), Ozone (O3), and Nitrogen Oxides (NO2) as well as particles. Moreover, IAQ plays a relevant role in reducing or containing bacterial and viral infections, possibly decreasing the risk for outbreaks. To preserve IAQ, the local system exploits the facilities offered by an ML construct able to identify possible risky situation caused by overcrowding of a room, then the ML construct act as a headcount app, processing air quality data, infer the number of people occupying a room, and taking countermeasures available(activate ventilation system, opening windows, activate a light signal warning the people to reduce the usage of the room, and so on). The solution identified is based on the Edge Computing facility; it runs the headcount application in-place, on the IoT board exploiting its MPU[1] to execute the ML-based workloads. SHIRS also exploits the full runtime personalization and (re)-programmability of the device provided by robust mechanisms for remote code injection provided by Stack4Things(S4T) [140], [141], an I/O cloud-driven [142] Things-as-a-Service platform.

The SHIRS's architecture is composed by three layers (see figure 5.1):

- **The device layer** is where the raw data is processed and exposes the Middleware layer's functionality to manage and coordinate device activities. The device layer's activities rely mainly on powerful single-board computers such as the Arancino boards4.3, that are equipped with both Microprocessor (MPU) and Microcontroller (MCU) units. The MPU of this device enables it to host a minimal Linux distro (e.g., OpenWRT) together with a set of Linux-based tools. The MCU instead enables the real-time interactions with the sensors and actuators they may host. The device layer's component is the board-side counterpart of the IoT management system used, Lightning-rod, a component of S4T.

- **The Middleware layer** is where the infrastructure activities are made mainly via a component called Stack4Things 4.2. The SHIRS's Middleware Layer exposes and leverages the Device Layer's capabilities to the upper layers. S4T is fully compliant with the Open-

[1]MPU: Micro Processing Unit.

Stack Cloud platform. This layer provides: ***i)*** Authentication/Authorization, ***ii)*** Remote Customization and access, and ***iii)*** Virtual Networking.

- **The application layer** is where the applications run, providing graphical IAQ indexes representation. Simultaneously, it produces a set of functions for the development of event-based applications also be distributed to manage the building rooms. It is also charged with the presentation of data to the end-user.

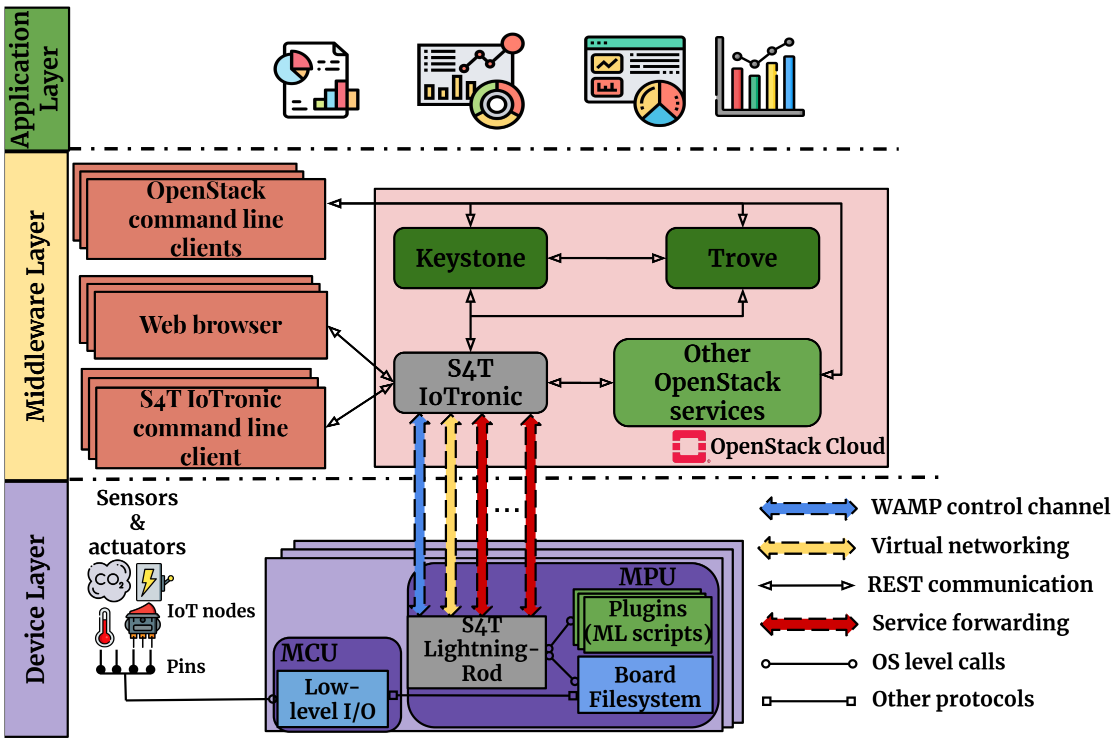


Figure 5.1: *SHIRS Architecture [6]*

The experimental part connected to this study is related to monitoring a room used for meetings, laboratories, seminars, and presentations of various kinds in our Engineering Department at University. This is to validate the SHIRS's functionalities in a real scenario, establishing a starting point for applying our infrastructure and methodology. To realize the Device layer as said previously, we used an Arancino MPU-MCU based board that is able to satisfy at the same time, the requirements due to sensing-derived needs and the computing-related needs. The board has two slots for host "click standard" sensors, but in the cases in which more than two sensors have to be managed, each pin can be used as a traditional input. Moreover, the board is provided with a WiFi module to simplify the device's placement by avoiding the Ethernet connectivity. The sensors used in our experiments are used to perceive gases in the air; in particular, I refer to steam water, CO, dust, CO_2, and temperature and atmospheric pressure. The first is mostly generated by the human presence in a room (i.e., breath, sweat), even the CO_2 is due to the human breathing; the correlation of CO and CO_2 produced useful information related to the combustion (e.g., the smoke of cigarettes). Moreover, a dust sensor can help in identifying the kind of activity, as any person entering or leaving a room naturally lifts dust in the air. Finally, temperature and atmospheric pressure are used to measure and identify the perceived data's deviation due to environmental conditions. A schema of the Edge node contained on the Device layer is shown in figure 5.2.

The IoT node is managed by the "Middleware layer" that is able to remotely inject new code/plugin on the device, modifying the behavior of

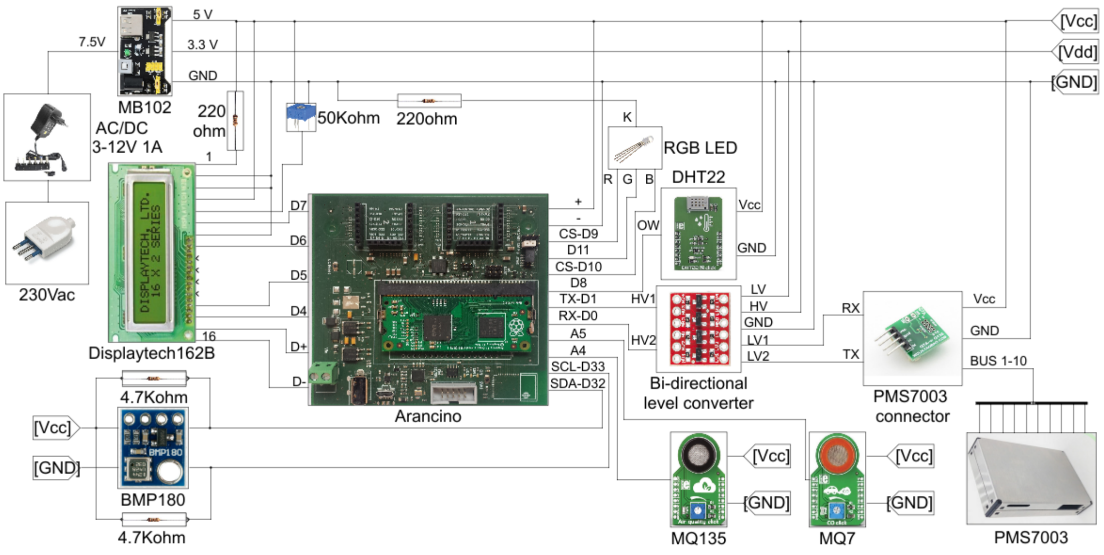


Figure 5.2: *Data acquisition and transmission schema [6].*

the computation element. This activity is Cloud-driven, and it enables the injection of custom Machine Learning code in all the IoT nodes managed by the SB at runtime. Thanks to the remote management facilities, an administrator may modify the behavior according to the need, so it is possible to contextualize the device to the environment characteristics. Moreover, this mechanism enables the IoT devices to run multiple parallel tasks because the plugin injected is an independent (either synchronous or asynchronous) process spawned by the LR agent on the IoT device. For our purposes, we inject an ML code based on opensource libraries and frameworks (e.g., Keras, Tensor-Flow) that is able to recognize the number of persons present in a room from the data perceived by the sensor shown in figure 5.2. The selection of the ML model to use in this application was made through performance comparison made on

Models	**Acc.(%)**	**MSE**	**Hyper-parameters**
MLP	56.4	0.127	$Optmiz.(Adam), Act.Func.(ReLU), HL(4), Neurons([25, 20, 15, 10]), Regular.(0.01)$ $LR(0.001), Loss(categ.\ crossentropy), Epochs(500), Output\ function(softmax)$
RF	82.3	0.454	$Rand.\ state(0), Criter.(gini), Max\ depth(2), Max\ features(auto), Estimators(100)$
SVM	96.1	0.235	$Kernel(rbf), C(1.0), Y(auto)$
KNN	97.3	0.037	$N.\ neighbors(1), Weights(uniform), Alg.(auto), Metric(minkowski), n\_jobs(1)$

Table 5.1: *Performance metrics comparison [6].*

our dataset, using 70% of the training data to evaluate the model and determine the optimal values to accurately represent our physical system. The comparison was made on four standard classification algorithms for this kind of problems: Multilayer Perceptron (MLP), k-nearest neighbors (KNN) algorithm, Support Vector Machine (SVM), and Random Forest (RF). We *tested* all the models on the test data (30% of the whole dataset) comparing one another in terms of accuracy, thus evaluating the ability to *generate* acceptable predictions when run on previously unseen samples. Finally, the model, here injected on the device, was able to reflect the accuracy reached by the tests carried out on the Cloud side, as shown in Table 5.1. An example of the data collected and presented by the system is shown in figure 5.3.

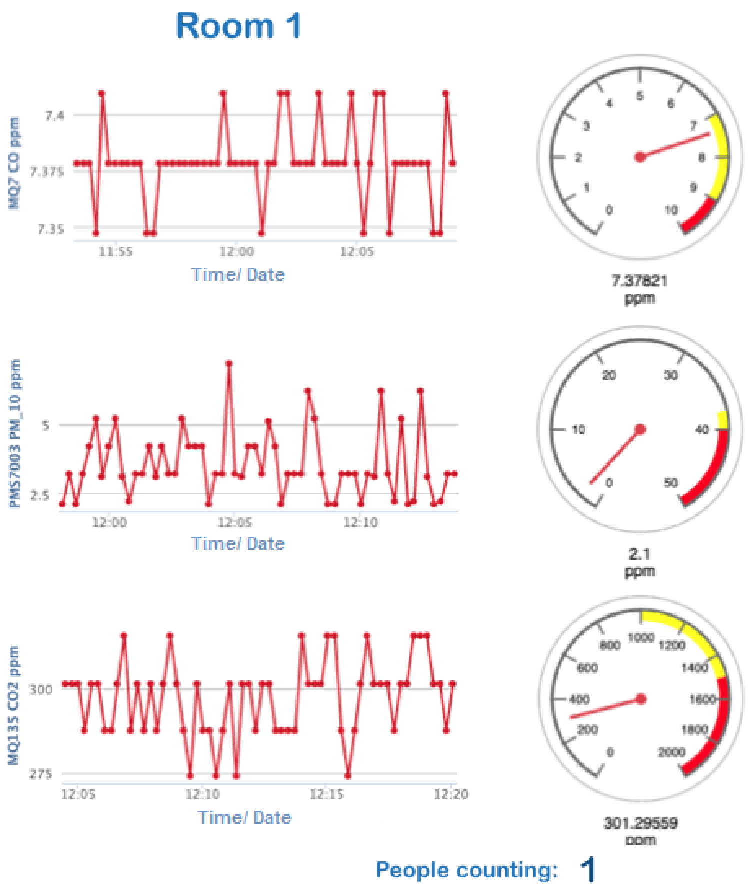


Figure 5.3: *Dashboard for data presentation [6].*

CHAPTER 6

Vehicles as CPSs supporting drivers through interaction with city infrastructure.

AN interesting work on CPS is the one related to the vehicles. A "*vehicle*" is a system rich of IoT devices that aggregates on a central control unit the data perceived by the peripheral devices to monitor the vehicle's status, such as the security system's status, vehicle's speed, motor's rpm, consumption estimation (fuel and CO_2 emission), and so on. Until now, a vehicle is a totally isolated CPS.

In this research step, even if we can not yet talk about first full cooperation among CPSs, I have presented some algorithms acting on a similar scenario to enhance the assisted drive. These works' cornerstone is a change of perspective because the vehicle is not considered an isolated environment. However, it is considered a CPS moving inside and through environments: compounds, cities, and countries. In the studies described in [7, 32], the environments did not represent CPSs cooperating with the vehicle but are considered a data source is offering useful services to the vehicles.

In particular, a reader can distinguish a two folds motivating scenario for this research. On the one hand, the intent to address the issues related to smart mobility infrastructures and services targets traffic monitoring, mobility as a service, route planning, autonomous vehicles, supply chain management, multi-modal transport, and generally any Intelligent Transportation System (ITS). On the other hand, the smart environment mainly focuses on global warming, acid rain, air pollution, urban sprawl, waste disposal, ozone layer depletion, water pollution, climate change, recycling, clean energy, and many more.

These studies are moving into the intersection of these two broad areas, intending to leverage the smart city infrastructure to implement a green/environmental smart mobility service that addresses both mobility and environmental issues.

The basic idea is to implement a system able to identify a route planning application that considers the information collected by both the smart city facilities and the vehicle. The proposed solutions are based on heuristics based on the exploitation of existing optimization algorithms already implemented by the cruise control system on-board the vehicle. The high-level control logic interacting with the Smart City facilities can help the driving system reduce gas emission and fuel consumption through a travel time balancement, or vice versa, reduce the traversal time balancing the emission and fuel consumption.

The architecture exploited by the two works is more or less reducible to the figure 6.4. The Smart City(SC) is equipped with a traffic management system based on a smart traffic lights system, connected to SC computing facilities for collecting, storing, and processing data, thus controlling traffic by acting on traffic lights and other devices (dynamic road signs, toll stations, smart parking devices, and so on). SC identifies each segment of a street delimited by two traffic lights as an oriented segment, and contextually circle, cross, and an intersection with other streets not regulated by traffic light are represented by the SC with a set of oriented segments sharing a portion of their path. A proper id uniquely identifies each simple road segment between two traffic lights.

A vehicular node (a generic vehicle) is modeled with an automatic gearbox and an advanced Cruise Control system, by which setting up speed and other vehicle travel parameters. Moreover, the vehicle interacts with the Smart City Infrastructure(SCI) through the specific system's interactions at the core of the approach. The Controller (see figure 6.4) is the component in which the algorithms run according to what is defined in [7, 32]. It is hosted into the vehicular node (a smart-board or similar) interacting both with the on-board Cruise Control and with the SC through a wide area networking system as could be a Metropolitan Area Network. The connection manager communicates with the Smart City Computing Facilities. It uses both synchronous and asynchronous messages to query the SC about "static route parameters" and "traffic-lights timing" related to the whole vehicle's path and the latter vehicle's position in the path established. So the data received from the Smart City become the inputs for the Route Control component, which is the component where the algorithms run. It produces the navigation speed to be used by the vehicle. The Cruise Control system receives from Route Control the computed values to set up its cruise speed, assuming an automatic gearbox vehicle, enforcing factory optimization on fuel consumption and CO_2 emissions. To reduce the impact of unexpected events, the vehicle frequently interacts with the SC.

As a result of the two works about vehicular CPS, a two-level intelligent cruise control system is obtained. This system proved quite effective in a case study of a conventional c-segment car of the internal combustion

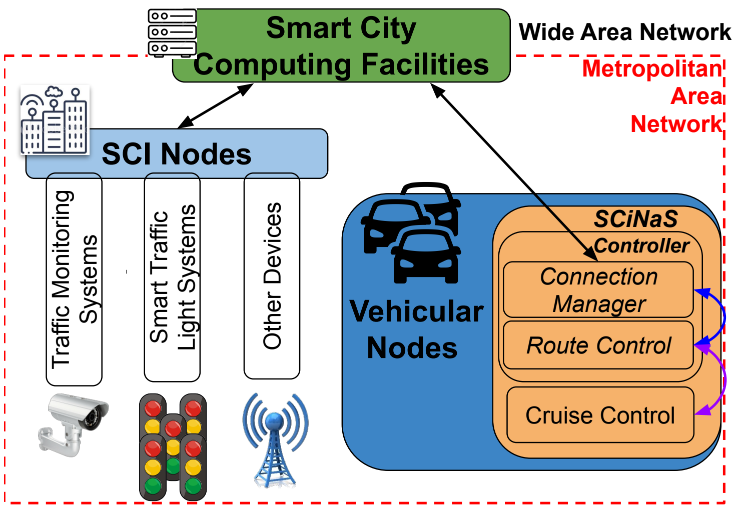


Figure 6.1: *Architectural overview of vehicular node and Smart City presented for SCINaS [7].*

engine in terms of emissions, travel time, and fuel consumption.

6.1 SCICC Algorithm

UNTIL now, the advancement of these two works was parallel; the difference indeed stays in the algorithm used to regulate the vehicle's city traversal. SCICC, *Smart City Intelligent Cruise Control*, is oriented to the reduction of fuel consumption and CO_2 emission minimization. In Algorithms 1,2 are presented the two main functions used by SCICC [32], to guide the cruise control in its duties.

