Modelling and placement optimization of compound services in a converged infrastructure of cloud computing and internet of things

Elie Rachkidi

Modélisation et Optimisation du Placement de Services Composés dans une Infrastructure Convergente de l'Informatique en Nuage et de l'Internet des Objets

Thèse de doctorat de l'Université Paris-Saclay
préparée à L'Université d'Evry Val d'Essonne et Telecom SudParis

École doctorale n°580 sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Evry, le 24 Octobre 2017, par

Elie EL RACHKIDI

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisors, Prof. Nazim AGOULMINE, Prof. Djamel BELAID, and Dr. Nada CHENDEB for giving me the chance to realize my Ph.D. research, for their knowledge, enthusiasm and support over the years. Without their guidance, this work would not have been achievable.

I am truly grateful to all members of the thesis jury, Prof. Joberto MARTINS, Prof. Vania CONAN, Prof. Jean-Marc DELOSME, Prof. Djamel BELAÏD, and Dr. Nada CHENDEB for their time, encouragement and constructive feedback. Their comments are very valuable for my future works.

I am thankful for my colleagues in the LRSM team, Elhadi CHERKAOUI, Thiago MOREIRA, and Mustapha AIT-IDIR for sharing a lot of ideas and for their friendly collaboration.

I would also like to thank my cousins Wissam RACHKIDI and Diana RACHKIDI for their support throughout the thesis. Their presence makes Paris home.

I am also indebted to my parents who always believe in me and encourage me to follow my dreams.

Finally, to all my friends, so many that I could not cite their names here, thank you for being with me for all these years. I greatly value their help and friendship.

Evry, 2017

Publications

E. H. Cherkaoui, E. Rachkidi, M. Santos, P. A. L. Rego, J. Baliosian, and J. N. De, “SLA4CLOUD : Measurement and SLA Management of Heterogeneous Cloud Infrastructures Testbeds,” in *3th International Workshop on ADVANCEs in ICT Infrastructures and Services*, pp. 1–6, 2014

T. Moreira, E. Rachkidi, L. M. Gardini, and R. Braga, “An Enhanced Architecture for LARIISA : An Intelligent System for Decision Making and Service Provision for e-Health using the cloud,” in *4th International Workshop on ADVANCEs in ICT Infrastructures and Services*, 2015

E. Rachkidi, E. H. Cherkaoui, M. Ait-idir, N. Agoulmine, N. C. Taher, M. Santos, and S. Fernandes, “Towards Efficient Automatic Scaling and Adaptive cost-optimized eHealth Services in Cloud,” in *2015 IEEE Global Communications Conference: Selected Areas in Communications: E-Health (GC’ 15 - SAC - E-Health)*, pp. 1–6, IEEE, dec 2015

E. Rachkidi, E. H. Cherkaoui, M. Ait-idir, N. Agoulmine, N. C. Taher, M. Santos, and S. Fernandes, “Cooperative dynamic eHealth service placement in Mobile Cloud Computing,” in *2015 17th International Conference on E-health Networking, Application & Services (HealthCom)*, (Boston, USA), pp. 627–632, IEEE, oct 2015

E. Rachkidi, N. Agoulmine, D. Belaïd, and N. Chendeb, “Towards an Efficient Service Provisioning in Cloud of Things (CoT),” in *2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, dec 2016

T. Moreira, H. Martin, E. Rachkidi, and N. Agoulmine, “An experiment on deploying a privacy-aware sensing as a service in the Sensor-Cloud,” in *5th International Workshop on ADVANCEs in ICT Infrastructures and Services*, pp. 1–8, 2017

E. Rachkidi, N. Agoulmine, N. Chendeb, and D. Belaïd, “Resources Optimization and Efficient Distribution of Shared Virtual Sensors in Sensor-Cloud,” in *2017 IEEE International Communications Conference (ICC)*, pp. 1–6, 2017

E. Rachkidi, N. Agoulmine, J. Baliosian, and J. Bustos, “VNET : Towards End-to-End Network Cloudification,” in *5th International Workshop on ADVANCEs in ICT Infrastructures and Services*, pp. 1–5, 2017

E. Rachkidi, D. Belaïd, N. Agoulmine, and N. Chendeb, “Cloud of Things Modeling for Efficient and Coordinated Resources Provisioning,” in *25th International Conference on COOPERATIVE INFORMATION SYSTEMS*, 2017 - Accepted 25 August 2017

Contents

List of Figures

List of Tables

Acronyms

Symbols

6LoWPAN IPv6 over Low-power Wireless Personal Area Networks.

A

API Application Programming Interface.
AWS Amazon Web Services.

B

BLE Low Energy Bluetooth.

C

CAMP Cloud Application Management for Platforms.
CIaaS City Infrastructure as a Service.
CIM Common Information Model.
CIMI Cloud Infrastructure Management Interface.
CoAP Constrained Application Protocol.
CoT Cloud of Things.
CPaaS City Platform as a Service.
CPU Central Processing Unit.
CRUD Create, Read, Update, and Delete.
CSaaS City Software as a Service.

D

DMTF Distributed Management Task Force.
DNS Domain Name System.
DPWS Devices Profile for Web Services.

E

EGI European Grid Infrastructure.
EPC Electronic Product Code.
ETSI European Telecommunications Standards Institute.
EU European Union.
EXI Efficient XML Interchange.

F

FI Future Internet.
FTP File Transfer Protocol.

G

GE Generic Enabler.
GSN Global Sensor Network.

H

HTTP Hypertext Transfer Protocol.

I

IaaS Infrastructure as a Service.
ID Identifier.
IEEE Institute of Electrical and Electronics Engineers.
IIoT Industrial Internet of Things.
IoT Internet of Things.
IoT-A ARM Internet of Things Architecture Reference Model.
IoT-A Internet of Things Architecture.
IoT-O Internet of Things Ontology.
IP Internet Protocol.
IPv4 Internet Protocol version 4.
IPv6 Internet Protocol version 6.
IT Information Technology.

J

JSON JavaScript Object Notation.

L

LAN Local Area Network.
LOV Linked Open Vocabularies.
LOV4IoT Linked Open Vocabularies for Internet of Things.
LP Linear Program.
LSM Linked Stream Middleware.
LTE Long Term Evolution.
LTE-A Long Term Evolution Advanced.

M

M2M Machine to Machine.
MCC Mobile Cloud Computing.
MQTT Message Queue Telemetry Transport.

N

NB-IoT Narrow Band IoT.
NFC Near Field Communication.
NGSI Next Generation Services Interface.
NIST national institute of standards and technology.

O

O&M Observations and Measurements.
OASIS Organization for the Advancement of Structured Information Standards.
OCCI Open Cloud Computing Interface.
OData Open Data.
ODP Ontology Design Patterns.
OFC Open Fog Consortium.
OGC Open Geospatial Consortium.
OGF Open Grid Forum.
OS Operating System.
OVF Open Virtualization Format.
OWL Ontology Web Language.
OWL-DL Ontology Web Language Description Logic.

P

PaaS Platform as a Service.
PAN Personal Area Network.

Q

QoI Quality of Information.
QoS Quality of Service.

R

RAN Radio Access Network.
RDF Resource Description Framework.
RDFS Resource Description Framework Schema.
REST REpresentational State Transfer.
RFID Radio Frequency Identification.
RMI Remote Method Invocation.
RuleML Rule Markup Language.

S

S^{2}aaS Sensing as a Service.
SAaaS Sensor/Actuator as a Service.
SaaS Software as a Service.
SAN Semantic Actuator Network.
SCA Service Component Architecture.
SDD Sensor Device Definition.
SensorML Sensor Model Language.
SLI Service Layer Integration.
SN Substrate Network.
SNPS Sensor Node Plug-in System.
SOA Software Oriented Architecture.
SOaaS Smart Object as a Service.
SOAP Simple Object Access Protocol.
SOS Sensor Observations Service.
SOSA Sensor, Observation, Sample, and Actuator.
SOSA-O Sensor, Observation, Sample, and Actuator Ontology.
SPARQL SPARQL Protocol and RDF Query Language.
SQL Structured Query Language.
SSN Semantic Sensor Network.
SSN-XG Semantic Sensor Network Incubator Group.

SSO Stimulus-Sensor-Observation.
SWE Sensor Web Enablement.
SWRL Semantic Web Rule Language.

T

TaaS Things as a Service.
TOSCA Topology and Orchestration Specification for Cloud Applications.

U

uCode Ubiquitous Code.
UDP User Datagram Protocol.
UML Unified Modeling Language.
UNB Ultra Narrow-Band.
URI Uniform Resource Identifier.
URL Uniform Resource Locator.
UWB Ultra Wide-Band.

V

vCPU Virtual Central Processing Unit.
VM Virtual Machine.
VN Virtual Network.
VNE Virtual Network Embedding.
VO Virtual Object.
VOG Virtual Object Group.

W

W3C World Wide Web Consortium.
WAN Wide Area Network.
WMSN Wireless Mesh Sensor Network.
WoT Web of Things.
WS Web Service.
WS-Discovery Web Services Dynamic Discovery.
WSN Wireless Sensor Network.
WWW World Wide Web.

X

X-GSN eXtended Global Sensor Network.
XML eXtensible Markup Language.
XMPP eXtensible Messaging and Presence Protocol.

Chapter 1

Introduction

Contents

1.1 Context and Motivation

1.1.1 General Context of the Research

The Internet of Things (IoT) paradigm was first coined in 1999 by Kevin Ashton [10]. The term referred to machines enhancement with the ability to provide contextual and environmental information over the Internet. Such idea aimed to replace humans and automatically provide data to computer systems with context-aware objects[1] (i.e. Things). Unlike humans who are error prone, these connected objects have better accuracy, reliability, and working time. The IoT evolved to encompass a broader vision as defined in [11]:

> The IoT allows people and things to be connected Any-time, Any-place, with Any thing and Anyone, ideally using Any path/network and Any service.

Furthermore, the IoT integrated an extensive range of domains such as healthcare, transportation, agriculture, industry, building management, energy, logistics, and many others [12–16]. Nowadays, the IoT optimizes processes regarding cost, efficiency, performance, and effectiveness. It minimizes the operational cost for businesses and industries by reducing the man power and thriving for autonomic systems able to operate without human intervention. Moreover, the IoT increases efficiency using sensor-driven analytics and decision making reasoning for optimizing real world resources consumption such as the energy, maintenance operations, and environments monitoring. The prompt reporting and actuation in complex autonomous systems allow the IoT to increase performance while saving time and reducing costs.

Benefits mentioned above accelerated the integration of IoT solutions in domestic environments (e.g. wearables, automated homes), businesses (e.g. retails, factories), and wide-scope deployments (e.g. smart cities, environmental monitoring). The quick adoption of the IoT resulted in an unprecedented growth rate of connected objects which are expected to reach 50 billion units in 2020 based on a study by Cisco [17]. Moreover, the proliferation of IoT device manufacturers alongside the diversity of application domains produced heterogeneous connected objects with different capabilities, properties, and functions. This huge amount of heterogeneous connected objects interacting over the network has been identified as one of the major open issues related to the IoT as discussed in several recent studies [12–16]. Consequently, the IoT produces a significant amount of unstructured data generated by connected objects [18]. These data need to be managed and handled by the network between IoT devices and third party applications. As a result, IoT infrastructures need to be scalable to cope with the huge amount of heterogeneous devices communicating over the Internet.

[1] We use the terms: ”objects,” ”things,” ”connected objects,” ”IoT nodes,” and ”IoT devices” interchangeably in this thesis to give the same meaning as they are frequently used in IoT related documentation. Other terms are also employed by the research community such as ”smart objects.”

In this context, Cloud Computing [19] emerged as a promising solution for the IoT scalability challenge. Cloud Computing offers on-demand network access to a theoretically unlimited pool of configurable virtual resources such as networking, computing, and storage. These resources can be automatically provisioned, scaled, and released with a pay-per-use business model on the fly . Hence, Cloud Computing presents several essential characteristics namely: on-demand self-service, broad network access, resource pooling, rapid elasticity, measured service. Such features are compatible with the IoT requirements as they transfer the processing power from resource-constrained devices to powerful data centres. Moreover, the cloud offers virtually infinite scalability which is needed to encompass large scale IoT infrastructures and corresponding big data. In this perspective, existing cloud providers such as Amazon[2] and Google[3] developed IoT related cloud services to ease the management, development, and maintenance of IoT applications. Besides, new cloud platforms specializing solely in IoT offerings have appeared [20] such as Xively[4].

The convergence of Cloud Computing and the IoT is possible in two distinct ways [21]: bringing the cloud to connected objects or bringing IoT devices to the cloud. The first case refers to the conventional approach which consists in using the cloud to compensate for low-powered connected objects. Such approach treats IoT devices as data sources with no additional capabilities while using the cloud to collect, process, store, and visualize generated data. Separately, bringing IoT devices to the cloud leverages connected objects capabilities with some of the cloud's characteristics such as the on-demand provisioning. This integration model provides a cloud environment aware of the underlying IoT resources (i.e. sensing, actuating) and able to offer them on-demand alongside cloud resources (i.e. compute, network, storage). This latter model is sometimes referred to as the Cloud of Things (CoT) [22].

1.1.2 Motivation of the Thesis

In a CoT context, things are abstracted and offered as cloud services accessible over the Internet from any place and at any time. Such software representations of connected objects in the cloud are known as Virtual Objects (VOs). They promote flexible on-demand provisioning of IoT resources. Indeed, VOs are only deployed when their corresponding connected objects are used. Furthermore, their allocated cloud resources can auto-scale as needed to cope with end-users demand. These VOs interconnect with each other or with traditional cloud services (e.g. data analytics, storage service, visualization dashboard) to deliver IoT applications. Consequently, the CoT environment provides means to perform end-to-end IoT applications provisioning, deployment, auto-scaling, and release on the fly with minimal management efforts. Hence, the CoT is a step closer towards realizing the IoT vision. Figure 1.1 illustrates the process of deploying an IoT application in a CoT infrastructure. A CoT customer specifies an IoT application request which is consumed by end-

[2]https://aws.amazon.com/iot/
[3]https://cloud.google.com/solutions/iot/
[4]https://www.xively.com/

users. We refer to this request as a CoT request. We also might refer to a CoT infrastructure as a CoT substrate throughout this thesis.

In this thesis, we aim to provide a means for CoT customers to describe their requests and CoT providers to represent their infrastructure. Furthermore, we provide a solution for CoT providers to orchestrate an incoming request. Let's consider a scenario where a CoT customer requests a weather forecasting application from a CoT provider to serve end-users. The weather forecasting application (see Figure 1.1) is composed by multiple interconnected atomic services. It contains two data sources (e.g. temperature sensors), two VOs (i.e. data collection services), and three distinct cloud services. These cloud services represent a storage service (to store collected temperature data), a data analytics service (to calculate the weather forecasting), and a visualization service (to provide a dashboard for end-users). The solutions proposed in this thesis aim to help CoT providers to represent and deploy such a CoT request in a CoT infrastructure.

Several challenges should be addressed to realize the described scenario. These challenges are the following:

- How to efficiently deliver selected IoT resources through VOs to deployed cloud services (i.e. cloud applications) while reducing the operational cost and maintaining minimal data transmission latency?
- How to efficiently provision and orchestrate the entire CoT request in a single-stage (i.e. provision IoT and cloud resources at the same time) while minimizing the operational cost, considering end-users demands, and respect Quality of Service (QoS) terms?

The first question addresses the optimization of cloud resources allocated for VOs as well as their placement across cloud data centres. The orchestration of VOs should account for the placement of cloud applications across data centres and the geographical location of corresponding connected objects. On the one hand, data streams produced by connected objects should be routed via VOs to cloud applications through least costly network paths with available bandwidth and low data transmission latency. On the other hand, VOs should be deployed in low cost cloud data centres to minimize computing and networking costs. Hence, a provisioning mechanism is needed to determine the optimal placement of VOs across data centres. Such optimal placement aims to minimize the operational cost for hosting VOs, the data transmission cost through selected hosts, and the communication latency over corresponding network links. Therefore, the first problem is, more specifically, how to optimally distribute VOs over different cloud data centres to minimize cloud resources cost (i.e. compute, storage, network) and communication latency while respecting required QoS terms.

This approach is in line with the current trend of major cloud providers such as Amazon and Google. They provide VOs for abstracting connected objects in the cloud. These VOs are internally managed and spanned across different geographically distributed data centres. Furthermore, these VOs can be connected to available services offered by cloud providers' catalogues. For example, the Amazon

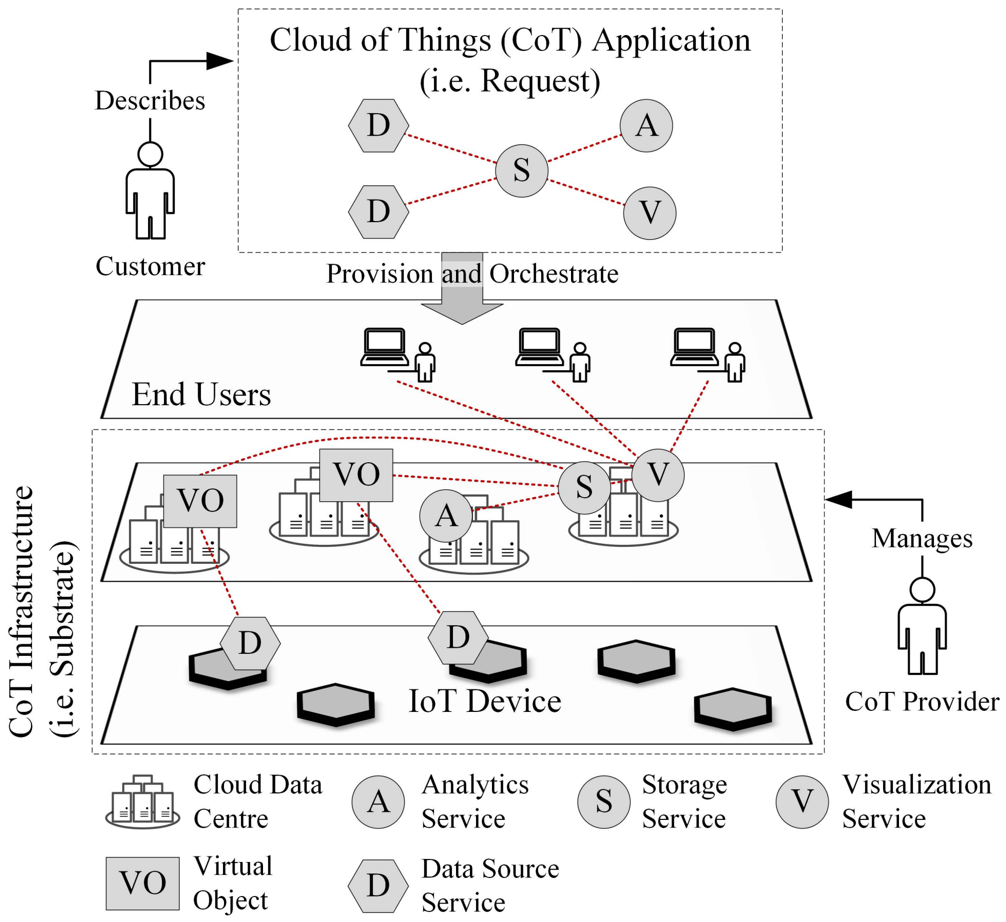


Figure 1.1: Problem Illustration.

Web Services (AWS) instantiates "device shadows" (i.e. VOs) to connect customers' IoT devices. It also automates the connection of these device shadows to available complementary services such as the Amazon storage services. Thus, there is a need to optimize the provisioning of these VOs to increase profits.

The second problem addressed in this thesis is related to a holistic view of the IoT application request and the CoT infrastructure. In fact, a CoT request provisioning consists of several steps: (1) the selection of connected objects, (2) the placement of cloud services, and (3) the orchestration of VOs linking connected objects and cloud services. These provisioning steps can be performed either without coordination, in a multiple-stage coordination, or a single-stage coordination. An uncoordinated provisioning implies that each step is performed independently. A multiple-stage coordination means the steps are executed separately but the relation between different steps is considered. For example, the orchestration of VOs is executed while considering the geographical location of selected connected objects. A single-stage coordination suggests that all steps are performed simultaneously while taking into account the effect each step has on the other. For example, the selection of connected objects and the orchestration of VOs are done at the same time. In this context, the execution of these provisioning steps in uncoordinated or coordinated multiple-stage is inefficient as different placement decisions are based on a partial view of the infrastructure [23]. Such an approach reduces the provisioning efficiency compared to a coordinated single-stage provisioning [23]. Therefore, a provisioning mechanism should map in a single-stage a given CoT request onto the CoT substrate. However, such a provisioning mechanism requires a novel resource-oriented model which provides a means to describe a requested IoT application. Furthermore, the model should also describe the CoT substrate on which the CoT request will be deployed.

These problems are the research challenges addressed in this thesis, and their corresponding solutions constitute the main achieved contributions. These contributions are briefly presented in the following section.

1.2 Contributions

Concerning the challenges identified in the previous section, the first contribution of this thesis consists in optimizing the provisioning of VOs in a cloud infrastructure. VOs hide the heterogeneity of underlying IoT resources and connect them to cloud applications via standardized Application Programming Interfaces (APIs). Therefore, it becomes crucial to optimize the placement of VOs across cloud data centres to minimize their operational cost and the network latency between connected objects and cloud applications. There exist different possible strategies when employing VOs to deliver IoT resources. In fact, each connected object can be linked to one or multiple VOs. Furthermore, each VO can serve one or several applications.

In our approach, we consider that each connected object is associated with a single VO that can be shared among multiple applications. The sharing strategy aims to minimize the number of connected objects that are needed to satisfy applications' requirements. Therefore, less VOs are required to be deployed which reduces the operational cost. However, a shared strategy increases the model complexity since the placement decision for each VO becomes dependent on multiple applications QoS requirements. We formulate this problem as a Linear Program (LP) with an objective function that aims to minimize VOs' operational cost alongside the data transmission latency between connected objects and cloud applications. This LP outputs the optimal placement of VOs in an infrastructure with no previously deployed VOs or cloud applications. We refer to this model as the static model. Once VOs are deployed in the infrastructure, they can be reused for subsequent cloud applications. In this case, the provisioning process should remap reused VOs based on the new cloud applications requirements and configuration. In this perspective, we introduce a second LP. Its objective function adds the migration cost in addition to the costs considered in the previous LP. This model aims to orchestrate a set of VOs which contains previously deployed and newly requested VOs. We refer to the latter model as the dynamic model.

The second contribution of this thesis focuses on defining a resource-oriented model able to describe a CoT request and substrate. A CoT request corresponds to a requested IoT application specified by a CoT customer as illustrated in Figure 1.1.

The CoT substrate corresponds to the interconnected entities on which a CoT request can be mapped. CoT entities correspond to cloud data centres, connected objects, gateways, and many others. Moreover, CoT requests and substrates can be specified on the infrastructure level or the platform level. On the infrastructure level, only hardware level resources are described such as compute, network, storage, sensors, and actuators. On the platform level, software components specifications are described alongside hardware level information. We identify the requirements of such requests and substrates to define a formalism for describing them at the infrastructure and platform levels. We focus on the orchestration aspects in our model. However, the described model can be extended to perform deployment operations in the CoT environment. We base our model on the Open Cloud Computing Interface (OCCI) specifications defined by the Open Grid Forum (OGF). We adopt the OCCI because it is simple, open, and expandable. We adapt and extend the OCCI infrastructure [24] and platform [25, 26] models previously defined for the cloud to encompass the CoT environment. Since we focus on provisioning aspects, we propose a graph-based model to represent the CoT requests and substrates. However, an associated mechanism to perform the mapping between both CoT graphs is needed which leads to the last contribution in this thesis.

Finally, the third contribution addresses the coordinated single-stage provisioning problem of a CoT request onto a CoT substrate. An IoT application request description includes end-users expected demands, requested cloud services (e.g. analytics, storage, visualization), needed connected objects, and required VOs for connecting needed IoT devices to requested cloud services. Nowadays, the provisioning of these components has been done separately, with or without coordination. For example, provisioning VOs while considering that connected objects are already selected and cloud services are previously deployed is identical to the first contribution. Consequently, the provisioning process of a given CoT request is not able to optimize efficiently all resources simultaneously which degrade the QoS and increases the operational cost. Therefore, we provide a global analytical model which provides a holistic view of a CoT substrate. We derive a LP able to orchestrate the entire CoT request at the same time. It also takes into consideration end-users demands and QoS requirements (i.e. latency).

1.3 Thesis Structure

The thesis is structured as follows:

Chapter 1 introduces the context and the motivation of the research. This chapter identifies the objectives of the thesis and presents the main contributions briefly.

Chapter 2 presents the state of the art on the integration of Cloud Computing and the IoT. It provides an overview of the basic elements forming the CoT. For this matter, existing integration strategies are also discussed. They highlight the different approaches used to integrate Cloud Computing and the IoT. Then, the main challenges to achieving a seamless convergence are discussed.

Chapter 3 addresses the placement optimization of VOs within a cloud infrastructure to deliver IoT resources for cloud applications. Firstly, related works are discussed to position our work which emphasizes VOs sharing. Then, challenges related to the placement of shared VOs in the cloud are presented. We propose two LPs to orchestrate VOs across cloud data centres. The problem is expressed in function of the average data traffic between connected objects, VOs, and cloud applications. Both LPs objective functions aim to minimize the operational costs as well as the data delivery latency. However, one model deals with a CoT environment without previously deployed VOs. The other considers that VOs are partially deployed.

Chapter 4 is dedicated to model the resources of a CoT environment. It discusses existing standards and resource-oriented models specified in the literature. The purpose is to identify a suitable set of specifications for modelling the CoT requests and substrates. We select the OCCI standard due to its flexibility and comprehensive description of the Cloud Computing service models. In this thesis, we propose to extend and adapt the OCCI standard to encompass the CoT infrastructure and platform resource models. Finally, several scenarios are provided to show how the proposed resource-oriented model enables the representation of a CoT request and substrate, as well as the execution of a single-stage mapping.

Chapter 5 deals with an end-to-end IoT application provisioning in converged Cloud Computing and IoT environments. It deals with the provisioning of a CoT request graph onto a CoT substrate graph in a single-stage. Two LPs are devised for CoT infrastructure and platform service levels mapping. In addition, we demonstrate by simulations the advantage of a coordinated single-stage provisioning process compared to a multiple-stage provisioning process.

Finally, Chapter 6 concludes this thesis. It synthesizes the overall contributions and highlights some perspectives for this research.

Chapter 2

State of the Art: Convergence of Cloud Computing and Internet of Things

Contents

2.1 Introduction

The Internet of Things (IoT) is a concept which evolved over the years and is enabled by a growing set of key technologies. Nowadays, the IoT envisions interconnecting every thing and person via the Internet. For example, IoT applications such as smart cities tend to attach sensors and actuators to every object in a city to facilitate our everyday lives and optimize the city's management processes such as transportation, garbage collection, traffic distribution, etc. The realization of such wide scope IoT applications called for the adoption of technologies including communication, computing, machine learning, data mining, and many others. Actually, large scale IoT applications promote pervasive computing in any thing, which generates big data that need to be stored and processed [18]. However, IoT devices are constrained objects unfit for dealing with the large amount of produced data.

In this context, Cloud Computing has emerged as a suitable technology for overcoming the technological intrinsic limitations of the IoT. It provides virtually unlimited computing, storage and networking resources with highly resilient energy supply which are required by IoT applications. Furthermore, such convergence enhances Cloud Computing service catalogue with IoT application offerings. As a result, Cloud Computing becomes able to provision IoT resources alongside its computing, networking, and storage resources.

On this basis, the integration of Cloud Computing and the IoT was inevitable [27] and resulted in the Cloud of Things (CoT) [28]. This integration has been also referred to as Sensor Cloud [29] or CloudIoT [27] in the literature. Moreover, the distinct visions of the IoT and Cloud Computing service models resulted in many possible integration strategies to realize the CoT.

In addition, different applications dictate diverse characteristics (i.e. mobility, geo-distribution) and requirements (i.e. low latency) which cannot be satisfied only by the cloud. Moreover, large scale IoT applications produce a large amount of data. Routing all these data for processing and storage at cloud data centres implies high bandwidth usage. Consequently, Fog Computing [30] was introduced as an intermediary layer between Cloud Computing and the IoT. Fog Computing allows parts of the application to execute closer to the network edge to reduce latency and bandwidth usage. In particular, Fog Computing is used for large-scale, geographically distributed, and latency sensitive applications.

This chapter aims to present the state of the art on the integration of Cloud Computing and the IoT. It is structured in 4 sections. Section 2.2 presents the basic concepts and key elements forming the CoT. We introduce the IoT enabling technologies, Cloud Computing, and Fog Computing. Section 2.3 presents the different possible integrations of Cloud Computing and the IoT. We highlight the resource allocation problem related to each integration model. In section 2.4, we present and discuss open research issues and challenges with respect to the integration of both paradigms. Finally, in section 2.5, we conclude this chapter.

2.2 Background and Basic Concepts

As previously mentioned, the CoT includes multiple main components, namely: the IoT, Cloud Computing, and Fog Computing. Moreover, the IoT is enabled by multiple key technologies. This section introduces briefly each of these key elements to clarify the context of this thesis.

2.2.1 Internet of Things

The evolution of embedded devices, communication technologies, and Internet protocols, eased the enhancement of physical objects with sensing, actuating, processing, and communication capabilities [12]. Hence, transforming dumb things into connected objects able to send and receive data over the Internet, sense their environment, and perform actions based on shared information. These connected objects collaborate with each other and with services over the Internet to deliver what we call today the IoT. This paradigm promotes IoT applications which rely on sensory data streams, actuators actions, and services (i.e. data analytics) to provide value-added information and functionalities for end-users and service providers. These IoT applications improve many real-world domains such as healthcare [31], ambient assisted living [32], smart cities [33], and many others. Nowadays, for example, instead of hiring nurses to watch seniors, connected objects and advanced analytics are used to monitor them in real-time and trigger alerts such as calling an ambulance in case of health problems [34]. Likewise, connected objects are used in agriculture to monitor climate, soil, and crops to optimize the cultivation process and detect anomalies without relying on human resources [35]. Several elements are needed however to deliver the functionality of the IoT [16] as illustrated in Figure 2.1, namely: identification/addressing, communication, computation, sensing/actuating, semantics, services.

Identification and Addressing

Connecting IoT devices to applications requires identifying the requested objects and configuring the network connections between them. Several methods assign Identifiers (IDs) to IoT devices to ensure they are uniquely identifiable. These identification methods provide universal hardware IDs such as the Electronic Product Code (EPC) and Ubiquitous Code (uCode) [36]. Alongside their object IDs, IoT devices need network addresses to be accessible over the Internet. Addressing methods applied to connected objects are the Internet Protocol version 4 (IPv4) and the Internet Protocol version 6 (IPv6). However, IPv6 is better adapted for the IoT due to its ability to encompass the large anticipated number [17] of connected objects [27, 37]. Moreover, the IPv6 over Low-power Wireless Personal Area Networks (6LoWPAN) mechanism [38, 39] provides a compression of IPv6 headers between the Internet and low power wireless networks to cope with resource-constrained devices. It reduces the overhead of IPv6 in resource-limited environments, thus enabling a seamless communication and integration of IoT devices to the Internet.

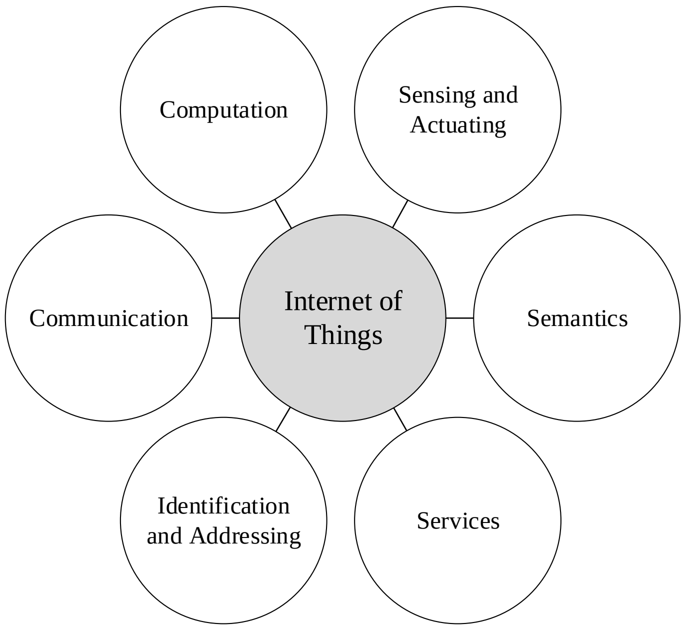


Figure 2.1: Internet of Things Essential Building Blocks.

Communication

The integration of connected objects in multiple domains involves different environments, hence different needs. Consequently, the IoT relies on several communication technologies to cope with applications' bandwidth and range requirements. Most of these technologies adopt wireless integration of the IoT due to its flexibility. Some of the communication technologies used for IoT are the following: RFID, Near Field Communication (NFC), Ultra Wide-Band (UWB), WiFi (i.e. IEEE 802.11 standards), Bluetooth, IEEE 802.15.4, Z-Wave, and Long Term Evolution (LTE).

Radio Frequency Identification (RFID) tags were the first enablers of the IoT. They emit universal IDs which allow readers to identify objects and rely on existing databases to retrieve additional information. These tags operate within 200 meters and can be passive, active, or semi-passive/active [40]. The NFC protocol has a smaller range (i.e. up to 10 cm) and permits a 424 Kbps transmission rate [41]. Another Personal Area Network (PAN) technology is the UWB. It is suitable for short range, low power, and high bandwidth transmissions [42]. Similarly, the IEEE 802.15.4 standard targets low-power PANs. It is used alongside the 6LoWPAN to enable IPv6 over low-powered wireless networks.

IEEE 802.11[1] standards are more suited to Local Area Network (LAN) such as home, healthcare, and industrial environments. These connected objects reach the Internet via access points. The IEEE 802.11ah[2] [43, 44] was introduced specifically

[1]http://standards.ieee.org/findstds/standard/802.11-2016.html

[2]https://standards.ieee.org/findstds/standard/802.11ah-2016.html

for the IoT to cope with devices requiring power efficient communications with up to 1 km range coverage and a minimum data rate of 100 Kbps.

Bluetooth, on the other hand, is used to exchange data between devices over short distances. Recently, the Low Energy Bluetooth (BLE) was introduced to provide more efficient power consumption while maintaining the same communication range and transmission rate. Furthermore, the BLE mesh[3] profile and model specifications were released in 2017 to enable many to many communication. The Bluetooth mesh supports sensor networks.

For long range communication, technologies such as LTE and LTE Advanced are used for high-speed data transfer, while others such as SigFox[4], LoRa[5] [45], and Narrow Band IoT (NB-IoT) [46] are adequate for low-rates and energy efficient data transfer. SigFox and LoRa are proprietary standards.

Computation

Connected objects are the key components of the emerging IoT. Traditionally, organizations needed to own, configure, and deploy sensors/actuators. Furthermore, they had to spend additional resources maintaining these connected objects. Therefore, connected objects were application specific. However, research efforts aim to decouple applications and underlying sensors/actuators networks to realize the IoT vision. Such efforts relied on sensing and actuating virtualization to share them among multiple applications. We notice several approaches in the literature to address connected objects virtualization: (1) node level virtualization [47], (2) network level virtualization [47], and (3) objects virtualization [48–50].

Node level virtualization consists in executing on the connected object, sequentially (i.e. event driven programming model) or simultaneously (i.e. thread-based programming model), several tasks. Each task serves a particular application [51]. Event driven solutions (i.e. SenSmart [52]) consist of executing tasks when an event occurs such as the temperature exceeding a given threshold. Event driven Operating Systems (OSs) have a simpler implementation. However, tasks should wait in a queue until previously triggered tasks finish. Thread-based solutions (i.e. RIOT[6] OS) execute threaded tasks in a time slicing fashion, hence, different tasks do not block each other. The disadvantage of thread-based OS is their complexity.

Network level virtualization enables several applications to share connected objects by dividing them into logical networks. Each application is assigned a logical network based on its needs. Therefore, formed logical networks contain different amounts and types of connected objects. Furthermore, logical networks dynamically change with time as applications needs change. They can be composed of connected objects belonging to different physical networks. Moreover, connected objects in the same physical network can be assigned to different virtual networks. However, a

[3]https://www.bluetooth.com/specifications/mesh-specifications
[4]https://www.sigfox.com/en
[5]https://www.lora-alliance.org/technology
[6]https://riot-os.org/

connected object belongs solely to one virtual network.

Objects virtualization consists in abstracting connected objects via Virtual Objects (VOs) which are wrappers encapsulating sensors or actuators to provide their functionalities to multiple applications. Such abstraction helps to mirror connected objects while providing additional resources for managing their non-functional aspects (e.g. availability, reliability). Also, wrappers provide unified Application Programming Interfaces (APIs) for applications and developers to ease the interaction with sensors and actuators. Hence, VOs hide the heterogeneity of connected objects. Generally, middleware solutions [53] manage VOs and provide necessary functionalities for applications to interact with the underlying IoT infrastructure.

Sensing/Actuating

Sensors and actuators constitute primary IoT resources. Sensors translate world phenomena into digitized information, while actuators transform logical states to actions in the physical world. Together, they allow information systems to gather knowledge about things and affect the physical world. Nowadays, the low cost of sensing and actuating technologies made it possible to integrate sensors and actuators in everyday objects, industrial machines, health devices, etc.

Semantics

The World Wide Web Consortium (W3C) thrives toward a standardized representation of knowledge on the web. It defines semantics for various domains and relationships between different concepts. Such technology enables the semantic interoperability between different systems. Furthermore, semantic web technology enables discovering, querying, and reasoning on top of available information. Therefore, the semantic web is seen as an enabler of the IoT. A semantic representation of heterogeneous connected objects and corresponding data facilitates the integration of IoT resources with domain oriented applications (e.g. healthcare, agriculture, transportation). Also, it enables data streams retrieval based on applications requirements. The W3C introduces the Resource Description Framework (RDF) and the Ontology Web Language (OWL) specifications which can be used to model concepts and their relationships.

Services

Sensors and actuators provide means to interact with the real world. However, collecting data or producing actions requires services to deliver needed IoT resources, aggregate data streams, take decisions, etc. IoT services are categorized in [54] as follows: identity-related, information aggregation, collaborative aware, and ubiquitous. Identity-related services focus on delivering the appropriate IoT resources based on application requests. These services are essential to identify existing sensors/actuators types, operational region, and properties to expose their functionalities for applications. VOs fall under this category of services. Information aggregation services summarize collected raw sensory data. Collaborative aware services analyse received data to provide insights, alerts, notifications, and decisions which can be presented to the user or transmitted directly to existing actuators. Ubiquitous services represent the previously described services when offered any-time and anywhere for end-users and applications.

2.2.2 Cloud Computing

Cloud Computing evolved as the future generation computing paradigm. The National Institute of Standards and Technology (NIST) presents Cloud Computing building blocks [19] as illustrated in Figure 2.2. Cloud Computing offers pools of compute, network, and storage resources which can be accessed from anywhere on-demand. Cloud Computing offers different service models, (1) Infrastructure as a Service (IaaS), (2) Platform as a Service (PaaS), and (3) Software as a Service (SaaS). Each service model defines the scope of control of the cloud provider and the cloud customer over the provisioned resources. For an IaaS service model, the cloud provider offers physical resources (processing, storage, and network), and the cloud customer can run over it arbitrary operating systems and applications. A PaaS service model provides the cloud customer an application hosting environment which is configurable, but the cloud customer does not control the underlying infrastructure and operating system. Finally, the SaaS offers cloud customers an application running in the cloud with limited configuration settings such as Dropbox. Cloud Computing does not bind clients to a particular service model, enabling a flexible environment for all IT needs. However, the benefits of the Cloud go beyond its service models. In fact, the cloud's essential characteristics render three major trends in Information Technology (IT): (1) agility, (2) elasticity, and (3) autonomous deployment.

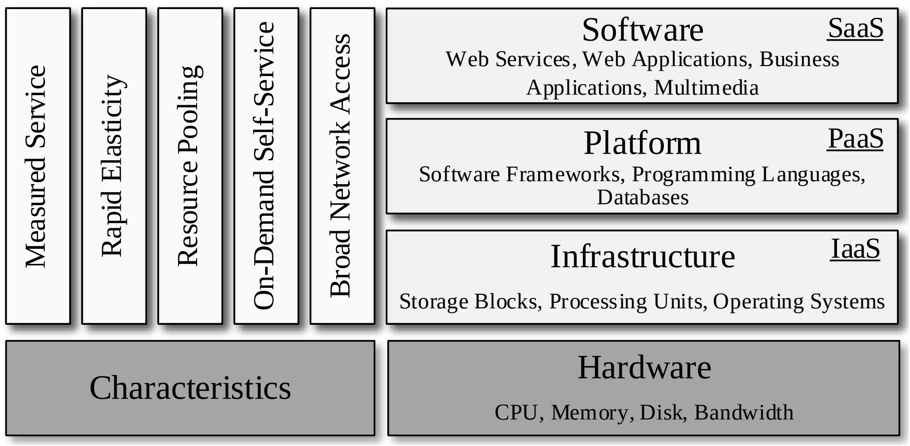

Measured Service
Rapid Elasticity
Resource Pooling
On-Demand Self-Service
Broad Network Access
Software
SaaS
Web Services, Web Applications, Business Applications, Multimedia
Platform
PaaS
Software Frameworks, Programming Languages, Databases
Infrastructure
IaaS
Storage Blocks, Processing Units, Operating Systems
Characteristics
Hardware
CPU, Memory, Disk, Bandwidth

Figure 2.2: Traditional Cloud Computing Service Layers and Key Characteristics.

Cloud Computing offers on-demand self-service for cloud customers, allowing them to provision resources (i.e. networks, servers, storage, applications, and services) without human intervention and to only pay for what they consume. Cloud users could also optimize their cost using cloud's elasticity to provision new resources on the fly or quickly release some reserved resources to keep the exact amount needed for satisfying their demand. Moreover, the capacity to measure the Quality of Service (QoS) of provisioned resources helps cloud providers and users monitor resources behaviour and state [55]. These characteristics encouraged the adoption of Cloud Computing and made possible the development of mechanisms for auto-scaling operations to dynamically optimize resources as the load varies. Furthermore, errors and faults (e.g. virtualization problems, work flow disruption) can be reported so appropriate actions can be enforced automatically. As a result, the cloud enables end-to-end autonomous service composition and delivery on demand, along with dynamic optimization of allocated resources. Such optimization maintains the QoS with minimal allocated resources. Consequently, the cloud minimizes the cloud customer cost, maximizes the cloud provider's profit, and optimizes energy consumption.

The capabilities of the cloud accelerated the adoption of this computing model by many businesses seeking to reduce their capital expenditure by moving their infrastructure to the cloud. Moreover, the "pay-as-you-go" business model of the cloud reduces the infrastructure cost. Businesses do not have to worry any more about maintenance, scalability issues, and hiring specialized staff to manage and deploy IT systems and software, thus reducing their operating expenditure. Furthermore, the wide range of services provided by service providers in the cloud creates an attractive marketplace for existing and emerging businesses. In fact, using tuned services with auto deployment and scaling mechanisms eases the development of IT solutions and allows businesses to focus on their main product without worrying about the back-end. For example, with Google App Engine Datastore[7] (No SQL based storage) and CloudSQL[8] (SQL based storage) businesses can deploy needed storage instantly without worrying about scalability, reliability, and disaster recovery.

2.2.3 Fog Computing

Cisco first introduced Fog Computing in 2011 [17, 56]. It was further developed and defined by the Open Fog Consortium (OFC). Fog Computing falls under the wider definition of Edge Computing which stands for pushing applications and services, completely or partially, to the network edge. As a result, some functions (e.g. processing, temporary storage, data aggregation) become closer to end-users and connected objects which improves applications and services response time [57]. Although Cloud and Fog Computing paradigms have virtualization as common ground, their characteristics are different. The cloud infrastructure is composed of large data centres with virtually unlimited capacity distributed in several countries or regions of a country. The cloud is a centric solution for service providers [56]. However, Fog Computing is characterized by highly distributed and location aware virtualized nodes with limited capacity.

[7]https://cloud.google.com/datastore/docs/
[8]https://cloud.google.com/sql/docs/

Another concept, cloudlets [58, 59], coincides with Fog Computing. A Cloudlet is a resource-rich computer like ”cloud in a box,” which is available for use by nearby mobile devices [30]. However, Fog Computing includes various types of nodes, differently to servers in the cloud or cloudlets. These nodes can be small-sized servers [60], gateways [28], routers [61, 62], and resource rich machines [58]. Hence, Fog nodes capabilities depend on their type and capacity, which can affect the nature and size of services these nodes can host. For example, authors in [57] consider all virtualized nodes between the cloud and connected objects as the Fog, while authors in [28] consider the Fog as the set of smart gateways at the network edge.

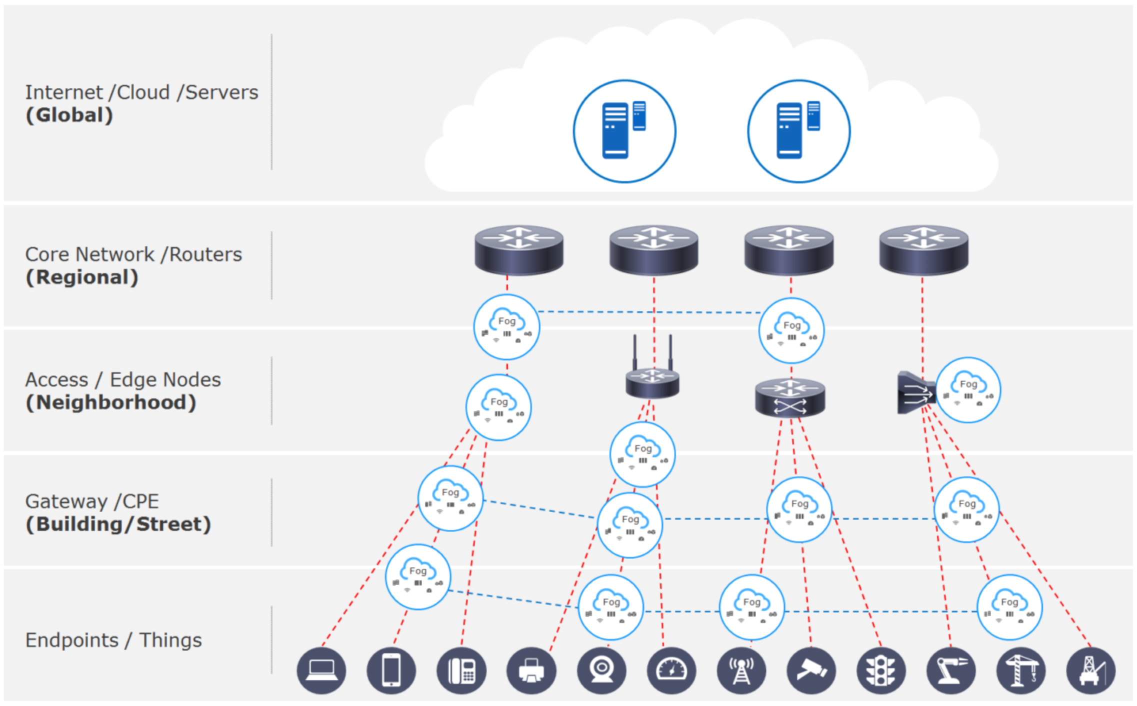


Figure 2.3: Fog Computing Conceptual Architecture (Source [63]).

Figure 2.3 shows the integration of Fog Computing between the cloud and end devices. The Fog complements the cloud and provides several benefits. Firstly, it extends services deployment to the edge of the network, enabling higher QoS for applications with low latency requirements such as video streaming, augmented reality, and gaming [61, 64]. Secondly. Fog Computing increases resources efficiency and QoS for widely distributed and large scale applications such as environment monitoring, and for applications introducing connected objects with high mobility such as vehicles. For example, it can decrease the traffic load on cloud applications by aggregating and processing data at the edge. Many uses cases are presented for Fog Computing in [63] such as IoT applications, mobile network acceleration, and content delivery networks.

2.3 Integrating Cloud Computing and the Internet of Things

Previous works have presented the benefits of converging Cloud Computing and the IoT. Botta et al. and Diaz et al. in [27, 53] show the importance of integrating these two domains. In fact, both technologies have complementary characteristics as represented in Table 2.1. Authors also presented the different drivers for Cloud Computing and the IoT integration such as Big Data and seamless IoT applications execution. More precisely, in [27], authors introduced novel applications resulting from the CoT paradigm and presented the state of the art of some research projects in this area. Diaz et al. [53] showed the different academic and industrial solutions enabling such integration with several case studies. They presented existing solutions which can be combined to deliver the CoT. In this perspective, Diaz et al. surveyed big data solutions such as Hadoop[9] and Apache Spark[10], Cloud Computing platforms such as OpenNebula [65], and middlewares for the IoT such as the Global Sensor Network (GSN).

Table 2.1: Complementary aspects of Cloud Computing and the Internet of Things (Source [27])

	Internet of Things	**Cloud Computing**
Displacement	pervasive	centralized
Reachability	limited	ubiquitous
Components	real world things	virtual resources
Computational Capabilities	limited	virtually unlimited
Storage	limited or none	virtually unlimited
Role of the Internet	point of convergence	means for delivering services
Big Data	source	means to manage

The broad definition of the IoT and the various service models of Cloud Computing made such integration possible using different approaches. As previously mentioned, there are two main manners to integrate both domains: bringing the cloud to connected objects or bringing IoT devices to the cloud. When Cloud Computing is used to shift the processing power from connected objects to powerful data centres, IoT devices become simple data sources. Such integration does not modify the respective functionalities of both domains. The IoT provides sensory data while the cloud provide services to process and store these data. We refer to this method as the loose integration. However, adding IoT devices to the cloud can be done by enhancing connected objects with cloud characteristics. Such convergence considers the IoT as part of the service models provided by the cloud. It can happen at the application level, the platform level, or the infrastructure level. Some works in the

[9]http://hadoop.apache.org/
[10]https://spark.apache.org/

literature focused on a single layer integration while others consider multiple-layer integration resulting respectively in partial or full integration strategies. In the following, we discuss in details the different levels of integration.

2.3.1 Loose Integration

A loose integration of Cloud Computing and IoT consists in a set of solutions that use both technologies without introducing a novel service model. Hence, preserving the traditional purposes of both domains. The IoT provides connected sensors and actuators, while the cloud offers compute and storage resources to host an application managing the latter IoT infrastructure. Moreover, the IoT application might expose sensors and actuators functionalities via APIs for developers. In the latter case, the cloud acts as the intermediary layer between the IoT infrastructure and domain specific applications. The IoT application deployed in cloud data centres benefits from the cloud characteristics such as the rapid elasticity and the ”pay as you go” business model. For example, allocated resources for the IoT application might scale up or down based on the applications usage [34]. However, the cloud platform remains unaware of IoT resources and rely solely on compute, storage, and network usage in cloud data centres for the decision making. Furthermore, the cloud characteristics are not transferred to the IoT. Therefore, IoT resources are not offered on-demand and their usage cannot be optimized. For example, the IoT application uses cloud resources for storing, analysing, and visualizing collected sensory data at all times without considering users actual needs which lead to inefficient use of IoT and cloud resources [66–70].

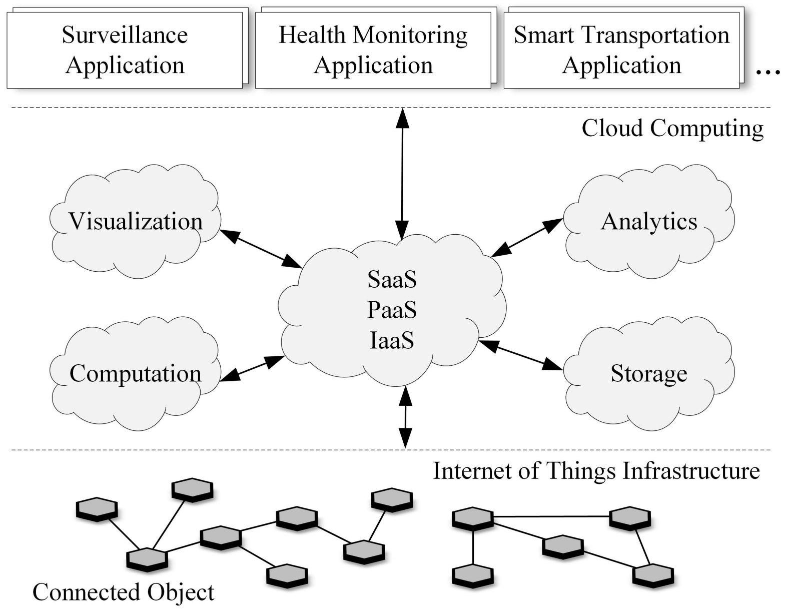


Figure 2.4: Conceptual IoT Framework with Cloud Computing at the Centre (Source [34]).

Several works follow a loose integration in their approach. For instance, authors in [32, 34] define a cloud application for managing and collecting sensory data. The work in [34] considers a general context, while [32] focuses on ambient assisted living. However, both studies rely on storing sensors data and provide them via APIs or a web interface to developers or end-users respectively. Furthermore, existing IoT platforms realize the loose integration of cloud and IoT such as Xively[11] [20]. This kind of platforms provides means to connect IoT devices, store their data, and expose them via APIs over the Internet. Some of these platforms offer also analytics services to process collected data and visualization tools to plot stored streams. Figure 2.4 illustrates the cloud and the IoT roles in a loose integration approach. This method is out of the scope of this thesis as it does not relate to the CoT vision.

2.3.2 Partial Integration

A partial integration consists in introducing a novel service model within Cloud Computing which delivers IoT resources [21, 27]. Such approach extends the cloud reach to the physical world. Therefore, cloud offerings become wider and include IoT resources. Furthermore, Cloud Computing characteristics are also passed on to IoT devices and resources. Hence, the pay as you go business model and the on-demand provisioning of resources become applicable on the IoT. This partial integration of Cloud Computing and the IoT is possible through: (1) a data-centric approach [50, 71], (2) a device-centric approach [50, 72, 73], or (3) a hybrid approach [50]. A data-centric approach relies on gathering and storing connected objects generated data which are shared among multiple applications. In this case, connected objects are used as data sources. Hence, end-users do not have control over the underlying IoT infrastructure nor the storage units. A data-centric approach provides additional service models such as the Sensing as a Service (S^2aaS) [74–77] for end-users. It is considered a PaaS [50] since it forbids access to IoT devices configurations.

Separately, a device-centric approach focuses on delivering Sensor/Actuator as a Service (SAaaS) [50, 72, 78–80]. It is also referred to as Smart Object as a Service (SOaaS) [81] or Things as a Service (TaaS) [82, 83]. In this case, a set of connected objects satisfying requested requirements are selected. Then, appropriate services such as VOs will be deployed to abstract the functionalities of selected connected objects via standardized APIs. Hence, the end-user gains control over IoT devices configurations such as data transmission rate. A device-centric approach allows end-users to provision IoT devices. A hybrid approach consists of a combination of the data-centric and device-centric approaches. It realizes the IaaS and PaaS service models for the IoT and enables the allocation of connected objects as well as sensory data streams. Figure 2.5 illustrates the difference between the data-centric (a) and the device-centric (b) approaches.

Despite the selected approach to achieve a partial integration of Cloud Computing and IoT infrastructures, orchestration mechanisms are required to optimize resource utilization. In contrast to the loose integration, a partial convergence provides an additional service model in the cloud responsible for delivering needed IoT

[11]https://www.xively.com/

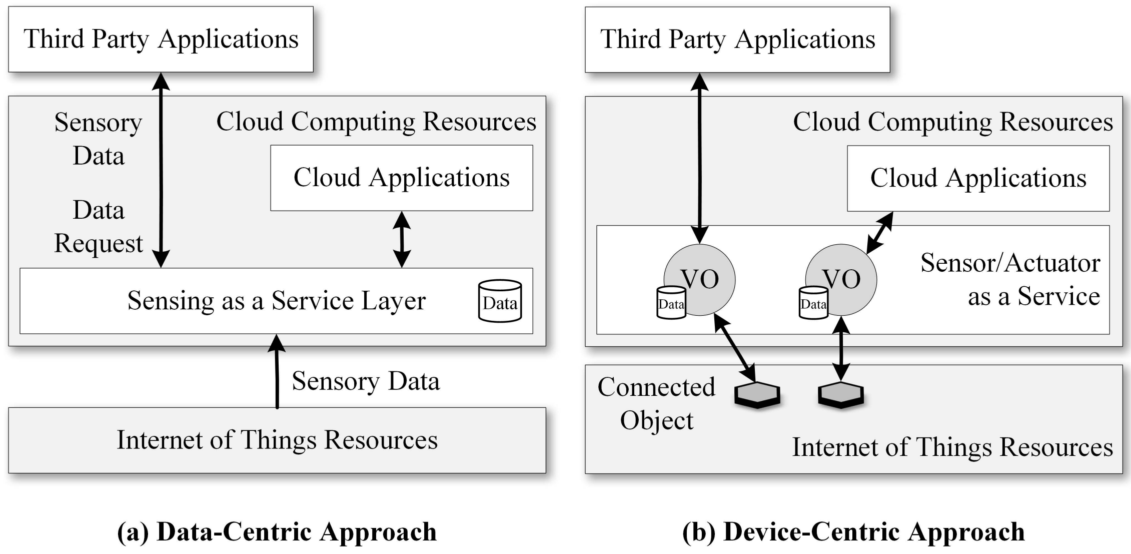

Third Party Applications
Sensory Data
Data Request
Cloud Computing Resources
Cloud Applications
Sensing as a Service Layer
Data
Sensory Data
Internet of Things Resources
(a) Data-Centric Approach
Third Party Applications
Cloud Computing Resources
Cloud Applications
VO
Data
VO
Data
Sensor/Actuator as a Service
Connected Object
Internet of Things Resources
(b) Device-Centric Approach

Figure 2.5: Difference between (a) the Data-Centric, and (b) the Device-Centric Approaches.

resources to upper applications as seen in Figure 2.5. Therefore, connected objects selection techniques are necessary to pick appropriate IoT resources for applications efficiently. Furthermore, cloud provisioning processes are crucial to optimize allocated compute, storage, and network for abstracting selected IoT resources. These orchestration mechanisms should be adaptable to the dynamic changes in Cloud Computing and IoT infrastructures. In the literature, several works have addressed the resource allocation problem in such an environment. Most techniques for selecting connected objects aim to minimize their energy consumption and extend their lifetime [66, 68–70, 84, 85]. However, some contributions focus on selecting the best set of IoT devices based on applications requirements [86, 87] without worrying about energy consumption. These studies perform the selection based on provided functional and non-functional properties of required connected objects. They consider properties such as accuracy, reliability, energy, availability, and cost. From the cloud perspective, resources optimization focuses on minimizing bandwidth consumption [68, 84, 88], storage usage [89], and QoS violation [68, 90, 91].

We refer throughout this work to platforms performing partial integration of the cloud and the IoT as cloud-based IoT platforms. There exist many research projects as well as commercial solutions which provide this kind of IoT platforms. In the following, we represent some of these works.

OpenIoT

The OpenIoT [71, 92] aims at providing an IoT platform with semantically interoperable data streams generated from heterogeneous IoT devices. This project presents the eXtended Global Sensor Network (X-GSN) which is an extension of the GSN middleware. The X-GSN connects to IoT devices and semantically annotates received raw data points which hide the heterogeneity of collected data, allow the unification of data description, and link related data. The OpenIoT follows a data-centric approach. It stores annotated collected data in a cloud storage and allows IoT applications to access them via APIs. The OpenIoT has a scheduler component which receives data streams requests from IoT applications. This component is responsible for allocating needed cloud resources and deploying appropriate services to retrieve and deliver needed data streams.

FIWARE

The FIWARE[12] project [93] aims to create a cloud-based IoT platform based on the Internet of Things Architecture (IoT-A) reference model [94]. The IoT-A defines a set of functional groups needed for a seamless delivery of IoT resources to third party applications. Some of these functional groups are: abstracting connected objects, discovering IoT resources, and storing sensory data. Furthermore, they include a set of services for delivering efficiently registered IoT resources. The FIWARE project implements these functional groups as Generic Enablers (GEs). Each GE is a software component which provides the key functionalities of a functional group. For example, the Backend Device Management GE is responsible for abstracting gateways, sensors, and actuators. The set of interconnected GEs compose the FIWARE cloud-based IoT platform. Such approach enables a modular composition of cloud-based IoT platforms depending on the providers needs. GEs use the Next Generation Services Interface (NGSI)[13] API to communicate with each other and with third party applications. The FIWARE project also includes GEs for managing, orchestrating, and provisioning cloud and IoT resources such as the IaaS GE, the PaaS Manager GE, and the IoT Broker GE.

Commercial Solutions

Multiple commercial solutions provide partial integration of Cloud Computing and IoT resources. We can cite the Google Cloud IoT[14], the Amazon Web Services (AWS) IoT[15], the IBM Watson IoT[16] platform, and many others. These platforms offer seamless connection and integration of IoT devices with cloud services such as storage, analytics, and visualization. They abstract IoT devices, collect their data, and provide means to manages these data. For example, the AWS IoT platform abstracts connected objects as virtual shadows (i.e. VOs) and expose their data via APIs for third party applications. Moreover, it enables the integration of AWS services to manage collected data within the AWS cloud. Some solutions, such as the IBM Watson IoT platform, enables even the automatic deployment of IoT workflows which can be defined using the Node-RED[17] tool. Commercial solutions uses also optimization mechanisms to maximize their infrastructure utilization. However,

[12]https://www.fiware.org/
[13]http://www.openmobilealliance.org/release/NGSI/
[14]https://cloud.google.com/solutions/iot/
[15]https://aws.amazon.com/iot-platform/how-it-works/
[16]https://www.ibm.com/internet-of-things/platform/watson-iot-platform/
[17]https://nodered.org/

they integrate Cloud Computing and IoT technologies solely on the platform level which remains a partial integration.

2.3.3 Full Integration

A full integration consists in extending all traditional Cloud Computing service models (i.e. IaaS, PaaS, and SaaS) to include the IoT. Such expansion enables cloud and IoT resources to be consumed seamlessly as integrated cloud services. Hence, clients are able to provision compute, network, storage, sensing, and actuating resources on-demand from cloud data centres and connected objects. Such resources allocation is possible at the infrastructure, the platform, or the software level. For example, a Raspberry PI connected to a virtual machine might be deployed for a given customer as an IaaS offering. Furthermore, a developer can allocate on the fly an android development environment alongside pollution data streams of multiple sensors spanned across a particular city. The latter scenario should be provisioned seamlessly without human intervention in a full Cloud Computing and IoT integration. In addition, a fully integrated environment must be able to provision IoT platforms similar to those defined in the loose and partial integrations.

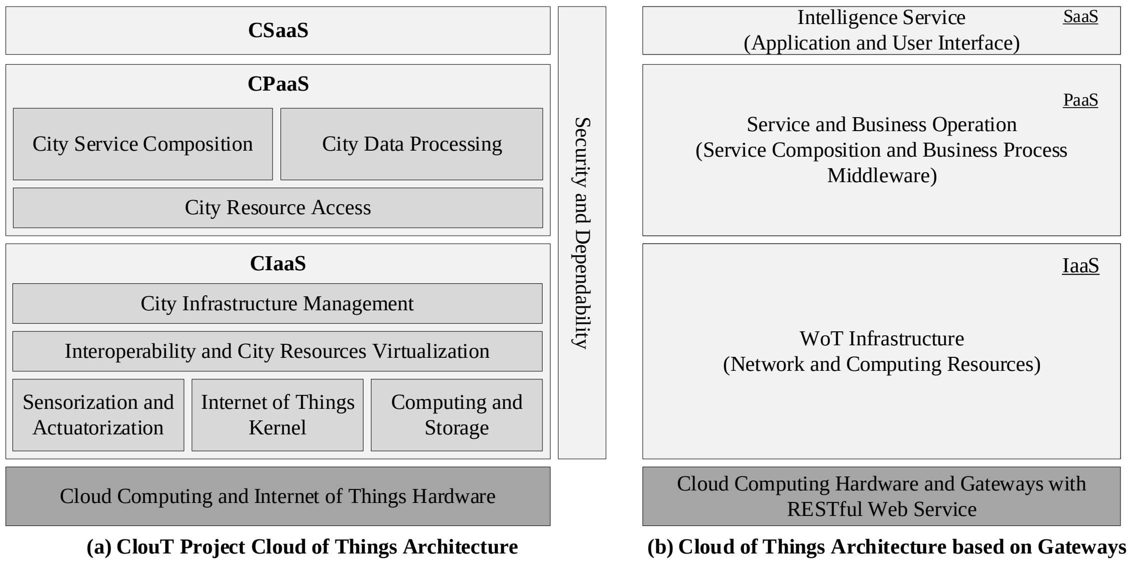

CSaaS
CPaaS
City Service Composition
City Data Processing
City Resource Access
CIaaS
City Infrastructure Management
Interoperability and City Resources Virtualization
Sensorization and Actuatorization
Internet of Things Kernel
Computing and Storage
Cloud Computing and Internet of Things Hardware
Security and Dependability
(a) ClouT Project Cloud of Things Architecture
Intelligence Service (Application and User Interface)
SaaS
PaaS
Service and Business Operation (Service Composition and Business Process Middleware)
IaaS
WoT Infrastructure (Network and Computing Resources)
Cloud Computing Hardware and Gateways with RESTful Web Service
(b) Cloud of Things Architecture based on Gateways

Figure 2.6: Cloud of Things Service Models as Defined by Existing Works (Sources [95, 96]).