Starting from the distance to the next traffic light and from the data about the traffic flow, SCICC applies the Algorithm 1 to assess the traversal speed aiming to implement the virtually continuous green wave.

When the vehicle is in proximity to the current segment traffic light, the SCICC system also considers the information of the next segment in the evaluation. The Route Control Algorithm 1 interacts with the Cruise Control through the functions $Decrease\_Speed(currentSpeed, newSpeed)$ and $Increase\_Speed(currentSpeed, newSpeed, activate)$ returning a data entry containing i) the vehicle final speed according to the request ($newSpeed$) and the acceleration (deceleration) profile and ii) the expected time to reach $newSpeed$, if enforced ($activate == true$). the Algorithm 2 computes the traversal speed for the segment as a function of the flow[1] and of the segment parameters, according to the navigation system constraints. The traffic flow evaluation is performed by the $EstimateTraffic()$ function based on the smart city traffic flows of the subsegments composing the segment of interest.

To evaluate the impact of SCICC on the vehicle gas emissions and fuel consumption, a simulation model working with Simulink was realized [32]. It computes the the quantities of fuel and CO_2 produced been evaluated on a well known driving cycle, the New European Driving Cycle (NEDC)[2], a standard cycle used to measure vehicle fuel consumption and gas emissions in EU. This is used to compare the advantages produced by SCICC's Algorithm, as shown in 6.2.

[1]Each city define its own threshold used to describe the traffic congestion in the segments.

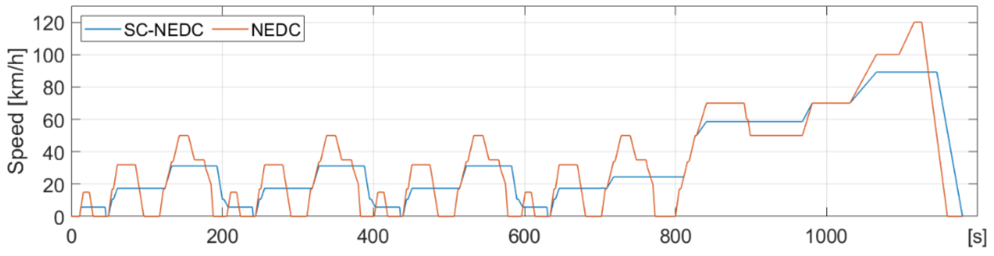
[2]https://web.archive.org/web/20050909051753/http://www.europarl.eu.int/commonpositions/1999/pdf/c5-0028-99_en.pdf

Algorithm 1 SCICC's Segment Analysis Coordination Algorithm

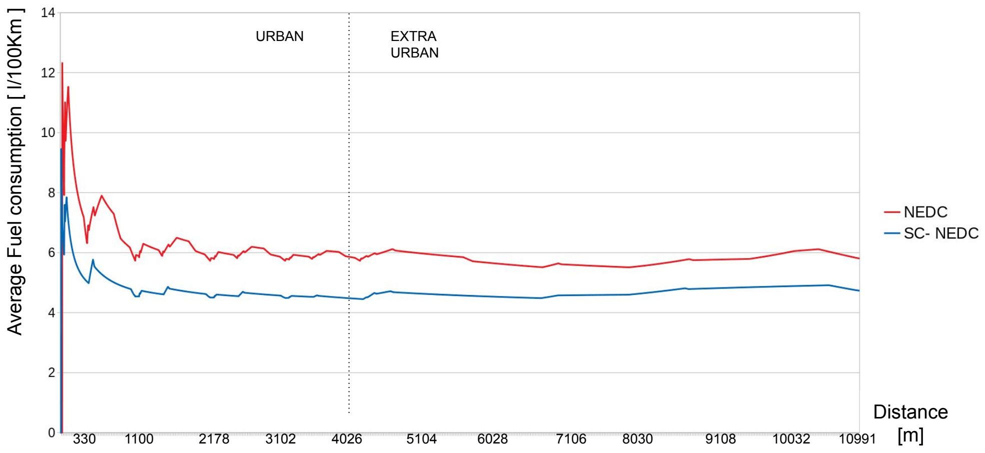
```
1: function SAA($v_{own}$, SegPars, Seg$_{next}$Pars)    ▷ This function coordinates the analysis of the segment parameters in the evaluation of the segment traversal speed. $v_{own}$= current speed of the vehicle, SegPars and Seg$_{next}$Pars contain: d=distance to the traffic light, trr=residual red time, tv= green time, tr=red time, vmax= max speed, flow= traffic flow.
2:     var k = EstimateTraffic();
3:     if (SegPars.d $\geq$ min-distance-to-traffic-lights) then
4:         if ($v_{own} \leq$ SegPars.vmax) then
5:             Struct$_{travs}$ = TravSpeed(SegPars,k);
6:             vseg = Struct$_{travs}$.vseg;
7:             if ($v_{own}$ = 0) then
8:                 Struct$_{acc/dec}$ = Increase_Speed($v_{own}$,vseg,False);
9:                 Struct$_{travs}$ = TravSpeed(SegPars,k);
10:                vseg = struct$_{travs}$.vseg;
11:            end if
12:        else:
13:            Struct$_{acc/dec}$.$v_{own}$ = Decrease_speed($v_{own}$,SegPars.vmax);
14:            SegPars.d = compute_distance();
15:            struct$_{travs}$ = TravSpeed(SegPars,k);
16:            vseg = Struct$_{travs}$.vseg;
17:        end if
18:    else:
19:        vseg$_{next}$ = SAA($v_{own}$,SegPars,Seg$_{next}$Pars);
20:        Struct$_{travs}$ = TravSpeed(SegPars,k);
21:        vseg = Struct$_{travs}$.vseg;
22:        if ((vseg$_{next} \geq$ vseg) & (vseg$_{next} \leq$ SegPars.vmax)) then
23:            Struct$_{acc/dec}$ = Increase_Speed(vseg,vseg$_{next}$,True);
24:        end if
25:        if (vseg$_{next} \leq$ vseg) then
26:            Struct$_{acc/dec}$.$v_{own}$ = Decrease_speed($v_{own}$,vseg$_{next}$);
27:        end if
28:    end if
29:    Return [vesg,$v_{own}$];
30: end function
```

Algorithm 2 SCICC's Traversal Speed computation Algorithm

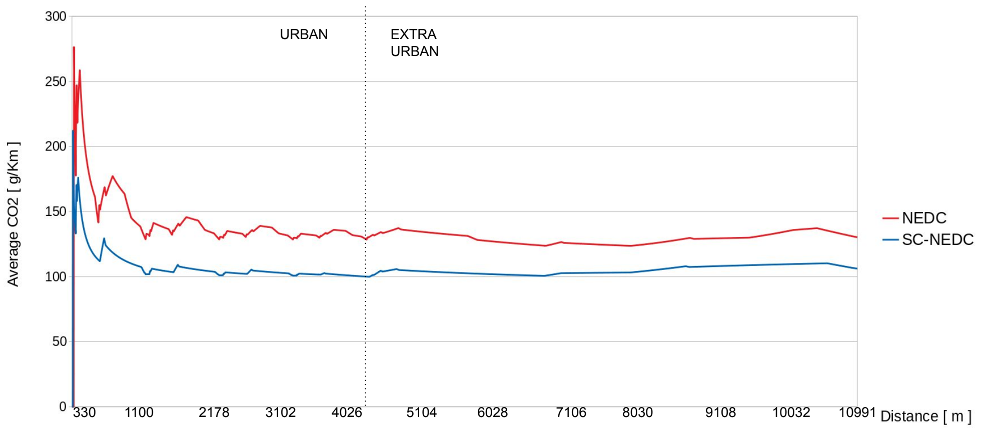
```
function TravSpeed(SegPars,k)    ▷
This function evaluates the traversal speed for the segment. k is a threshold used by the city to identify the congestion; SegPars contains: d=distance to the traffic light, trr=residual red time, tv= green time, tr=red time, vmax= max speed, flow= traffic flow, NavTime= traversal time estimated by Navigator.
    if (SegPars.flow $\geq$ k) then
        Struct.vseg = SegPars.d / (SegPars.trr+SegPars.tv+SegPars.tr);
        time = (SegPars.trr+SegPars.tv+SegPars.tr);
    else:
        vseg = (SegPars.d / SegPars.trr);
        if (vseg $\geq$ SegPars.vmax) then
            if (SegPars.NavTime $\leq$ (SegPars.trr+(3*SegPars.tv/4))) then
                if (SegPars.NavTime $\geq$ (SegPars.trr)) then
                    vseg = SegPars.d / SegPars.NavTime;
                    time = SegPars.NavTime;
                end if
            else:
                vseg = SegPars.d / (SegPars.trr+(3*SegPars.tv/4))
                time = (SegPars.trr+(3*SegPars.tv/4));
                if (vseg $\geq$ SegPars.vmax) then
                    Struct.vseg = SegPars.d / (SegPars.trr+SegPars.tv+SegPars.tr);
                    time = (SegPars.trr+SegPars.tv+SegPars.tr);
                end if
                Struct.vseg = vseg;
            end if
        end if
    end if
    if (SegPars.NavTime $\leq$ time) then
        Struct.compliantNav = False;
    else:
        Struct.compliantNav = True;
    end if
    Return Struct;
end function
```



(a)



(b)



(c)

Figure 6.2: *SCICC experiments results: a) NEDC and SC-NEDC speed comparison; b) Fuel Consumption; c) CO_2 emissions.*

6.2 SCINaS Algorithm

IN SCiNaS, *Smart City Navigation System* the goal is to reduce the traversal time aiming to balance the CO_2 emission and fuel consumption. The Route Control (RC) identifies the traversal time valid for the traversed segment (Tsmin in Figure 6.3), enabling the vehicle to avoid stopping due to the traffic light. This activity is strongly dependant on the starting condition. For SCiNaS's algorithm, the starting condition is represented by the traffic light status when the vehicle begins the segment (or when the vehicle makes the computation). These conditions are depending on the value of two (green and red) remaining time parameters received by the SC, **trg**, and **trr**, respectively. The one greater than zero represents the traffic light status at query time. As shown in Figure 6.3, the traversal time computed at the beginning of the computation in some cases may correspond to red light. To avoid this, the RC has to increase the traversal time of an additional time (**Tadd** in Figure 6.3).

In order to enable the Route Control to compute the additional time, we have to make some assumptions:

- Traffic light cycle begins with a green period, and it contains even the yellow one if any.

- The Residual time, **Tres**, represents the time needed by the traffic light to reach the beginning of the first traffic light cycle (if $trr > 0$ $Tres = trr$, else $Tres = trg + Ttlr$).

- The algorithm verifies if the value of the variable **td** (see Figure 6.3) is lower or higher than zero, to compute a prediction of the status assumed by the traffic light at the end of the segment. A positive value indicates a red light.
- The final traversal time proposed by the RC is equal to the sum of the previous traversal time, and the **td** value if greater than zero.

The computation made by the RC for the speed profile computation begins analyzing as a first option a segment traversal made at the maximum speed. The output obtained could result in a march with constant velocity (**Vmax**) or a uniform accelerated motion that depends on the vehicle's speed when the computation is done. This first option could break the vehicle's constraint about the minimum speed admitted in the vehicle's first gear (the engine idle speed). In this case, a tentative to identify a suitable deceleration pattern is made, and if it does not produce any result, the vehicle is forced to stop its march. When the vehicle's speed constraint is not broken, the system verifies if the profile is able to get the green light. Then the Cruise Control actuates the profile received by the RC. In opposition to the latter case, if the profile is not able to drive the vehicle towards a green light, it is re-computed changing the **Tsmin** used; this is done by adding the value **Tadd** (described above and shown in Figure 6.3). The algorithm tries to split the profile into two parts as a solution to the profile identification problem. In the first one, the vehicle accelerates (or decelerates) with a uniformly accelerated motion. The decision about the acceleration depends on the threshold:

- 1 m/s^2 if the initial speed is below the engine idle speed,
- 0.5 m/s^2 if the initial speed is higher than previous but below standard urban speed limitation (50 km/h),
- 0.25m/s^2 in the other cases.

A constant velocity motion instead characterizes the second part for the remaining time. Data about both motion profiles are tested to verify if the resulting profile can get a green light. If not, the system computes a profile again based on the previous hypothesis of traversal time, the new one, and re-iterates the profile identification. Figure 6.5 shows some code snippets implementing the algorithm mentioned above.

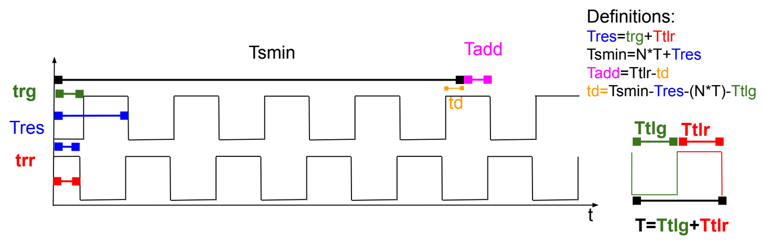


Figure 6.3: *SCINaS Traffic Light cycles and definition [7].*

Even in the SCiNaS case, to evaluate the algorithm's impact on the vehicle gas emissions and fuel consumption, a simulation model working

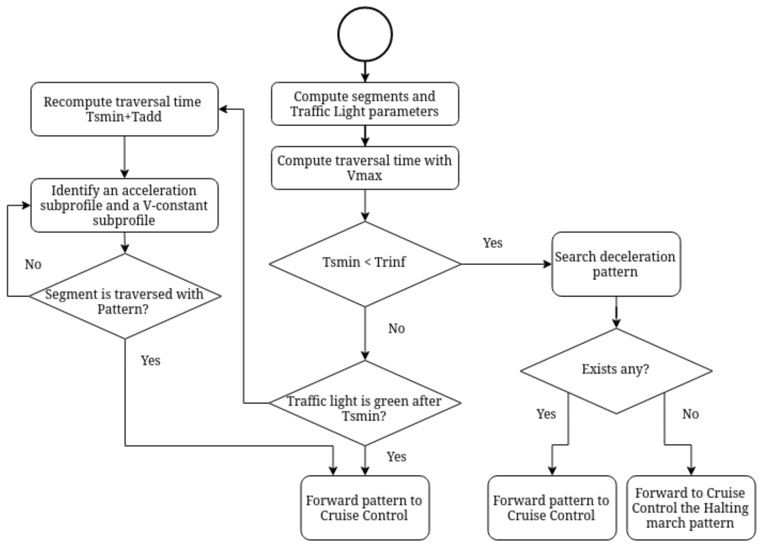
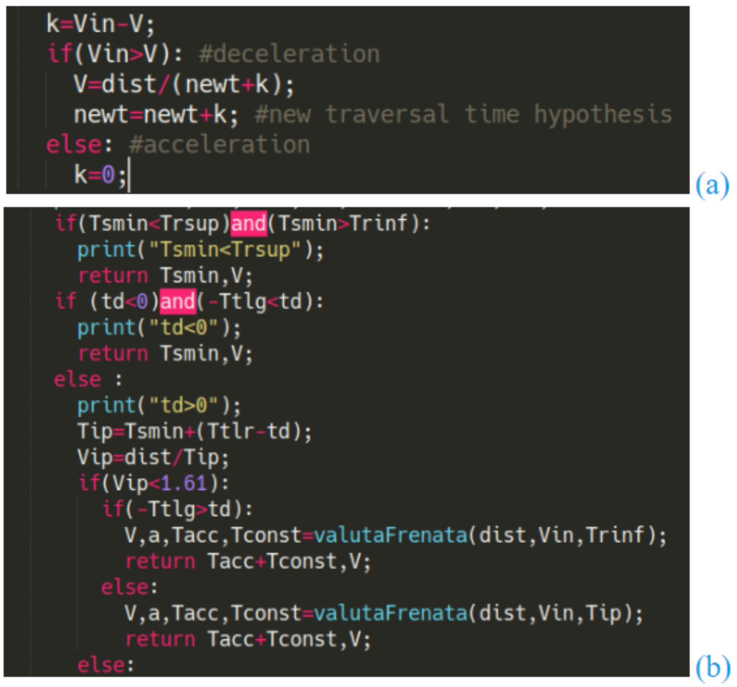

Compute segments and Traffic Light parameters
Compute traversal time with Vmax
Tsmin < Trinf
Yes
Search deceleration pattern
Exists any?
Yes
No
Forward pattern to Cruise Control
Forward to Cruise Control the Halting march pattern
No
Traffic light is green after Tsmin?
Yes
Forward pattern to Cruise Control
Recompute traversal time Tsmin+Tadd
Identify an acceleration subprofile and a V-constant subprofile
Segment is traversed with Pattern?
No
Yes

Figure 6.4: *SCINaS Algorithm's Flow chart minimizing the city traversal time [7].*



```
k=Vin-V;
if(Vin>V): #deceleration
  V=dist/(newt+k);
  newt=newt+k; #new traversal time hypothesis
else: #acceleration
  k=0;
```

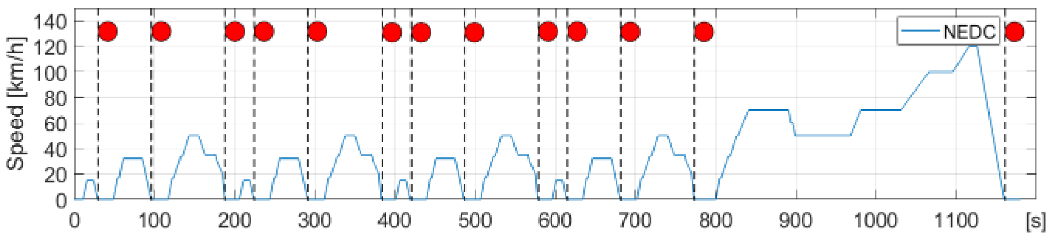
```
if(Tsmin<Trsup)and(Tsmin>Trinf):
  print("Tsmin<Trsup");
  return Tsmin,V;
if (td<0)and(-Ttlg<td):
  print("td<0");
  return Tsmin,V;
else :
  print("td>0");
  Tip=Tsmin+(Ttlr-td);
  Vip=dist/Tip;
  if(Vip<1.61):
    if(-Ttlg>td):
      V,a,Tacc,Tconst=valutaFrenata(dist,Vin,Trinf);
      return Tacc+Tconst,V;
    else:
      V,a,Tacc,Tconst=valutaFrenata(dist,Vin,Tip);
      return Tacc+Tconst,V;
  else:
```