In this perspective, some works have defined a reference architecture for the CoT [95, 96] as depicted in Figure 2.6. Authors in [96] describe the CoT for smart cities in the context of the ClouT[18] project (Figure 2.6.a). They represent the City Infrastructure as a Service (CIaaS), the City Platform as a Service (CPaaS), and the City Software as a Service (CSaaS). The CIaaS layer is responsible for delivering abstraction services for the IoT as well as traditional infrastructure level resources

[18]http://clout-project.eu/

for the cloud. The CPaaS provides development environments with accessible data streams. In this layer, developers are able to produce applications which consume sensory data or actuate in the real world while having a set of available integrated services such as data processing. The CSaaS layer enables the deployment of IoT applications for smart cities. Similarly, the study in [95] provides a three layered architecture to integrate IoT resources with cloud offerings (Figure 2.6.b). It defines the lowest layer as the Web of Things (WoT) infrastructure which is composed of gateways exposing underlying connected objects. Each gateway hosts a RESTful WoT web service. The second layer is the PaaS layer which handles service composition and business processes deployment. The uppermost layer represents the SaaS and offer visualization services. To the best of our knowledge, there exists no work addressing the resource allocation problem in a fully integrated environment.

2.4 Open Issues and Challenges

Although some work has been done to define the CoT, this new paradigm is still in its infancy. Many open issues still need to be addressed by the research community. In this section we present some challenges from the resources management and provisioning perspectives in the CoT.

2.4.1 Interoperability

Individual Cloud Computing and IoT have interoperability challenges [13, 97]. On the one hand, Cloud Computing suffers from vendors lock-in due to proprietary solutions which prevent applications portability and interoperability between cloud providers. On the other hand, the IoT encompasses heterogeneous devices with a wide range of capabilities, types, data encodings, and properties. Such diversity makes it harder to build interoperable IoT solutions. In this perspective, several initiatives aimed to enhance interoperability by developing standards for Cloud Computing and the IoT which are discussed with more details in Chapter 4. However, these standards were designed specifically for each domain which make them unfit for leveraging the convergence of Cloud Computing and the IoT. Moreover, the lack of a clear definition and a reference architecture [98] for the CoT resulted in different solutions for combining both technologies as seen in Section 2.3. This variety of existing solutions decreases further the interoperability of CoT platforms. Hence, the need to define a CoT reference architecture and standards for managing compute, network, storage, sensing, and actuating resources.

Reference Architecture

In the IoT, several reference architectures have been proposed such as the IoT-A [94] to provide guidelines for developing IoT platforms. They define domain, information, and functional models alongside needed security measures for the IoT. Furthermore, Cloud Computing also has a well defined reference model [19] describing its service models and characteristics. However, integrating Cloud Computing and the IoT requires novel reference models which consider both domains simultaneously. In fact, existing reference architectures are unfit for representing the CoT. As a result, current solutions for describing the CoT propose different strategies to connect IoT devices and cloud applications which do not fit in neither cloud service models nor the IoT-A functional blocks. For example, studies in [72, 80] used a software component named the SAaaS framework to deliver IoT resources in a cloud environment. This component extends the IoT-A functionalities to enable on-demand and elastic provisioning of sensing and actuation resources in the cloud. Other works [29, 48, 81, 99] used VOs to link between connected objects and cloud applications. However, they do not integrate seamlessly cloud service models and result in additional offerings such as the S^2aaS, the SAaaS, the SOaaS, and many others. Fully integrated solutions provide a clearer representation of IoT offerings applied to Cloud Computing service models. However, different works [95,96] present multiple definitions. Since the IoT-A is an established reference architecture for IoT platforms and has been used in several works such as [72, 80] and cloud-based IoT projects such as OpenIoT and FIWARE [93], it can be used as a starting point for defining a CoT reference architecture. In such case, the IoT-A concepts need to be adapted for a cloud-like service model and encompass cloud characteristics such as rapid elasticity, on-demand provisioning, etc.

Standards

Integrating Cloud Computing and the IoT creates a highly heterogeneous environment. Firstly, connected objects offer proprietary interfaces which do not follow any sort of standardization. Secondly, different sensors generate raw data in distinct formats and different actuators encode their state information variously. Finally, multiple solutions for connecting IoT devices to cloud applications are presented in the literature as stated in Section 2.3. However, these methods are not standardized nor compatible with one another. Moreover, CoT platforms need to support IoT applications portability between different providers to avoid vendors lock-in. Currently, there exist several standards which address some of the standardization issues. The Semantic Sensor Network (SSN) ontology, the Sensor Web Enablement (SWE) Sensor Model Language (SensorML), and the SWE Observations and Measurements (O&M) provide means to describe sensors and their data streams (see Section 4.2). However, some work is still needed to include actuators description within these standards. Furthermore, the SWE suite of specifications also defines standardized interfaces to interact with IoT resources such as the Sensor Observations Service (SOS). Other standards related to Cloud Computing ensure applications portability such as the OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA)[19]. Also, standardized resources management interfaces exist for Cloud Computing such as the Open Cloud Computing Interface (OCCI). They can be extended or combined with IoT related interfaces such as the SWE SOS to provide a CoT resource management interface. As we notice, existing standards solve interoperability issues in individual Cloud Computing and

[19]https://www.oasis-open.org/committees/tosca

IoT environments. However, further efforts are needed to adapt existing standards to a CoT environment similarly to the work in [100] which uses TOSCA to describe and deploy IoT applications.

2.4.2 Resource Provisioning

Resource provisioning is one of the major challenges in Cloud Computing [101] and the IoT [102, 103]. On the one hand, resource provisioning in Cloud Computing consists in mapping Virtual Networks (VNs) onto a Substrate Network (SN). the VN is a set of interconnected nodes which represent Virtual Machines (VMs) in an IaaS deployment, or software components in PaaS and SaaS. Moreover, the SN represents the cloud infrastructure (i.e. cloud data centres and network links). Allocating resources consists in mapping the VN nodes onto candidate SN data centres then selecting the best candidates as illustrated in Figure 2.7.a. Therefore, the optimal solution is the set of candidates which ensures the mapping of the entire VN and optimizes the objective function of the cloud provider. This function aims to maximize the provider's profit, minimize the energy consumption, etc. On the other hand, the IoT resource provisioning process aims to choose the set of sensors and actuators which satisfy applications requests (see Figure 2.7.b). Similarly to Cloud Computing, IoT devices are picked based on an objective such as maximizing QoS, minimizing energy consumption, etc.

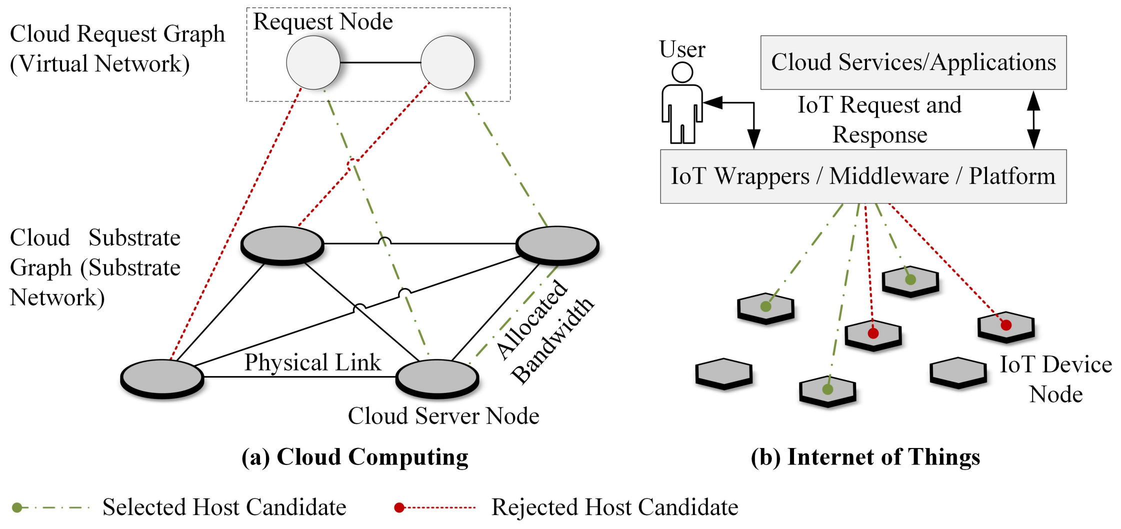


Figure 2.7: Cloud Computing and the IoT Separate Provisioning Processes.

However, provisioning separately cloud and IoT resources in a CoT environment prevents reaching an optimal resource utilization and control. In fact, a seamless integration of Cloud Computing and the IoT requires a holistic approach with global orchestration mechanisms that consider all CoT resources simultaneously (i.e. compute, network, storage, sensing, actuating). Such holistic provisioning cannot be reached with previous works as it needs to account for multiple aspects regarding Cloud Computing and IoT at the same time:

- Selecting the set of connected objects which satisfy the requested functional (e.g. type) and non-functional (e.g. accuracy) requirements specified by the IoT application.

- Provisioning cloud services responsible for managing previously selected connected objects across cloud data centres.

- Provisioning connectivity services (e.g. VOs) to link selected connected objects and deployed cloud services.

These aspects should be considered while aiming to minimize the CoT infrastructure resource utilization and maximizing the QoS experienced by end-users consuming the deployed IoT applications. In addition, orchestrating IoT connectivity services such as IoT middlewares or VOs requires the consideration of multiple delivery strategies as defined in the IoT-A project [94]. For example, an IoT device might be connected directly, through a VO, or through an IoT middleware. Other methods also exist such as the data-centric approach. Therefore, the provisioning process should be aware of all possible deployment strategies and be able to select the optimal configuration which reduces the operational cost of the deployment. Furthermore, these delivery methods can be provisioned to serve one or multiple applications. Hence, methods for considering previously deployed IoT delivery services and re-using them are needed to avoid the excessive use of resources. These challenges are still to be addressed when composing and provisioning IoT applications in a CoT environment.

2.5 Conclusion

The CoT offers a new scope of IoT applications and services which, in contrast to traditional IoT infrastructures, can be self-managed, self-configured, and automatically deployed without human intervention. In this chapter, we surveyed several works in the literature thriving for realizing such vision. We classified these works based on the way they integrate the IoT and Cloud Computing to realize the CoT. Such classification resulted in three categories: the loose integration, the partial integration, and the full integration. We highlighted the resource allocation problem addressed within each integration strategy. Furthermore, we presented the open issues related to resource modelling and provisioning in the CoT. More contributions are needed to achieve autonomous end-to-end IoT applications provisioning and deployment.

Chapter 3

Efficient Provisioning of Shared Virtual Objects

Contents

3.1 Introduction

Cloud-based Internet of Things (IoT) platforms were first proposed in [29]. These platforms aim to take benefit from cloud resources and characteristics to deliver IoT service models such as the Sensing as a Service (S^2aaS) [74, 75, 77, 104] and the Sensor/Actuator as a Service (SAaaS) [79, 80]. These platforms provide IoT resources to multiple applications running in the Cloud by abstracting heterogeneous physical and logical IoT devices using Virtual Objects (VOs). A VO is a software component which abstracts the interaction with a real IoT device. It wraps all the functionalities of the object (i.e. sensing, actuating, configuration) and provide them via common Application Programming Interfaces (APIs). Cloud-based IoT platforms deploy and manage VOs on-demand to provide required IoT resources for cloud applications. Such platforms benefit from cloud features to scale up/down or migrate VOs to cope with applications requirements and Quality of Service (QoS) terms. The dynamic allocation of IoT resources permits also to increase the lifetime of IoT devices reducing their solicitation when information is already available in their associated VO [67].

VOs are not deployed individually in independent Virtual Machines (VMs) which will be inefficient and resources consuming. The large-scale nature of the IoT makes it impossible to run one VM for each VO. Therefore, IoT middlewares manage collections of VOs and provide APIs to access them individually. Middlewares are themselves executed in VMs hosted in a Cloud Computing infrastructure. It starts and stops VOs as needed. Launching a VO consists on instantiating a wrapper that connects to a given IoT device. In such context, the distribution of VOs and VMs within the cloud infrastructure affects the resources utilization efficiency, the IoT applications QoS, and the operation cost of the overall system. For example, provisioning two VOs in a single VMs costs less than provisioning them in multiple VMs. However, if these VOs communicate with different applications, separating them might have an important impact on the QoS (e.g. latency for each application). Consequently, provisioning mechanisms are very important and have an important impact on the QoS and deployment cost of VOs.

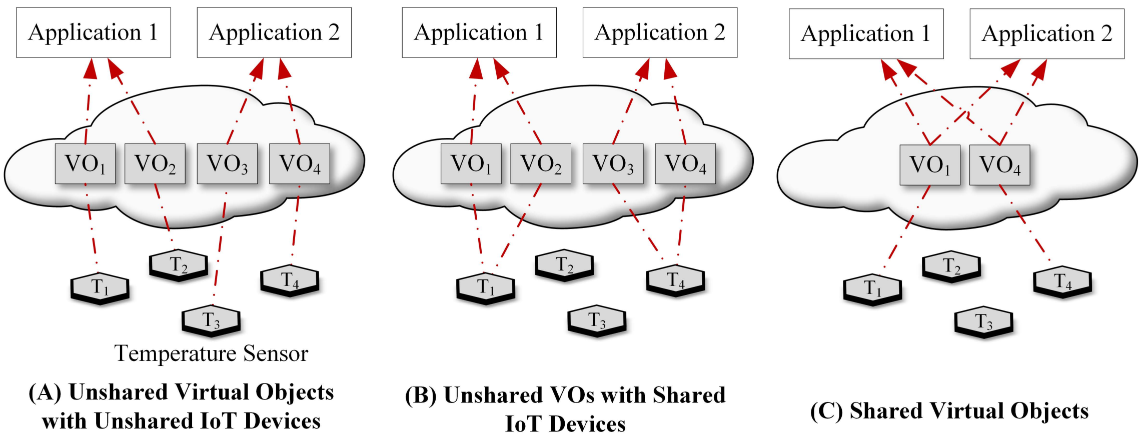

Application 1
Application 2
VO_1
VO_2
VO_3
VO_4
T_1
T_2
T_3
T_4
Temperature Sensor
(A) Unshared Virtual Objects with Unshared IoT Devices
(B) Unshared VOs with Shared IoT Devices
(C) Shared Virtual Objects

Figure 3.1: Virtual Object Provisioning Approaches.

Moreover, several applications might request similar IoT resources (e.g. temperature sensing in the same location) and the same IoT object might serve multiple applications. In such case, there are several possible configurations to abstract sensing and actuating resources in cloud-based IoT platforms. These configurations are depicted in Figure 3.1. In part (a) of the figure each IoT device connects to one VO which belongs to a single application, in part (b) connected objects are shared between applications in contrast to VOs, in part (c) IoT resources and VOs can be shared. In cases (b) and (c), connected objects might be shared or not based on their capabilities. To the best of our knowledge, most previous works in the area have only considered the first case (i.e. case a). However, sharing connected objects and VOs reduces needed cloud resources to deliver sensing and actuating functionalities to applications. In this thesis, we highlight the benefit of sharing IoT resources between different applications in the context of cloud-based IoT platforms. We propose two analytical formulations of the problem and associated solutions to optimize the placement of shared IoT resources (i.e. VOs) in the Cloud Computing in order to satisfy the global system performances.

The chapter is organized as follows. First, We present the specific related works in Section 3.2. we present the challenges to share efficiently IoT resources in Section 3.3. In Sections 3.4 and 3.5 we present our model to provision shared VOs in the Cloud Computing. We provide two analytical formulations of the VOs placement optimization: (a) static (Section 3.4) and (b) dynamic (Section 3.5). Both analytical models are based on the Linear Program (LP). The static LP considers a new environment with no previously deployed VOs, while the dynamic model takes into consideration the previously deployed VOs when placing a new request. Afterwards, simulations results are discussed in section 3.6. Finally, we conclude the chapter with an overview outlining the benefits of our approach.

3.2 Related Works

The selection of connected objects is an important functionality of cloud-based IoT platforms. There are several techniques proposed in the literature [105] to achieve this selection. Contributions made by [86, 87] propose several techniques to select the best k connected objects based on an application request: comparative-priority based heuristic filtering, relational-expression based filtering and a distributed object searching technique. The search techniques take into consideration non functional requirements of IoT devices with distinct priorities in the selection process. Authors consider accuracy, reliability, energy, availability, and cost of connected objects. However they do not try to optimize the number of used objects. This issue was addressed later in [68]. The focus of this work is the optimal selection of connected objects to satisfy the applications requirements while preserving the efficiency of resource utilization. The authors propose two algorithms `CoV-I` and `CoV-II` addressing two particular cases: when physical objects are homogeneous and fall within the same geographical area (i.e. `CoV-I`), and when objects are heterogeneous and geographically distributed (i.e. `CoV-II`). Results show that both algorithms enhance the resources utilization by selecting only the necessary IoT devices for each application.

Other related works have addressed the network resources utilization as described in [3, 69, 88, 90, 106]. The approach presented in [70, 88] relies on the control of data transmission rates between different entities in a cloud-based IoT platform to reduce the consumption of the network resources and increase the physical connected objects lifetime. Phan et al. [88] focus on optimizing communication bandwidth in SC-iPaaS (Sensor-Cloud Integration PaaS). In SC-iPaaS, IoT devices send data periodically to corresponding sinks, who relay them at different frequency rates to VOs. When end-users request data streams to their applications running in the cloud, applications invoke the corresponding VOs. VOs reply directly if the requested data streams are locally available, otherwise they request up to date information to the connected objects. Authors seek in this work the Pareto-optimal data transmission rate for each connected to object and sink node to maximize applications requests success rate while minimizing objects' energy consumption and network resource utilization. Phan et al. define three objective functions: (a) maximizing the objects data yield (i.e. data availability) for cloud applications, (b) minimizing the bandwidth consumption between the cloud layer and the edge layer, and (c) minimizing the energy consumption of connected objects in the IoT infrastructure.

Moreover, other related works have addressed the limited processing, storage, and energy capabilities of connected objects and how to use them in an optimal manner. In [106], Xu et al. increase IoT devices lifetime by optimizing the data requests from these devices, and the data demand from applications. Authors propose a bi-directional waterfall optimization framework which relies on data/application caching. Instead of only caching data while it is moving up to the applications in the cloud-based IoT platform, authors move parts of the applications logic down to the edges. Hence, the amount of data to send to the cloud applications decreases which reduces the bandwidth overhead for data transmission. In [69], authors optimize the transmission of data streams by assigning the closest data centre to host virtual objects, then schedule a particular data centre to aggregate data from these VOs.

Finally, several works focused on the QoS in an IoT environment, in particular, delay and response times. Misra et al. introduced in [90] a gateway selection mechanism to establish an efficient, reliable and cost-effective health monitoring system and to minimize transmission delay with cloud applications. Authors considered static connected object devices while authors in [3] considered instead a Mobile Cloud Computing (MCC) environment where IoT devices can be mobile and VOs deployed in the cloud process the incoming data streams. Authors formulated the problem as a linear program to jointly optimize the gateway selection and the services deployment cost in the cloud. Authors objective was to minimize data flows fluctuation in each cloud data centre and the rate of required reconfiguration operations to satisfy the changing load.

In the mentioned related works, several approaches have been used to optimize network resources when transmitting data streams to applications via virtual objects, however, all these approaches considered that virtual objects are dedicated to a particular application and not shared among them. In our work, we consider that sharing VO is very important and can help to derive a more efficient solution. Therefore, we take a different approach and we propose that virtual objects can be shared among multiple applications when they are requesting the same data and that the objective is to further minimize resources utilization while maintaining the same level of QoS for the applications. This allows to go one step forward in the optimization of the resources, QoS assurance and reduction of operational cost for such applications.

3.3 Problem Statement

A provisioning mechanism of cloud resources to deliver shared IoT devices must address several challenges. Firstly, the IoT large-scale infrastructure increases substantially the problem size. Such infrastructure consists of highly distributed numerous connected objects spanned over multiple geographical areas. Consequently, orchestrating VOs for a large number of allocated sensing and actuating resources can be time consuming. The provisioning algorithm must cope with such large scale IoT environment. It should scale with large number of requests and produce solutions in an acceptable response time. Furthermore, such algorithm should consider two data flows: (a) the data exchange between connected objects and VOs hosted in cloud data centres, and (b) the data transmission between VOs and cloud applications.

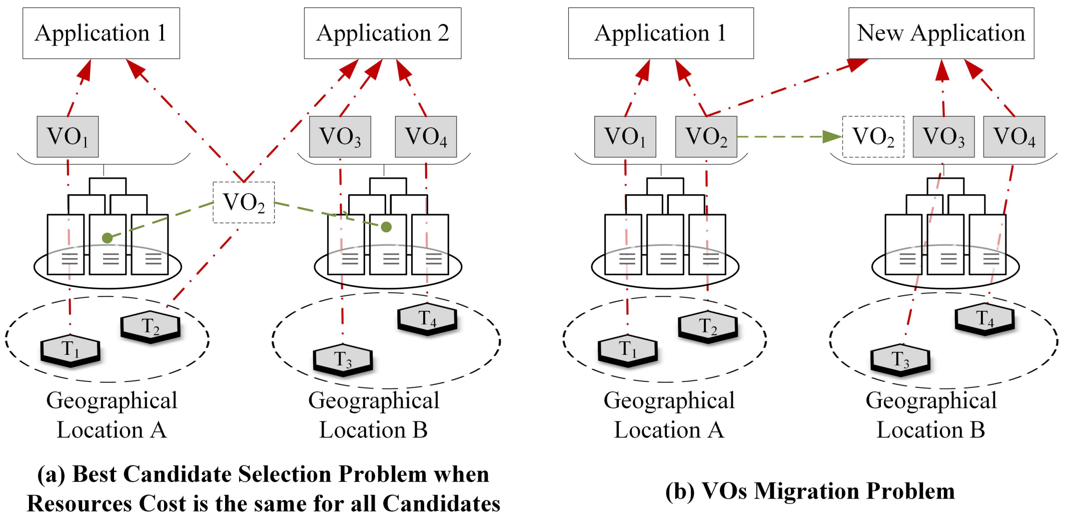


Figure 3.2: Virtual Object Sharing Challenges Examples.

Secondly, sharing VOs amongst multiple applications requires a trade-off between optimal placements of each application individually. The provisioning process should optimize the placement of each request without penalizing a particular one. For example, let's consider 4 VOs requested by two applications. $\texttt{VO}_1$ belongs to one of these applications, while $\texttt{VO}_3$ and $\texttt{VO}_4$ serve the other one. Each application's VOs are deployed in separate data centres as shown in Figure 3.2.a. The $\texttt{VO}_2$ sends data to both applications. In this example, $\texttt{VO}_2$ has two possible candidates with already deployed VMs. Both candidates satisfy applications QoS terms. We consider that the costs of hosting $\texttt{VO}_2$ in both data centres are similar for all types of resources (i.e. compute, network, and storage). Therefore, the provisioning decision is mainly impacted by the QoS requirements of each application (e.g. latency). In this particular case, an efficient orchestration algorithm should find the placement that best satisfies all the applications QoS requirements instead of finding a placement based on other criteria.

Finally, a sharing strategy of connected objects might require that new requests are served by already deployed VOs. However, VOs might need to satisfy some particular QoS requirements for the initiating applications. In this case, the migration feature of Cloud Computing might be used to move VOs to a location that permits to satisfy these QoS requirements instead of creating new VOs. Figure 3.2.b illustrates such use case: a new application requires $\texttt{VO}_2$ which is already deployed for another one. Since the previous placement violates the QoS of the new application, the $\texttt{VO}_2$ is migrated to another host that is more appropriate in term of QoS assurance. The migration operation induces additional costs for the cloud operator. Therefore, the proposed model needs to take into account the migration cost and try to minimize the reconfiguration operations performed on previously deployed VOs. Additional granularity might be considered as well during the provisioning process. For example, if the migration cost of a previously deployed VO is high, another VO is instantiated without performing the migration operation. In such case, two VOs connect multiple applications to the same IoT device. However, such considerations increase considerably the algorithm complexity and its computation time. Therefore, we decided to not consider such a solution since we aim to propose only a scalable algorithm with acceptable time complexity. Therefore, in the proposed approach, we consider that a VO and an IoT device can be shared between several applications with the restriction that only one VO can be associated with an IoT device.

3.4 Static Virtual Objects Placement Optimization Model

In this section, we address the challenges of sharing VOs among different applications. First of all, we formulate an analytical model of the problem which is an optimization problem under constraints, then we propose solutions to optimize the placement of VOs in such environment. Table 3.1 is a notation table that illustrates the significant variables used in the model.

3.4.1 Internet of Things Objects Clustering

In the IoT, connected objects usually belong to various geographical areas. When IoT devices belong to the same geographical areas, communication costs with their associated cloud data centre are similar. Indeed, since the data follows the same routing path, experienced bandwidth capacity and QoS (such as latency) are similar. Therefore, there is no need to consider each connected object individually and it is therefore possible to treat them as a collection. We propose therefore to reduce the problem size by clustering the connected objects in the IoT based on different geographical areas. Such grouping does not alter the efficiency of the placement process. However, since selected IoT devices in each geographical area may communicate with different applications hosted in different data centres, QoS experience may be different. Therefore, in order to assure the required level of QoS, we propose to subdivide further the cluster and group together connected objects serving the same set of applications. Each obtained cluster at the end of the partitioning should belong to only one geographical area. Each cluster is also associated with the same group of cloud applications and QoS requirements.

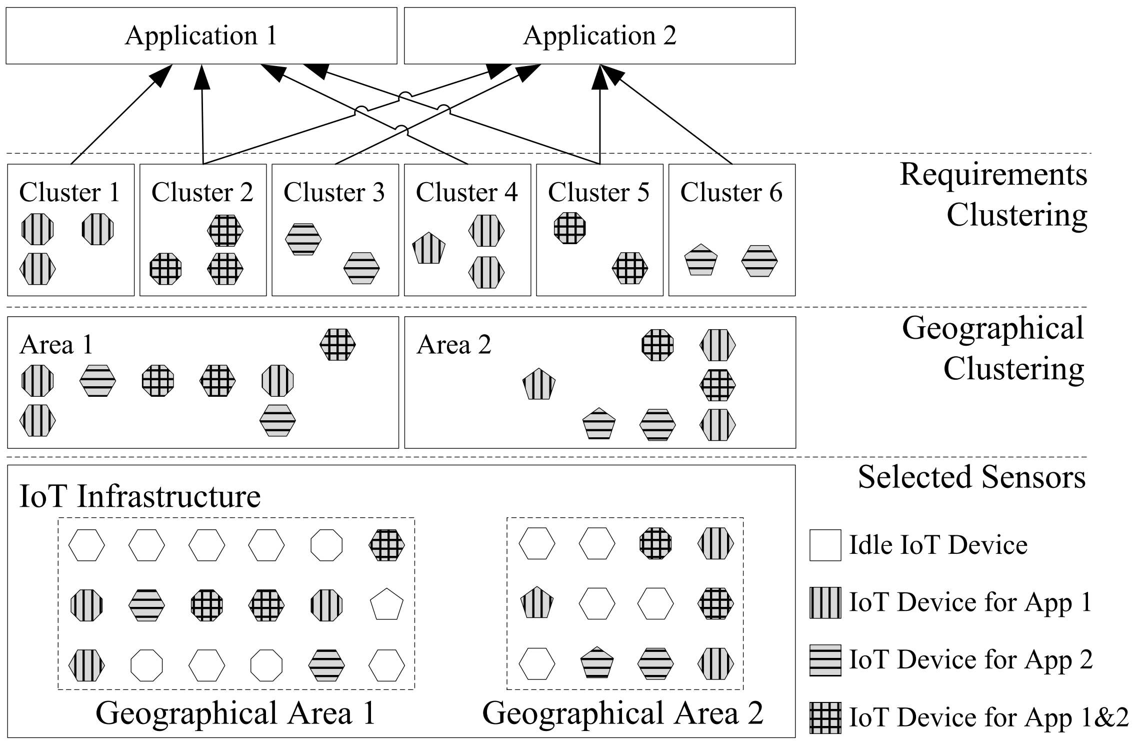


Figure 3.3: Clustering Steps Before the Provisioning Process.

Figure 3.3 shows the proposed clustering process. First, the selected pool of connected objects is separated into geographical groups. Then, they are further separated by similar requirements groups (i.e. same set of served applications). Let's consider a set of applications $A = a_1, ..., a_n$ having each a set of associated connected objects (IoT devices) $O = O_1, ..., O_n$. Each set of connected object has a cardinality n_i and is represented as $O_i = o_{i,1}, ..., o_{i,n_i}$. Having these elements, we propose the Algorithm 3.1 to compute the set of clusters C. This clustering approach permits to reduce the size of the problem to derive a solution with a reduced complexity.

Algorithm 3.1 Clustering Algorithm for the Static Optimization Model

Cluster-Static(A, O)

In: Set of n applications $A = \{a_1, ..., a_n\}$; Sets of assigned connected objects for each application $O = \{O_1, ..., O_n\} = \{o_1, ...\}$.

Out: Sets of clusters $C = \{C_1, ...\}$ where each cluster C_i groups connected objects with similar requirements. Sets of requirements $R = \{R_1, ...\}$ for each cluster.

```
1: $C \leftarrow \emptyset; R \leftarrow \emptyset$;
2: for all selected connected objects $o_i \in O$ do
3:     create a temporary set $T$ containing the geographical location of $o_i$.
4:     add to $T$ all the cloud applications $A_j \in A$ served by $o_i$.
5:     if $\exists R_k$ such that $R_k = T$ then
6:         $C_k \leftarrow o_i$
7:     else
8:         $R \leftarrow \{T\}$ {add the new group of requirements to $R$}
9:         $C \leftarrow \{o_i\}$ {create a new cluster for new requirements $T$}
10:    end if
11: end for
12: return $C, R$
```

3.4.2 Placement Optimization Problem Formulation

Cluster-Static(A, O) produces a finite number of clusters $C = \{C_1, ..., C_m\}$. Each cluster C_i represents a set of connected objects, has a geographical location $g_i \in R_i$, and serves a set of applications $\Delta_i \subset R_i$ such that $\Delta_i = R_i \setminus \{g_i\}$. VOs mirror connected objects within these clusters in data centres $D = \{d_1, ..., d_p\}$ and provide their services to cloud applications. Figure 3.4 shows the network schema of such environment. We define $e^{cd}_{i,j}$ and $e^{da}_{i,j}$ as network links. An edge $e^{cd}_{i,j}$ lies between a cluster C_i and a data centre d_j, while $e^{da}_{i,j}$ connects a data centre d_i and an application a_j. Each network link possesses functional and non-functional properties such as available bandwidth ($b^{cd}_{i,j}$, $b^{da}_{i,j}$) and network latency ($l^{cd}_{i,j}$, $l^{da}_{i,j}$) respectively.

Network Model

IoT devices exchange data with VOs. Therefore, each cluster C_i produces an average transmission rate represented as λ^c_i. The generated average transmission rate for a cluster C_i is equal to the sum of all individual average rates produced by connected objects within this cluster. However, VOs abstracting IoT devices in one cluster might not be deployed in the same data centre. Hence, a data centre receives a portion of a cluster's average transmission rate. This portion is represented by the variable $\alpha_{i,j}$. It corresponds to the fraction of connected objects within the cluster C_i managed by VOs in the data centre d_j. As a result, the arrival rate (i.e $\theta^{cd}_{i,j}$) generated by a given cluster C_i at a data centre d_j is:

$$\theta^{cd}_{i,j} = \alpha_{i,j}\lambda^c_i \tag{3.1}$$

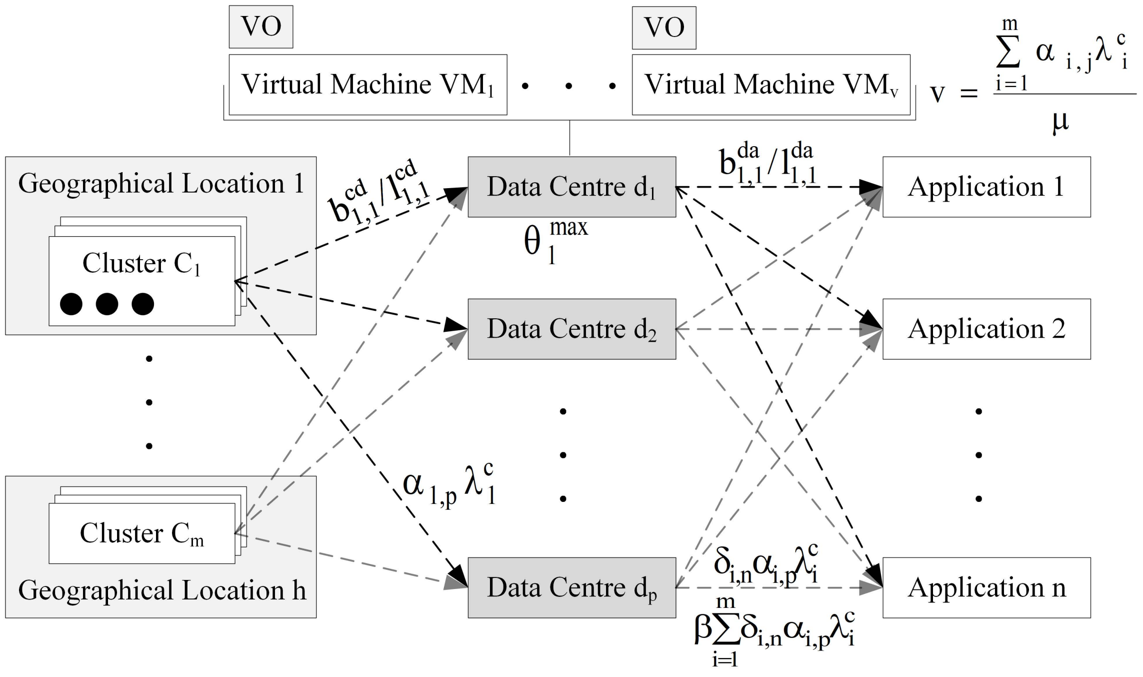

VO
Virtual Machine VM$_1$
VO
Virtual Machine VM$_v$
$v = \frac{\sum_{i=1}^{m} \alpha_{i,j}\lambda_i^c}{\mu}$
Geographical Location 1
Cluster C$_1$
$b_{1,1}^{cd}/l_{1,1}^{cd}$
Data Centre d$_1$
θ_1^{max}
$b_{1,1}^{da}/l_{1,1}^{da}$
Application 1
Data Centre d$_2$
Application 2
$\alpha_{1,p}\lambda_1^c$
Cluster C$_m$
Geographical Location h
Data Centre d$_p$
$\delta_{i,n}\alpha_{i,p}\lambda_i^c$
$\beta\sum_{i=1}^{m}\delta_{i,n}\alpha_{i,p}\lambda_i^c$
Application n

Figure 3.4: Network Model of the Placement Problem.

Consequently, we can derive the overall ingress rate at any data centre d_j as following:

$$\theta_j^d = \sum_i \theta_{i,j}^{cd} = \sum_i \alpha_{i,j}\lambda_i^c \quad \forall i : 1 \rightarrow m \tag{3.2}$$

In some cases, VOs hosted in data centres do aggregation operations on data or add additional information such as semantic annotations. Therefore, the egress traffic of VOs is not equal to the ingress traffic of connected objects. We use a coefficient $\beta > 0$ to represent the effect of VOs operations on received data. If $\beta \in]0, 1[$, the deployed VOs aggregate data. However, for $\beta > 1$, the instantiated VOs provide additional information regarding the data for cloud applications. Also, the communication overhead difference between ingress and egress traffic can be handled by the coefficient β. In our work, we consider that the cloud-based IoT platform deploys a single type of VOs similarly to existing platforms discussed in Section 2.3. As a result, the value of β is constant for all VOs. Thus, the egress traffic for a data centre d_j is the following:

$$\lambda_j^d = \beta \sum_i \alpha_{i,j}\lambda_i^c \quad \forall i : 1 \rightarrow m \tag{3.3}$$

Each application receives the amount of data transmitted by its assigned connected objects. However, the latter connected objects are spread across different clusters. Also, they are represented by VOs in cloud data centres. Hence, we can express the traffic received by each application as a function of: (a) the transmitted rate of clusters, or (b) the egress traffic of VOs. We define the binary variable $\delta_{i,k}$. It equals 1 if the application a_k belongs to the set Δ_i, it is 0 otherwise. As a result,

Table 3.1: Notation Table

Symbol	**Definition**
C	Set of connected objects clusters; $C = \{C_1, ..., C_m\}$.
D	Set of data centres; $D = \{d_1, ..., d_p\}$.
A	Set of cloud applications; $A = \{a_1, ..., a_n\}$.
Δ_i	The set of applications served by the cluster C_i.
$\delta_{i,k}$	Binary value equal to 1 if $a_k \in \Delta_i$, and 0 otherwise.
g_i	Geographical location of a cluster C_i.
λ_i^c	Total egress traffic of a cluster C_i.
$b_{i,j}^{cd}$	Network bandwidth between a cluster C_i and a data centre d_j.
$l_{i,j}^{cd}$	Network latency between a cluster C_i and a data centre d_j.
$\theta_{i,j}^{cd}$	Ingress traffic of a data centre d_j from a cluster C_i.
θ_j^d	Total ingress traffic of a data centre d_j.
$\theta_{j,k}^{da}$	Ingress traffic of an application a_k from a data centre d_j.
$b_{j,k}^{da}$	Network bandwidth between a data centre d_j and an application a_k.
$l_{j,k}^{da}$	Network latency between a data centre d_j and an application a_k.
$\phi_{i,j}^{cd}$	Price of a data unit between a cluster C_i and a data centre d_j.
$\phi_{j,k}^{da}$	Price of a data unit between a data centre d_j and an application a_k.
ϕ_j^{vm}	Price of one virtual machine in the data centre d_j.
$\alpha_{i,j}$	The fraction of VOs mirroring cluster C_i and hosted in data centre d_j.

the average arrival rate sent by a cluster C_i or a data centre d_j to an application a_k can now be expressed respectively as follows:

$$\theta_{i,k}^{ca} = \delta_{i,k}\lambda_i^c \tag{3.4}$$

$$\theta_{j,k}^{da} = \lambda_{j,k}^{da} = \beta \sum_i \delta_{i,k}\theta_{i,j}^{cd} = \beta \sum_i \delta_{i,k}\alpha_{i,j}\lambda_i^c \;\; \forall i : 1 \to m \tag{3.5}$$

with:

$$\delta_{i,k} = \begin{cases} 1 & \text{if } a_k \in \Delta_i \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

As well, the overall traffic received by an application a_k is:

$$\theta_k^a = \beta \sum_i \delta_{i,k}\lambda_i^c \quad \forall i : 1 \to m \tag{3.7}$$

The traffic between clusters, data centres, and applications, is handled by network links. Transmitted data rates on a given network link should not exceed its available bandwidth capacity (constraints 3.8 and 3.9). Furthermore, the data centres has to be capable of handling the received traffic (constraint 3.10). In fact, each data centre is able to manage a maximal amount of traffic represented by θ_j^{max}. Also, the received traffic by an application a_k should comply with its requested QoS terms. In our work, we consider the latency of network links as the QoS indicator. Each application a_k QoS is represented by its highest acceptable latency l_k^{max} (constraint 3.11). These network constraints must be satisfied when mapping VOs to cloud data centres and are expressed as follows:

$$\theta_{i,j}^{cd} \leq b_{i,j}^{cd} \quad \forall i : 1 \rightarrow m,\ \forall j : 1 \rightarrow p \tag{3.8}$$

$$\theta_{j,k}^{da} \leq b_{j,k}^{da} \quad \forall j : 1 \rightarrow p,\ \forall k : 1 \rightarrow n \tag{3.9}$$

$$\theta_j^{d} \leq \theta_j^{max} \quad \forall j : 1 \rightarrow p \tag{3.10}$$

$$\delta_{i,k}\alpha_{i,j}(l_{i,j}^{cd} + l_{j,k}^{da} - l_k^{max}) \leq 0 \quad \forall i : 1 \rightarrow m,\ \forall k : 1 \rightarrow n,\ \forall j : 1 \rightarrow p \tag{3.11}$$

Quality of Service Index

During the deployment process, some VOs might have several hosts candidates with similar cloud resources costs (see Section 3.3) as illustrated in Figure 3.5. In this case, the provisioning process should select the data centre which provides the best QoS for all applications. Therefore, we define a normalised parameter, the QoS index $q_{i,j} \in [0, 1]$. This index reflects the inverse of the QoS level experienced by the set of applications Δ_i using cluster C_i when their VOs are installed in data centre d_j. The lower the value of $q_{i,j}$ is, the higher the QoS level is. We use this to prioritize hosts providing better QoS for cloud applications (See equation 3.16). The QoS index is expressed as follows:

$$q_{i,j} = \frac{l_{i,j}^{cd} + \sum_k \delta_{i,k} l_{j,k}^{da}}{(1 + \sum_k \delta_{i,k})L} \quad \forall k : 1 \rightarrow n \tag{3.12}$$

with:

$$L = \max_{i,j,k}(l_{i,j}^{cd}, l_{j,k}^{da}) \quad \forall i : 1 \rightarrow m,\ \forall j : 1 \rightarrow p,\ \forall k : 1 \rightarrow n \tag{3.13}$$

Cost Function

We provide a model for orchestrating VOs in the cloud. The selection of connected objects suitable for each application is out of the scope of this work. Consequently, we do not consider costs related to the allocation of IoT resources. We only represent costs associated with cloud resources. The basic resource in the cloud is the VM. As mentioned previously, VMs manage instantiated VOs which are software components within VMs. These VOs consume VMs resources based on the

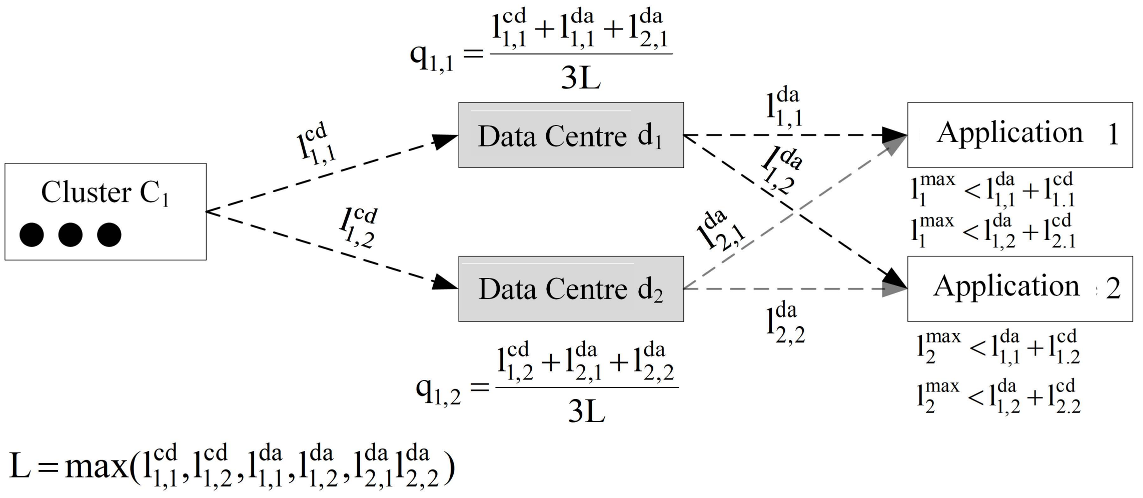

$q_{1,1} = \frac{l^{cd}_{1,1} + l^{da}_{1,1} + l^{da}_{2,1}}{3L}$
Cluster C_1
$l^{cd}_{1,1}$
$l^{cd}_{1,2}$
Data Centre d_1
Data Centre d_2
$l^{da}_{1,1}$
$l^{da}_{1,2}$
$l^{da}_{2,1}$
$l^{da}_{2,2}$
Application 1
Application 2
$l^{max}_1 < l^{da}_{1,1} + l^{cd}_{1.1}$
$l^{max}_1 < l^{da}_{1,2} + l^{cd}_{2.1}$
$l^{max}_2 < l^{da}_{1,1} + l^{cd}_{1.2}$
$l^{max}_2 < l^{da}_{1,2} + l^{cd}_{2.2}$
$q_{1,2} = \frac{l^{cd}_{1,2} + l^{da}_{2,1} + l^{da}_{2,2}}{3L}$
$L = \max(l^{cd}_{1,1}, l^{cd}_{1,2}, l^{da}_{1,1}, l^{da}_{1,2}, l^{da}_{2,1} l^{da}_{2,2})$

Figure 3.5: Provisioning Scenario with the QoS Index.

traffic they handle. In our work, we consider that all VMs are able to handle the same arrival rate μ. Therefore, we can calculate the total number of VMs needed in each data centre based on the overall traffic managed by it. Each data centre d_j has a different VM cost represented as ϕ_j^{vm}. As a result, the cost of needed VMs for orchestrating requested VOs is expressed as follows:

$$F^{vm}(\alpha) = \sum_j \phi_j^{vm} \left\lceil \frac{\theta_j^d}{\mu} \right\rceil = \sum_j \phi_j^{vm} \left\lceil \frac{\sum_i \alpha_{i,j} \lambda_i^c}{\mu} \right\rceil \quad \forall i : 1 \to m, \ \forall j : 1 \to p \tag{3.14}$$

In order to linearise $F^{vm}(\alpha)$ we introduce the variable u_j to replace the ceiling function. The relation between the variable u_j is defined as $u_j = \lceil \theta_j^d / \mu \rceil$ and modelled with constraints (3.15a), (3.15b), and (3.15c). The VM cost function is now represented as follows:

$$F^{vm}(\alpha) = \sum_j \phi_j^{vm} u_j \quad \forall i : 1 \to m, \ \forall j : 1 \to p \tag{3.15}$$

with:

$$u_j \geq \frac{\theta_j^d}{\mu} \quad \forall j : 1 \to p \tag{3.15a}$$

$$u_j \leq \frac{\theta_j^d}{\mu} + 1 \quad \forall j : 1 \to p \tag{3.15b}$$

$$u_j \in \mathbb{Z}^+ \quad \forall j : 1 \to p \tag{3.15c}$$

Moreover, the network cost represents the price of data exchanges between connected objects, data centres, and applications. We define the cost of transmitting a unit of data on a network link between two nodes n_i and n_j as $\phi_{i,j}^n$. The network cost is the following:

$$F^n(\alpha) = \sum_i \sum_j q_{i,j}\phi^{cd}_{i,j}\theta^{cd}_{i,j} + \beta \sum_i \sum_j \sum_k q_{i,j}\phi^{da}_{j,k}\delta_{j,k}\alpha_{i,j}\lambda^c_i$$
$$\forall i : 1 \to m,\ \forall j : 1 \to p,\ \forall k : 1 \to n \tag{3.16}$$

Our objective is to calculate the provisioning plan which minimizes the previously defined costs. Therefore, the objective function is defined as follows:

$$\min_{\alpha} F(\alpha) = \omega F^{vm}(\alpha) + \gamma F^n(\alpha)$$
$$\text{s.t. } \omega + \gamma = 1 \tag{3.17}$$

subjected to:

Bandwidth constraints:

$$\theta^{cd}_{i,j} \le b^{cd}_{i,j} \quad \forall i : 1 \to m,\ \forall j : 1 \to p \tag{3.17a}$$

$$\theta^{da}_{j,k} \le b^{da}_{j,k} \quad \forall j : 1 \to p,\ \forall k : 1 \to n \tag{3.17b}$$

Data centre capacity constraint:

$$\theta^d_j \le \theta^{max}_j \quad \forall j : 1 \to p \tag{3.17c}$$

QoS related constraint:

$$\delta_{i,k}\alpha_{i,j}(l^{cd}_{i,j} + l^{da}_{j,k} - l^{max}_k) \le 0$$
$$\forall i : 1 \to m,\ \forall j : 1 \to p,\ \forall k : 1 \to n \tag{3.17d}$$

Domain variable constraints:

$$\sum_j \alpha_{i,j} = 1 \quad \forall i : 1 \to m \tag{3.17e}$$

$$\alpha_{i,j} \ge 0 \quad \forall i : 1 \to m,\ \forall j : 1 \to p \tag{3.17f}$$

$$u_j \ge \frac{\theta^d_j}{\mu} \quad \forall j : 1 \to p \tag{3.17g}$$

$$u_j \le \frac{\theta^d_j}{\mu} + 1 \quad \forall j : 1 \to p \tag{3.17h}$$

$$u_j \in \mathbb{Z}^+ \quad \forall j : 1 \to p \tag{3.17i}$$

ω and γ are coefficients to specify the weight of each cost in the total value of the function F. The constraint (3.17e) enforces the orchestration of all needed VOs, and verifies that each connected object is assigned to only one VO. Also, the constraint (3.17f) limits the values of $\alpha_{i,j}$ to positive real numbers only.

3.5 Dynamic Virtual Objects Placement Optimization Model

Unlike the static approach described in Section 3.4, a dynamic orchestration process must adapt to changes at the software and infrastructure levels. Software level changes are characterized by new applications or variations in previous applications requests, while infrastructure level changes are initiated by connected objects mobility or failure (e.g. empty battery). On both levels, the orchestration process needs to reallocate and adapt continuously cloud and IoT resources to cope with these variations. In this work, we do not deal with the dynamic selection of IoT resources. Such problem is well studied in the literature and various solutions are provided [103]. We focus on the provisioning of cloud resources, and therefore the distribution of VOs in such a dynamic environment. As a result, changes in the infrastructure are presented and processed as variations in existing applications needs. In this section, we adapt the model presented in Section 3.4 to include the dynamic aspect of cloud-based IoT platforms.

3.5.1 Internet of Things Objects Clustering

The dynamic arrival of applications requests populates data centres with VOs over time. These VOs are shareable and therefore might be reused by forthcoming applications requests. However, the placement of reused VOs might violate the QoS required by new requests. Therefore, the provisioning process should investigate the validity of such VOs placement and migrate them to suitable hosts if needed. Such control is not necessary for unshared VOs. In fact, shared VOs should satisfy all the QoS requirements of applications consuming them as mentioned in Section 3.4. Hence, migrating VOs should be performed while accounting for all their connected applications requirements, and not only new ones. Considering these applications in the orchestration process enforces their QoS terms.

Figure 3.6 depicts this scenario. It illustrates a new application requesting connected objects o_2, o_3, and o_4, while an existing application uses o_1 and o_2. Both applications share the connected object o_2 and therefore its related VO (i.e. $\texttt{VO}_2$). In this example, we notice that the current placement of $\texttt{VO}_2$ does not comply with the QoS required by the new application. Therefore, $\texttt{VO}_2$ should be migrated. If the provisioning process considers solely the QoS requirements of the new application, it might migrate $\texttt{VO}_2$ to the data centre in location B. The latter data centre violates the QoS terms of the existing application. Hence, the provisioning process should include both applications requirements in the provisioning process to select the best placement for both (i.e. data centre at location C).

In this perspective, clustering connected objects in the dynamic model should not operate solely on new applications. It should consider previously deployed applications consuming these connected objects as well. Therefore, we consider two sets of applications $A^t = \{a_1^t, ..., a_x^t\}$ and $A^{t-1} = \{a_1^{t-1}, ..., a_y^{t-1}\}$ corresponding to new and existing applications respectively. The set of new applications A^t requests sets of connected objects $O^t = \{O_1^t, ..., O_x^t\}$, while existing applications A^{t-1} con-

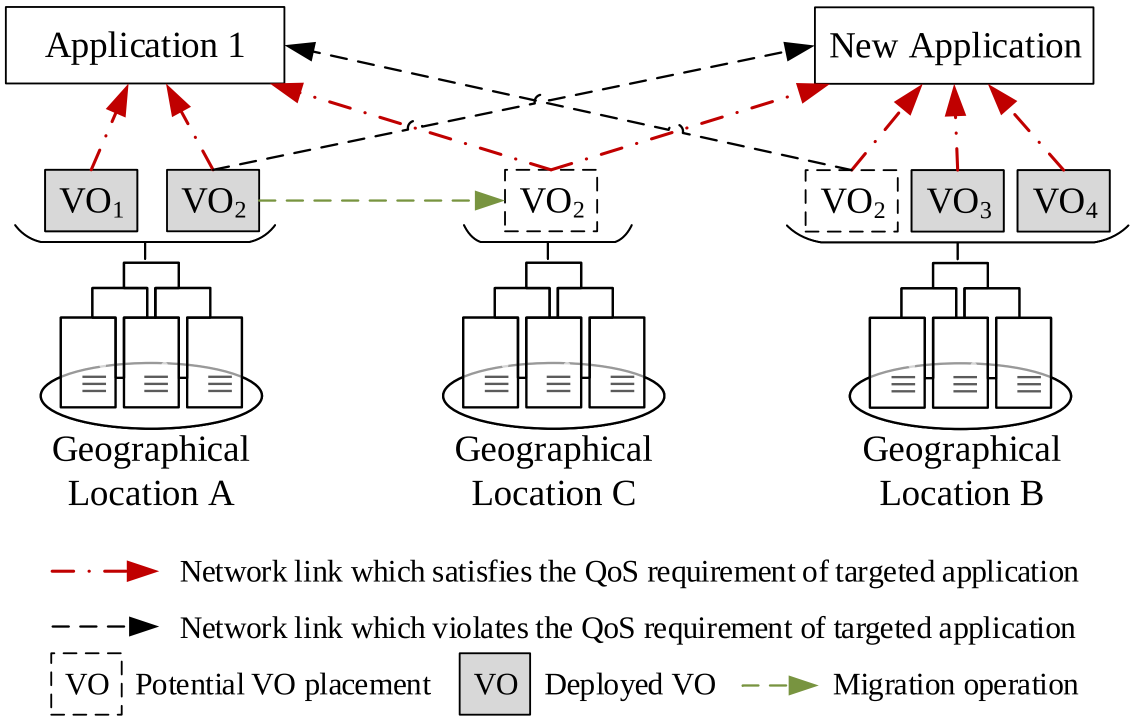

Application 1
New Application
VO$_1$
VO$_2$
VO$_2$
VO$_2$
VO$_3$
VO$_4$
Geographical Location A
Geographical Location C
Geographical Location B
Network link which satisfies the QoS requirement of targeted application
Network link which violates the QoS requirement of targeted application
VO Potential VO placement
VO Deployed VO
Migration operation

Figure 3.6: Scenario of A Migration Operation.

sume sets of connected objects $O^{t-1} = \{O_1^{t-1}, ..., O_y^{t-1}\}$. Based on these elements, we propose the Algorithm 3.2 to compute the set of clusters C and their requirements.

3.5.2 Domain Variable Definition

CLUSTER-DYNAMIC$(A^t, O^t, A^{t-1}, O^{t-1})$ produces a finite number of clusters $C = \{C_1, ..., C_m\}$. The notation of clusters is similar to Section 3.4. Hence, each cluster C_i has a set of requirements R_i containing a geographical location g_i and a set of served applications $\Delta_i \subset R_i$. Furthermore, a cluster C_i generated an average transmission rate λ_i^c. However, unlike the static model, connected objects in clusters might have corresponding VOs assigned to different data centres. Therefore, we define $\alpha_{i,j}^{t-1}$ as the fraction of connected objects belonging to the cluster C_i and have corresponding VOs in data centre d_j at time $t-1$. It describes the previous placement of shared VOs in the infrastructure. Similarly, we introduce the variable $\alpha_{i,j}^t$ which represents the distribution of VOs after orchestrating the new set of applications A^t. $\alpha_{i,j}^t$ is related to $\alpha_{i,j}^{t-1}$ as follows:

$$\alpha_{i,j}^t = \alpha_{i,j}^{t-1} + r_{i,j}^{t-1} \quad \forall i : 1 \to m, \ \forall j : 1 \to p \tag{3.18}$$

with:

$$\alpha_{i,j}^t \geq 0 \forall i : 1 \to m, \ \forall j : 1 \to p \tag{3.18a}$$

Algorithm 3.2 Clustering Algorithm for the Dynamic Optimization Model

```
CLUSTER-DYNAMIC($A^t, O^t, A^{t-1}, O^{t-1}$)
In: Set of x new applications $A^t = \{a_1^t, ..., a_x^t\}$; Sets of assigned connected objects
    for each new application $O^t = \{O_1^t, ..., O_x^t\} = \{o_1, ...\}$; Set of y existing ap-
    plications $A^{t-1} = \{a_1^{t-1}, ..., a_y^{t-1}\}$; Sets of assigned connected objects for each
    existing application $O^{t-1} = \{O_1^{t-1}, ..., O_y^{t-1}\} = \{o_1, ...\}$.
Out: Sets of clusters $C = \{C_1, ...\}$ where each cluster $C_i$ groups connected objects
    with similar requirements. Sets of requirements $R = \{R_1, ...\}$ for each cluster.

1: $C \leftarrow \emptyset; R \leftarrow \emptyset;$
2: for all selected connected objects $o_i \in O^t$ do
3:     create a temporary set $T$ containing the geographical location of $o_i$.
4:     add to $T$ all the cloud applications $A_j^t \in A^t$ and $A_k^{t-1} \in A^{t-1}$ served by $o_i$.
5:     if $\exists R_k$ such that $R_k = T$ then
6:         $C_k \leftarrow o_i$
7:     else
8:         $R \leftarrow \{T\}$ {add the new group of requirements to $R$}
9:         $C \leftarrow \{o_i\}$ {create a new cluster for new requirements $T$}
10:    end if
11: end for
12: return $C, R$
```

$$\sum_j \alpha_{i,j}^t = 1 \quad \forall j : 1 \rightarrow p \tag{3.18b}$$

The variable $r_{i,j}^{t-1}$ corresponds to the reconfiguration operations to perform on cluster i with respect to data centre j. A negative reconfiguration operation $r_{i,j}^{t-1} = -v$ means that a portion v of VOs in data centre d_j mirroring connected objects in C_i need to be migrated. A positive value $r_{i,j}^{t-1} = +v$ means that a portion v of VOs corresponding to the cluster C_i needs to be in data centre d_j. Figure 3.7 illustrates an example of reconfiguration operations between times $t-1$ and t.

We can notice that the cluster C_1 at time $t-1$ has a third partition which does not correspond to any data centre. This addition part reflects the portion of unshared VOs in the cluster before the orchestration process. We refer to this portion at cluster C_i by n_i and calculate it as follows:

$$n_i = 1 - \sum_j \alpha_{i,j}^{t-1} \quad \forall i : 1 \rightarrow m \tag{3.19}$$

Moreover, we can see that the 40% fraction was divided between data centres d_1 and d_2 with respective portions 35% and 5%. The second cluster suffered from a migration operation. 55% of connected objects previously mapped to the data centre d_2 were shifted to d_1. As we can see, the migration operation is characterized by a negative $r_{2,2}^{t-1}$ and a positive $r_{2,1}^{t-1}$. If a data centre does not satisfy a cluster's QoS requirements, all related VOs of the latter cluster in this data centre should be

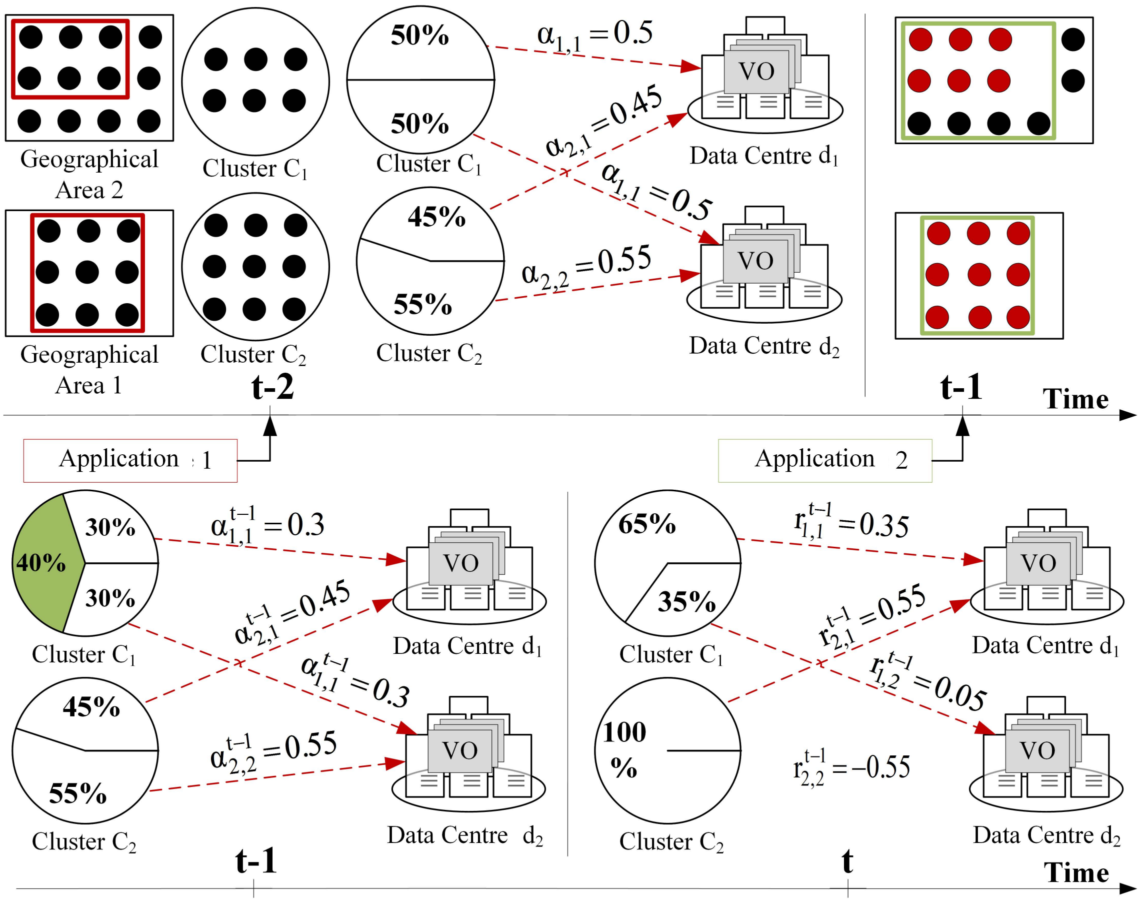


Figure 3.7: Example of A Provisioning Reconfiguration Between Times $t-1$ and t.

moved. This is mandatory because a cluster represents applications with the same QoS.

Our objective is to find the values of $r_{i,j}^{t-1}$ which minimize the cost of VOs distributed on data centres while maintaining the applications QoS requirements. Once the variables $r_{i,j}^{t-1}$ are determined, the final distribution values of VOs ($\alpha_{i,j}^{t}$) can be deduced. The constraint (3.18b) guarantees that each connected object has a VOs instance in a data centre. Also, the constraint (3.18a) stops migration operations from exceeding available VOs.