Figure 6.5: *SCINaS Algorithm's fragment minimizing the city traversal time [7].*

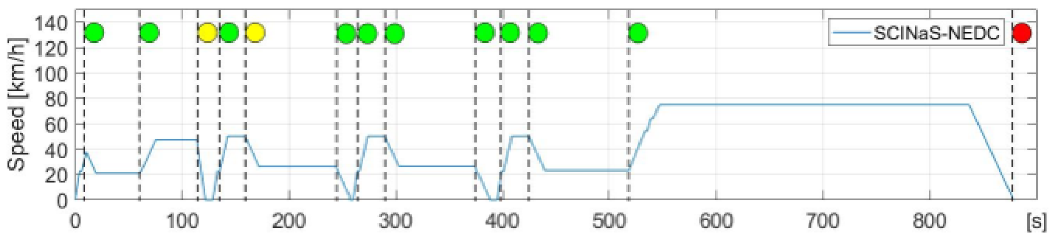
with Simulink was realized [7]. It computes the quantities of fuel and CO_2 produced been evaluated on a well known driving cycle, the NEDC. The comparison among consumption (fuel and CO_2 emission) produced by SCiNaS's Algorithm and NEDC is shown in figures 6.6, and 6.7. Trying to investigate the computational overhead of the SCINAS algorithm, we tried to execute it 100 times on the same path (segments composing the path are described in Table 6.1), the one representing the NEDC cycle that we already evaluate during the simulation discussed before. To analyze the impact on the Route Control of the analysis, we run the experiments on constrained LXC containers modifying the RAM and the number of CPUs available to the underneath system (i.,e., an Ubuntu 18.04 server containerized and running on a laptop equipped with an Intel i7 6500U CPU and 8GB of RAM). The experiments are done on containers that are assigned 1 to 2 CPUs and 32Mb to 256 Mb of RAM. On the graph 6.8, it is possible to see that the algorithm execution doesn't benefit from the increment of resources both in RAM and the number of CPUs available (for every segment and in any container configuration, the execution times are comparable). In graph 6.9, the standard deviation of the experimental data obtained on each segments' elaboration is very low, except for some segments (e.g., segments 2, 6, and 7).

Seg.ID	Lenght	Vmax: Maximum Admitted Speed	Trr: Red Residual Time	Trg : Green Residual Time	Ttlr: Traffic Light Red duration	Ttlg: Traffic Light Green duration	Ttl: Traffic Light Cycle duration
1	54.166667	13.88	0	16	33	27	60
2	317.576	13.88	0	17	33	27	60
3	659.722	13.88	24	0	33	27	60
4	54.166667	13.88	10	0	33	27	60
5	317.576	13.88	7	0	33	27	60
6	659.722	13.88	8	0	33	27	60
7	54.166667	13.88	0	26	33	27	60
8	317.576	13.88	4	0	33	27	60
9	659.722	13.88	10	0	33	27	60
10	54.166667	13.88	1	0	33	27	60
11	317.576	13.88	5	0	33	27	60
12	659.722	13.88	20	0	33	27	60
13	6865.279	36.11	0	18	33	27	60

Table 6.1: *Segment's parameter passed to SCiNaS in order to compute the speed profile.*

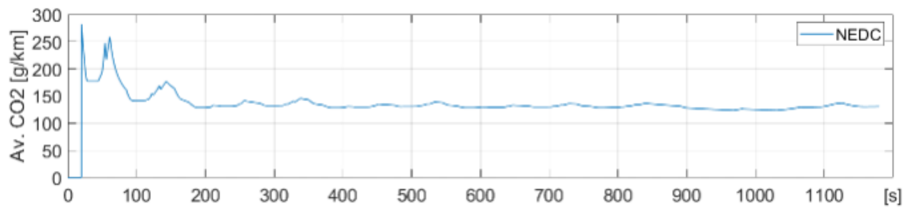


(a)

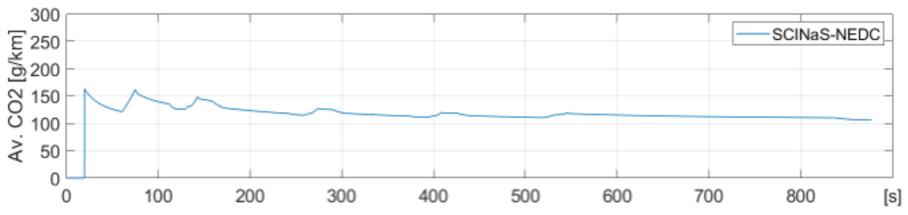


(b)

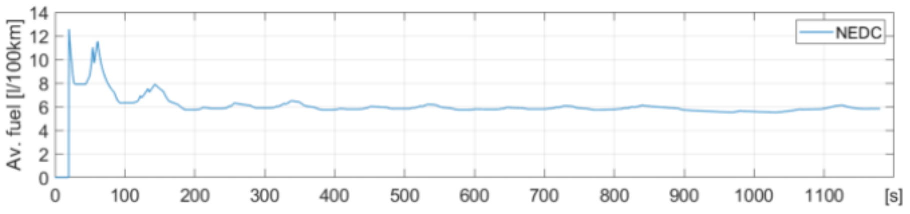
Figure 6.6: *SCiNaS experiments results: traffic light states. a) NEDC's traversal time; b) SCiNaS's traversal time.*



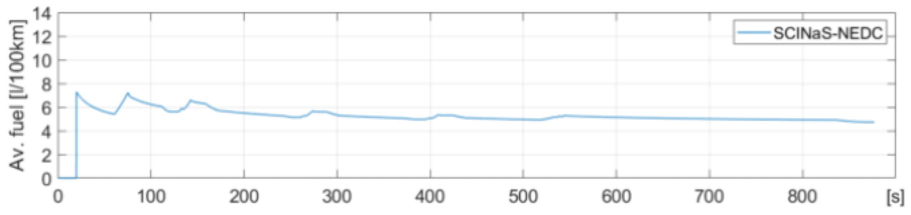
(a)



(b)



(c)



(d)

Figure 6.7: *SCiNaS experiments results: a) NEDC's emission of CO_2; b) SCiNaS's emission of CO_2; c) NEDC's Fuel Consumption; d) SCiNaS's Fuel Consumption.*

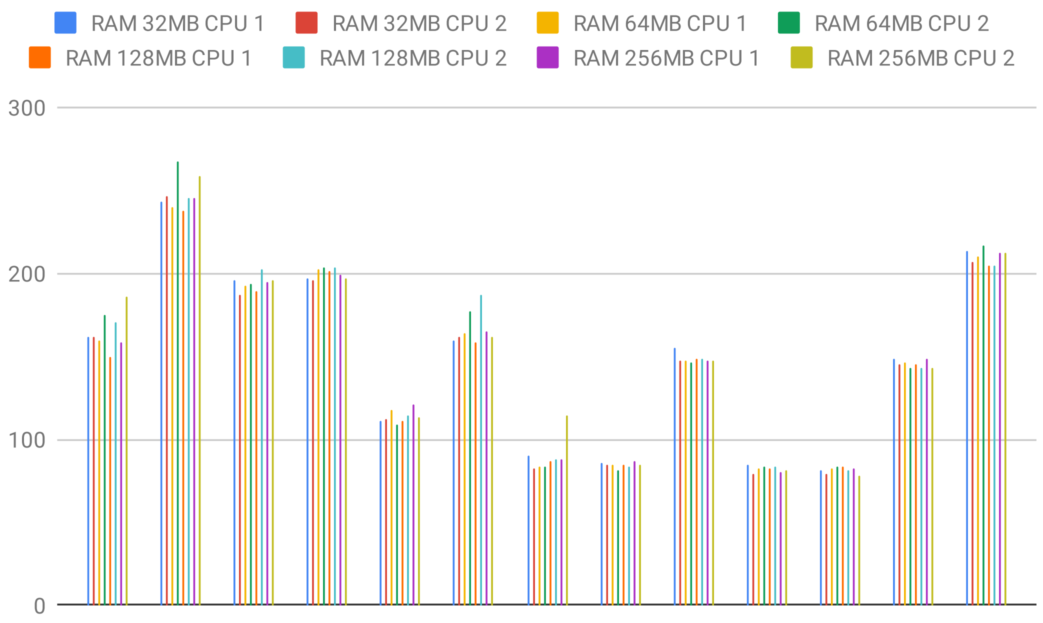


Figure 6.8: *Average execution times of SCiNaS's algorithm for each segments and for each combination of RAM and CPU.*

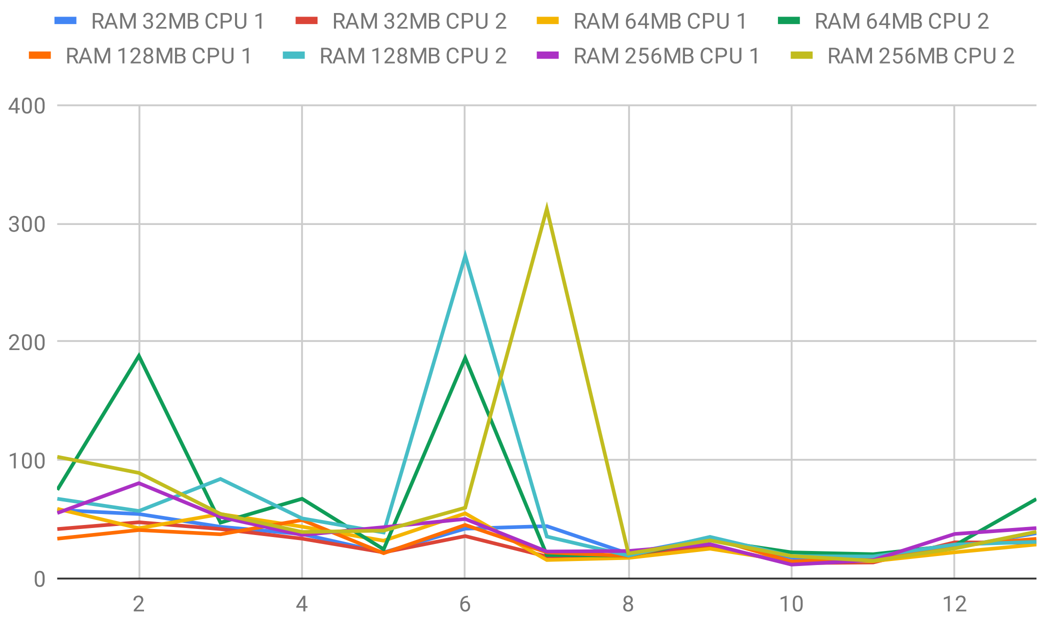


Figure 6.9: *Execution time Standard Deviation of SCiNaS's algorithm for each segments and for each combination of RAM and CPU.*

CHAPTER 7

CPS as set of computational units: Exploitation of Serverless paradigms to deploy pipelines on CPS

THE Serverless paradigm (see section 2.1.6), in particular the FaaS one, is a computing style in which the computation is made through the execution of functions somewhere that producing an output to the requestor. Behind the definition of this paradigm, there is the idea that reduces the computation tasks requested (by an application) into a series of functions considering all inputs received to produce an output. It is clear that function becomes blocks composing an application, and their execution in pipeline produces the aimed results. In a CPS's application, the IoT devices perceive the physical values used as inputs, and they implement the actuation actions required by the computation's results. The inputs perceived are not computed in the same step, but for some application, the choice of an input source may depend on a previous computation's step. To understand better, a reader can imagine the situation in which a temperature sensor perceives a high temperature, and it engages the smoke sensor in a workflow to understand if there is a fire in the room or the nearest environments. At this point, if the smoke sensor does not perceive any warning, maybe it is an issue with the air-conditioner, of the environment, and it can shut off the air-conditioner, and open the window. This makes the CPS's application the perfect candidate to exploit the FaaS paradigm.

Another reason making the CPS the perfect candidate to work in this scenario is due to the computational power owned by the IoT node. As explained in the previous sections 2.1.4, 2.1.5, and 2.3, a CPS is exploitable for a distributed computation able to react in real-time, avoiding network latency.

The previous consideration about the CPS and the FaaS paradigm's consideration become a research cue for this study. So an infrastructure implementing the Edge computing principle to host the functions related to a FaaS-based pipeline was presented in [8]. It is an extension of the OpenStack environment obtained from the cooperation put in place among several OpenStack's projects, in particular: Qinling [143][1], Stack4Things4.2, and Zun [144][2].

As it is possible to see in figure 7.1 the system is based on three main sections:

- **Interface section** represents the point of access where the administrators define their application; it is used to provide commands to Cloud section components that will physically set up the application running on the Edge worker node, then on the CPS.
- **Cloud section** contains all the coordination components used to orchestrate the functions delivered on the IoTs. It also configures all the infrastructure, physical and virtualized exploited by both the Edge worker node and CPS's applications.
- **Edge Worker section** contains the components running on the IoT devices to execute the functions.

The solution is realized using IoTronic, and modified version of Qinling and Zun subsystems, so the OpenStack framework can provide FaaS services using distributed IoT devices located at the Edge of the network.

[1]**Qinling** is a project of Openstack meant to implement Function as a Services paradigm.

[2]**ZUN** is a project of Openstack meant to act as a Container Orchestration Environment, then it orchestrates and manages the Containers during their lifecycle on the computational nodes.

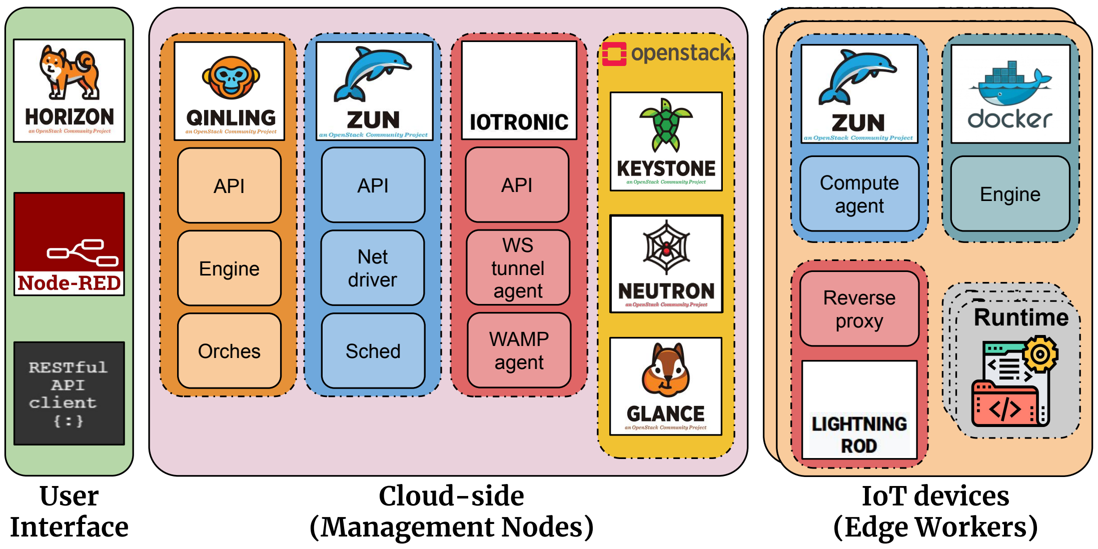


Figure 7.1: *The edge-based FaaS system architecture [8].*

To do this, Qinling and ZUN were extended using a driver enabling Qinling to interact with ZUN, and ZUN was extended with a driver to interact with IoTronic (see figure 7.2). This newly introduced functionality in OpenStack is achieved through RESTful interactions with Qinling that uses, in the backend, Zun, and IoTronic to deploy functions on the remote IoT devices.

The main difference between the data center-based OpenStack deployments and the described solution is the deployment of the three components: Zun-compute, Docker engine, and the runtimes; they are deployed on the Edge worker node, the CPS's IoTs. In more detail, on the Cloud-side are deployed the Qinling subsystem (engine, orchestrator, and API server), Zun (API server and Zun scheduler), and IoTronic. The other components, such as Zun-computes, Docker engines, and the

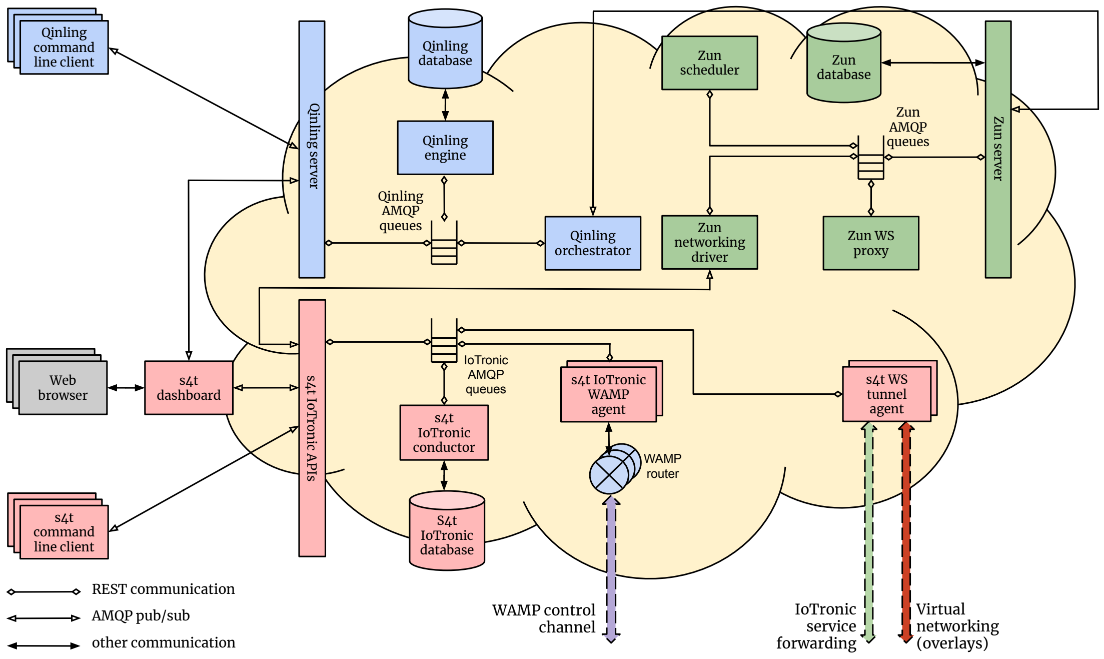


Figure 7.2: *Stack4Things FaaS Cloud-side subsystem design [8].*

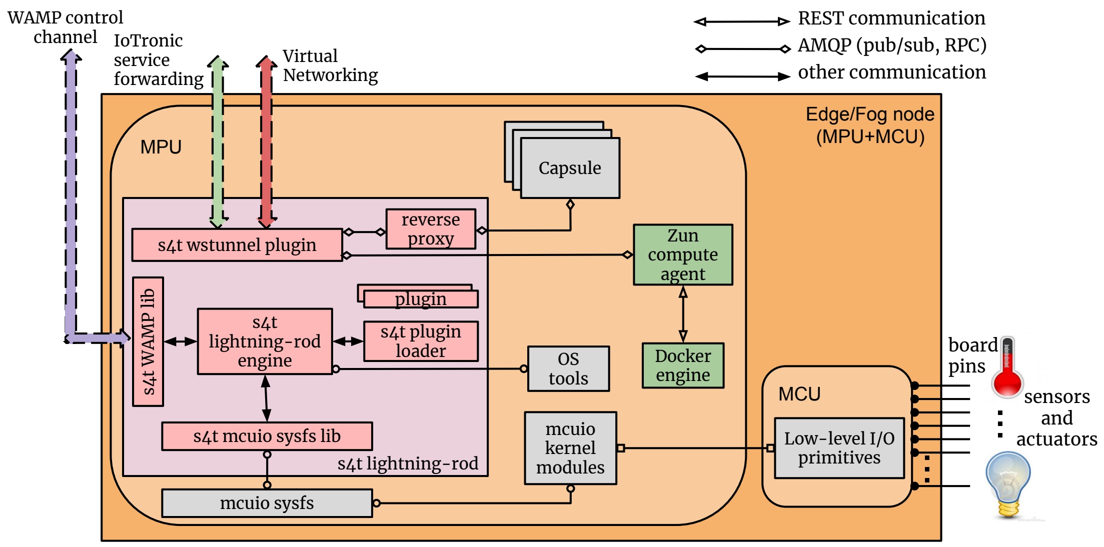


Figure 7.3: *Stack4Things FaaS Edge/Fog-side subsystem design [8].*

runtimes stand in our case on the IoT devices (see figure 7.3), instead of being placed on the Cloud as commonly happen in a standard OpenStack deployment.

The typical workflow for the solution described is the following: A CPS's administrator creates his pipeline interacting with the *Interface section* through the NodeRED-based dashboard, even if it stays at the same level of typical OpenStack's tools (Horizon dashboard and CLI), the usage of NodeRED makes easier the pipeline deployment (see figure 7.4.a). With the NodeRED dashboard, the admin is enabled to define blocks able to invoke function's instantiation; these blocks are the central elements in the workflow composition as a pipeline of blocks. When the pipeline is defined, the administrator deploys the flow with a simple click on the deployment button; it corresponds to a series of requests to deploy function delivery to the Qinling orchestrator (going through Qinling-API)(see figure 7.4.b). The Qinling orchestrator uses ZUN as the Container Orchestration Environment (COE), where the *runtimes* (execution environment of the Qinling's function) are instantiated thanks to a specific driver created for [8]. Firstly, the Qinling orchestrator interacts with Zun-scheduler to identify IoT device, and after this, the Zun-API server sends a request to create, on the device, the containers needed (i.e., the capsule[3]). At this point, ZUN involves IoTronic in the deployment workflows to provide the networking facilities to the *Capsule*. IoTronic exposes a pair of IP-port that exploits a WS tunnel to reach a Capsule's Runtime. On the device-side, LR uses the reverse proxy to

[3]A capsule is a group of containers cooperating, it is analogous to the *Pod* concept in Kubernetes. Containers within a *capsule* share the same network configuration, namespaced PID, Mount, etc.

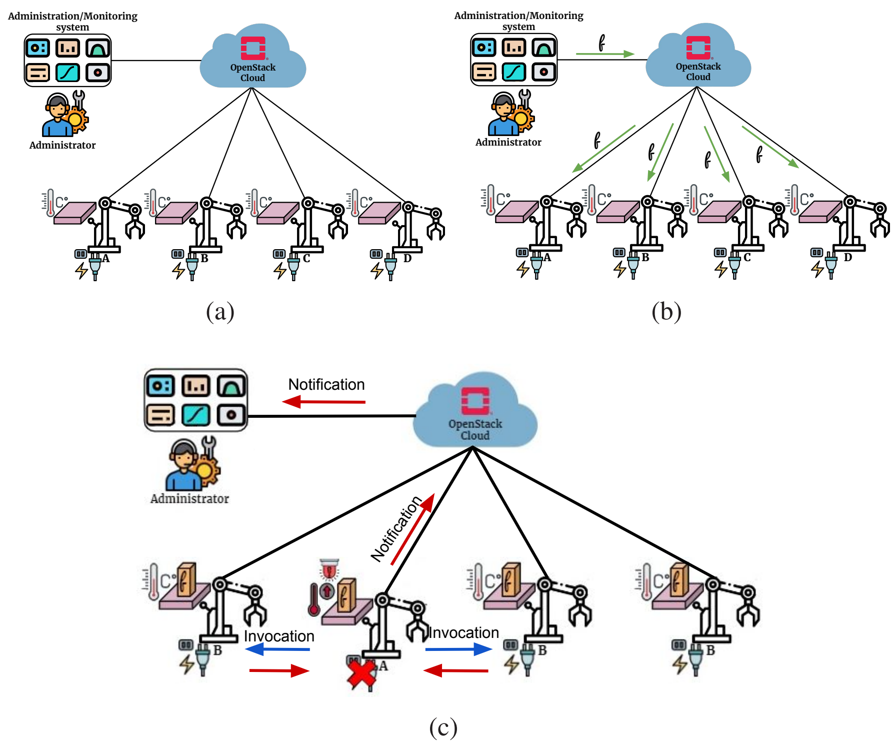


Figure 7.4: *Overview of a use case when the edge-based FaaS system can be deployed for enhanced tasks management [8].*

route the traffic. When the pipeline is instantiated, it executes the CPS's IoT functions according to the administrator's definition in the first step. Figure 7.4.c shows the system behavior in the example described above in this chapter when a node perceives a peak of temperature.

CHAPTER 8

Software-Defined City Infrastructure supporting a Dynamic Intrusion Surveillance System

THE possibility of constructing CPS-based applications exploiting the FaaS paradigm associated with Edge computing infrastructure is one of the possible solutions working on a CPS or in cooperative CPSs. Another approach explored is related to the application of software-defined principles and the networking and the Input/Output operations. This way, it is possible to orchestrate the cooperation between CPS simply through the definition of virtualized (i.e., defined via software) Input/Output paths between devices, even in case there is no real network connection neither direct nor through NAT or proxy systems.

The basic idea is meant to extend what previously defined in [21], where the concepts of Software Defined paradigm was applied on an abstraction of CPS, managing them with a high-level approach that does not consider what happens inside the CPS. Notwithstanding this, [21] presents an interesting point of reflection that has generated the following idea. The Fog computing needs to be pushed to an even lower logical level as a set of mechanisms enabling the deployment and execution of multiple location-aware, low-latency, peer-to-peer IoT-like applications, where most of the logic runs on the involved smart objects. So the Fog infrastructure works as a (programmable) coordinator only, evaluating custom rules and subsequently acting upon the ones which apply to events related to a specific situation, modifying the infrastructure topology where needed. This is a clear usage of the Computing Continuum paradigm 2.1.5. This situation is reflected by schematic shown in figure 8.1.d. The whole figure 8.1 represents several ways in which the interaction with IoT devices may be realized.

The most basic configuration, shown in Figure 8.1.a, features a “plain”

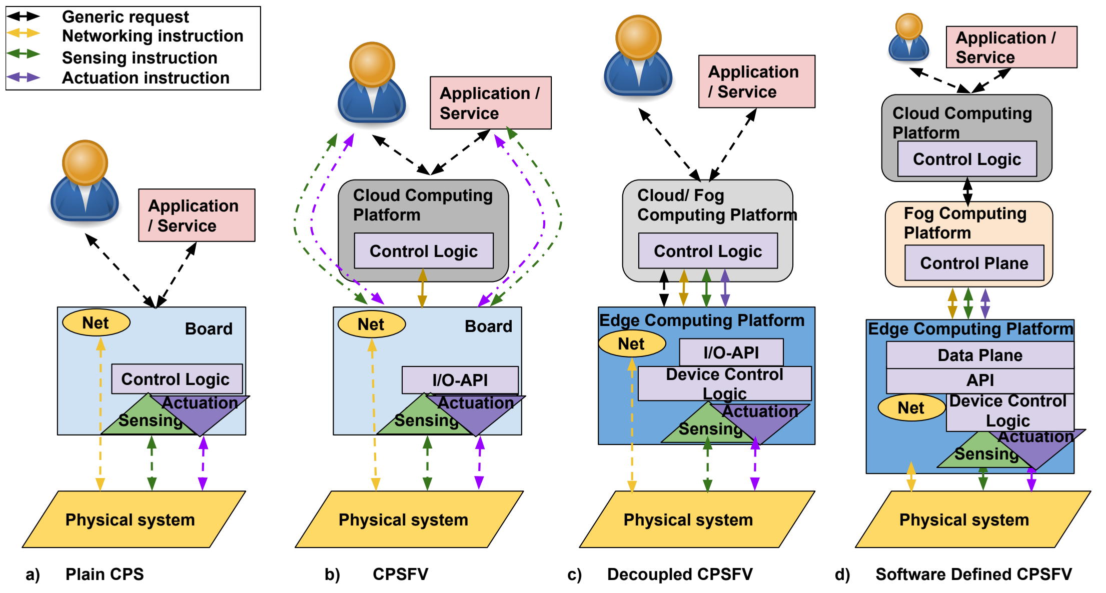


Figure 8.1: *Approaches for Cyber-Physical System Functions Virtualization (CPSFV) [9]*

CPS where the interface subsystem (e.g., a board) acts independently from any end-user or application interaction and provides exclusive access to the physical resources, mediated by the custom-developed control logic. To manage the functionalities related to sensing and actuation devices, it is possible to follow another approach, as discussed in [21], and shown in Figure 8.1.b. This approach is meant to decouple the application and the service requests from the physical system's management. An evolution of this approach is shown in Figure 8.1.c. Here, the generic idea is to decouple application or service requests from the physical system. The overarching Cloud-based approach has first been described in [145], where the authors propose a device-centric paradigm to provide services based on IoT infrastructure, whereas in [142], a way to provide virtual instances on I/O devices (*virtIO*) that run in an Edge computing environment, the I/Ocloud system, is presented. Lastly, Figure 8.1.d is shown the solution designed to enable the orchestration, and the definition of Input/Output virtualized paths useful for the cooperation of CPSs.

The concept previously described becomes concrete when we map it on multiple cooperating CPSs into a Smart City.

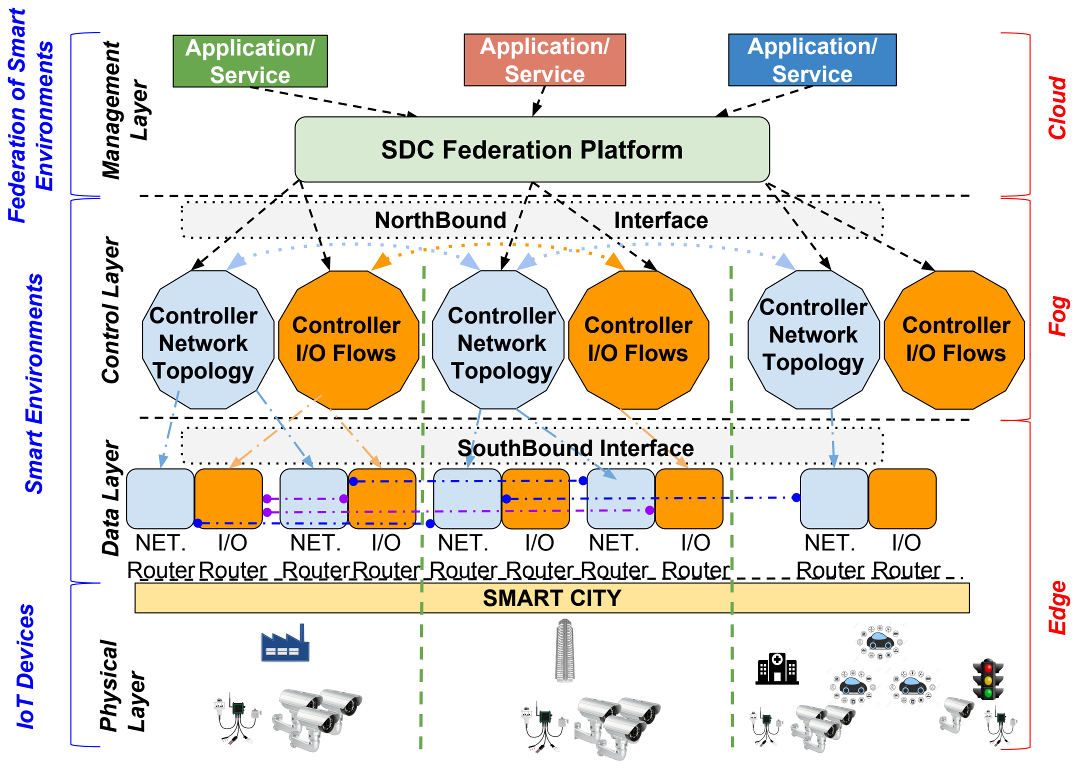


Figure 8.2: *High Level architecture of Software Defined City. [9]*

To better understand the scenario presented in figure 8.2, some definitions are needed:

- **Domain**: it represents an administrative domain managing one or more CPS, typically inside the same environment (even if it is not mandatory). The devices contained in the same domain share with flexible policies (or SLA) their functionalities.

- **Physical Layer**: it represents the layer in which the IoT devices (the Edge node) operate. They receive data from the sensor, send commands to the actuator, run scripts/batch/functions to pre-elaborate locally the data owned, interact with upper layers to forward the information they own.

- **Data Layer**: it represents the layer where the action to orchestrate the system is put in place, in line with the Software-Defined paradigm definition. Here, the involved devices are systems with medium computation capabilities (the Fog node, such as a router, small server, and similar). This layer receives from the physical one the data and acts as a relayer, distributing the data according to the upper layer's directive.

- **Control Layer**: it represents the place in which the coordination decisions are taken. The components running in this layer know the structure of the domain's infrastructure and are authoritative for the coordination of the domain's elements. Layer's components interact with the Data layer's elements passing orchestration commands and with the upper layer to receive the requests of data and resources facilities coming from the federation.

- **Management Layer**: it represents the federation system, here the applications, aware of the federation existence, run and interact with the *SDC federation platform* to request data and services. The *SDC federation platform* is interfaced with each domain and delivery requests and information to the coordination components of the Control Layer.

In Figure 8.2, these layers are further grouped based on their logical domain (leftmost blue layering) and based on the deployment (rightmost red layering). The latter ranges from Cloud computing hosting applications and services offered to the end-users, but also the software needed to maintain and manage the Smart City (a kind of federation among CPS of the Smart City), to Fog computing at the control layer and Edge computing for device networking on IoT devices.

Figure 8.2 is the reference architecture for a stack implementing an interdomain *Software Defined City Infrastructure* (SDCI). As defined above, the bottom of the SDCI stack is the *physical layer*, which refers to heterogeneous IoT nodes at the edge, operating in the physical world under different administrative domains and corresponding smart environments. The *data layer* provides interoperability, customizability, and programmability mechanisms for networking and I/O operations. Above this, the *control layer* offers tools and facilities for building network topologies and I/O flows by enforcing policies based on the application logic requirements. Data and control layers implement the Software-Defined approach of the SDCI. From this point of view, they operate in smart environments, which implies that a pair of controllers (network and I/O) is required to coordinate the underlying IoT devices for each smart environment. Cooperation between peering controllers is established by *federating entities* provided by the *management layer*, here adapting a hierarchical management approach, akin to the one defined in [146] , for Software-Defined Networking in Cloud environments, to support flows across different domains without any non-datapath relay in-between.