3.5.3 Placement Optimization Problem Formulation

Similarly to the domain variable, the network model variables are computed at the time t. We distinguish between (1) variables that are not affected by the VOs distribution across data centres, (2) variables that are affected only by the new deployment, and (3) variables that represent the global status of the infrastructure. Variables in category (1) remain as defined in Section 3.4 and are $\theta_{i,k}^{a}$, θ_{k}^{a}, and $q_{i,j}$. Category (2) variables are related to clusters processed during the provisioning process such as $\theta_{i,j}^{cd,t}$, $\theta_{j,k}^{da,t}$, and $\lambda_{j,k}^{da,t}$. They are adapted by replacing $\alpha_{i,j}$ with $\alpha_{i,j}^{t}$ as follows:

$$\theta_{i,j}^{cd,t} = \alpha_{i,j}^{t}\lambda_i^c = \left(\alpha_{i,j}^{t-1} + r_{i,j}^{t-1}\right)\lambda_i^c \tag{3.20}$$

$$\theta_{j,k}^{da,t} = \lambda_{j,k}^{da,t} = \beta\sum_i \delta_{i,k}\theta_{i,j}^{cd,t} = \beta\left[\sum_i \delta_{i,k}\left(\alpha_{i,j}^{t-1} + r_{i,j}^{t-1}\right)\lambda_i^c\right] \quad \forall i : 1 \to m \tag{3.21}$$

Category (3) refers to variables $\theta_j^{d,t}$ and $\lambda_j^{d,t}$ which represent the overall traffic handled by each data centre. They are expressed as a function of their value at the time $t-1$ and performed reconfiguration operations $r_{i,j}^{t-1}$. Their value might increase or decrease since the value of $r_{i,j}^{t-1}$ can be negative or positive. Accordingly, they are calculated as follows:

$$\theta_j^{d,t} = \theta_j^{d,t-1} + \sum_i r_{i,j}^{t-1}\lambda_i^c \quad \forall i : 1 \to m \tag{3.22}$$

$$\lambda_j^{d,t} = \beta\theta_j^{d,t} = \beta\theta_j^{d,t-1} + \beta\sum_i r_{i,j}^{t-1}\lambda_i^c \quad \forall i : 1 \to m \tag{3.23}$$

Since our objective is to minimize the distribution of VOs, we do not consider the migration cost of VOs from one data centre to the other. In fact, the migration is mandatory as it will affect VOs that do not comply with the requested QoS. Based on these elements, the dynamic objective function is formulated as follows:

$$\begin{aligned} \min_r \; F(r) = \omega F^{vm}(r) + \gamma F^{n}(r) \\ \text{s.t. } \omega + \gamma = 1 \end{aligned} \tag{3.24}$$

with:

$$F^{vm}(r) = \sum_j \phi_j^{vm} u_j^t \quad \forall j : 1 \to p \tag{3.24a}$$

$$\begin{aligned} F^{n}(r) = &\sum_i\sum_j q_{i,j}\phi_{i,j}^{cd}\theta_{i,j}^{cd,t} + \\ &\beta\sum_i\sum_j\sum_k q_{i,j}\phi_{j,k}^{da}\delta_{i,k}\left(\alpha_{i,j}^{t-1} + r_{i,j}^{t-1}\right)\lambda_i^c \\ &\forall i : 1 \to m,\; \forall j : 1 \to p,\; \forall k : 1 \to n \end{aligned} \tag{3.24b}$$

subjected to:

Bandwidth constraints:

$$\theta_{i,j}^{cd,t} \le b_{i,j}^{cd} \quad \forall i : 1 \to m,\; \forall j : 1 \to p \tag{3.24c}$$

$$\theta_{j,k}^{da,t} \le b_{j,k}^{da} \quad \forall j : 1 \to p,\; \forall k : 1 \to n \tag{3.24d}$$

Data centre capacity constraint:

$$\theta_j^{d,t} \leq \theta_j^{max} \quad \forall j : 1 \rightarrow p \tag{3.24e}$$

QoS related constraint:

$$\delta_{i,k}(\alpha_{i,j}^{t-1} + r_{i,j}^{t-1})(l_{i,j}^{cd} + l_{j,k}^{da} - l_k^{max}) \leq 0$$
$$\forall i : 1 \rightarrow m, \;\; \forall j : 1 \rightarrow p, \;\; \forall k : 1 \rightarrow n \tag{3.24f}$$

Domain variable constraints:

$$\sum_j (\alpha_{i,j}^{t-1} + r_{i,j}^{t-1}) = 1 \quad \forall i : 1 \rightarrow m \tag{3.24g}$$

$$r_{i,j}^{t-1} \geq -\alpha_{i,j}^{t-1} \quad \forall i : 1 \rightarrow m, \;\; \forall j : 1 \rightarrow p \tag{3.24h}$$

$$u_j^t \geq \frac{\theta_j^{d,t}}{\mu} \quad \forall j : 1 \rightarrow p \tag{3.24i}$$

$$u_j^t \leq \frac{\theta_j^{d,t}}{\mu} + 1 \quad \forall j : 1 \rightarrow p \tag{3.24j}$$

$$u_j^t \in \mathbb{Z}^+ \quad \forall j : 1 \rightarrow p \tag{3.24k}$$

Constraints (3.24g) and (3.24h) are adaptations of constraints (3.18a) and (3.18b) presented previously. They hide the term $\alpha_{i,j}^t$.

3.6 Implementation and Evaluation

In this section, we investigate the benefit of our static and dynamic approaches against a non sharing approach through simulations. Furthermore, we evaluate the role of clustering in reducing the problem size and the processing time of our algorithm with respect to requested applications and connected objects. We use JAVA CPLEX to implement our model.

3.6.1 Evaluation Settings

Table 3.2 summarizes simulations setting. We generate 10 interconnected data centres spanned across 4 geographical areas. We fix the price of a running VM to 50$ per month in all data centres. We consider no network charges between data centres in the same availability zone similarly to commercial cloud providers such as Amazon Web Services (AWS)[1]. However, we set communications costs between data centres in different geographical locations at 0.01$ per GB. We refer to the network between data centres as the intra-cloud network. Links latencies are selected randomly from the range $[100, 300]$ (milliseconds) if communicating nodes (i.e. data centres, connected objects) belong to the same region. Otherwise, we pick a latency value from $[200, 500]$.

[1]https://aws.amazon.com/

Table 3.2: Configuration Settings

Parameters	**Values**
Number of data centres	10
Data centre network capacity	Uniform in $[40000, 60000]$
Virtual machine capacity	1200
Network latency for data transfer in one region (ms)	Uniform in $[100, 300]$
Network latency for data transfer across regions (ms)	Uniform in $[200, 500]$
Network links available bandwidth (blocks per minute)	2000
Applications requested latency (ms)	Uniform in $[600, 1000]$
Number of geographical locations	4
Total number of connected objects	100000
Connected object throughput (blocks)	Uniform in $[0.1, 5]$
Price of one million blocks($)	Uniform in $[5, 8]$
Price of intra-cloud data transfers ($ per GB)	$\{0, 0.01\}$
Price of a virtual machine ($ per month)	50
First Simulation	
Number of applications	10
Connected objects per application	Uniform in $[200x, 200(x+1)]$
x	$\{0, 1, 2, 3, ..., 44\}$
Second & Third Simulations	
Number of applications	$\{4, 6, 8, ..., 50\}$
Connected objects per application	Uniform in $[200, 400]$

Cloud providers charge for connected objects data units separately. Cloud-based IoT platforms such as AWS IoT charge for the number of data blocks received from IoT devices. In particular, the AWS IoT sets the price of a million data blocks at 5$ to 8$ depending on the geographical location. One data block corresponds to 512 bytes. We use the same pricing strategy. Moreover, we simulate up to $100K$ connected objects spread across the 4 geographical areas. Their throughput is expressed in number of blocks per minute and is generated randomly from the range $[0.1, 5]$. We consider that the traffic handled by a data centre cannot exceed a maximal value selected from the range $[40000, 60000]$ (blocks per minute). Also, a VM can manage up to 1200 blocks per minute. Furthermore, the bandwidth capacity for all network links is fixed at 2000 blocks per minute.

We perform three simulations. In the first one, we fix the number of applications to 10 while varying the number of connected objects per application. At each iteration, applications select a random number of connected objects in range $[200x, 200(x+1)]$ with $x = 0 \rightarrow 44$. We use the static approach for provisioning VOs cloud resources. In the second and third simulations, we fix the range of connected objects per application to $[200, 400]$ while varying the number of applications from 4 to 50 with a step 2. We use the static approach to map VOs for all applications at each step in the second simulation, while we rely on the dynamic approach to orchestrate VOs dynamically for applications as they arrive in the final simulation. In all simulations, the latency requested by an application belong to the range $[600, 1000]$ (ms). In this context, we measure in simulations 1 and 2 the effect of the

QoS index on the provisioning process.

3.6.2 Evaluation Results

Overall Performance

As argued, clustering connected objects reduces the problem size and decreases the needed processing time to calculate a mapping solution. Figure 3.8 shows the number of clusters against the number of connected objects in simulations 1 and 2. We notice the considerable reduction in the problem size when relying on clusters to solve the provisioning problem. Furthermore, the numbers of clusters increases in the shared approach even though the number of selected IoT devices decreases. In fact, the clustering algorithm groups connected objects with the same served applications which leads to additional clusters when these IoT devices are shared. Furthermore, when the number of applications increases linearly, the number of clusters increase rapidly (Figure 3.8.a) but remains lower than connected objects amount. However, increasing the number of connected objects per application has minimal effect of clusters numbers (Figure 3.8.b).

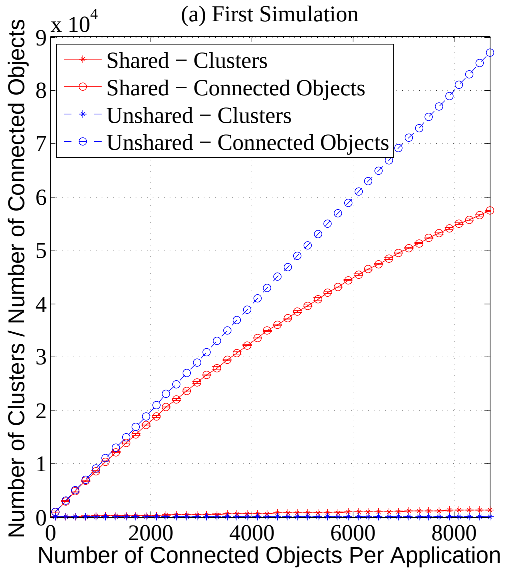
It is obvious in Figure 3.9 that the processing time of the LP solver increases proportionally to the number of applications (i.e. clusters). However, such time can be reduced using the dynamic approach. In real life scenarios, applications are not requested simultaneously. In fact, they are distributed in time and therefore are not considered collectively when mapping VOs to cloud data centres. However, as mentioned in Section 3.5, previously deployed applications need to be considered in the provisioning process when their connected objects are shared with incoming applications.

As a result, even if only one new application is being requested, multiple applications might be considered during the orchestration process. However, the number of applications considered at each orchestration process did not exceed 15 applications in our simulations which kept the performance time acceptable and slightly higher than the unshared approach (Figure 3.9). Therefore, the dynamic model provides faster mapping than the static approach. Moreover, algorithms applied for selecting connected objects can be tuned to maximize shared connected objects while minimizing the number of shared sets of connected objects between applications.

Cost

Since the number of allocated connected objects decreases with the shared approach as seen in Figure 3.8, the physical resources needed for VOs decrease. Figures 3.10 and 3.11 show the cost of VMs and bandwidth reserved to deliver IoT services for cloud applications. The costs related to the dynamic approach are similar to the static one. The main difference between both approaches is the way applications are considered during the mapping as mentioned earlier.

We notice that the cost of physical resources in the shared approach increases slowly when the number of applications or connected objects increases. In contrast, this cost increases rapidly with the unshared approach. It is worth noting that



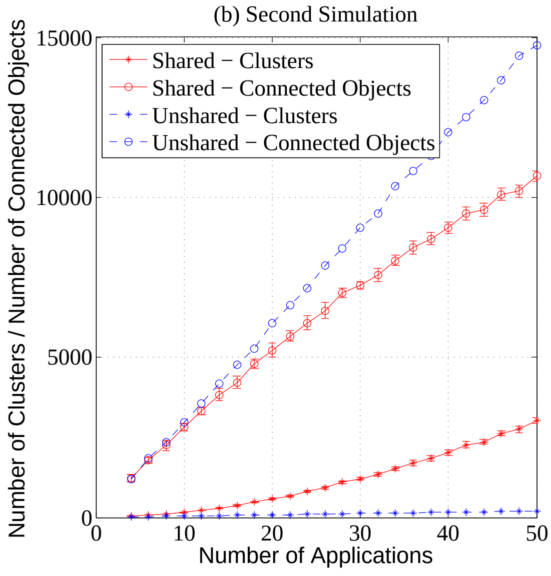


Figure 3.8: Number of Sensors and Clusters for Shared and Unshared Approaches.

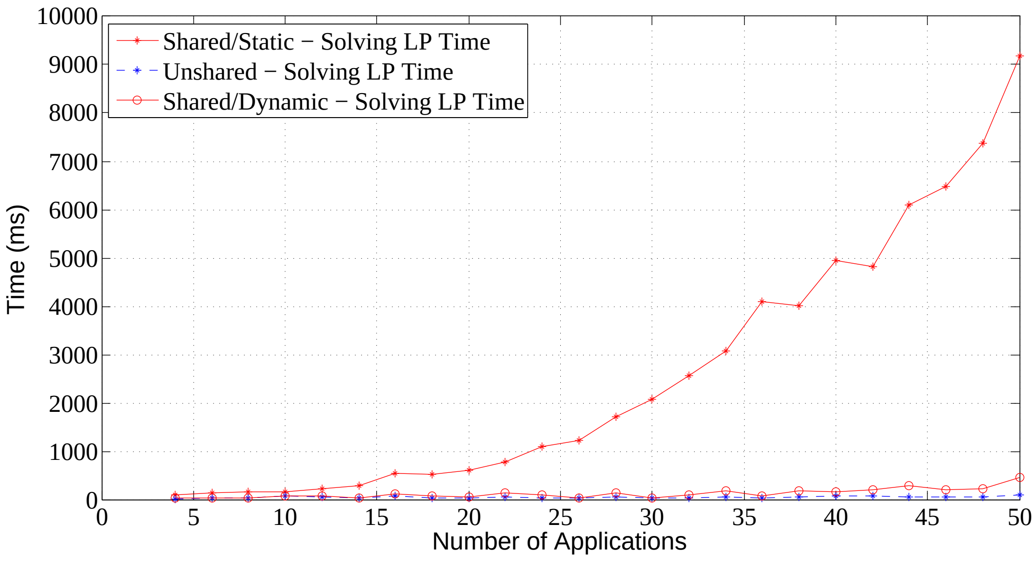


Figure 3.9: Time in (ms) for Solving the LP with Shared and Unshared Approaches.

connected objects can be linked directly to applications without using VOs in the unshared approach. This is not possible when sharing connected objects. Since IoT devices are resource-constrained, they are not able to manage requests from multiple applications at the same time. However, Figure 3.10 shows that when the number of connected objects increases significantly, the bandwidth cost exceeds the network and compute costs combined for the shared approach. Therefore, it is more profitable to use VOs to share IoT resources between applications.

Quality of Service Index

As mentioned, the QoS index provides means to balance the latency between different applications. Hence, the analytical model can satisfy applications latency constraints without prioritizing an application placement over the other. The results in Figures 3.12, 3.13, 3.14, and 3.15 show the effect of the QoS index on VOs resources cost and the mean latency experienced by cloud applications. The similar results in simulations 1 and 2 indicate that the QoS index affects the provisioning whether we increase the number of applications, the number of connected objects per application, or both.

We notice in Figures 3.12 and 3.14 that the latency perceived by applications is lower if the QoS index is included in the model. However, achieving a better deployment regarding the QoS (i.e. latency) leads to greater resources allocation cost. As a result, the cost of provisioning shared VOs can be reduced even more while satisfying the applications QoS requirements. However, such approach would penalize some applications. It is clear that without the QoS index, applications suffer from high latencies. Moreover, when the number of selected connected objects increases, the effect of the QoS index decreases. Such behavior appears better in Figures 3.13 and 3.15. The relative gain plots for resources cost and latency, with and without the QoS index, become closer to 0 when the number of applications or connected objects increases. This evolution of relative gains is due to the reduction of available resources which decreases possible candidates and therefore converges for similar solutions. Therefore, the QoS index effect on the provisioning decreases.

3.7 Conclusion

In this chapter, we proposed a mechanism to share VOs among multiple IoT applications deployed in cloud-based IoT infrastructure. We aimed to optimize network and system resources utilization as well as satisfy the applications' QoS requirements (i.e. latency). This proposition was made to leverage previous solutions that do not permit VOs to be shared among applications, which could lead to a waste of resources and increase operational cost. We formulated the problem as an optimization problem under constraints. The problem was described as a Linear Programming Problem with an objective function. The latter function formulates the relation between physical resources usage and the data transmission rate of connected objects to each application. The proposed algorithm aims to optimize the distribution of VOs and VMs across data centres taking into consideration the infrastructure's capacity and the applications' QoS requirements. We provided two models: a static model for the initial deployment of VOs and a dynamic model for reconfiguring VOs as needed to cope with incoming applications requirements. We implemented the solution and performed several simulations that show how such an approach reduces the number of deployed VMs (system resources saving) while satisfying the QoS constraints of the applications deployed in the Cloud Computing infrastructure. Our solution was calculated based on the placement of applications within the cloud data centre, and the selected connected objects. The orchestration processes of latter elements happen in separate phases and therefore prevent a holistic optimization of cloud and IoT resources. However, cloud-based IoT platforms are not able to provide the global vision of Cloud Computing and IoT infrastructures needed for such a global optimization. Hence, the need for a full integration of cloud and IoT platforms to enable a seamless deployment and optimization of IoT applications.

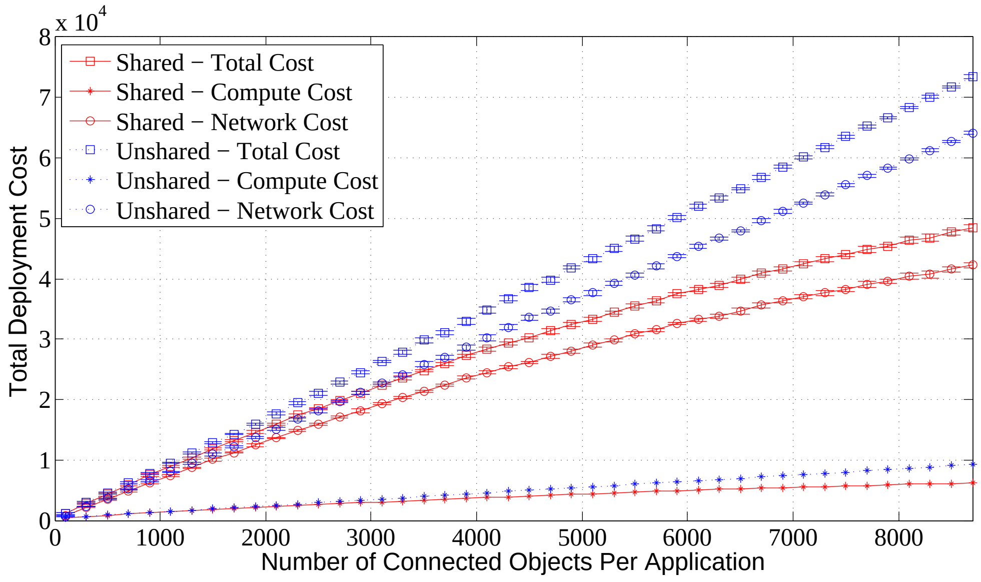


Figure 3.10: Cost of Allocated Physical Resources for Applications using Shared and Unshared Approaches (Simulation 1).

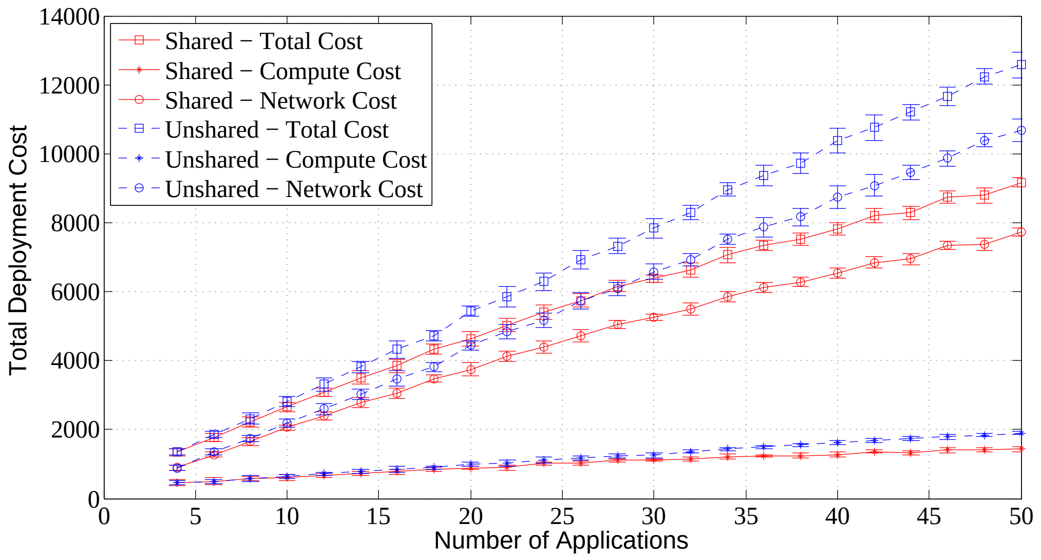


Figure 3.11: Cost of Allocated Physical Resources for Applications using Shared and Unshared Approaches (Simulation 2 & 3).

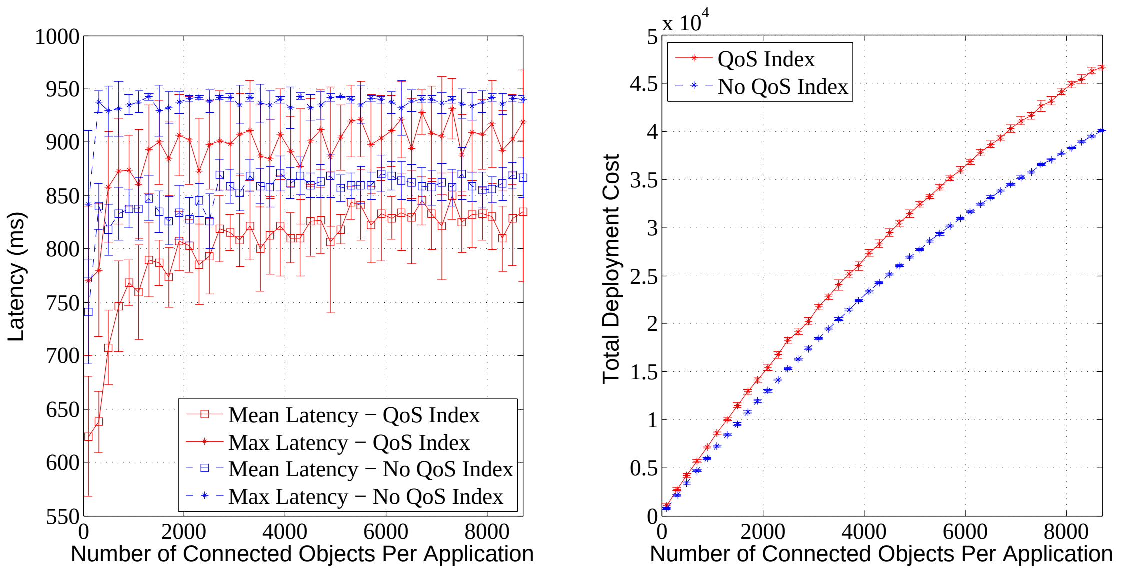


Figure 3.12: Mean Latency and Cost of Physical Resources for Applications with and without the QoS Index (Simulation 1).

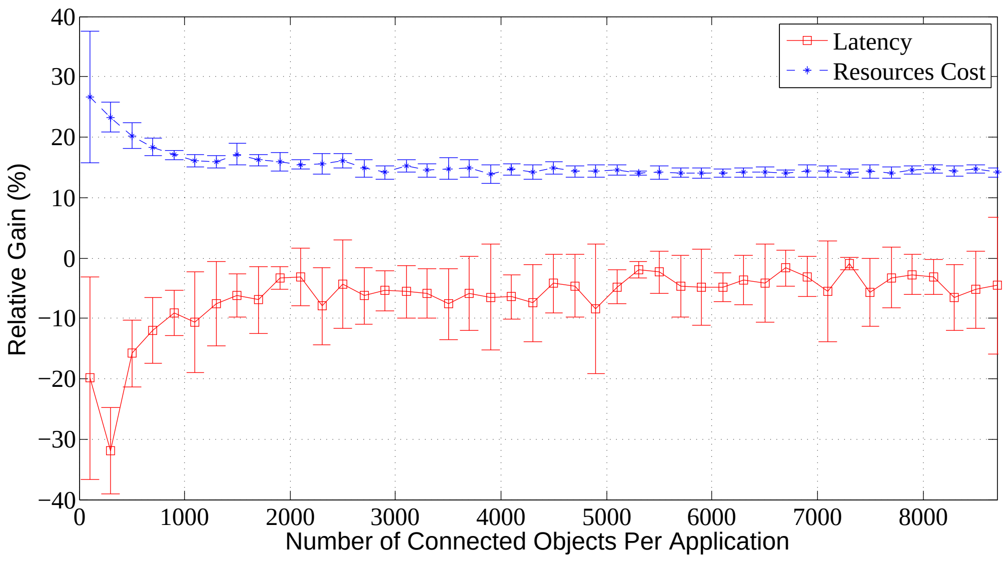


Figure 3.13: The Relative Gain in terms of Latency and Cost with and without the QoS Index (Simulation 1).

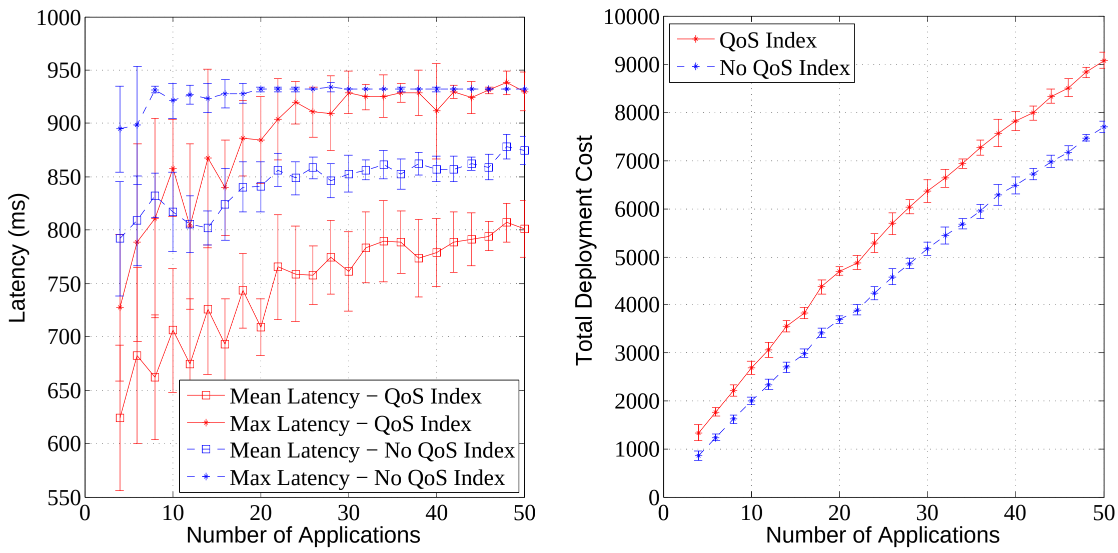


Figure 3.14: Mean Latency and Cost of Physical Resources for Applications with and without the QoS Index (Simulation 2).

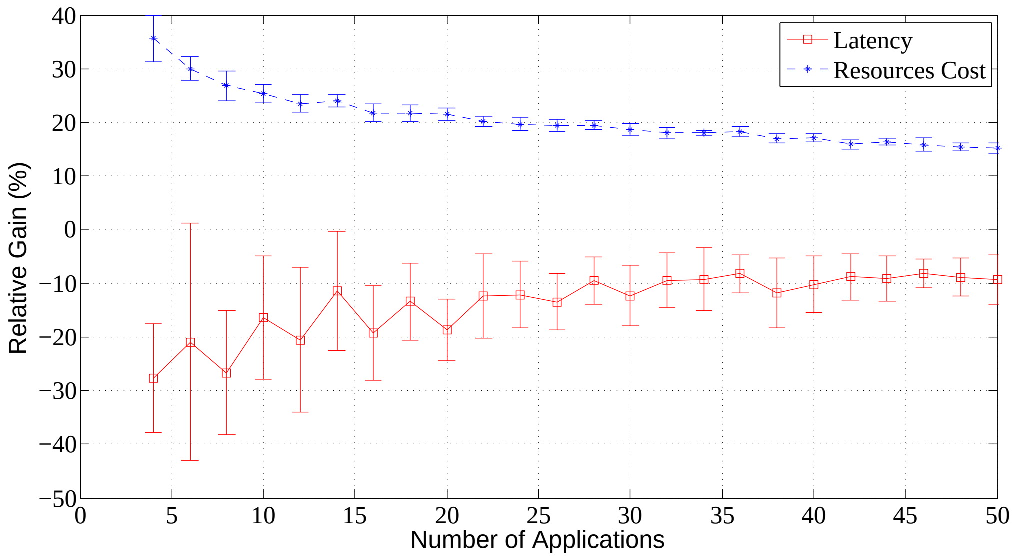


Figure 3.15: The Relative Gain in terms of Latency and Cost with and without the QoS Index (Simulation 2).

Chapter 4

Cloud of Things Resources Modelling

Contents

4.1 Introduction

Nowadays, proposed Cloud of Things (CoT) platforms [95,96] are either domain-specific or represent a simplified view of the Internet of Things (IoT) infrastructure. Furthermore, these works do not provide a resource management framework to deal with an integrated Cloud Computing and IoT infrastructure. Currently, cloud platforms and cloud-based IoT platforms are used alongside one another to manage end-to-end IoT applications provisioning and deployment. Figure 4.1.a illustrates the roles of these two types of platforms in the management and orchestration of IoT applications. Cloud-based IoT platforms abstract connected objects capabilities and offer them on-demand as cloud services; while cloud platforms provide means to manage the life cycle of end-users oriented cloud services such as data analytics services and web applications. We can then state that existing platforms do not achieve a holistic approach of the integration of Cloud Computing and the IoT and there is a need to leverage them to build the CoT vision.

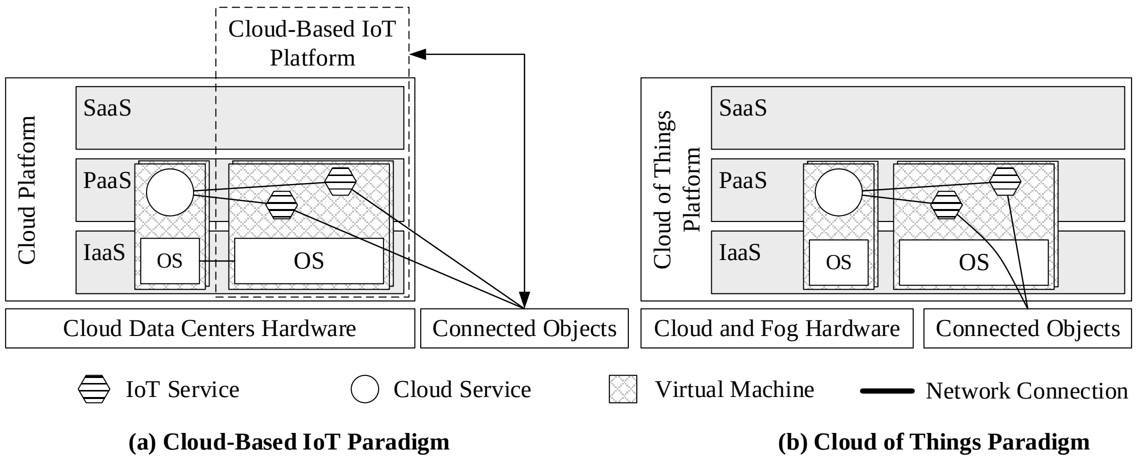


Figure 4.1: Cloud-Based IoT Platform and and Cloud of Things Architectures

As previously introduced, CoT envisions the seamless integration of Cloud Computing and IoT. It promotes a holistic management of both domains. Figure 4.1.b depicts the environment architecture of CoT platforms. Such platforms might manage in a homogeneous way resources hosted in cloud data centres as well as connected objects. They might intervene at all Cloud Computing layers namely: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Moreover, CoT platforms provide IoT related service models such as the Sensing as a Service (S^2aaS) [74, 75] and the Sensor/Actuator as a Service (SAaaS) [80] might as part of the PaaS and SaaS. The CoT platforms should be able to orchestrate in one stage requests involving both Cloud Computing and IoT resources. Such integrated orchestration mechanism will permit a global optimization of underlying infrastructures resources and eventually the introduction of new types of services. However, existing Cloud Computing and IoT management platforms are not able to describe such an integrated CoT resource model [21]. Also, they prevent a one stage provisioning mechanism for CoT requests. Therefore, there is a clear need to leverage existing resource models to design a new one that is able to support the CoT vision and its service provisioning models for the infrastructure (i.e. IaaS), the development environment (i.e. PaaS), and the CoT applications (i.e. SaaS). It is worth noting that the SaaS and PaaS models are similar. Their difference will only be in the scope of the control each model provides over deployed applications. This chapter will then focus on the proposition of a new model for CoT IaaS and PaaS.