8.1 Software-Defined I/O

THE cornerstone of the system described in the previous section is a distributed *I/O* layer, as well as a centralized subsystem modeled as an *Infrastructure-as-a-Service* layer. This way, it is possible to extend the approach to include dynamic reconfiguration of the underlying nodes' networking subsystem. For this purpose, the SDCI approach can be adopted to manage both network equipment on urban-scale and sensor/actuator-based infrastructure. So, it is introduced a mixed SDN and SDI/O control plan at the IaaS level and basic functionality (networking and I/O forwarding) at the I/O level (data plan), as shown in Figure 8.2. In this way, there is a generic ability to inject code (through the Edge computing platform) on physical boards, unlocking a certain degree of freedom in the customization of intelligent objects. This way, it is allowing the edge-oriented approaches towards Fog computing; at the same time, the system becomes able to be customized at the infrastructure level, such as virtual networking structures board side. Furthermore, a middleware devoted to the management of both sensor- and actuator-hosting resources may help in the establishment of higher-level services such as policies for “closing the loop”; an example is the configuration of triggers for a range of (dispersed) actuators based on sensing activities from (geographically non-overlapping) sensing resources. In line with the SDN approach, the solution proposed in [9] for the policy-based management of I/O resources through a purely software-based approach leads to Software-Defined Input/Output (SDI/O) as a core mechanism.

SDI/O is an approach to I/O flow management; it decouples the infrastructure topology and application-level request dispatching (used by SDCI entities) according to their internal policies. In summary, SDI/O decouples the definition of acquisition/actuation logic from the (procedural) details of the implementation. In the Control Plane all the decisions are taken according to policies, that encode how to translate high level I/O operations into forwarding paths, implemented in the data plane (e.g., routers along the paths). Instead in the data plane reside all the mechanisms related to how interactions with the transducers have to be relayed or terminated. Software-Defined I/O provides a two-fold benefit for an SDCI environment:

1. the ability to easily define and set up workflows and functional pipelines that involves resources owned by different actors, in a scenario where the number of actors involved can be significant.

2. the possibility to use routed RPCs, provided as a way to interact with an SDCI at a suitable, user-defined level of abstraction.

SDI/O involves mainly the elements of the central layers (Data and Control).Respectively, the former contains the Net and I/O Routers exploiting both the *Publish/Subscribe* communication mechanisms and the *Remote Call Procedure*, that receives the data from the IoTs (exploiting the Publish/Subscribe mechanism) and, actuates or activate procedures on the IoTs (exploiting the Remote Call Procedure). The latter instead hosts a pair of controllers (one for the Network topology and one for the I/O) that are able to coordinate the setup of communication path among peer router in the same domain or to create a path to directly forward the data to others domains.

8.2 Use Case and Evaluation

To demonstrate the suitability of the approach, in the following, SDCI is presented in an explanatory use case in the field of physical security, which is one of the most used in in the literature related to Smart Cities. Figure 8.3 depicts the use case,that is describing a situation in which a smart home, detects an unauthorized intrusion.

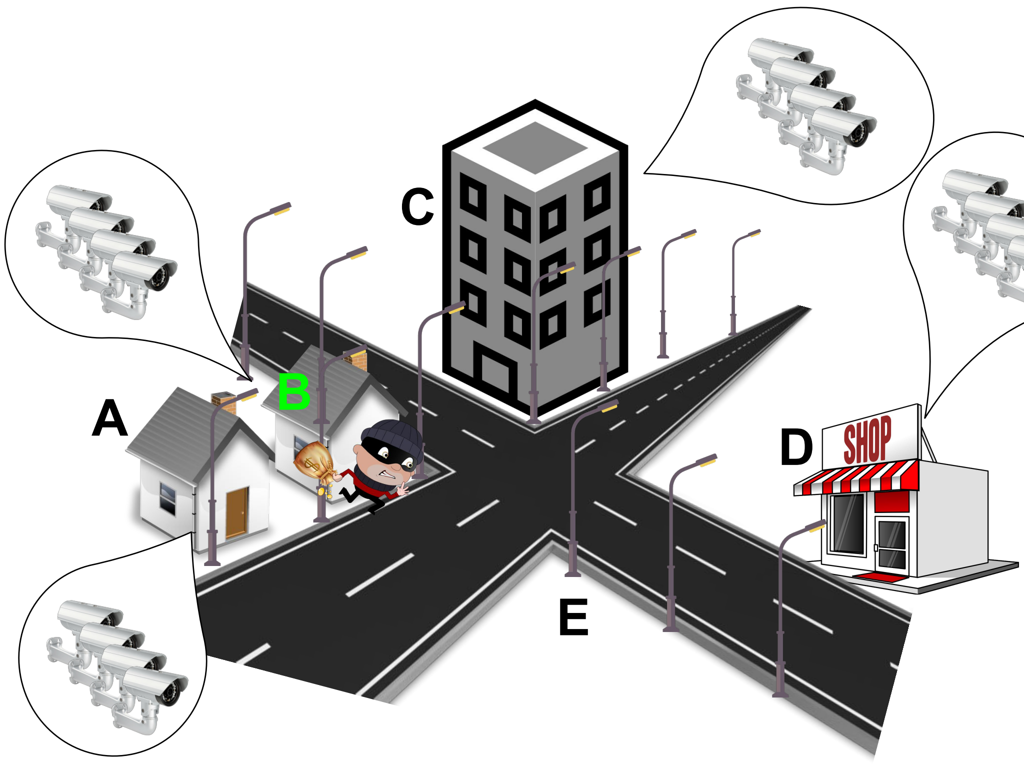


Figure 8.3: *Intrusion Surveillance System use case scenario [9].*

In this use-case, if SDCI-federating capabilities are put in place among the environments, the smart home (identified with the green letter B) can use the resources shared by nearby smart environments to increase the possibility of identifying the thieves. In fact, nearby smart environments - another smart home (A in the figure), a smart building (C in the figure), and a smart shop (D in the figure) can make their external video surveillance systems available for the identification of the thief. This means that even if the thieves try to compromise the internal/external video-surveillance system of the target environment (smart home at the letter B), the environment itself can interact through the federation to engage the nearest external video surveillance systems. An example of action made through the federation is sending specific commands to change the external cameras' position, to capture life-saving clips, or at the least shots, of the scene, which may help law enforcement officers identify the thieves. Even the smart city's public smart lighting system (E in the figure) may interact through an increment of the overall brightness in the streets on demand, so the quality of the videos recorded by all the involved cameras is improved.

As a reader can image, the scenario can involve in the workflow multiple devices, so an index to be measured is the scalability; in the following, a measurement of the scalability of a proof-of-concept implementation based on the use of the WAMP protocol[1] as publish/subscribe and RPC system, and on the Crossbar.io[2] *application messaging* router, working either as forwarding, thus belonging to the data layer, and as controller (control) layer as well.

Figure 8.4 reports the propagation time of the commands issued by smart home B for an increasing number of federated smart environments and requested IoT nodes. The evaluation is made by removing the unpre-

[1]https://wamp-proto.org/
[2]https://crossbar.io

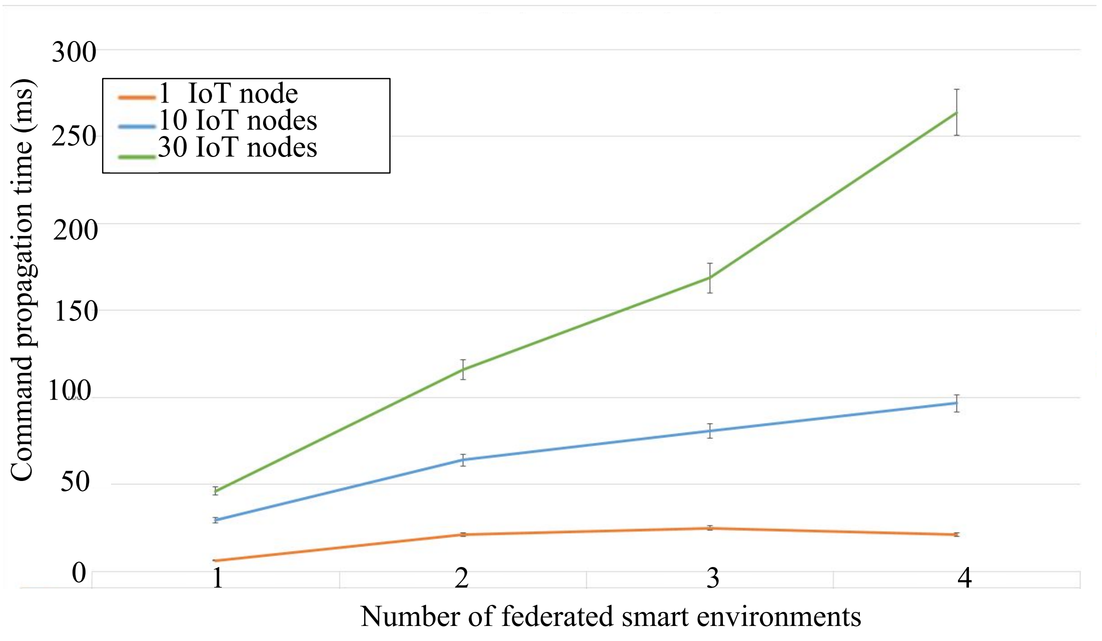


Figure 8.4: *Command propagation time with respect to the number of federated smart environments and IoT nodes involved.*

dictable effect of the network latencies, thanks to each VMs' connection with a *host-only* network enabling the communication inside the same server. Simulated infrastructure is composed of a Crossbar.io router enhanced that is used in the control layer and from 1 to 4 Crossbar.io router of the Data layer. The simulation consists of forwarding RPC towards the four Crossbar.io routers in the Data layer. In more detail, each RPC is delivered from another router in the data layer (representing the Home B, violated by the thieves) that is delivering RPC towards other data layer's router that is managed by federated domains.

This involves, at first, the interaction with the I/O flow controller for the creation of the appropriate forwarding tables, within the data plane, or integration of entries in existing ones, and, in a second moment, the lookup of the aforementioned forwarding tables for the identification of the designed destination. A total of 100 runs have been performed for each unique data point. Figure 8.4 depicts the average propagation times, together with the corresponding confidence intervals. The results are very promising as they show a quasi-linear behavior with little, if not negligible, slope.

CHAPTER 9

Federated Fire Protection System as an implementation of Software Defined City Infrastructure.

THE base of knowledge provided by *SDCI*, presented in the previous chapter (the chapter 8), became the foundation for the application related to Federated Fire Protection System management discussed in the following and presented in [10].

The main idea is to exploiting the Continuum computing approach to distribute the computation among each environment related to the industrial district. This way, even if a disruptive fire occurs in a supply line or between buildings, the system can continue to work, offering rescuers the facilities essential to support the firefighters; some examples are: access to sensors data, control on the environment such as open-close windows, reduce the flux of water in the area safe to increase the water where it is needed, and so on. The Cloud computing part is meant to enable automatic reactions to a fire event aiming to make slower the fire diffusion, extending in this way the intervention window of the firefighters (this increases the chance to minimize the damages made by the fire).

In particular, we have focused the Software-Defined approach towards the Factories to be applied to an industrial district fire system. Again, the starting point is the layered architecture presented in figure 8.2, but it is adapted to a fire system.

Software-Defined Factories (SDF) can be considered a Smart Building specialized for an industrial context, including all business processes. This common root has suggested to follow and adopt the concepts at the base of Software-Defined Building into SDF.

An SDF is a *programmable factory* where the business processes (e.g., administration, manufacturing, distribution, design, and so on), as well as the related to life-cycle of the environment (e.g., lighting, HVAC, surveillance, energy management), are supported by smart, programmable and interoperable devices, policies, and services. Nevertheless, these devices can be dynamically adapted to variable operating conditions (by, e.g., code injection); this not only increases the source of data available in a system (thanks to the multiple purposes that assume IoT sensing and actuating devices). Moreover, it increases the accuracy of the data perceived, and the monitoring process grows up (thanks to the redundancy created by the shared usage of IoT sensing and actuating devices).

The architecture of an SDF is reflected on the FFPS shown in the left-most side of Figure 9.1, collecting the SDCI's functionalities into three primary levels: “Infrastructure", “Management", “Application and Services". At the bottom, the Infrastructure layer includes all the devices and nodes composing the factory's ICT infrastructure, including any sensor and actuator, robot, workstation, gateway, router, smart object, device, and machine connected to the factory communication network. All the IoT devices can communicate, interact, and interoperate, even through M2M or similar protocols adopting WSN or other IoT solutions, in line with the SDF vision. This way, it is clear that an SDF should abstract its devices by exposing API that allows their customization and enables the programmability through a unique and uniform interface.

As is shown in figure 9.1, and in analogy with the software-defined paradigm, this layer represents the SDF Data Plane. This plane provides

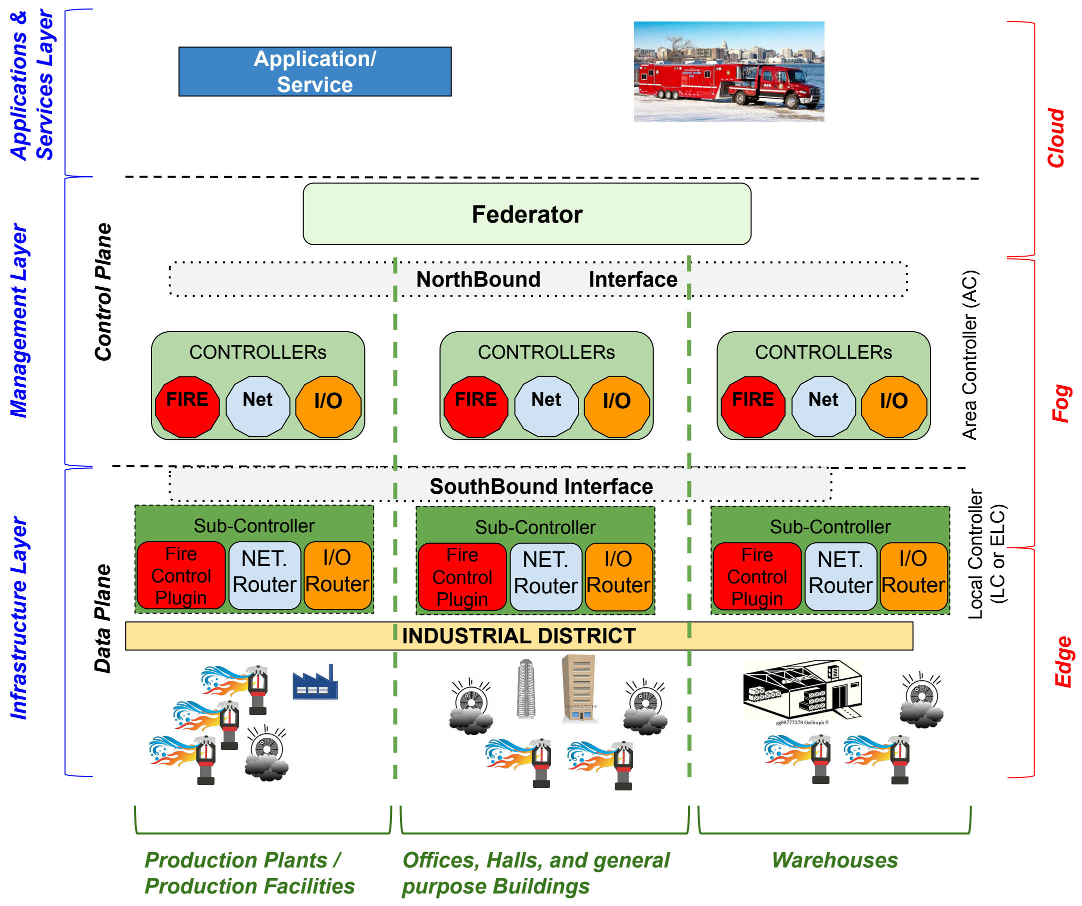


Figure 9.1: *High Level architecture of the factory FFPS [10].*

the basic mechanisms and API to program the device managed by using a serverless architecture to allow code injection into a containerized environment, as discussed in chapter 7.

To control and manage the Data Plane are required mechanisms for managing both living processes (that can be derived/inherited from SDB [147] and section 2.3.1.1) and business ones (that can be analyzed from the literature about Software-Defined Networking [148] or Software-Defined Manufacturing [18]). As before, referring to figure 9.1, the architecture reflects the SDF principles owning a layer acting as Control Plane. The upper Management layer exploits the lower layer's features to specify (hardware-physical and software-data) resource management policies. These are related to orchestration, virtualization, networking, pooling, monitoring, and generic mechanisms similar to those conceived for the factory business and living processes (manufacturing, safety, energy management, etc.).

To deeper analyze the architecture, we have to talk more in detail about Data and Control Plane. The Data Plane represents the bottom part of the SDF architecture, where IoT devices acquire data from physical phenomena and actuate the action defined by the fire protection system controlling part. This plane is tailored to follow a Fog/Edge computing fashion. It represents the Infrastructure layer, where physical devices (sensors and actuators) of each SDF sub-domain own the fire system identification of the countermeasure elements. Furthermore, the Infrastructure layer owns the sub-controllers driving the physical actions requested by the upper plane. That involves the IoT devices in the interaction with the sensors and actuators.

More in general, the lower part of this plane is not managed directly with an approach HTTP/REST-based, as commonly happens with some control devices used in the industrial environment that expose only M2M interfaces. The IoT devices directly control these peripheral devices. Those are directly linked, or in some cases, integrated directly with the peripheral devices. IoTs are able to receive instructions from the nearest sub-controllers. The sub-controllers are local node or IoT gateway equipped with sensing and actuation devices. These elements are meant to translate commands into actions connecting outgoing and incoming data streams generated by the (existing and newcomer) devices in the managed environments, thus providing a virtual bus in which devices exchange data. The DP has to expose functionalities and Remote Procedure Calls (RPC) to receive directives meant to shape the virtual bus dynamically. To contextualize this plane with the fire system's environment, a reader can analyze Figure 9.2. Here, the devices taken into account are Smoke, Temperature, Pressure, and Flow sensors; moreover, as actuators, we use Valves, Sprinklers, Acoustic Alarms, and Electro-mechanical Hold-open(doors).

Concerning the right side of Figure 9.1, it is possible to associate the DP's sub-controllers to the Local Controller (LC) or Enhanced-LC (ELC) depending on their processing and networking capabilities. One of the main differences between an ELC and an LC is wireless connectivity in the former. In case of fire, if it is needed, it can be used to deliver and receive the command via a wireless connection created with the firefighters. An ELC can be promoted to Area Controller (AC) if it needs to work in "disconnected mode" (e.g., when the connection with some buildings of the SDF might be down due to the fire).

An Area Controller, or an external entity when the sub-controller acts

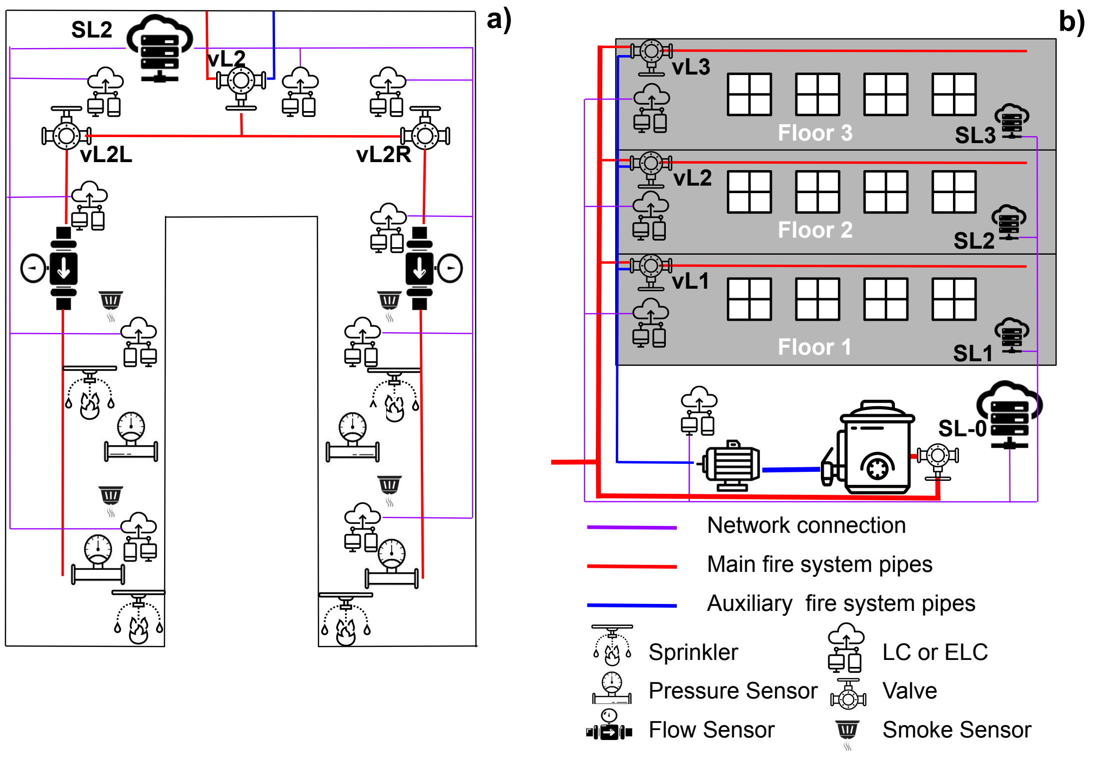


Figure 9.2: *SB schematics [10]. a) Floor view. b) Whole SB view.*

as Enhanced-LC interact with the LC or ELC exploiting the I/O Router RPC requests (or subscribing events) to interact with the Fire System's devices controlled by it. The same approach is used by the Fire Controller Plugin that stays side by side to the DP's Routers; see Figure 9.1.

The Control plane corresponds to the Management layer of the SDF; it contains two parts of the FFPS: the "Controllers" (AC/ELC) and the "Federator". The Control Plane is usually deployed in Cloud and/or Fog nodes. In the lower part are running the ACs, or in exceptional cases, an ELC promoted to AC (it happens when the logical infrastructure topology modification obliges the FFPS to provide an isolated area with an AC). Here, the paradigm used to manage the workflow, and the computation task is Fog Computing. The Controllers instruct the entities (e.g., traffic relayers) in the Data Plane, via Southbound interface, following the requests received through the Northbound interface (corresponding to a request coming from the Federator or the DP Fire Control Plugin[1]). The CP Fire Controller is shown in figure 9.1 as a component of the *Controllers* block, represents a typical *fire alarm control unit* (FACU), receiving and delivering data (raw and processed) from/to Federator and Fire Control Plugins. Control Plane's commands are referring to fire management, networking, and/or I/O aspects and functionalities of the target smart environments, acting on the corresponding controller (Fire, Net, I/O, respectively, as shown in figure 9.1) running on each AC.

In the upper part of this plane, the "Federator" runs; it is a central element of the infrastructure with stronger computation facilities. According to its purposes, it does not need to stay near the federated domains because its interaction is more related to management and reporting than real-time tasks. It is charged with coordination activities about the requests coming by both a) the CP domains requesting action autonomously and b) the users of the FFPS (Fire-Fighters, Security-vigilant, Factory responsible, and so on). So it is easy to understand that Federator exploits the Cloud Computing paradigm.

The last advantage provided by the FFPS platform, through the medi-

[1]DP Fire Control Plugin can make requests via Northbound interface to the Controllers in case of an emergency appears, and an immediate reaction is due, e.g., open the valve to 100% to maximize the water flowing in the pipes.

ation of the Federator, is the possibility to enable the firefighter mobile control center[2] to interact with the FFPS. This way, the firefighters will be updated about the environmental condition before entering a building and can not only identify the better strategy to face the fire but even use the automatism and actuation available in a building to extinguish the fire.

[2]The firefighter's mobile control center, is a supporting structure providing (data aggregation and processing, fire monitoring, communication) management facilities to firefighter.

CHAPTER 10

An effort to create and distribute a template for Smart Cities: the TOO(L)SMART Project

As discussed in the introduction and shown in Figure 1.3, the long term goal of this research path is the definition and the creation of an environment composed of cooperating CPSs. Moreover, this research aims to create a

"CPS Layer" overlapping the physical environment to support people's lives fully.

Even if the final goal is very far to be reached, an impressive intermediate result may make each city "Smart". In the last years, several initiatives are conducted from the government (national but even coming from the European Community) to promote and push the cities' transformation. This way surely has produced a big jump forward, but without a real guideline, each path followed in this process, even if pointing to the same destination, follows different directions. The previous issue appears when the transformation process is made in different geographical areas and over time. This happens because the different initiatives' owners do not always belong to the same administration and/or designed with the same criteria as the previous one. Trying to solve this unpleasant practice, national governments have released some guidelines to unify digitization processes. In particular, in Italy in recent years, a special agency (Agenzia per l'Italia Digitale, AgID) has been created to digitize infrastructure and public administration. AgID has defined some criteria for the adoption of technologies and systems related to the digitization process, pushing administrations to "Reuse" the solutions adopted by administrations that had already carried out this digitization process. Following the directives issued by AgID, the TOO(L)SMART project [149] aims to reuse what was previously realized within the #SmartMe project [150], enhancing it from the technological point of view and allowing a wide experimentation scenario for the solution. In fact, the architecture has been prepared in different cities (Turin, Padua, Lecce, Syracuse), which, being geo-located in different areas of Italy, allowed to verify the goodness of the architecture into the different operative conditions (weather, infrastructure, social engagement, and so on).

The idea is to create a template shareable between the city to set up a CPS's skeleton pluggable and extensible over time. For this purpose, a reference model was followed to design the template for a Smart City. Bawany et al. in [151] describe a 4-layer model to summarize the Smart City systems. At the bottom, the foundation of a Smart City is the (ICT) infrastructure composed of the actuators and sensors located in the urban area, including both private and public devices (e.g., smart cameras, air pollution, and weather stations, traffic lights, and so on). Furthermore, network resources and related issues have to be considered in the Infrastructure layer, as well as storage and processing facilities able to collect, manage, and process data. The management layer is meant to provide basic, core mechanisms for enabling the Smart City infrastructure, exploited by users/citizens, to access all the available resources. Then it requested a Management layer working on top of the infrastructure one, providing platform-advanced features based on and extending the infrastructure core mechanisms (e.g., security, privacy, monitoring, profiling, SLA, and QoS mechanisms and policies, and so on). The Application layer supports the Smart City services and applications that are referring to contexts such as mobility, energy, water, waste, public safety, and mobility building management, to name a few. Finally, the Stakeholder layer includes all the entities involved in a Smart city (e.g., the municipality, citizens, enterprises, telecommunication operators, officers, and so on).

TOO(L)SMART is a project aiming to reuse and improve the output of the #SmartME project [152] that was a crowd-funded initiative aiming at morphing Messina into a Smart City [153]. In line with its ancestor, TOO(L)SMART's main goal is to disseminate IoT resources throughout the territory of the city adopting its architecture, to obtain an infrastructure for ubiquitous sensing and actuation acting as a virtual laboratory to which multiple stakeholders can contribute with their own resources and on top of which they can develop applications and services for research, business, and administrative activities.

The TOO(L)SMART's template of a Smart City architecture includes the first three layers of the 4-layer model defined by [151], starting from the lower layer.

On the Infrastructure Layer, there are the Edge Nodes, these devices are spread into the city, and they are exposing the sensors and actuators facilities. The Edge nodes that provide their facilities to the upper layer host a software called Lighting-Rod(LR) that is a part of the Stack4Things(S4T) framework (see section 4.2). S4T is composed of two main components: the first one (LR) is running on the Edge de-

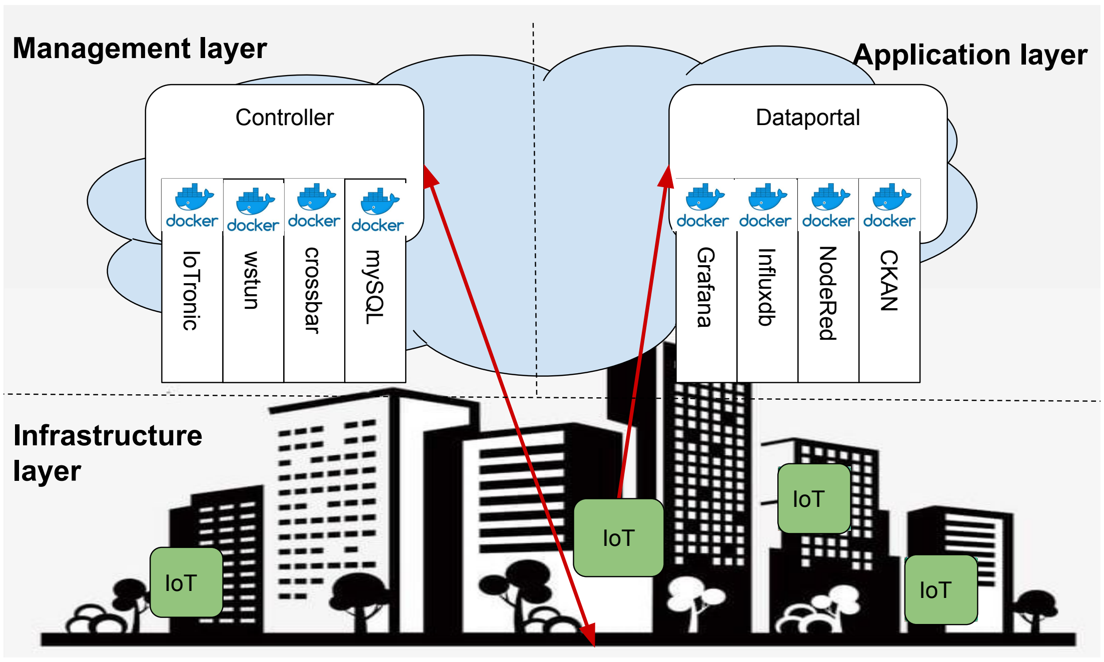


Figure 10.1: *High Level architecture of the TOO(L)SMART template.*

vices; instead, the other one is IoTronic (see section 4.2) running in the Cloud with management purposes; for this reason, with other few components, it runs inside the Controller node that represents the Management Layer of this architecture. The S4T framework follows an on-demand, service-oriented provisioning model to manage IoT nodes. This is possible moving the IoT paradigm towards the Cloud for, on the one hand, providing control and management capabilities to IoT nodes and, on the other, extending the Cloud paradigm with pervasiveness capabilities to interact with the physical world. The architecture so proposed in the TOO(L)SMART template is modeled on the client-server paradigm. On the IoT node side, the Stack4Things lighting-rod runs under the device-native environment available for developers and interacts with the OS tools and services available. Interactions are implemented by UNIX-style, exploiting file system-based abstractions of the underlying interfaces, either GPIO for embedded boards or API-mediated for mobiles, and enforced by WebSocket-based tunneling and WAMP-based messaging between the Stack4Things lightning-rod and IoTronic service. Regarding the technologies implemented in the project experimentation, we are referring to a system called Arancino described in section 4.3.

The last layer, the Application layer, is represented by the Dataportal node where are contained the API, the dashboard, and the Open Data repository. The services exposed by this component enable developers to define applications that exploit the data and interacts with the devices (according to the right access owned) through the tools available, as defined in the template via Node-Red interaction. An example of application realized upon this template is shown in Figure 10.2; the instruments provided both by the application and the management layers are used in a Node-Red flow that, in this case, produces a map enriched by data coming from a weather station located in the cities.

Each city involved in the TOO(L)SMART project has provided her portal, the dataportal (shown in Figure 10.1) accessible with an URL in the format **https://dataportal.comune.XXX.it:1880/worldmap**, where XXX is the name of the city owning the dataportal.

As said before, the template represents a skeleton upon which is attached to the muscle and the tendons of the Smart City:

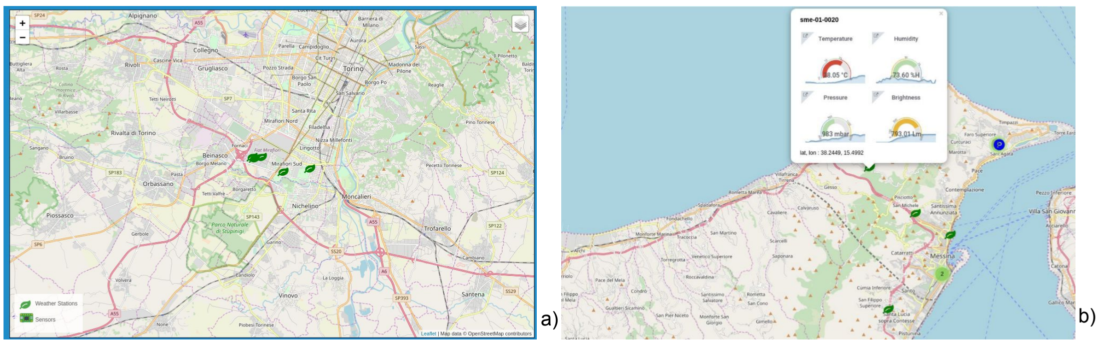


Figure 10.2: *Examples of the map developed through tools available in the template.*

- it offers a framework (S4T) ables to manage the IoT devices behavior and even their lifecycle;
- it offers an Open Source data repository connected to a time-series database for high-performance visualization and elaboration of data;
- it offers a system enabling the SC Administrator(or more in general a User) to create applications involving the IoT devices owned by the city.

Anyway, this is not the only reason why we can define this template as a skeleton. This is a skeleton because, thanks to the adoption of this model by several Italian cities and even because it is easily replicable (all the template can be hosted with 2 VMs), it will become the spinal column for disseminating Smart Cities made through few steps.

CHAPTER 11

Conclusions

WITHOUT any doubt, “Cyber Physical Systems” are becoming increasingly part of our daily routine. In the foreseeable future, the number of environments bearing at least a CPS will drastically increase, so creating and deploying applications working within a cooperation among several CPSs will become a natural evolution of CPSs close to people’s daily life.

Similarly to Cloud cooperation instances, where the cooperation agreement and the resource sharing are transparent to the end-user, a cooperative system composed of CPSs is better established with a federated cooperation scheme. Indeed, the federation of CPSs seems to be a suitable approach working effectively for environments owned by different domains that can interact with each other reducing the risks of misuse or weakness exploitation. This is valid both in cooperation mediated by a third party agent (a broker) than when there is any autonomous management (i.e., a peer management). In fact, federating domains is a very broad concept that encompasses many branches of research, such as interoperability between different systems, definition and compliance with SLA policies, users' data security, creation of trusted and high-quality environments in terms of both the availability and reliability of the services provided. After a preliminary study of the literature, on the well-trodden, and comparable scenario of cooperation among Clouds, this thesis presents some solutions that may be applied to a single CPS, but easily extensible towards cooperation with other CPSs. Indeed, in chapter 5 a structure of a CPS is presented that may be easily integrated with pre-existing devices (sensors and actuators) in an environment. This structure is meant to be managed from a centralized element (the Stack4Things framework) that enables the administrator (or who owns the system) to update or modify AI running on the device, under the guise of a plugin, according to the environment's requirements. This way, the system can benefit from several advantages: i) easy functionality restore in case of error or fault, ii) a centralized data repository, iii) the chance to extend the system functionalities through definition, injection, and execution of new AI-based modules that are executed on the Edge devices involved in the system (according to the available resources at the edge). The results obtained highlight mostly two underlying limitations that can be overcome thanks to cooperation among CPSs in the analyzed scenarios and use cases: the need to increase the compute power available on systems at the edge of the network, often constrained by design, and the requirement to interact with a neighbor to increase the data available to improve hosted services.

The latter consideration is supported by the results shown by Figures 6.2 and 6.6 in which the application computing the data coming from another CPS (the Smart City) is able to produce an improvement of the CPS behavior (the vehicle) and consequently provide benefits to the people interacting with it. The system envisioned and discussed in chapter 6 exploits communications that connect the vehicle with the centralized (possibly cloud-based) Smart City computation facilities, so the computation and all the periodical "*update-requests*" coming from the vehicles are sent towards a centralized system. Even if the cooperation works fine and, as it is proved, the system is able to improve the traversal time (or the fuel consumption) of vehicles moving inside the city, it appears clear that something could be improved in this infrastructure;

an example may be the need to manage the huge amount of requests received by the centralized system and the computation connected to these requests in closer proximity to the requesting vehicles. This way, it will be possible to increase the efficiency of the communication between the SC and the vehicles.