The chapter is organized as follows. First, we present and discuss existing approaches for modelling Cloud Computing and the IoT in Section 4.2. We compare existing models and select a suitable design pattern for the foundation of our CoT model in Section 4.3. In the following section 4.4, we study the CoT infrastructure requirements and extend the core model accordingly. We provide scenarios to show the capability of our proposed design to describe a CoT infrastructure and perform resource provisioning in one stage IaaS level orchestration. Afterwards, we detail possible data delivery methods at the PaaS layer in Section 4.5. In particular, we present a model that is capable of (1) capturing different possible data delivery configurations, and (2) mapping a CoT request graph considering the different configurations. Finally, we conclude the chapter with an overview outlining key aspects of our contribution.

4.2 Existing Models and Standards

To the best of our knowledge, there exist no standards for the CoT. Current works dealing with the convergence of Cloud Computing and the IoT are significantly different from one to another [21]. They focus also on the functional and architectural aspects of the CoT. Moreover, proposed management platforms in each domain have distinct resource models and Application Programming Interfaces (APIs), even though they offer sometime similar functionality. This heterogeneity of approach that exists already in each domain has motivated several standardization initiatives to be launched mainly focusing on platform architectures and resource models. These standardized models for Cloud Computing and the IoT have been specified independently without having in mind that these two domains may converge in the future.

These models indeed do have part of the information that is required to build the CoT model since several concepts are similar. Therefore, the objective of this work was to analyse, reuse, extend, and adapt current standards to leverage them to the level that is required in CoT. Moreover, we aimed to remain aligned with existing specifications to ensure the compatibility with standardized models and definitions. However, we only considered standards which focus on the modelling of the orchestration process in Cloud Computing and the IoT. The two sections 4.2.1 and 4.2.2 detail the considered IoT and cloud standards.

4.2.1 Internet of Things Environment

IoT related specifications could be divided into three categories as depicted in Figure 4.2: (1) sensor web, (2) sensor semantic web, and (3) IoT middlewares. These groups of solutions are complementary and can work alongside one another. Basically, in a heterogeneous IoT environment, sensor web solutions provide syntactic interoperability, sensor semantic web ontologies apply more meaningful representation of sensory data, while IoT middlewares provide further management operations such as resource orchestration.

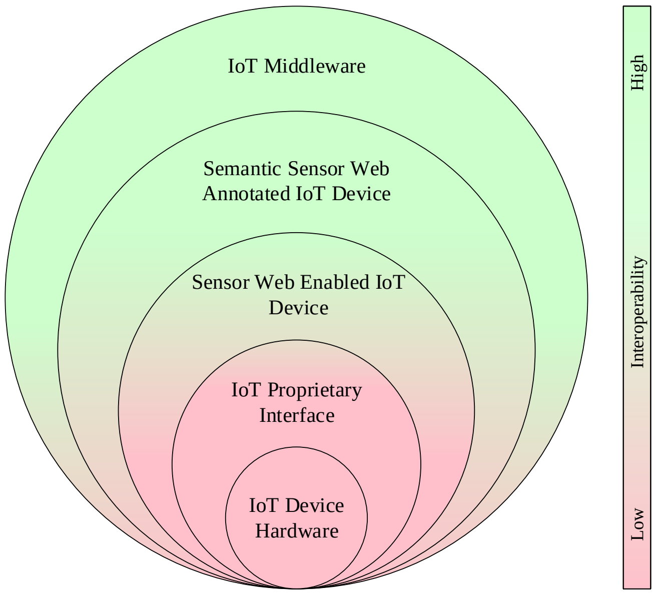

IoT Middleware
Semantic Sensor Web Annotated IoT Device
Sensor Web Enabled IoT Device
IoT Proprietary Interface
IoT Device Hardware
High
Interoperability
Low

Figure 4.2: Internet of Things Web Model Levels

Sensor Web Initiatives

The Open Geospatial Consortium (OGC) established the Sensor Web Enablement (SWE) [107, 108] suite of specifications. These standards define models to describe the IoT environment and web service interfaces to offer a high-level management layer over low-level sensing and actuating resources. The main adopted SWE framework standards are the Sensor Model Language (SensorML)[1], the Observations and Measurements (O&M)[2] model, and the Sensor Observations Service (SOS)[3] interface. The SensorML describes sensor systems capabilities, properties, measurements, and processes. It provides information which helps to manage, discover, locate, and identify sensors. Moreover, it provides an eXtensible Markup Language (XML) document model to represent this information. The O&M offers an XML schema for encoding connected objects observed and measured data. The SOS interface describes methods (e.g. **`DescribeSensor()`**, **`GetObservation()`**) for

[1]http://www.opengeospatial.org/standards/sensorml
[2]http://www.opengeospatial.org/standards/om
[3]http://www.opengeospatial.org/standards/sos

managing, discovering, requesting, filtering, and retrieving sensor descriptions and their generated data. The SOS methods use O&M and SensorML schemata to encode exchanged messages. There exist several implementations of the OGC SWE specifications [109, 110].

The OGC SWE suite is relatively complex as it supports a wide variety of sensor types ranging from simple Wireless Sensor Networks (WSNs) to earth imaging satellites. Therefore, it is heavyweight and unfit for resource-constrained IoT devices. Accordingly, the OGC presents SensorThings[4] [111] APIs which are lightweight and designed specifically for the IoT. The SensorThings APIs are based on existing SWE standards hence easily integrated with OGC services such as the SOS. In contrast to prior specifications, SensorThings is RESTful[5] and adopts the OASIS Open Data (OData) URL pattern and query options. Moreover, it uses JSON encoding instead of XML and supports the MQTT messaging protocol. Figure 4.3 illustrates the SensorThings data model.

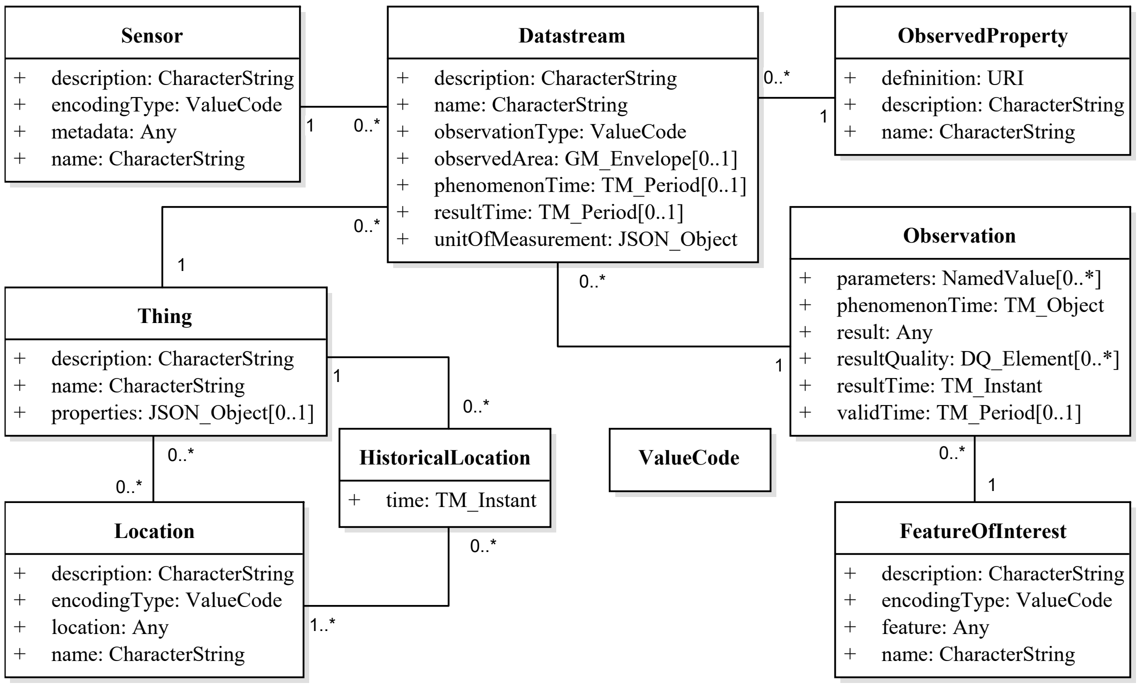

Sensor
+ description: CharacterString
+ encodingType: ValueCode
+ metadata: Any
+ name: CharacterString
Datastream
+ description: CharacterString
+ name: CharacterString
+ observationType: ValueCode
+ observedArea: GM_Envelope[0..1]
+ phenomenonTime: TM_Period[0..1]
+ resultTime: TM_Period[0..1]
+ unitOfMeasurement: JSON_Object
ObservedProperty
+ defninition: URI
+ description: CharacterString
+ name: CharacterString
Observation
+ parameters: NamedValue[0..*]
+ phenomenonTime: TM_Object
+ result: Any
+ resultQuality: DQ_Element[0..*]
+ resultTime: TM_Instant
+ validTime: TM_Period[0..1]
Thing
+ description: CharacterString
+ name: CharacterString
+ properties: JSON_Object[0..1]
HistoricalLocation
+ time: TM_Instant
ValueCode
Location
+ description: CharacterString
+ encodingType: ValueCode
+ location: Any
+ name: CharacterString
FeatureOfInterest
+ description: CharacterString
+ encodingType: ValueCode
+ feature: Any
+ name: CharacterString
1
0..*
1
0..*
1
0..*
1
0..*
1
0..*
0..*
0..*
0..*
1..*
0..*
1

Figure 4.3: SensorThings UML Diagram (Source [111])

Another model, the Web of Things (WoT), is based on work undertaken within the COMPOSE European project [78] and submitted to the World Wide Web Consortium (W3C) for processing. The WoT [112, 113] aims to seamlessly integrate connected objects to the World Wide Web (WWW) by applying on them the REpresentational State Transfer (REST) architectural style. Consequently, each IoT device becomes a uniquely identified resource via a Uniform Resource Identifier (URI) on which HTTP operations can be performed (e.g. GET, POST, PUT,

[4]https://github.com/opengeospatial/sensorthings
[5]follows the REpresentational State Transfer specifications.

DELETE). Furthermore, each resource links to other related resources. For example, an IoT device might connect to multiple sensors (e.g. temperature, humidity) or can reference its model description on a remote machine. For further interoperability, the WoT proposes a common model to describe connected objects resources on the WWW, called the Web Things Model[6]. It defines a set of APIs and syntactic messages data structures. The evrythng[7] platform supports the WoT concept.

Similarly, the Devices Profile for Web Services (DPWS) [114, 115] brings Web services to resource-limited connected objects. It is based on the W3C Web services[8] standards. However, the DPWS uses a minimal set of Web services specifications to enable messaging, discovery, description, and eventing for resource-constrained devices. Moreover, it adopts the SOAP-over-UDP binding[9] to minimize connection overhead and to use multicast addressing for discovery mechanisms. In fact, the DPWS uses the Web Services Dynamic Discovery (WS-Discovery). It applies a decentralized discovery mechanism adapted for large scale networks with resource-limited devices and no centralized registry. This mechanism employs an ad-hoc mode to leverage networks with minimal networking services (e.g. no DNS). It can switch to managed mode with multicast suppression behaviour if networking services exist to reduce network traffic. Web services joining or leaving the network send announcement messages to minimize the need for polling. Therefore, DPWS is suitable for IoT devices. Although the DPWS is lightweight and can operate on low-power devices [116], several issues still need to be addressed [37] such as its integration with the IPv6 and 6LoWPAN [117, 118]. The DPWS is used with the Constrained Application Protocol (CoAP) to minimize the communication cost on constrained devices.

Sensor Semantic Web Initiatives

Sensor web solutions provide syntactic interoperability but not a domain semantic compatibility [119]. In fact, semantic technologies add a shared domain knowledge layer that gives common meaning to data across different platforms. The work in [120] extends an SWE SOS implementation with an O&M-OWL ontology. An O&M-XML to O&M-OWL adapter integrates the ontology with other SWE components such as the SOS. The O&M ontology enables reasoning on top of sensor observations. It was further studied and aligned with upper ontologies in [121, 122]. However, the resulting ontology is limited to measurements and observations and does not represent the IoT domain knowledge.

Subsequently, the W3C Semantic Sensor Network Incubator Group (SSN-XG) produced the Semantic Sensor Network (SSN) ontology [119]. The SSN[10] ontology includes the O&M vocabulary. It is capable of describing sensors (e.g. location, type), their properties (e.g. precision, resolution, unit), and the measurements ob-

[6]http://www.w3.org/Submission/wot-model/
[7]https://evrythng.com
[8]http://www.w3.org/TR/ws-arch/
[9]http://schemas.xmlsoap.org/ws/2004/09/soap-over-udp
[10]http://purl.oclc.org/NET/ssnx/ssn

served by a sensor (e.g. values). Furthermore, the SSN enables context awareness by linking sensor observations to features of interest and events. Hence, data evolve from a sensor measurement (e.g. temperature, humidity) to a part of a broader context (e.g. soil condition) enabling a higher level of knowledge representation. Consequently, the SSN improves and simplifies sensors identification and selection processes. The OpenIoT[11] project [71] uses the SSN ontology to annotate collected data semantically and store them in an RDF cloud store managed by the LSM middleware [123]. Therefore, applications can use high-level SPARQL queries to fetch data streams originally generated as raw data by various underlying sensors.

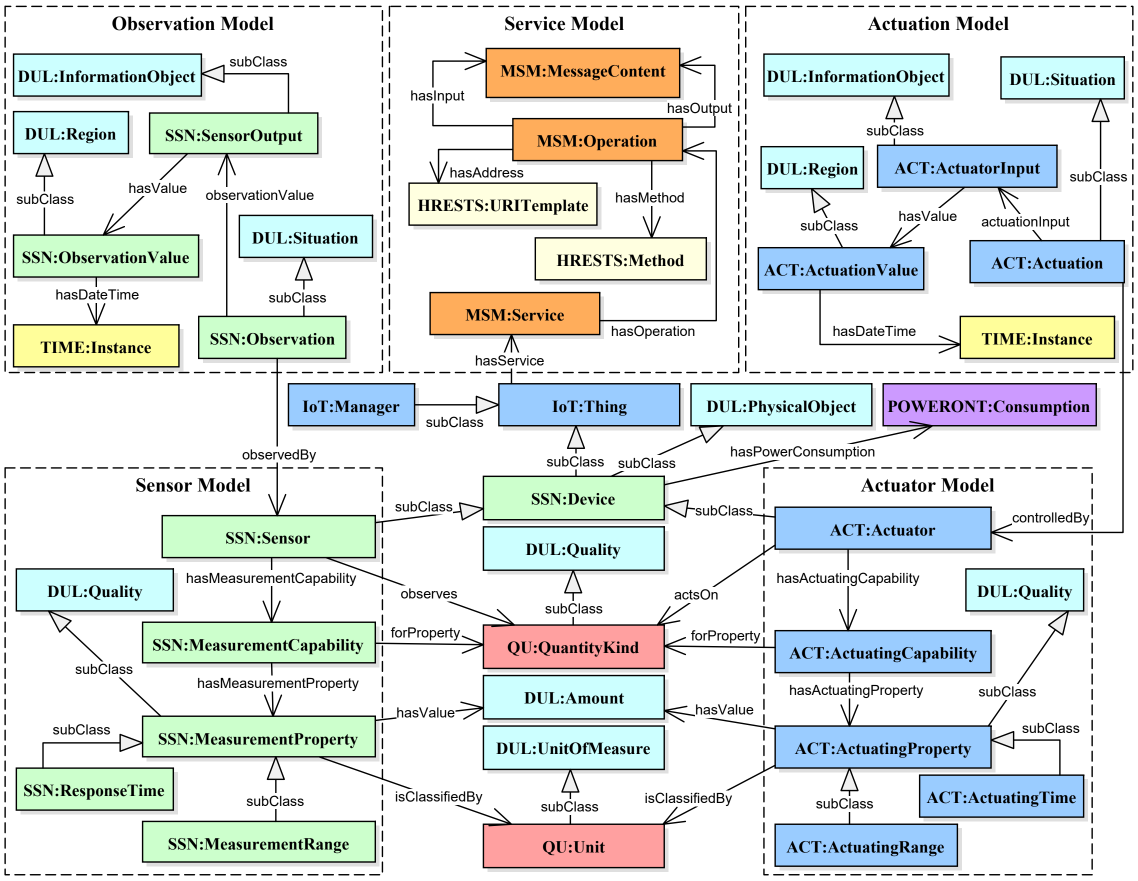


Figure 4.4: Overview of the IoT-O Architecture (Source [124])

However, the SSN does not capture the full extent of the IoT which stimulated several initiatives for creating an IoT ontology. Although many ontologies were designed for the IoT (e.g. OWL-IoT-S[12], IoT-Lite[13] [125]), not all of them apply good practices. In fact, they do not follow the Ontology Design Patterns (ODP) introduced in [126, 127], reuse existing sources, align with upper ontologies, or com-

[11]http://www.openiot.eu/
[12]http://personal.ee.surrey.ac.uk/Personal/P.Barnaghi/ontology/OWL-IoT-S.owl
[13]http://iot.ee.surrey.ac.uk/fiware/ontologies/iot-lit

ply with the Linked Open Vocabularies (LOV)[14] for IoT (LOV4IoT[15]). Authors in [128] study these aspects for several ontologies and present the Internet of Things Ontology (IoT-O)[16]. It respects previously mentioned design rules, defines some missing concepts relevant to the IoT such as **`Thing`**, **`Actuator`**, and **`Actuation`**. The IoT-O reuses the SSN, the Semantic Actuator Network (SAN)[17], the Stimulus-Sensor-Observation (SSO) [129], and several other ontologies as illustrated in Figure 4.4. Although the IoT-O is not an approved standard, it aligns with existing ontologies. Similarly, the W3C is integrating the SSN and the Sensor, Observation, Sample, and Actuator (SOSA)[18] ontologies for a better representation of the IoT.

Internet of Things Middlewares

The complex nature of IoT devices put in motion several attempts, from industry and academia, to encapsulate connected objects and offer their resources on the web. Wrapping an IoT device hides its vendor specific interface thus simplifying IoT resources discovery, management, and access. However, the large-scale nature of IoT makes it impossible to manage functional and non-functional aspects of each IoT device separately via its wrapper. Consequently, many middlewares for IoT were developed [53, 103] to aggregate connected objects, hide their heterogeneity, and optimize their utilization. IoT middlewares act as virtual gateways and offer management, querying, and configuration operations for the myriad of underlying heterogeneous connected objects. Furthermore, they implement optimization operations on managed IoT resources such as orchestration, search techniques, and data aggregation. There exist indeed no specific standard for IoT middlewares. In fact, they reuse previously presented sensor web and semantic web solutions to provide IoT resources for applications.

Cloud4Sens [50] middleware implements the previously mentioned OGC SWE suite of specifications to connect, discover, and provision sensors. It provides a uniform interface for third party applications based on the eXtensible Messaging and Presence Protocol (XMPP). Cloud4Sens defines two provisioning options for IoT consumers: (1) data-centric, and (2) device-centric. The first option provisions only sensory data streams without giving direct access to connected objects, while the second allocates sensor devices instead and enables the client to access them (pull produced data, configure properties).

The Sensor Node Plug-in System (SNPS) middleware [130] offers sensors resources following Software Oriented Architecture (SOA). It provides a Service Layer Integration (SLI) which transforms underlying sensors into web services. Accordingly, the SNPS implements a registry which holds sensors information (e.g. geographic position), deployed wrapper components, and other provided services. As a result, the sensors composition service component of the middleware is now able to

[14]http://lov.okfn.org

[15]http://www.sensormeasurement.appspot.com/?p=ontologies

[16]http://www.irit.fr/recherches/MELODI/ontologies/IoT-O

[17]https://www.irit.fr/recherches/MELODI/ontologies/SAN

[18]http://www.w3.org/ns/sosa/

aggregate sensors based on a given request and expose them to third party applications. The SNPS implements methods for searching, retrieving, composing, and configuring sensors. It uses the SensorML and O&M specifications as data models for describing sensors and exchanging sensor observations.

The eXtended Global Sensor Network (X-GSN) middleware [131] uses system-generated wrappers based on XML sensor descriptions documents described in [132] to enable automatic connection to sensors. The X-GSN semantically annotates collected sensory data using the SSN ontology. It stores collected data in Resource Description Framework (RDF) format. Therefore, SPARQL Protocol and RDF Query Language (SPARQL) queries can be used to fetch desired data streams based on sensors properties (e.g. location, type, precision). Similarly, the SemSOS [120] uses semantic technology with the SWE specifications suite. It extends the SOS with the O&M ontology enabling a high-level knowledge of the environment.

4.2.2 Cloud Infrastructure Management Initiatives

Cloud Computing standards[19] deal with management interfaces (e.g. CIMI), applications portability (e.g. TOSCA[20]), virtual appliances packaging (e.g. OVF[21]), and many others. In our work, we focus on standards providing a comprehensive view of Cloud Computing resources. Therefore, we consider management interfaces specifications. They specify entity-relationship models for manageable resources in the cloud. Also, these specifications provide standardized APIs and protocols to manage these resources as depicted in Figure 4.5. In this perspective, we study current cloud resource management interfaces. We identify the Cloud Infrastructure Management Interface (CIMI), the Open Cloud Computing Interface (OCCI), and the Cloud Application Management for Platforms (CAMP).

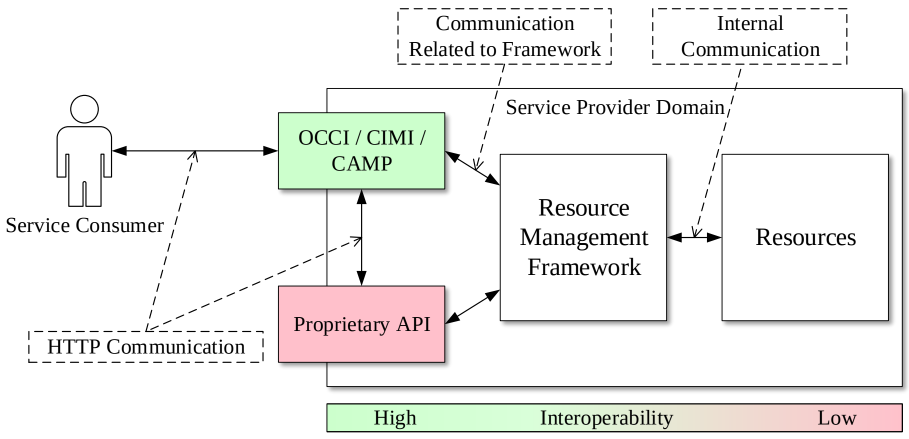


Figure 4.5: CIMI, OCCI, and CAMP Role in Cloud Architecture

[19]http://cloud-standards.org/
[20]https://www.oasis-open.org/committees/tosca/
[21]https://www.dmtf.org/standards/ovf

Cloud Infrastructure Management Interface

CIMI is a standard developed by the Distributed Management Task Force (DMTF) to describe the IaaS model in Cloud Computing. The CIMI specification defines a data model and a RESTful communication protocol for cloud platforms. The responsible working group is also considering a Simple Object Access Protocol (SOAP) interface. The CIMI represents the cloud infrastructure as a set of resources based on key entities managed at the IaaS layer. It defines such basic resources as machines, storage volumes, networks, and cloud environment monitoring components.

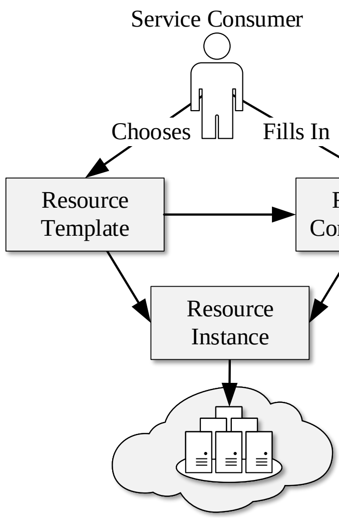


Figure 4.6: Resources Forms in CIMI (Source [133])

The CIMI model captures the steps a cloud consumer undergoes to deploy a resource in the cloud. It includes selecting the product from the cloud operator's catalog, tuning the product, and validating the process for the deployment phase. Accordingly, each resource has three forms as illustrated in Figure 4.6: (1) template resource, (2) configuration resource, and (3) resource instance. The template is the operator's predefined resources properties which belong to the operator's catalogue of offerings. A configuration resource is the client's modified version of an existing template. However, an instantiated resource is a deployed resource in the cloud infrastructure.

These resources follow a RESTful architectural style. Thus, they are uniquely addressable via URIs and can be created, retrieved, updated, and deleted using Hypertext Transfer Protocol (HTTP) methods. Furthermore, CIMI resources are interconnected via embedded URIs links. The root endpoint is the **`CloudEntityPoint`** which describes and locates the resources available in the cloud infrastructure. Moreover, the CIMI model describes an additional class named **`System`**. It represents a set of interconnected machines, storage volumes, networks, and monitoring components to form a more complex composition of deployable resources.

Open Cloud Computing Interface

The OCCI[22] is a set of specifications delivered by the Open Grid Forum (OGF)[23] and led by community contributions. The OCCI defines a RESTful protocol and APIs for resource management frameworks. An extensible and domain independent core model is at the heart of the OCCI specifications as depicted in Figure 4.7. It can be extended to describe various resources, relations, and possible actions on both. The core's essential classes are the **Kind**, **Entity**, **Action**, and **Mixin**.

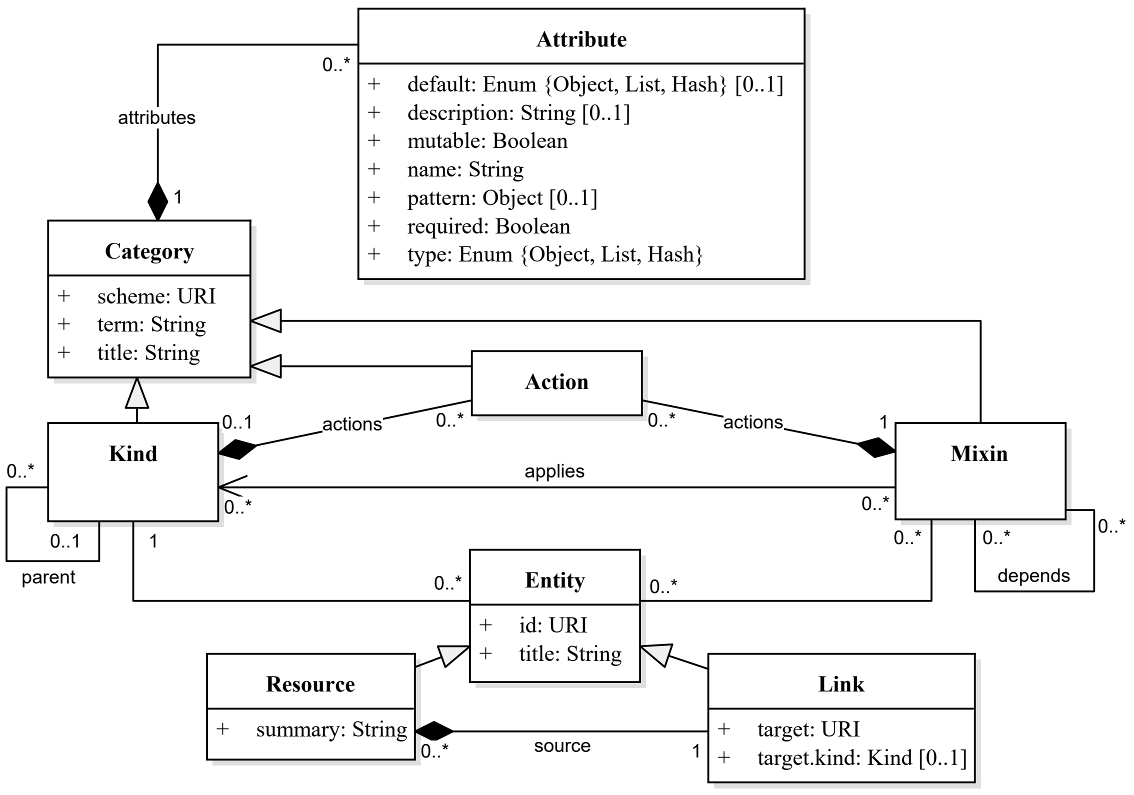


Figure 4.7: Open Cloud Computing Interface Core Model (Source [134])

The **Kind** class classifies managed resources. Each **Kind** instance represents a type of resource with related attributes and possible associated actions. The **Attribute** and **Action** classes provide details about attributes and actions respectively. For example, a Cloud Computing infrastructure has the compute **Kind**. It defines compute attributes (e.g. cores), attributes properties (e.g. name, type, required), and possible actions (e.g. start, stop).

The **Entity** is an abstract class with **Resource** and **Link** sub-classes. In contrast to **Kind** instances which specify a platform's types of resources, entity instances represent available resources within the platform. Each entity has one **Kind** instance which defines its properties and invocable actions. Moreover, **Link** instances bind together related **Resource** instances. For example, in the Cloud Computing case,

[22]http://occi-wg.org/
[23]https://www.ogf.org/ogf/doku.php

the compute `Kind` induces a `Compute` class which inherits the `Resource` class. Each `Compute` instance mirrors a Virtual Machine (VM) in the cloud which can be started or stopped. Also, the VM might be linked to other `Resource` instances using `Link` items.

The `Mixin` class is an extension mechanism which provides additional flexibility to the OCCI model. It defines attributes and actions that can be associated with `Kind` and `Entity` instances at creation time or run time. Therefore, a `Mixin` adds new capabilities for available resources without changing the predefined model.

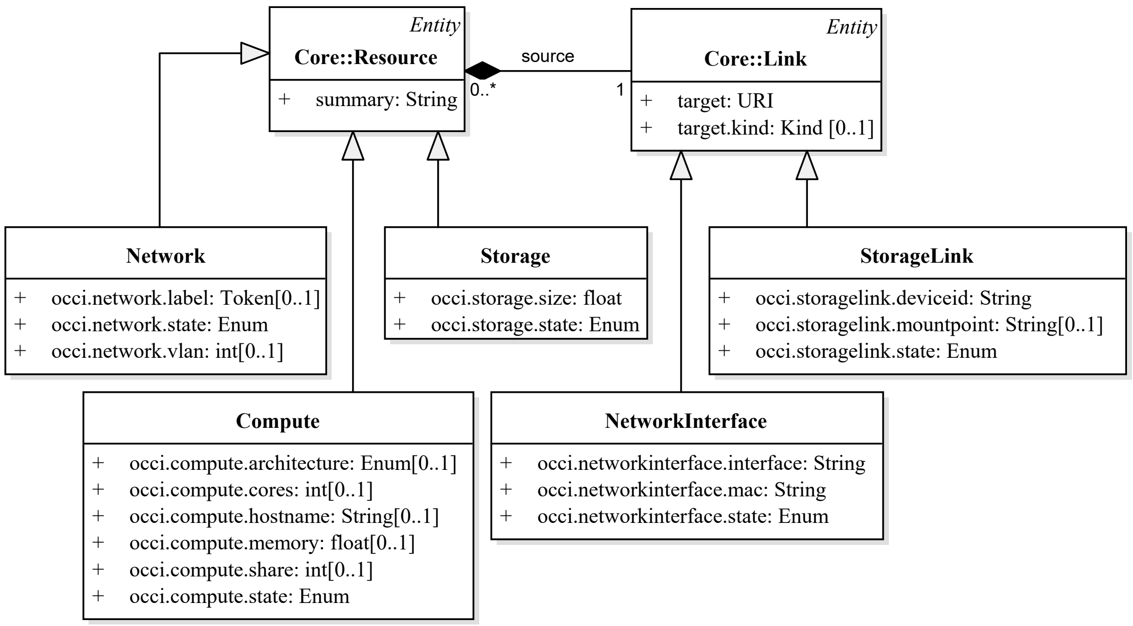

Entity
Core::Resource
+ summary: String
source
0..*
1
Entity
Core::Link
+ target: URI
+ target.kind: Kind [0..1]
Network
+ occi.network.label: Token[0..1]
+ occi.network.state: Enum
+ occi.network.vlan: int[0..1]
Storage
+ occi.storage.size: float
+ occi.storage.state: Enum
StorageLink
+ occi.storagelink.deviceid: String
+ occi.storagelink.mountpoint: String[0..1]
+ occi.storagelink.state: Enum
Compute
+ occi.compute.architecture: Enum[0..1]
+ occi.compute.cores: int[0..1]
+ occi.compute.hostname: String[0..1]
+ occi.compute.memory: float[0..1]
+ occi.compute.share: int[0..1]
+ occi.compute.state: Enum
NetworkInterface
+ occi.networkinterface.interface: String
+ occi.networkinterface.mac: String
+ occi.networkinterface.state: Enum

Figure 4.8: OCCI Infrastructure UML Representation (Source [24])

At first, the OCCI was used to create an interface for managing IaaS [24] resources for cloud platforms. Such initiative enabled IaaS level operations such as the deployment to have standard APIs. The OCCI IaaS specification promotes interoperability between distinct platforms. Several frameworks adopted it such as OpenStack, OpenNebula[24], cloudStack[25], European Grid Infrastructure (EGI)[26], and FI-WARE[27]. Figure 4.8 shows the OCCI infrastructure classes. Then, the community extended existing models to manage additional aspects of the cloud (e.g. PaaS [135], IaaS monitoring and automatic scaling [26]).