For these reasons, the research scope has been extended to include serverless approaches, i.e., those that applied on a CPS enable the whole computing continuum infrastructure (Cloud, Fog, and Edge) to cooperate, splitting bigger and higher-complexity application tasks into smaller and simpler ones. This way, it is possible to enable one or more CPSs to interact quickly with the users or the environment, in near real-time, because computation gets shifted near the site where it is needed the most. Another interesting feature that a CPS may provide lies in sharing I/O data with other CPSs to enable remote computation upon such data. This feature is particularly adaptable for scenarios where the SLA agreed by cooperating CPSs is very strict, and the degree of cooperation is very loose. The experiments and the applications here investigated underline that, even in a loosely coupled scenario, it is possible to define interesting and dynamic configurations for cooperating CPSs. As demonstrated by the experimental results presented in chapter 8, the cooperation among CPSs can be realized without affecting the performance of the resulting combination. As shown in Figure 8.4, the implementation of cooperation among environments is very promising thanks to the quasi-linear scalability for data and command stream brokering with little (if not negligible) additional delays, and this means that the cooperation among CPSs is exploitable without counteracting the advantages introduced by shifting computing duties to the Edge.

Finally, to foster (federated) CPSs adoption, in particular, in the context of Smart Cities, a base template for CPSs has been proposed and deployed in five major cities under an Italian government-sponsored project called Too(L)smart.

In future works, the experiences gathered along the investigations I pursued have to be collected and integrated into a coherent worldview made up of a hierarchical-composable CPS able to be easily interfaced with both pre-existing systems and other CPSs. This way, the resulting system can provide computing and storage facilities more granularly, avoiding the problems related to latency and communication, thanks to the exploitation of Computing Continuum principles. Furthermore, we are going to define an approach that can mix the benefits coming from the Serverless-FaaS with control of the owner coming from the interaction with the I/O based approach. The solution thus created could be included in the template to be applied in a realistic scenario, e.g., a Smart City. Moreover, the adoption of CPSs and the efforts for cooperation among them will introduce another complex problem to face: the definition of a standard and agreed communication interface by each CPSs. Researchers and governments' actions are fundamental to reach the cooperation goal, even if the experience obtained from the previous initiatives made in Cloud Computing fields could guide us to make the right decision and avoid some locking situation. Examples of issues suffered by the Cloud Computing cooperation schema are the late definition of a standard common communication interface that has to adapt to the pre-existent technologies or the difficulties of setting up cooperative schema due to the reticence change market balances.

Other interesting topics to analyze as next steps for my research in the near future about interconnected (and possibly, federated) CPSs are related to "*Security*", "*Authorization and Delegation*", "*Dynamic management*", "*Autonomous Integration*", and so on. These cooperation-based approaches, utilities and applications may be the cornerstone for the autonomous management of cooperating CPSs. Blockchains and Machine Learning can help in addressing various aspects pertaining to previous topics. Blockchain-based approaches mostly to ensure the *integrity*, *immutability*, and *transparent* behaviours of contextual data, possibly able to disclose the privacy of the people; instead, Machine Learning empowers the investigation and design of mechanisms for dynamic and autonomous management of the interactions to broker among cooperating CPSs.

Bibliography

[1] [Online]. Available: https://www.plesk.com/blog/various/iaas-vs-paas-vs-saas-various-cloud-service-models-compared/

[2] (2013). [Online]. Available: https://mycloudblog7.wordpress.com/2013/06/19/who-manages-cloud-iaas-paas-and-saas-services/

[3] T. Kurze, M. Klems, D. Bermbach, A. Lenk, S. Tai, and M. Kunze, "Cloud federation," in *Proceedings of the 2nd International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2011)*, vol. 1971548541. Citeseer, 2011.

[4] Berkeley. Cyber physical systems conceptual map. [Online]. Available: http://CyberPhysicalSystems.org

[5] OpenStack Community, 2020, https://docs.openstack.org/install-guide/get-started-logical-architecture.html.

[6] G. Cicceri, C. Scaffidi, Z. Benomar, S. Distefano, A. Puliafito, G. Tricomi, and G. Merlino, "Smart healthy intelligent room: Headcount through air quality monitoring," in *SmartSys 2020 workshop held in Smartcomp*. IEEE, sep 2020.

[7] C. Scaffidi, G. Tricomi, S. Distefano, and Puliafito, “Scinas, a smart city-driven navigation system to catch green waves,” in *Conference on Sustainable Mobility, CSM 2020*. SAE, sep 2020.

[8] G. Tricomi, Z. Benomar, F. Aragona, G. Merlino, F. Longo, and A. Puliafito, “A nodered-based dashboard to deploypipelines on top of iot infrastructure,” in *SMARTCOMP*. IEEE, sep 2020.

[9] G. Tricomi, G. Merlino, F. Longo, S. Distefano, and A.Puliafito, “Software-defined city infrastructure: a control plane for rewireable smart cities,” in *2019 IEEE 5th Intl Conf on Smart Computing and Its Associated Workshops (SMARTCOMP)*.

[10] G. Tricomi, C. Scaffidi, G. Merlino, F. Longo, A. Puliafito, and S. Distefano, “An adaptive fire protection system for software-defined factories,” oct 2020, submitted to COMPUTERS IN INDUSTRY waiting for response.

[11] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of things (iot): A vision, architectural elements, and future directions,” *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645 – 1660, 2013, including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services Cloud Computing and Scientific Applications, Big Data, Scalable Analytics, and Beyond. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X13000241

[12] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelfflé, “Vision and challenges for realising the internet of things,” *Cluster of European Research Projects on the Internet of Things, European Commision*, vol. 3, no. 3, pp. 34–36, 2010.

[13] L. Srivastava, T. Kelly *et al.*, “The internet of things,” *International Telecommunication Union, Tech. Rep*, vol. 7, 2005.

[14] J. Bélissent *et al.*, “Getting clever about smart cities: New opportunities require new business models,” *Cambridge, Massachusetts, USA*, vol. 193, pp. 244–77, 2010.

[15] A. Ting-pat So and W. Lok Chan, *Intelligent Building Systems*, 1st ed., ser. The International Series on Asian Studies in Computer and Information Science 5.

Springer US, 1999. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4615-5019-8_13

[16] B. Qolomany, A. Al-Fuqaha, A. Gupta, D. Benhaddou, S. Alwajidi, J. Qadir, and A. C. Fong, "Leveraging machine learning and big data for smart buildings: A comprehensive survey," *IEEE Access*, vol. 7, pp. 90 316–90 356, 2019. [Online]. Available: https://doi.org/10.1109/access.2019.2926642

[17] E. Commission, D.-G. for the Information Society, and Media. (Luxembourg, 2009.) Ict for a low carbon economy: Smart electricity distribution networks. [Online]. Available: https://ec.europa.eu/information_society/activities/sustainable_growth/docs/sb_publications/smartbuildings-ld.pdf

[18] L. Thames and D. Schaefer, "Software-defined cloud manufacturing for industry 4.0," *Procedia CIRP*, vol. 52, pp. 12 – 17, 2016, the Sixth International Conference on Changeable, Agile, Reconfigurable and Virtual Production (CARV2016). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2212827116307910

[19] B. Chen, J. Wan, L. Shu, P. Li, M. Mukherjee, and B. Yin, "Smart factory of industry 4.0: Key technologies, application case, and challenges," *IEEE Access*, vol. 6, pp. 6505–6519, 2018.

[20] M. Brettel, N. Friederichsen, M. A. Keller, and M. Rosenberg, "How virtualization, decentralization and network building change the manufacturing landscape: An industry 4.0 perspective," 2014.

[21] G. Merlino, D. Bruneo, F. Longo, A. Puliafito, and S. Distefano, "Software defined cities: A novel paradigm for smart cities through iot clouds," in *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*, Aug 2015, pp. 909–916.

[22] M. S. Q. Z. Nine, M. A. K. Azad, S. Abdullah, and N. Ahmed, "Dynamic load sharing to maximize resource utilization within cloud federation," in *Cloud Computing and Big Data*. Springer Science + Business Media, 2015, pp. 125–137.

[23] L. Barreto, J. Fraga, and F. Siqueira, "Conceptual model of brokering and authentication in cloud federations," in *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*. Institute of Electrical and Electronics Engineers (IEEE), oct 2015.

[24] A. Kertesz, G. Kecskemeti, M. Oriol, P. Kotcauer, S. Acs, M. Rodríguez, O. Mercè, A. C. Marosi, J. Marco, and X. Franch, "Enhancing federated cloud management with an integrated service monitoring approach," *J Grid Computing*, vol. 11, no. 4, pp. 699–720, Jun. 2013.

[25] G. Tricomi, A. Panarello, G. Merlino, F. Longo, D. Bruneo, and A. Puliafito, "Orchestrated multi-cloud application deployment in OpenStack with TOSCA," in *2017 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, may 2017.

[26] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. DaSilva, C. Lee, and O. Rana, "The internet of things, fog and cloud continuum: Integration and challenges," *Internet of Things*, vol. 3, pp. 134–155, 2018.

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

[28] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[29] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "Serverless programming (function as a service)," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 2658–2659.

[30] G. Tricomi, A. Panarello, G. Merlino, and A. Puliafito, "Optimal selection techniques for cloud service providers," *IEEE Access*, Nov 2020, DOI: 10.1109/ACCESS.2020.3035816.

[31] BEACON, “The beacon - enabling federated cloud networking project,” 2015. [Online]. Available: https://cordis.europa.eu/project/rcn/194143/factsheet/en

[32] C. Scaffidi, G. Tricomi, S. Distefano, and A. Puliafito, “Continuous green^2 waves for surfing smart cities,” in *SSC 2020 workshop held in Smartcomp*. IEEE, sep 2020.

[33] G. Tricomi, D. Giosa, G. Merlino, O. Romeo, and F. Longo, “Toward a function-as-a-service framework for genomic analysis,” in *SmartSys 2020 workshop held in Smartcomp*. IEEE, sep 2020.

[34] G. Tricomi, Z. Benomar, G. Merlino, F. Longo, A. M. Longo, and A. Puliafito, “Too(l)smart: A template to make cities "smart",” in *i-CITIES*, 2020.

[35] G. Tricomi, F. Longo, G. Merlino, and A. Puliafito, “From a smart city service platform towards a smart city template: smartme & too(l)smart projects.” 2020, submitted for Smart City Workshop in Hanoi.

[36] P. Mell and T. Grance, “The nist definition of cloud computing,” National Institute of Standards and Technology (NIST), Gaithersburg, MD, Tech. Rep. 800-145, September 2011. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[37] A. Botta, W. De Donato, V. Persico, and A. Pescapé, “Integration of Cloud computing and Internet of Things: a survey,” *Future generation computer systems*, vol. 56, pp. 684–700, 2016.

[38] M. Díaz, C. Martín, and B. Rubio, “State-of-the-art, challenges, and open issues in the integration of internet of things and cloud computing,” *Journal of Network and Computer applications*, vol. 67, pp. 99–117, 2016.

[39] P. P. Ray, “A survey of iot cloud platforms,” *Future Computing and Informatics Journal*, vol. 1, no. 1-2, pp. 35–46, 2016.

[40] C. Puliafito, E. Mingozzi, F. Longo, A. Puliafito, and O. Rana, “Fog computing for the internet of things: A survey,” *ACM Transactions on Internet Technology (TOIT)*, vol. 19, no. 2, p. 18, 2019.

[41] T. H. Luan, L. Gao, Z. Li, Y. Xiang, G. Wei, and L. Sun, “Fog computing: Focusing on mobile users at the edge,” *arXiv preprint arXiv:1502.01815*, 2015.

[42] P. Hu, S. Dhelim, H. Ning, and T. Qiu, “Survey on fog computing: architecture, key technologies, applications and open issues,” *Journal of network and computer applications*, vol. 98, pp. 27–42, 2017.

[43] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[44] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The case for vm-based cloudlets in mobile computing,” *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[45] A. Davis, J. Parikh, and W. E. Weihl, “Edgecomputing: extending enterprise applications to the edge of the internet,” in *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, 2004, pp. 180–187.

[46] A. Rabay’a, E. Schleicher, and K. Graffi, “Fog computing with p2p: Enhancing fog computing bandwidth for iot scenarios,” in *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2019, pp. 82–89.

[47] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, “On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration,” *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1657–1681, thirdquarter 2017.

[48] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, *Mobile Edge Computing A key technology towards 5G*. ETSI, 2015, vol. White Paper No. 11.

[49] I. Morris. (2016, sep) Etsi drops ’mobile’ from mec. [Online]. Available: https://www.lightreading.com/mobile/mec-(mobile-edge-computing)/etsi-drops-mobile-from-mec/d/d-id/726273

[50] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. DaSilva, C. Lee, and O. Rana, "The internet of things, fog and cloud continuum: Integration and challenges," *Internet of Things*, vol. 3-4, pp. 134 – 155, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2542660518300635

[51] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy, "Task scheduling strategies for workflow-based applications in grids," in *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.*, vol. 2. IEEE, 2005, pp. 759–767.

[52] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 26–35, 2017.

[53] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, "Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3702–3712, 2016.

[54] H. Xu and B. Li, "Joint request mapping and response routing for geo-distributed cloud services," in *2013 Proceedings IEEE INFOCOM*. IEEE, 2013, pp. 854–862.

[55] C.-C. Hung, L. Golubchik, and M. Yu, "Scheduling jobs across geo-distributed data-centers," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, 2015, pp. 111–124.

[56] B. Heintz, A. Chandra, R. K. Sitaraman, and J. Weissman, "End-to-end optimization for geo-distributed mapreduce," *IEEE Transactions on Cloud Computing*, vol. 4, no. 3, pp. 293–306, 2014.

[57] T. Lynn, P. Rosati, A. Lejeune, and V. Emeakaroha, "A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Dec 2017, pp. 162–169.

[58] A. Glikson, S. Nastic, and S. Dustdar, “Deviceless edge computing: Extending serverless computing to the edge of the network,” in *In Proceedings of the 10th ACM International Systems and Storage Conference (SYSTOR)*, 2017.

[59] A. Alvarado. (2019, Sep) Serverless vs. FaaS: A beginner’s guide. [Online]. Available: https://www.liquidweb.com/kb/serverless-vs-faas-a-beginners-guide/

[60] P. Johnston. (2018, jul) Serverless: It’s much much more than FaaS. [Online]. Available: https://medium.com/@PaulDJohnston/serverless-its-much-much-more-than-faas-a342541b982e

[61] L. E. Hecht. (2018, oct) Add it up: Faas not equal serverless. [Online]. Available: https://thenewstack.io/add-it-up-serverless-faas/

[62] CNCF-Serverless-Working-Group. (2018, mar) Serverless whitepaper v1.0, tech. rep., cloudnative computing foundation. [Online]. Available: https://github.com/cncf/wg-serverless/tree/master/whitepapers/serverless-overview

[63] Apache. (2019) Apache OpenWhisk. [Online]. Available: https://openwhisk.apache.org/

[64] Microsoft. (2019) Azure IoT Edge. [Online]. Available: https://azure.microsoft.com/en-us/services/iot-edge/

[65] Amazon. (2019) AWS Greengrass. [Online]. Available: https://aws.amazon.com/greengrass/

[66] IBM. (2019) Ibm Watson IoT platform. [Online]. Available: https://www.ibm.com/cloud/internet-of-things

[67] B. Cheng, J. Fuerst, G. Solmaz, and T. Sanada, “Fog function: Serverless fog computing for data intensive iot services,” in *2019 IEEE International Conference on Services Computing (SCC)*. IEEE, 2019, pp. 28–35.

[68] P. Persson and O. Angelsmark, “Kappa: serverless iot deployment,” in *Proceedings of the 2nd International Workshop on Serverless Computing*, 2017, pp. 16–21.

[69] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14–22, 2009.

[70] R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, "An elasticity model for high throughput computing clusters," *Journal of Parallel and Distributed Computing*, vol. 71, no. 6, pp. 750 – 757, 2011, special Issue on Cloud Computing. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731510000985

[71] A. Celesti, F. Tusa, M. Villari, and A. Puliafito, "Integration of clever clouds with third party software systems through a rest web service interface," in *2012 IEEE Symposium on Computers and Communications (ISCC)*, 2012, pp. 000 827–000 832.

[72] A. Celesti, F. Tusa, M. Villari, and A. Puliafito, "How the dataweb can support cloud federation: Service representation and secure data exchange," in *2012 Second Symposium on Network Cloud Computing and Applications*, 2012, pp. 73–79.

[73] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers," *Future generation computer systems*, vol. 28, no. 2, pp. 358–367, 2012.

[74] D. Petcu, "Multi-cloud," in *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds - MultiCloud*. Association for Computing Machinery (ACM), 2013.

[75] Artemis industry associations. [Online]. Available: https://artemis-ia.eu/

[76] C. Someswara Rao, K. V. S. Murthy, S. V. Appaji, and R. Shiva Shankar, "Cyber-physical systems security: Definitions, methodologies, metrics, and tools," in *Smart Intelligent Computing and Applications*, S. C. Satapathy, V. Bhateja, J. R. Mohanty, and S. K. Udgata, Eds. Singapore: Springer Singapore, 2020, pp. 477–488.

[77] [Online]. Available: :http://newsinfo.nd.edu/news/17248-nsf-funds-cyber-physical-systems-project/.

[78] J. Wan, H. Yan, H. Suo, and F. Li, “Advances in cyber-physical systems research.” *KSII Transactions on Internet & Information Systems*, vol. 5, no. 11, 2011.

[79] D. Snoonian, “Smart buildings,” *IEEE Spectrum*, vol. 40, no. 8, pp. 18–23, Aug 2003.

[80] M. R. Alam and M. A. M. Reaz, M. B. I.and Ali, “A review of smart homes past, present, and future,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1190–1203, Nov 2012.

[81] J. Pan, R. Jain, and S. Paul, “A survey of energy efficiency in buildings and microgrids using networking technologies,” *IEEE Communications Surveys Tutorials*, vol. 16, no. 3, pp. 1709–1731, Third 2014.

[82] Berkeley. (2016) Software defined building. [Online]. Available: http://sdb.cs.berkeley.edu/sdb

[83] S. Dawson-Haggerty, A. Krioukov, J. Taneja, S. Karandikar, G. Fierro, N. Kitaev, and D. Culler, “{BOSS}: Building operating system services,” in *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, 2013, pp. 443–457.

[84] G. Fierro and D. E. Culler, “Poster abstract: Xbos: An extensible building operating system,” in *Proceedings of the 2Nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, ser. BuildSys ’15. New York, NY, USA: ACM, 2015, pp. 119–120. [Online]. Available: http://doi.acm.org/10.1145/2821650.2830311

[85] C. Blumstein, D. Culler, G. Fierro, T. Peffer, and M. Pritoni. (2015) Open software-architecture for building monitoring and control. paper and presentation at sustainable places, pp 16 - 18. [Online]. Available: https://people.eecs.berkeley.edu/~gtfierro/papers/open_arch.pdf

[86] B. Rochwerger, D. Breitgand, A. Epstein, D. Hadas, I. Loy, K. Nagin, J. Tordsson, C. Ragusa, M. Villari, S. Clayman, E. Levy, A. Maraschini, P. Massonet, H. Mu, and G. Tofetti, “Reservoir - when one cloud is not enough,” *Computer*, vol. 44, no. 3, pp. 44–51, Mar. 2011.

[87] Stratuslab, "Stratuslab, darn simple cloud." 2010-2012. [Online]. Available: http://www.stratuslab.eu/

[88] Bonfire, "Bonfire project," http://www.bonfire-project.eu/, 2010-2013. [Online]. Available: http://www.bonfire-project.eu/

[89] A. C. Hume, Y. Al-Hazmi, B. Belter, K. Campowsky, L. M. Carril, G. Carrozzo, V. Engen, D. García-Pérez, J. J. Ponsatí, R. Kűbert, Y. Liang, C. Rohr, and G. V. Seghbroeck, "BonFIRE: A multi-cloud test facility for internet of services experimentation," in *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. Springer Science + Business Media, 2012, pp. 81–96.

[90] D. García-Pérez, J. Á. L. del Castillo, Y. Al-Hazmi, J. Martrat, K. Kavoussanakis, A. C. Hume, C. V. López, G. Landi, T. Wauters, M. Gienger, and D. Margery, "Cloud and network facilities federation in BonFIRE," in *Euro-Par 2013: Parallel Processing Workshops*. Springer Science + Business Media, 2014, pp. 126–135.

[91] CONTRAIL, "Contrail - cloud federation computing project," 2010-2014. [Online]. Available: http://contrail-project.eu/

[92] R. G. Cascella, L. Blasi, Y. Jegou, M. Coppola, and C. Morin, "Contrail: Distributed application deployment under SLA in federated heterogeneous clouds," in *The Future Internet*. Springer Science + Business Media, 2013, pp. 91–103.

[93] VISION, "Vision cloud project, funded by the european commission seventh framework programme (fp7/2006-2013) under grant agreement n. 257019." 2010-2013. [Online]. Available: http://www.visioncloud.eu/

[94] Mosaic, "Mosaic - multi-modal situation assessment and analytics platform." 2011-2014. [Online]. Available: http://www.mosaic-fp7.eu/

[95] C. Project, "Cloudwave project, funded by the european commission seventh framework programme (fp7/2006-2013) under grant agreement n. 610802." 2013. [Online]. Available: http://cloudwave-fp7.eu/

[96] A. Nus and D. Raz, “Migration plans with minimum overall migration time,” in *2014 IEEE Network Operations and Management Symposium (NOMS)*. Institute of Electrical & Electronics Engineers (IEEE), May 2014.

[97] S. Yangui, I.-J. Marshall, J.-P. Laisne, and S. Tata, “CompatibleOne: The open source cloud broker,” *J Grid Computing*, vol. 12, no. 1, pp. 93–109, Nov. 2013.

[98] R. Moreno-Vozmediano, E. Huedo, I. Llorente, R. Montero, P. Massonet, M. Villari, G. Merlino, A. Celesti, A. Levin, L. Schour, C. VÃÂ¡zquez, J. Melis, S. Spahr, and D. Whigham, “Beacon: A cloud network federation framework,” *Communications in Computer and Information Science*, vol. 567, pp. 325–337, 2016.

[99] SUNFISH, “Sunfish - secure information sharing in federated heterogeneous private clouds project,” 2015-2017. [Online]. Available: https://cordis.europa.eu/project/rcn/194230/factsheet/en

[100] SUPERCLOUD, “Supercloud - user-centric management of security and dependability in clouds of clouds project,” 2015-2018. [Online]. Available: https://cordis.europa.eu/project/rcn/194123/factsheet/en

[101] FIESTA, “Fiesta - federated interoperable semantic iot/cloud testbeds and applications project,” 2015-2018. [Online]. Available: https://cordis.europa.eu/project/rcn/194117/factsheet/en

[102] S. K. Garg, S. Versteeg, and R. Buyya, “SMICloud: A framework for comparing and ranking cloud services,” in *2011 Fourth IEEE International Conference on Utility and Cloud Computing*. Institute of Electrical & Electronics Engineers (IEEE), Dec. 2011.

[103] T. Subramanian and N. Savarimuthu, “Application based brokering algorithm for optimal resource provisioning in multiple heterogeneous clouds,” *Vietnam J Comput Sci*, vol. 3, no. 1, pp. 57–70, Dec. 2015.

[104] H. Kurdi, A. Al-Anazi, C. Campbell, and A. A. Faries, “A combinatorial optimization algorithm for multiple cloud service composition,” *Computers & Electrical Engineering*, vol. 42, pp. 107–113, Feb. 2015.

[105] M. Caballer, I. Blanquer, G. Moltó, and C. de Alfonso, "Dynamic management of virtual infrastructures," *J Grid Computing*, vol. 13, no. 1, pp. 53–70, Apr. 2014.

[106] Q. Duan and A. V. Vasilakos, "Federated selection of network and cloud services for high-performance software-defined cloud computing," *International Journal of High Performance Computing and Networking*, vol. 9, no. 4, p. 316, 2016.

[107] Z. ur Rehman, F. K. Hussain, and O. K. Hussain, "Towards multi-criteria cloud service selection," in *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. Institute of Electrical & Electronics Engineers (IEEE), Jun. 2011.

[108] J. O. de Carvalho, F. Trinta, and D. Vieira, "PacificClouds: A flexible MicroServices based architecture for interoperability in multi-cloud environments," in *Proceedings of the 8th International Conference on Cloud Computing and Services Science*. SCITEPRESS - Science and Technology Publications, 2018.

[109] D. Lin, A. C. Squicciarini, V. N. Dondapati, and S. Sundareswaran, "A cloud brokerage architecture for efficient cloud service selection," *IEEE Transactions on Services Computing*, vol. 12, no. 1, pp. 144–157, Jan 2019.

[110] C. Redl, I. Breskovic, I. Brandic, and S. Dustdar, "Automatic SLA matching and provider selection in grid and cloud computing markets," in *2012 ACM/IEEE 13th International Conference on Grid Computing*. Institute of Electrical & Electronics Engineers (IEEE), Sep. 2012.

[111] S. Sundareswaran, A. Squicciarini, and D. Lin, "A brokerage-based approach for cloud service selection," in *2012 IEEE Fifth International Conference on Cloud Computing*. Institute of Electrical & Electronics Engineers (IEEE), Jun. 2012.

[112] J. Carvalho, D. Vieira, and F. Trinta, "Dynamic selecting approach for multi-cloud providers," in *Cloud Computing – CLOUD 2018*, M. Luo and L.-J. Zhang, Eds. Cham: Springer International Publishing, 2018, pp. 37–51.

[113] J. B. Abdo, J. Demerjian, H. Chaouchi, K. Barbar, and G. Pujolle, “Broker-based cross-cloud federation manager,” in *8th International Conference for Internet Technology and Secured Transactions (ICITST-2013)*. Institute of Electrical & Electronics Engineers (IEEE), Dec. 2013.

[114] H. Mezni and M. Sellami, “Multi-cloud service composition using formal concept analysis,” *Journal of Systems and Software*, vol. 134, pp. 138 – 152, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121217301760

[115] S. Farokhi., F. Jrad., I. Brandic., and A. Streit., “Hs4mc - hierarchical sla-based service selection for multi-cloud environments,” in *Proceedings of the 4th International Conference on Cloud Computing and Services Science - Volume 1: MultiCloud, (CLOSER 2014)*, INSTICC. SciTePress, 2014, pp. 722–734.

[116] F. Jrad, J. Tao, and A. Streit, “A broker-based framework for multi-cloud workflows,” in *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds - MultiCloud 13*. Association for Computing Machinery (ACM), 2013.

[117] F. Jrad, J. Tao, A. Streit, R. Knapper, and C. Flath, “A utility-based approach for customised cloud service selection,” *IJCSE*, vol. 10, no. 1/2, p. 32, 2015.

[118] C. Esposito, M. Ficco, F. Palmieri, and A. Castiglione, “Smart cloud storage service selection based on fuzzy logic, theory of evidence and game theory,” *IEEE Transactions on Computers*, pp. 1–1, 2016.

[119] F. DAndria, S. Bocconi, J. G. Cruz, J. Ahtes, and D. Zeginis, “Cloud4Soa: Multi-cloud application management across PaaS offerings,” in *2012 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. Institute of Electrical & Electronics Engineers (IEEE), Sep. 2012.

[120] P. Massonet, J. Luna, A. Pannetrat, and R. Trapero, “Idea: Optimising multi-cloud deployments with security controls as constraints,” in *Lecture Notes in Computer Science*. Springer Science + Business Media, 2015, pp. 102–110.

[121] Y. Mansouri, A. N. Toosi, and R. Buyya, “Brokering algorithms for optimizing the availability and cost of cloud storage services,” in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*. Institute of Electrical & Electronics Engineers (IEEE), Dec. 2013.

[122] T. G. Papaioannou, N. Bonvin, and K. Aberer, “Scalia: An adaptive scheme for efficient multi-cloud storage,” in *2012 International Conference for High Performance Computing, Networking, Storage and Analysis*. Institute of Electrical & Electronics Engineers (IEEE), Nov. 2012.

[123] W. Yao and L. Lu, “A selection algorithm of service providers for optimized data placement in multi-cloud storage environment,” in *Communications in Computer and Information Science*. Springer Science + Business Media, 2015, pp. 81–92.

[124] M. Hadji, B. Aupetit, and D. Zeghlache, “Cost-efficient algorithms for critical resource allocation in cloud federations,” in *2016 5th IEEE International Conference on Cloud Networking (Cloudnet)*. Institute of Electrical and Electronics Engineers (IEEE), Oct. 2016.

[125] M. Giacobbe, M. Scarpa, R. D. Pietro, and A. Puliafito, “An energy-aware brokering algorithm to improve sustainability in community cloud,” in *Proceedings of the 6th International Conference on Smart Cities and Green ICT Systems*. SCITEPRESS - Science and Technology Publications, 2017.

[126] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, “Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers,” *Future Generation Computer Systems*, vol. 28, no. 2, pp. 358–367, Feb. 2012.

[127] C. Negru, F. Pop, O. C. Marcu, M. Mocanu, and V. Cristea, “Budget constrained selection of cloud storage services for advanced processing in datacenters,” in *2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER)*. Institute of Electrical & Electronics Engineers (IEEE), Sep. 2015.

[128] Y. Kajiura, S. Ueno, A. Kanai, S. Tanimoto, and H. Sato, "An approach to selecting cloud services for data storage in heterogneous-multicloud environment with high availability and confidentiality," in *2015 IEEE 12*th *International Symposium on Autonomous Decentralized Systems*. Institute of Electrical & Electronics Engineers (IEEE), Mar. 2015.

[129] F. Longo, D. Bruneo, S. Distefano, G. Merlino, and A. Puliafito, "Stack4Things: An OpenStack-based framework for IoT," in *2015 3rd International Conference on Future Internet of Things and Cloud*, Aug 2015, pp. 204–211.

[130] S. Distefano, G. Merlino, and A. Puliafito, "Device-centric sensing: An alternative to data-centric approaches," *IEEE Systems Journal*, vol. 11, no. 1, pp. 231–241, March 2017.

[131] D. Bruneo, S. Distefano, F. Longo, G. Merlino, and A. Puliafito, "I/Ocloud: Adding an IoT dimension to cloud infrastructures," *Computer*, vol. 51, no. 1, pp. 57–65, 2018.

[132] G. Merlino, D. Bruneo, F. Longo, S. Distefano, and A. Puliafito, "Cloud-based network virtualization: An IoT use case," in *Ad Hoc Networks*, N. Mitton, M. E. Kantarci, A. Gallais, and S. Papavassiliou, Eds. Cham: Springer International Publishing, 2015, pp. 199–210.

[133] G. Campobello, S. Serrano, A. Leonardi, and S. Palazzo, "Trade-offs between energy saving and reliability in low duty cycle wireless sensor networks using a packet splitting forwarding technique," *EURASIP Journal on Wireless Communications and Networking*, vol. 2010, no. 1, Aug. 2010. [Online]. Available: https://doi.org/10.1155/2010/932345

[134] O. Chenaru, G. Florea, A. Stanciu, V. Sima, D. Popescu, and R. Dobrescu, "Modeling complex industrial systems using cloud services," in *2015 20th International Conference on Control Systems and Computer Science*. IEEE, 2015, pp. 565–571.

[135] G. B. Fioccola, R. Sommese, I. Tufano, R. Canonico, and G. Ventre, "Polluino: An efficient cloud-based management of iot devices for air quality monitoring," in *2016*

IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), 2016, pp. 1–6.

[136] C.-T. Yang, S.-T. Chen, W. Den, Y.-T. Wang, and E. Kristiani, "Implementation of an intelligent indoor environmental monitoring and management system in cloud," *Future Generation Computer Systems*, vol. 96, pp. 731–749, 2019.

[137] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the internet of things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.

[138] V. Mihai, C. Dragana, G. Stamatescu, D. Popescu, and L. Ichim, "Wireless sensor network architecture based on fog computing," in *2018 5th International Conference on Control, Decision and Information Technologies (CoDIT)*, 2018, pp. 743–747.

[139] M. J. Jafari, A. A. Khajevandi, S. A. M. Najarkola, M. S. Yekaninejad, M. A. Pourhoseingholi, L. Omidi, and S. Kalantary, "Association of sick building syndrome with indoor air parameters," *Tanaffos*, vol. 14, no. 1, p. 55, 2015.

[140] F. Longo, D. Bruneo, S. Distefano, G. Merlino, and A. Puliafito, "Stack4things: a sensing-and-actuation-as-a-service framework for iot and cloud integration," *Annals of Telecommunications*, vol. 72, no. 1-2, pp. 53–70, 2017.

[141] D. Bruneo, S. Distefano, F. Longo, G. Merlino, and A. Puliafito, "Iot-cloud authorization and delegation mechanisms for ubiquitous sensing and actuation," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, Dec 2016, pp. 222–227.

[142] D. Bruneo, S. Distefano, F. Longo, G. Merlino, and A. Puliafito, "I/Ocloud: Adding an IoT dimension to Cloud infrastructures," *Computer*, vol. 51, no. 1, pp. 57–65, January 2018.

[143] Qinling Openstack community, 2019, https://docs.openstack.org/qinling/latest/.

[144] ZUN Openstack community, 2019, https://docs.openstack.org/zun/latest/.

[145] S. Distefano, G. Merlino, and A. Puliafito, "Device-centric sensing: An alternative to data-centric approaches," *IEEE Systems Journal*, vol. 11, no. 1, pp. 231–241, 2017, cited By :9. [Online]. Available: www.scopus.com

[146] A. Levin, D. Lorenz, G. Merlino, A. Panarello, A. Puliafito, and G. Tricomi, "Hierarchical load balancing as a service for federated cloud networks," *Computer Communications*, vol. 129, pp. 125 – 137, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366418303165

[147] M. Mazzara, I. Afanasyev, S. R. Sarangi, S. Distefano, V. Kumar, and M. Ahmad, "A reference architecture for smart and software-defined buildings," in *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2019, pp. 167–172.

[148] R. Bauer, R. Bless, C. Haas, M. Jung, and M. Zitterbart, "Software-based smart factory networking," *at - Automatisierungstechnik*, vol. 64, no. 9, pp. 765 – 773, 28 Sep. 2016. [Online]. Available: https://www.degruyter.com/view/journals/auto/64/9/article-p765.xml

[149] *Official site Toolsmart Project*, https://www.torinocitylab.it/it/toolsmart.

[150] D. Bruneo, S. Distefano, F. Longo, and G. Merlino, "An iot testbed for the software defined city vision: The #smartme project," in *2016 IEEE Int. Conf. on Smart Computing (SMARTCOMP)*, May 2016, pp. 1–6.

[151] N. Z. Bawanyn and A. Jawwad Shamsi, "Smart city architecture: Vision and challenges," *International Journal of Advanced Computer Science and Applications*, vol. 6, no. 11, 2015.

[152] *An IoT Testbed for the Software Defined City Vision: The #SmartMe Project*, 2016, dOI: 10.1109/SMARTCOMP.2016.7501678.

[153] *Software defined cities: A novel paradigm for smart cities through IoT clouds*, 2015, proceedings - 2015 IEEE 12th International Conference on Ubiquitous Intelligence and Computing.