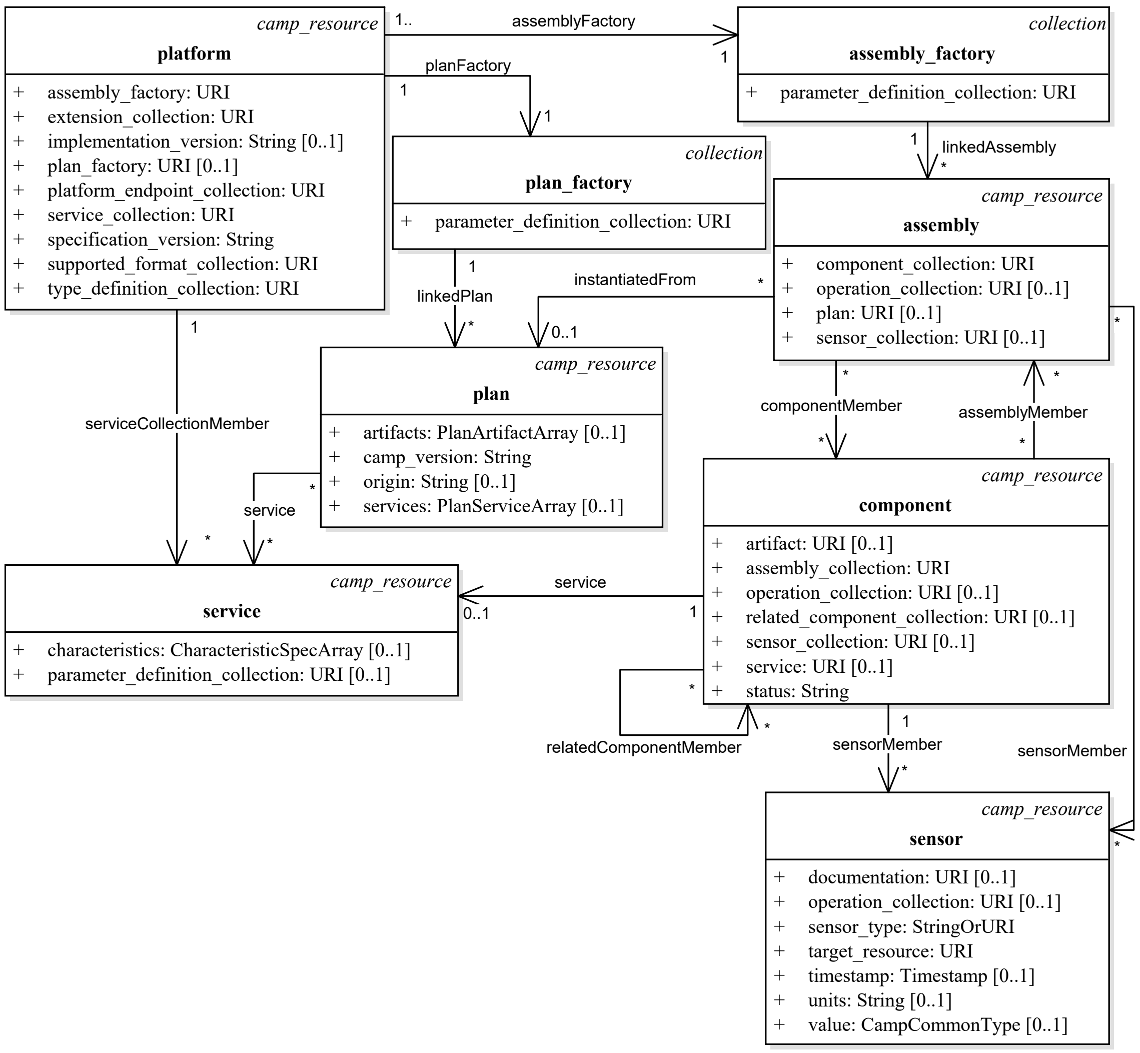
OCCI specifications are nowadays more adopted than the CIMI model [133, 136] by cloud management frameworks (e.g. OpenStack, OpenNebula) and European projects (e.g. FI-WARE, OCCIware[28] [137], EGI). Also, many libraries[29] implementing the OCCI protocol exist in numerous programming languages such as JAVA,

[24]https://opennebula.org/
[25]https://cloudstack.apache.org/
[26]https://www.egi.eu/
[27]https://www.fiware.org/
[28]http://www.occiware.org
[29]http://occi-wg.org/author/alansill/

Ruby, and Erlang.

Cloud Applications Management Initiatives

The Organization for the Advancement of Structured Information Standards (OASIS) advances the CAMP [138, 139] specification. It describes APIs, models, mechanisms, and protocols for packaging and deploying applications in the cloud. It enables a cloud provider independent interface to perform PaaS level activities such as provisioning, monitoring, and control on applications. Therefore, CAMP eliminates vendor lock-in by providing a standardized application description model. CAMP promotes interoperability among PaaS clouds by specifying artefacts and defining APIs which are necessary to manage the building, running, administration, monitoring and patching of applications in any PaaS cloud. It defines resources in a PaaS environment as depicted in Figure 4.9. Main resources are: **`platform`**, **`plan`**, **`assembly`**, **`service`**, **`component`**, and **`collection`**.



camp_resource
platform
+ assembly_factory: URI
+ extension_collection: URI
+ implementation_version: String [0..1]
+ plan_factory: URI [0..1]
+ platform_endpoint_collection: URI
+ service_collection: URI
+ specification_version: String
+ supported_format_collection: URI
+ type_definition_collection: URI
1..
assemblyFactory
1
planFactory
1
1
collection
assembly_factory
+ parameter_definition_collection: URI
collection
plan_factory
+ parameter_definition_collection: URI
1
linkedAssembly
*
camp_resource
assembly
+ component_collection: URI
+ operation_collection: URI [0..1]
+ plan: URI [0..1]
+ sensor_collection: URI [0..1]
1
linkedPlan
*
instantiatedFrom
*
0..1
camp_resource
plan
+ artifacts: PlanArtifactArray [0..1]
+ camp_version: String
+ origin: String [0..1]
+ services: PlanServiceArray [0..1]
1
serviceCollectionMember
*
service
*
*
componentMember
*
*
assemblyMember
*
*
camp_resource
component
+ artifact: URI [0..1]
+ assembly_collection: URI
+ operation_collection: URI [0..1]
+ related_component_collection: URI [0..1]
+ sensor_collection: URI [0..1]
+ service: URI [0..1]
+ status: String
camp_resource
service
+ characteristics: CharacteristicSpecArray [0..1]
+ parameter_definition_collection: URI [0..1]
service
0..1
1
*
*
relatedComponentMember
1
sensorMember
*
sensorMember
*
*
camp_resource
sensor
+ documentation: URI [0..1]
+ operation_collection: URI [0..1]
+ sensor_type: StringOrURI
+ target_resource: URI
+ timestamp: Timestamp [0..1]
+ units: String [0..1]
+ value: CampCommonType [0..1]

Figure 4.9: CAMP Basic Resources Relationships (Source [139])

They inherit the root resource **`camp_resource`**. The **`platform`** represents the primary view of the platform and reference provided services, running applications (i.e. **`assembly`** resources), supported resources. Furthermore, it provides sharing permissions across deployed applications. The **`component`** resources are dynamic atomic elements composing **`assembly`** items (e.g. database instance), while **`plan`** resources describe static artefacts required for an application (e.g. SQL script). Moreover, the **`service`** resource defines a blueprint of a **`component`** or exposes a platform-provided service. A **`collection`** instance represents a homogeneous set of any resource described in CAMP. Figure 4.9 shows the relations between these resources.

Also, **`operation`** and **`sensor`** elements are defined. Their objective is to provide means of interacting with a deployed application via the CAMP APIs. On the one hand, the **`operation`** resource describes a set of actions which can be performed on a resource. On the other hand, the **`sensor`** class represents dynamic data produced by a resource (e.g. state). It does not abstract a connected object, however, it is used to expose a provided information or a configurable property of a **`camp_resource`**.

4.3 Cloud of Things Core Model

A myriad of separate standards exists for respectively cloud and IoT environments. We cited related specifications in Section 4.2. Figure 4.10 illustrates the standards landscape. Our objective is to identify the proper standard for modelling the CoT. Once selected, we aim to adapt and extend it as necessary to enable a one-stage provisioning process in CoT for IaaS and PaaS levels. Also, we reuse concepts from other existing standards when possible to stay aligned with current specifications and vocabulary. First, we classify studied specifications based on several criteria. Table 4.1 summarizes the classification synthesis. Then, we select the appropriate standard and define the core model which will be used in the rest of the Chapter.

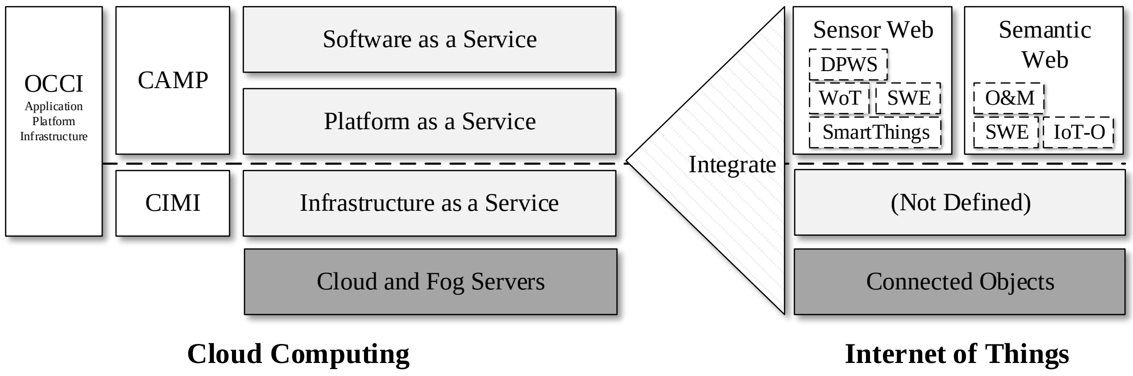


Figure 4.10: Cloud and IoT Studied Standards

4.3.1 Standards Classification

Cited standards cover Cloud Computing and the IoT. They were designed for different environments with distinct properties and challenges. For example, Cloud Computing standards target the vendor lock-in problem for various service models. However, IoT specifications aim to homogenize connected objects description and semantics. Also, these standards target precise domain-dependent issues and do not provide a comprehensive overview of all the domain aspects. For instance, the CAMP specification targets application description and deployment in the PaaS. Mentioned specifications use different technologies as well, such as W3C Web Services and W3C Ontology Web Language (OWL). Moreover, they define different structures which are incompatible with one another. Hence the necessity to select one standard as the foundation of our model. Afterwards, we extend the chosen standard to cope with the CoT provisioning model requirements.

Classification Criteria

The CoT combines Cloud Computing and the IoT. Each studied standard encompasses a partial part of the CoT. Thus, we aim to identify the correlation between these standards and required factors for the CoT model. In particular, we focus on factors related to the provisioning aspect. We select the standard with the highest correlation. Also, we determine specifications which are complementary to the selected core model. The defined classification criteria are the following:

- **Resources**: We consider compute, network, storage, sensing, and actuating resources which represent Cloud Computing and IoT infrastructures offerings. They are the core elements of the provisioning process at the IaaS level. We add the feature of interest resource which shows the ability of the standard to provide a high-level representation of data. It allows users to request connected objects using their context. Furthermore, we include the component resource. It implies that the standard adopts the Service Component Architecture (SCA) model and adapts it to Cloud Computing or IoT environments. We are interested in the component resource because the provisioning process deals with software components at the PaaS level.

- **Service Models**: Since the CoT inherits the cloud's characteristics, it provides resources on different service levels. Furthermore, our objective is to model the CoT environment on infrastructure and platform layers. Therefore, the provisioning of resources in the CoT requires a separation of service models. We consider the IaaS and PaaS levels for the classification. There exist other IoT related service models such as the S^2aaS model [22, 74, 75, 80, 105]. However, they belong to the broader PaaS level. Hence, we do not include them.

- **Extensibility**: The selected base standard will be extended to model the CoT. Therefore we define an extensibility property which indicates how much the standard is flexible and extensible. It is low (L) if the standard is rigid and any extension requires major modifications to the model's structure. An average (A) value represents standards that are not hard to expand but require several extensions and considerable effort to achieve the CoT model. A high (H) extensibility refers to standards which are easily extended, require no modifications to their core model, and already cover a large part of the CoT aspects. Thus, they minimize the effort needed to model the CoT.

- **Graph Modelling**: Orchestration mechanisms for network based environments rely on the graphical representation [23] of requests and their targeted substrate. The CoT falls under this category. It consists of a network of interconnected cloud data centres, fog elements, and IoT devices. Therefore, a standard which enables networks modelling facilitates describing the CoT environment and requires no further extensions.

4.3.2 Synthesis of Existing Works on Cloud and IoT Models and Standards

Sensor and semantic web models are similar. They focus on describing connected objects capabilities and represent their data in a unified manner. Most of these models represent sensing and actuating resources except SSN and O&M specifications. They model these resources on the PaaS and SaaS levels. However, they achieve their objectives with different technologies. Sensor web standards use syntactic approaches to enforce interoperability while semantic web specifications rely on technology standards defined by the W3C (e.g. RDF, OWL) to design and describe data on the web. Syntactic approaches are rigid and difficult to extend. In particular, sensor web approaches are focus on modelling data types rather than device networks. They are not flexible, and their extensibility is low. On the contrary, ontologies are extensible and able to describe complex systems. Moreover, they rely on the W3C standards to encode defined classes, properties, and relations. Hence, existing models are easily expandable. However, extending ontologies requires some effort to validate new classes and properties' alignment with existing definitions. Therefore, we consider that semantic web standards have an average level of extensibility. It is worth noting that the DPWS and the IoT-O offer the component resource.

On the cloud's side, OCCI and CIMI standards focus on the IaaS level, while the CAMP aims to model the PaaS. However, community contributions to the OCCI model has expanded its reach today all service layers. In the IaaS level, cloud specifications consider the compute, network, and storage resources. The component resource exists on higher service layers such as the PaaS. It is provided by the OCCI and the CAMP. Since the CAMP standard focuses on application description, packaging and deployment, it does not support cloud infrastructure modelling. Moreover, CIMI and CAMP standards enable a graphical representation of IaaS and PaaS resources respectively.

The OCCI core separates `Resource` and `Link` entities thus loosely coupling resources and their relations. This split makes the OCCI defined models more ex-

Table 4.1: Summary of Studied Cloud Computing and IoT Models and Standards

			Resources							**Service Models**			
			Compute	Network	Storage	Sensing	Actuating	Feature of Interest	Component	IaaS	PaaS	Graph Modeling	Extensibility
IoT	**Sensor Web**	SWE SensorML				X	X				X		(L)
		SWE O&M				X					X		(L)
		SmartThings				X	X	X			X		(L)
		WoT				X	X				X		(L)
		DPWS				X	X		X		X		(L)
	Semantic Web	SSN				X		X			X		(A)
		IoT-O				X	X	X	X		X		(A)
		SWE O&M-OWL				X					X		(A)
		SSN & SOSA				X	X	X	X		X		(A)
Cloud		OCCI	**X**	**X**	**X**				**X**	**X**	**X**		**(H)**
		CIMI	X	X	X					X		X	(L)
		CAMP							X		X	X	(L)

tensible and reusable with no major changes to their structure. Unlike the OCCI, the CIMI and the CAMP standards are more rigid with tightly coupled resources and links. Furthermore, they define complete data model structures in contrast to the OCCI which introduces the **`Mixin`** concept [133, 136]. The **`Mixin`** class enables augmenting a given data model with additional capabilities without affecting its initial structure. For example, VM images are added as **`Mixin`** objects to the OCCI infrastructure **`Compute`** instances. Such addition does not change the predefined standard but enables cloud management frameworks to link a **`Compute`** instance with an Operating System (OS) image. We consider that the OCCI has a high extensibility. It describes most of the needed resources and has specifications for all service layers. Therefore, it covers a large part of the CoT. Moreover, it is the most adopted standard for the cloud.

The OCCI provides a high extensibility in contrast to other cloud specifications and IoT related standards. Also, it is the only analysed standards which describes PaaS and IaaS service models simultaneously. Therefore, we select the OCCI specifications to model the CoT environment. However, we reuse cloud and IoT specifications when necessary to stay aligned with predefined concepts and vocabulary.

Some recent works [140, 141] extended existing OCCI models to represent the IoT. Ciuffoletti et al. [140] define an **`Aggregator`** and **`RealWorldObject`** **`Resource`**s with a **`Sensor Link`**. They consider that one or multiple sensors observe a real world object and report measured information to aggregators. They perform an orchestration scenario to show how their model can be used to deploy a CoT request in a converged cloud and IoT environments. However, their approach does not consider actuators nor represent the IaaS service model in the CoT environment. Furthermore, authors in [141] define an extension of the OCCI model to fill the gap between the cloud and the robotics world. The work focuses on developing an OCCI enabled gateway to hide underlying heterogeneous and mobile robots. Unlike our approach, they consider solely a gateway integration pattern (see Section 4.4) and define a domain specific extension.

4.4 Cloud of Things Infrastructure

Cloud infrastructure includes servers with virtualization support, power source, and high bandwidth connections. However, integrating IoT devices with cloud servers alters the infrastructure's homogeneity. Consequently, deploying IaaS level request graphs requires a more granular representation of the infrastructure. In fact, the IoT physical nodes belong to various categories (sensors, gateways, Fog nodes) and objects within the same category represent different hardware capabilities. In this section, we describe the OCCI CoT infrastructure model.

4.4.1 Network Graph Model

The provisioning process in the CoT is similar to the Virtual Network Embedding (VNE) problem described in [23]. The set of requested virtual machines, sensors, actuators, and the relations between them can be considered as the Virtual Network (VN) (i.e. CoT request graph). Furthermore, the set of cloud data centres, IoT devices, and network links can be mapped as the Substrate Network (SN) (i.e. CoT substrate graph). Hence, the OCCI CoT should enable a network graph representation of its resources.

Table 4.2: `Attribute`s Defined for the `NetworkLink` Type

Attribute	**Type**	**Mult.**	**Mutability**	**Description**
occi.networklink.bandwidth	Double	0..1	Mutable	Bit-rate of available or consumed information capacity (Mbps).
occi.networklink.hops	Integer	0..1	Mutable	Number of intermediate devices (routers) through which data must pass.
occi.networklink.latency	Double	0..1	Mutable	Delay that happens in data communication over the link.

A graph is a set of interconnected nodes. In the CoT, nodes represent cloud data centres, Fog elements, and IoT devices. Moreover, edges are network links connecting these nodes. Hence, we add **`Node`** and **`NetworkLink`** entities as shown in Figure 4.11. A **`Node`** instance uses **`Link`** items to identify its available and provided resources (e.g. **`Compute`**, **`Component`**). A **`NetworkLink`** instance connects two **`Node`** objects and provides their network connection attributes such as latency

(**`NetworkLink.latency`**) and bandwidth (**`NetworkLink.bandwidth`**). Table 4.2 details the **`NetworkLink`** type. Section 4.4.5 illustrates an infrastructure CoT graph created using the **`Node`** and **`NetworkLink`** entities.

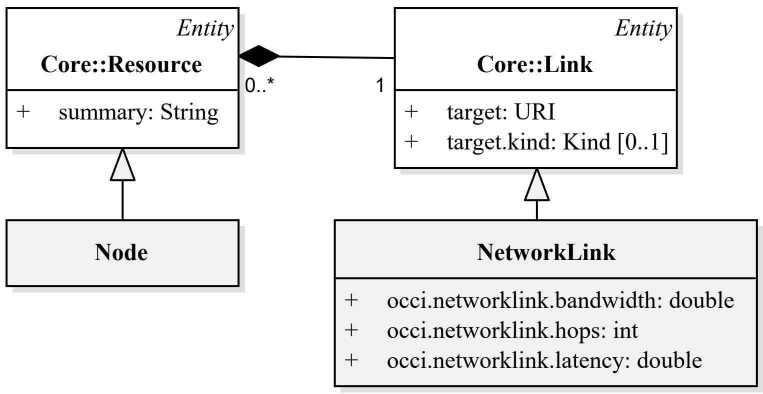

Entity
Core::Resource
+ summary: String
0..*
1
Entity
Core::Link
+ target: URI
+ target.kind: Kind [0..1]
Node
NetworkLink
+ occi.networklink.bandwidth: double
+ occi.networklink.hops: int
+ occi.networklink.latency: double

Figure 4.11: OCCI Extensions (coloured boxes) to Enable a CoT Graph Representation.

4.4.2 Sensing and Actuating

Sensors and actuators are the main functional blocks of IoT that permit to interact with the real world. Sensors observe a physical or logical object property and provide therefore sensing resources. Observable properties can be logical (e.g. system state) or physical (e.g. temperature, humidity). On the other side, actuators act on a property of an object. They do offer therefore actuating resources. In this case, the resource property must be changeable and its modification should trigger a modification in the corresponding physical object (e.g. turn light on/off) or logical one (e.g. tweet).

Hence, we added **`Sensor`** and **`Actuator`** resources to represent sensor and actuator properties in the CoT model as illustrated in Figure 4.12. The sensor entity is presented in CAMP, SSN, and SmartThings under different names but they all represent sensors. A sensor is identified by the type of observation (**`Sensor.quantityKind`**), the measurement unit (**`Sensor.unit`**), the feature of interest (**`Sensor.featureOfInterest`**), and the geographical location. In our model, we adopted the vocabulary used by the SSN ontology. These elements are the same for an actuator. Tables 4.3 and 4.4 detail the attributes of sensing and actuating resources.

A sensing or actuating resource might need to be bound to a geographical area. For example, climate temperature measurement is irrelevant if not bound to a location. However, a location is not a resource in the system and it does not exist on its own. The location is always attached to an existing resource in the model. Therefore, the added **`Location`** class inherits **`Mixin`** and can be associated with sensors or actuators. A location point is represented with **`Location.longitude`** and **`Location.latitude`** attributes as specified in Table 4.5. Moreover, the CoT includes

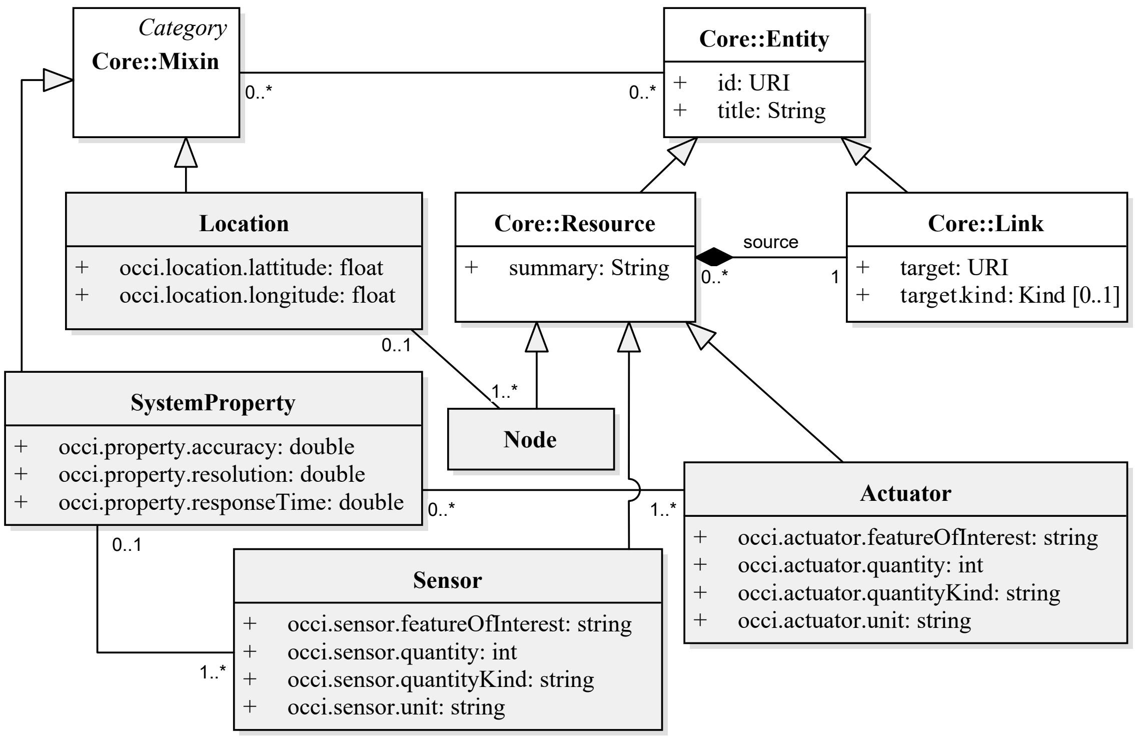

Category
Core::Mixin
0..*
0..*
Core::Entity
+ id: URI
+ title: String
Location
+ occi.location.lattitude: float
+ occi.location.longitude: float
Core::Resource
+ summary: String
source
0..*
1
Core::Link
+ target: URI
+ target.kind: Kind [0..1]
0..1
1..*
SystemProperty
+ occi.property.accuracy: double
+ occi.property.resolution: double
+ occi.property.responseTime: double
Node
0..*
1..*
Actuator
+ occi.actuator.featureOfInterest: string
+ occi.actuator.quantity: int
+ occi.actuator.quantityKind: string
+ occi.actuator.unit: string
0..1
Sensor
1..*
+ occi.sensor.featureOfInterest: string
+ occi.sensor.quantity: int
+ occi.sensor.quantityKind: string
+ occi.sensor.unit: string

Figure 4.12: Extensions of the OCCI Infrastructure for the Cloud of Things.

devices with several sensing and actuating resources such as high-powered connected objects (e.g. Raspberry PI). Consequently, we added the **`Sensor.quantity`** and **`Actuator.quantity`** to avoid representing similar sensors and actuators (i.e. with same properties) several times for the same device. This is possible since we focused on the provisioning aspects of the CoT, hence there is no need to model each sensor as a separate instance. In fact, aggregating the information decreases the network overhead since the amount of data to update a device's information is reduced.

Previously described properties represent functional requirements. However, several non-functional requirements such as accuracy, response time, and resolution can be associated with sensors and actuators. The SSN ontology considers these requirements as subclasses of the **`SystemProperty`** class. We did not model all possible non-functional requirements, however, we illustrate how additional attributes are added with a **`SystemProperty Mixin`**. Table 4.6 shows considered attributes in details.

4.4.3 Things Virtualization

IoT devices have different capabilities regarding virtualization resulting in distinct characteristics, thus the need to specify how to model this aspect in the CoT. It is possible that an IoT device does not retain any virtualization capabilities incurring no pool of available compute and storage resources. However, it still provides sensing and actuating resources through an API. Another type of possible virtualization is the network-level virtualization [47]. This kind of virtualization aims to

Table 4.3: **`Attribute`**s Defined for the **`Sensor`** Type

Attribute	**Type**	**Mult.**	**Mutability**	**Description**
occi.sensor.featureOfInterest	String	0..1	Mutable	Object whose property is being measured, estimated, or calculated
occi.sensor.quantity	Integer	1	Mutable	Number of sensor objects.
occi.sensor.quantityKind	String	1	Mutable	Kind of quantity that can be measured using defined and unrestricted units of measurement.
occi.sensor.unit	String	0..1	Mutable	Definite magnitude of a the measured quantity.

Table 4.4: **`Attribute`**s Defined for the **`Actuator`** Type

Attribute	**Type**	**Mult.**	**Mutability**	**Description**
occi.actuator.featureOfInterest	String	0..1	Mutable	Object whose property is being manipulated
occi.actuator.quantity	Integer	1	Mutable	Number of actuator objects.
occi.actuator.quantityKind	String	1	Mutable	Kind of quantity that can be manipulated.
occi.actuator.unit	String	0..1	Mutable	Definite magnitude of a quantity being manipulated.

Table 4.5: **`Attribute`**s Defined for the **`Location`** Type

Attribute	**Type**	**Mult.**	**Mutability**	**Description**
occi.location.longitude	Double	1	Mutable	Lines between the North and South Poles. They measure east-west position.
occi.location.latitude	Double	1	Mutable	Angle which ranges from $0°$ at the Equator to $90°$ (North or South) at the poles.

Table 4.6: **`Attribute`**s Defined for the **`SystemProperty`** Type

Attribute	**Type**	**Mult.**	**Mutability**	**Description**
occi.property.accuracy	Double	0..1	Mutable	Maximum difference in percentage that will exist between the actual value and measured value.
occi.property.resolution	Double	0..1	Mutable	Smallest reliable measurement that a system can make.
occi.property.responseTime	Double	0..1	Mutable	Time for a sensor to respond from no load to a step change in load.

create sub-networks by deploying an overlay network on top of connected objects or by physically clustering a group of IoT devices. The latter virtualization type is performed during the deployment process and does not affect the resource provisioning step thus it is not considered in the model.

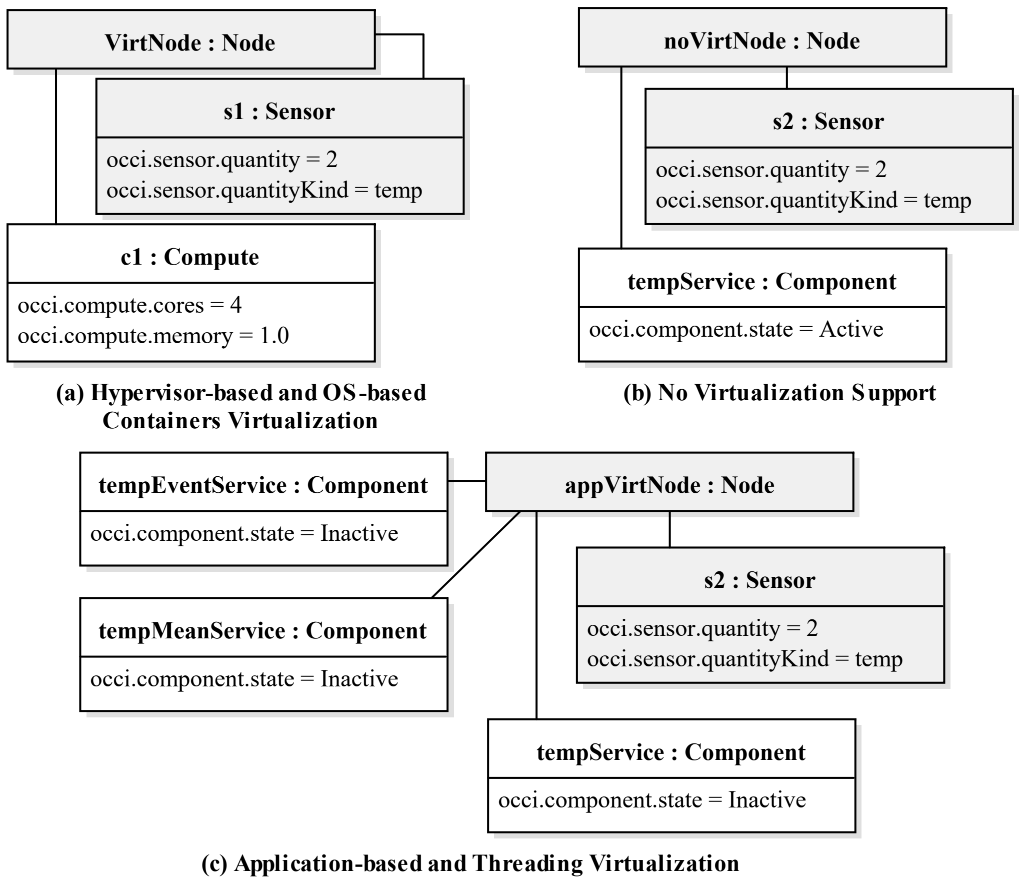

VirtNode : Node
s1 : Sensor
occi.sensor.quantity = 2
occi.sensor.quantityKind = temp
c1 : Compute
occi.compute.cores = 4
occi.compute.memory = 1.0
(a) Hypervisor-based and OS-based Containers Virtualization
noVirtNode : Node
s2 : Sensor
occi.sensor.quantity = 2
occi.sensor.quantityKind = temp
tempService : Component
occi.component.state = Active
(b) No Virtualization Support
tempEventService : Component
occi.component.state = Inactive
appVirtNode : Node
s2 : Sensor
occi.sensor.quantity = 2
occi.sensor.quantityKind = temp
tempMeanService : Component
occi.component.state = Inactive
tempService : Component
occi.component.state = Inactive
(c) Application-based and Threading Virtualization

Figure 4.13: Representation of `Node` Instances with Different Virtualization Types.

The last kind is the node-level virtualization. It defines the virtualization technology supported on top of the node's hardware. As illustrated in Figure 4.13, it consists of: (1) hypervisor-based, (2) OS-based containers, (3) application-based containers, and (4) threading virtualization. On the one hand, hypervisor-based and OS-based containers virtualization are on the infrastructure level. Nodes capable of such virtualization are able to provide compute, storage, network, sensing, and actuating resources. Figure 4.13.a shows such a node representation. Furthermore, they allow VMs or OS containers deployment over the latter resources. On the other hand, the application-based containers and threading virtualization are on the platform service level. Therefore, related nodes do not have virtualized infrastructure resources such as compute and storage. However, they can host concurrent software components (e.g. containers, threads). Hence, they are linked to **`Component`** instances. The **`Component`** class is defined in the OCCI platform specification as shown in Figure 4.17.

It's worth noting that the main difference between nodes with no virtualization support, and nodes with application-based virtualization is the possibility to deploy software components in this node. A node with no virtualization support has no compute or storage resources therefore it should be linked in the model to only one **`Component`** instance which is already deployed. The latter **`Component`** instance offers sensing and actuating resources via a web interface. Figure 4.13.b shows a node with no virtualization support. In contrast, a node with application-based virtualization has multiple associated software components which can be deployed on-demand as illustrated in Figure 4.13.c. Even though **`Component`** instances are visible at the PaaS level, they can also be offered in the IaaS layer. Anyhow, in the latter case, **`Component`** instances are not shared between different applications, and provide full control over the allocated sensing and actuating resources (e.g. configuration settings).

4.4.4 Things Integration Patterns

Integration patterns represent different ways IoT devices connect to the Internet and get accessed by third party applications. There exist three integration patterns: (1) direct connectivity, (2) gateway-based connectivity, and (3) cloud-based connectivity. Figure 4.14 depicts these integration patterns. Direct connectivity means that the IoT device has an Internet Protocol (IP) address and is able to communicate directly over the Internet. The device might be a high-powered connected object with compute, storage, sensing and actuating resources (e.g. Raspberry PI) as illustrated in Figure 4.14.a. It might be also a constrained device offering sensing and actuating resources via an HTTP or WS interface with no compute resources.

Gateway-based connectivity represents connected objects in a non-IP network (e.g. IEEE 802.15.4). These IoT devices lie behind a gateway which performs protocol translation between the Internet and its internal network. Attached connected objects might be uniquely identified and accessible from the Internet (Figure 4.14.b) or hidden behind the gateway (Figure 4.14.c). In the first case, we represented them individually in the infrastructure (e.g. **`gatewayBasedSensor1`** and **`gatewayBasedSensor2`** in Figure 4.14.c). In the second case, there is no need to model them independently. Therefore, we represented the gateway as the association of attached connected objects (e.g. **`s4`** in Figure 4.14.b). It describes all the sensing and actuating resources provided by abstracted IoT devices. Such aggregation of available resources reduces the number of nodes within the infrastructure. Likewise, the cloud-based connectivity consists of a virtual gateway in the cloud such as IoT middlewares. Its representation is identical to gateway-based connectivity in both cases.

4.4.5 Scenarios

The OCCI CoT infrastructure model enables representing CoT IaaS requests and substrates as a network graphs. Also, it facilitates mapping incoming requests on available resources during the provisioning process. We illustrate two scenarios to show the ability of the proposed model to describe the CoT.

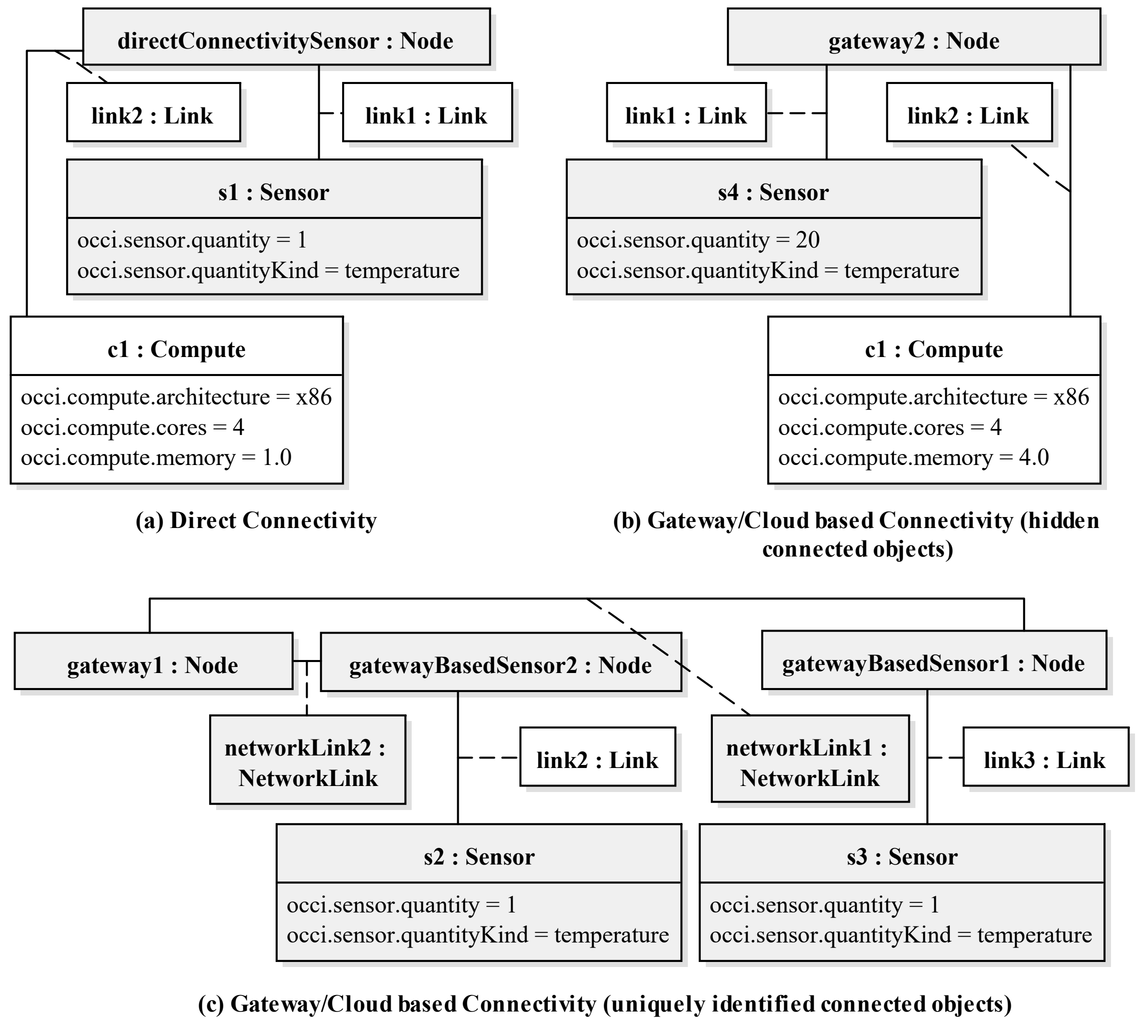

directConnectivitySensor : Node
link2 : Link
link1 : Link
s1 : Sensor
occi.sensor.quantity = 1
occi.sensor.quantityKind = temperature
c1 : Compute
occi.compute.architecture = x86
occi.compute.cores = 4
occi.compute.memory = 1.0
(a) Direct Connectivity
gateway2 : Node
link1 : Link
link2 : Link
s4 : Sensor
occi.sensor.quantity = 20
occi.sensor.quantityKind = temperature
c1 : Compute
occi.compute.architecture = x86
occi.compute.cores = 4
occi.compute.memory = 4.0
(b) Gateway/Cloud based Connectivity (hidden connected objects)
gateway1 : Node
gatewayBasedSensor2 : Node
gatewayBasedSensor1 : Node
networkLink2 : NetworkLink
link2 : Link
networkLink1 : NetworkLink
link3 : Link
s2 : Sensor
occi.sensor.quantity = 1
occi.sensor.quantityKind = temperature
s3 : Sensor
occi.sensor.quantity = 1
occi.sensor.quantityKind = temperature
(c) Gateway/Cloud based Connectivity (uniquely identified connected objects)

Figure 4.14: Integration Pattern Modelled using the OCCI CoT Infrastructure.

Resources Description

We define only a substrate graph for the demonstration, however, creating a CoT request graph is similar. The main difference is that a request graph represents needed resources, while a substrate graph describes available resources. Figure 4.15 illustrates the needed OCCI instances and their relations for representing a CoT substrate. The described infrastructure graph corresponds to two physical nodes **`infra_n1:Node`** and **`infra_n2:Node`**. The first one is a Raspberry PI with four cores and 1 GB of RAM. The other one is a cloud server. The Raspberry PI has 10 connected temperature sensors, while the cloud server has a 1000 GB storage unit. We can associate nodes with the `Location Mixin` to extend their attributes. Furthermore, both physical nodes are connected with a 100 Gbps network link having a latency of 20 ms.

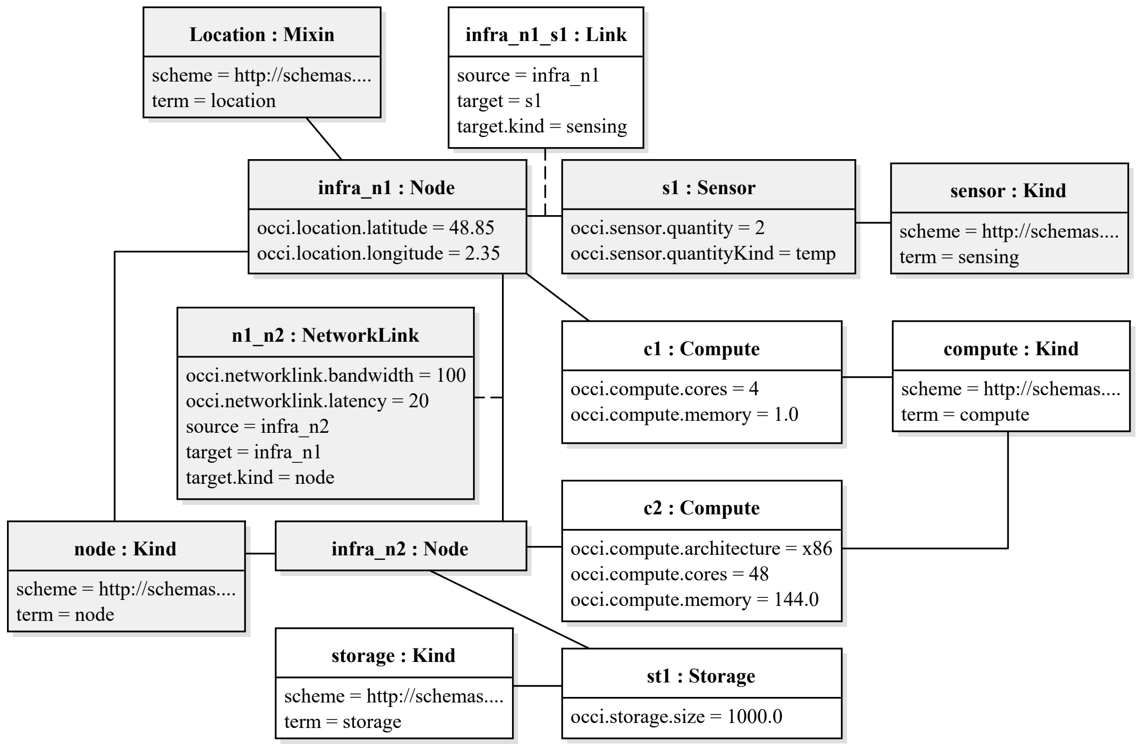


Figure 4.15: Cloud of Things Substrate Graph Description with OCCI Infrastructure Extended Model.

Mapping Process

Moreover, the proposed OCCI CoT model enables mapping requests onto a CoT infrastructure. In order to simplify the mapping example in Figure 4.16, we represented graphically the OCCI CoT instances and their relations. We give a brief description for nodes as well. It includes their **`Node.id`** attribute and attached infrastructure resources. Moreover, **`NetworkLink`** instances are presented as inter-nodes connectors.

In the mapping process, candidate hosts are identified for each requested node based on infrastructure resources. We noticed that nodes requiring sensing and actuation resources are mapped towards IoT devices and gateways in the infrastructure. However, nodes demanding more computational power are mapped to cloud servers. Furthermore, the request node C (i.e. **`requestDataAgg1`**) has an additional resource constrained connected object (i.e **`infraDataSource2`**) candidate. Consequently, we achieved a one-stage mapping. In fact, Cloud Computing and IoT requests are combined, and they can be mapped to available hosts in both domains infrastructures simultaneously.

Furthermore, the gateway (i.e. **`infraGateway1`**) is exposing the sensors resources collectively. This model is similar to a WoT enabled gateway with temperature and humidity sensors. As a result, we minimized the problem size. Otherwise, request nodes A and B would have been mapped to additional candidate hosts. In large-scale IoT infrastructures, such aggregation of available resources reduces considerably the

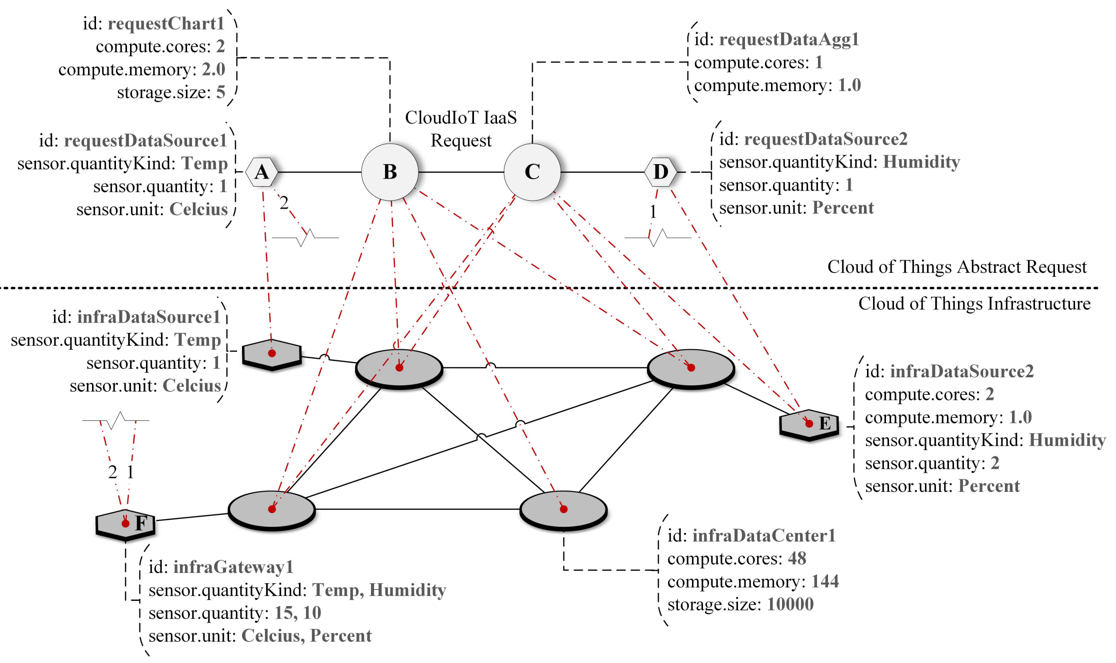


Figure 4.16: IaaS Mapping Process using the defined OCCI Infrastructure Model.

problem size and the solution computation time. After the mapping, a selection process picks the best candidates to host a given CoT request. This stage of the provisioning process is studied in Chapter 4.

4.5 Cloud of Things Platform

A PaaS level request consists of interconnected atomic components. In a cloud environment, the provisioning process identifies appropriate hosts based on components and infrastructure resources availability. Also, it decides whether software elements mapped to the same potential host should be deployed in a single or multiple VMs. Afterwards, the provisioning process identifies the best candidate host and deployment configuration (i.e. install multiple components on one or several VMs). Anyhow, a CoT application at the PaaS layer requires data streams from connected objects towards some of its components. Even though different IoT devices deliver similar resources, they host device-specific software with proprietary APIs. Also, connected objects can be accessed individually or collectively via IoT middlewares. Therefore, additional aspects such as data delivery methods, IoT devices selection, components compatibility check, and many others should be considered during the CoT PaaS provisioning process. In this section, we describe the OCCI CoT platform model which enables such operations.

4.5.1 Cloud of Things Deployment Options

The heterogeneous nature of IoT devices leads to distinct architecture specifications for different IoT systems and use cases. Consequently, different deployment

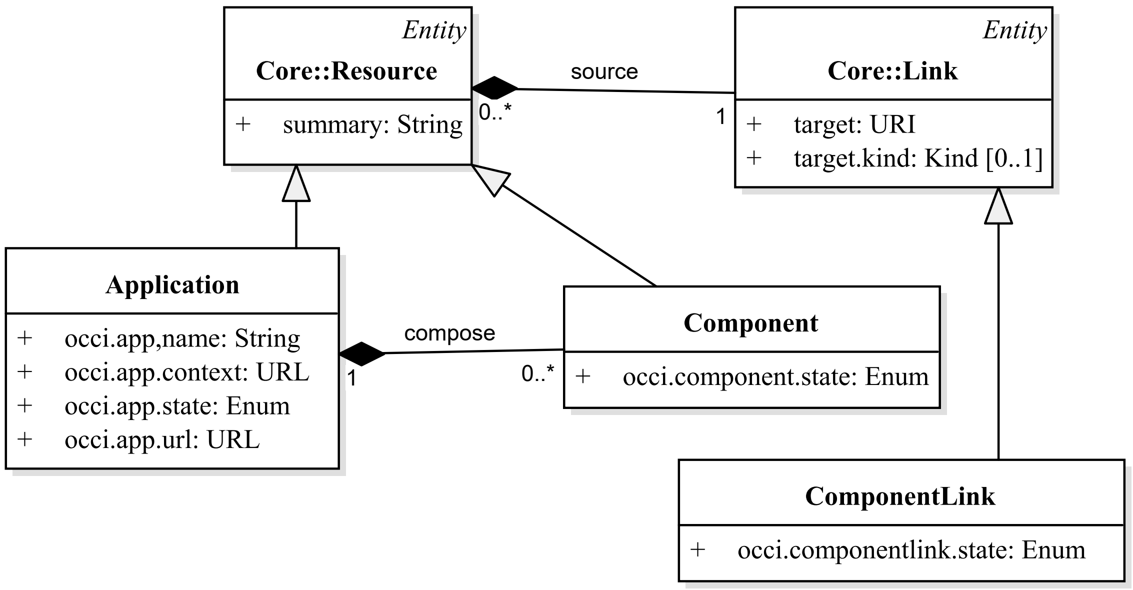


Figure 4.17: OCCI Platform UML Representation

models were introduced to link IoT devices and applications. The diverse approaches motivated several academic and industrial projects [142] to define reference models for the IoT and offer generic guidelines for implementing IoT platforms. Although these reference models differ in some aspects, their deployment models are similar. However, the Internet of Things Architecture (IoT-A) project [94] provides a more thorough and abstract representation in contrast to industrial approaches [143, 144] which focus more on business features. Furthermore, the IoT-A Reference Model (IoT-A ARM) is adopted by several European projects (e.g. FI-WARE [93], OpenIoT [71]) and illustrates a reverse mapping to validate its ability to model existing standards (ETSI M2M [145], EPCgobal[30]) and concrete architectures.

IoT-A Deployment Configurations

IoT devices have various manufacturers and offer distinct capabilities. Hence, consuming IoT devices requires deploying hardware-dependent software components to access their sensing and actuation capabilities. The IoT-A ARM defines these components as device-specific (i.e. `DeviceComponent`) which act on physical entities. However, `DeviceComponent`s are bound to particular devices and specific hardware architecture and expose a proprietary interface. Therefore, they do not allow easily interoperability. Two types of device-specific components could exist: (1) network (i.e. `NetworkComponent`) and (2) on-device (i.e. `OnDeviceComponent`). Network components are usually deployed on remote hardware such as gateways to access IoT devices (e.g. WSN). Such approach is needed whenever the IoT devices cannot be accessed individually and directly via a public network. Whilst, on-device

[30]http://www.gs1.org/epcglobal

software components are deployed locally and are bound to the devices' capabilities.

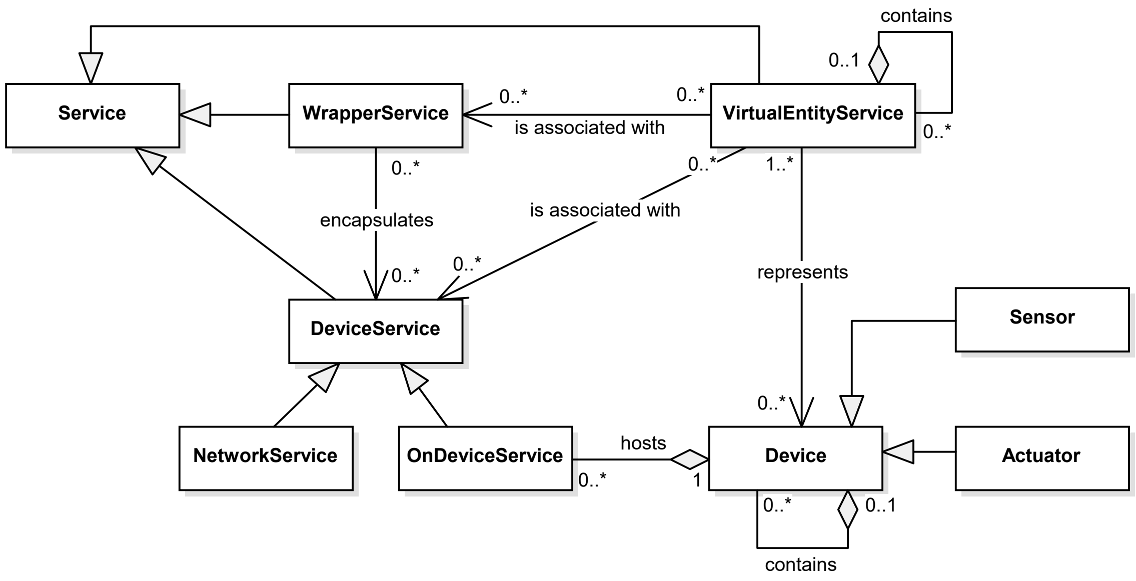


Figure 4.18: UML Representation of the IoT Environment Services Based on the IoT-A.

The shortcomings of device-specific components are overcome by using IoT wrappers that encapsulate their proprietary functionality and provide a standard API. The wrappers also aim to manage the related non-functional aspects as seen in Section 4.2.1. In fact, IoT wrapper components can be hosted on powerful hardware (e.g. capable devices, fog nodes, cloud servers), therefore, they can perform additional processing such as aggregation, filtering, access control, etc. to reduce bandwidth, energy consumption, and enhance security. Furthermore, wrappers handle non-functional aspects for device-specific components. They are divided into two categories: (1) wrapper components (**`WrapperComponent`**) and (2) virtual entity components (**`VirtualEntityComponent`**). The first category accesses IoT devices directly, connects to them, and manages quality aspects of device's resources such as dependability, resilience, security, and performance. However, virtual entity components offer higher level of abstraction and manage multiple wrapper instances or other virtual entities without connecting directly to things. They may add additional processing for underlying IoT devices such as data streams querying. Virtual entities imitate a virtual gateway for virtual IoT devices (wrapped connected objects). Therefore, **`WrapperComponent`** and **`VirtualEntityComponent`** portray cloud-based connectivity for connected objects. The difference is the multiplicity of managed sensors and the level of abstraction and management enabled by each. Figure 4.18 illustrates the relation between described services while Figure 4.19 depicts the possible deployment options for the IoT based on the IoT-A.

It's worth mentioning that since the device-specific components have proprietary APIs, establishing a connection between these components and wrappers can be challenging. However, multiple researchers have already addressed this issue and provided valuable solutions. For example, the Global Sensor Network (GSN) mid-

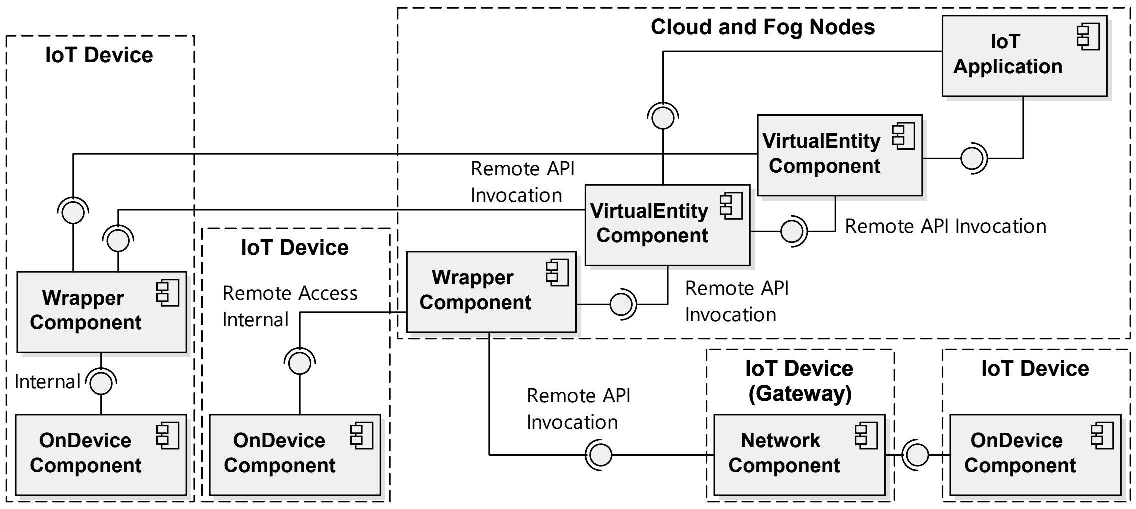


Figure 4.19: Cloud of Things Deployment Options Based on IoT-A.

dleware [132] generates device-specific wrappers relying on a devices' description file known as the Sensor Device Definitions (SDDs). Furthermore, recent specifications for constrained devices such as DPWS enable more standardized and defined interfacing with constrained devices [37].

Provisioning Deployment Configurations

Even though multiple delivery methods are referenced by the IoT-A project as illustrated in Figure 4.19, the provisioning process does not require such granularity. Mapping a CoT request graph aims to select candidates for cloud components, data delivery components, and data sources. At the PaaS layer, **`DeviceComponent`**s represent data sources, while **`VirtualEntityComponent`**s and **`WrapperComponent`**s perform data delivery operations. Moreover, **`WrapperComponent`**s are generally deployed alongside the **`VirtualEntityComponent`**s as seen in Chapter 2. For example, wrappers are instantiated within the X-GSN to abstract sensors as illustrated in Figure 4.20.

Also, CoT users have no control over infrastructure resources and the deployment architecture on the PaaS level. Consequently, IoT applications request data streams without constraints on deployed delivery methods. CoT operators are responsible for managing data delivery for cloud applications and should avoid inefficient deployment configurations. Authors in [68] show that using virtual entities to aggregate multiple data streams and distribute them among cloud applications consumes less energy than individual IoT devices access. Therefore, we used **`VirtualEntity-Component`**s to deliver data streams and did not rely on **`WrapperComponent`**s which manage devices independently.

As a result, **`WrapperComponent`**s connect solely IoT devices to **`VirtualEntityComponent`**s which relay data streams to applications. Therefore, the explicit description of **`WrapperComponent`**s in the CoT request increases the problem size and the provisioning phase complexity without affecting its decision. We defined

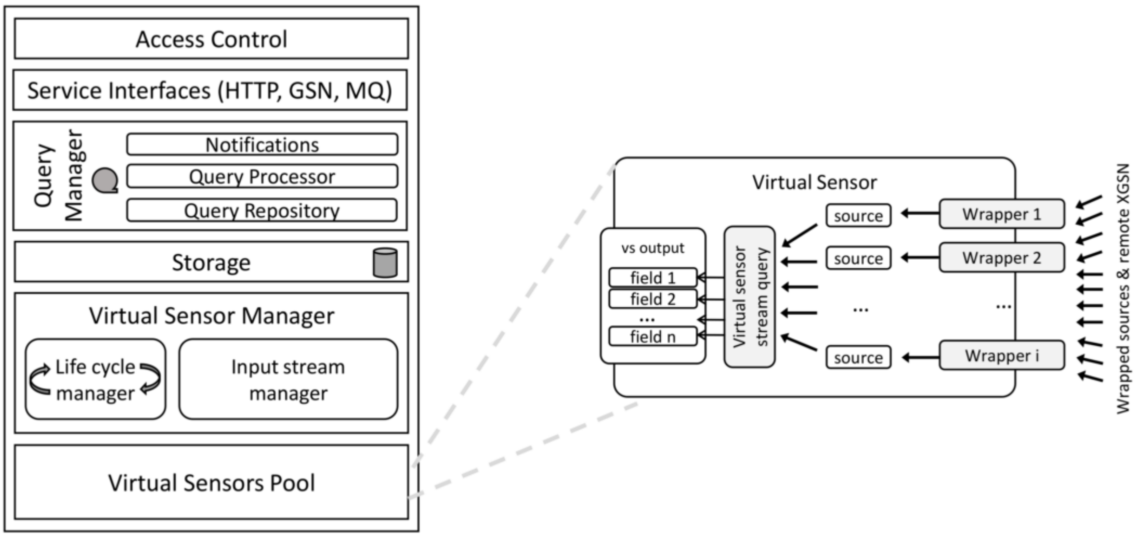


Figure 4.20: X-GSN Container Architecture. Sources [131, 146]

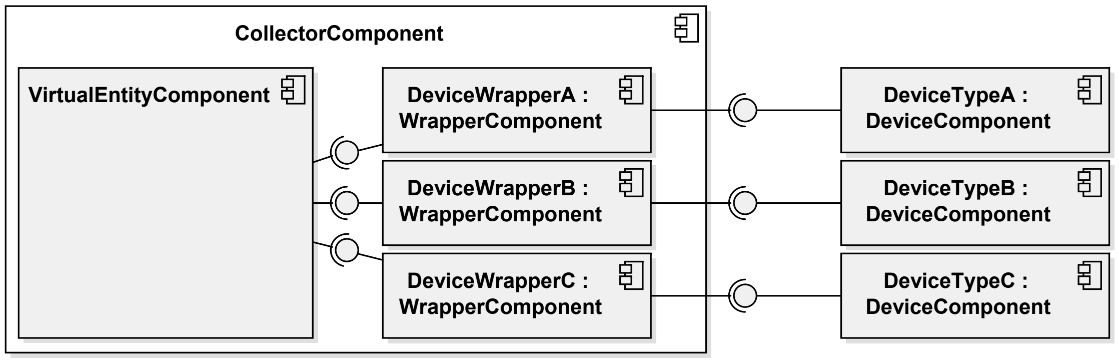

CollectorComponent
VirtualEntityComponent
DeviceWrapperA : WrapperComponent
DeviceWrapperB : WrapperComponent
DeviceWrapperC : WrapperComponent
DeviceTypeA : DeviceComponent
DeviceTypeB : DeviceComponent
DeviceTypeC : DeviceComponent

Figure 4.21: Collector Component Architecture Example

the **CollectorComponent** as illustrated in Figure 4.21. Consequently, we only represented **DeviceComponent**s and **CollectorComponent**s in the CoT request. Such representation does not affect the provisioning phase result and global cost. The **ComponentLink** indicates whether a **DeviceComponent** connects to a particular **CollectorComponent** such as X-GSN or not.

4.5.2 Data Components Sharing

In CoT, data streams should be reused to maximize resources utilization and minimize energy consumption. Several works [7, 57, 67–69, 84] show that sharing sensing and actuating resources among various applications increases usage efficiency. Hence, **CollectorComponent**s instances which already exist in the CoT environment should be reused. They result from previous deployed requests or from instances provided by third party data providers. These instances can be partially or completely reused by incoming requests based their sensing/actuating needs. Furthermore, reusing existing components speeds up the deployment process.

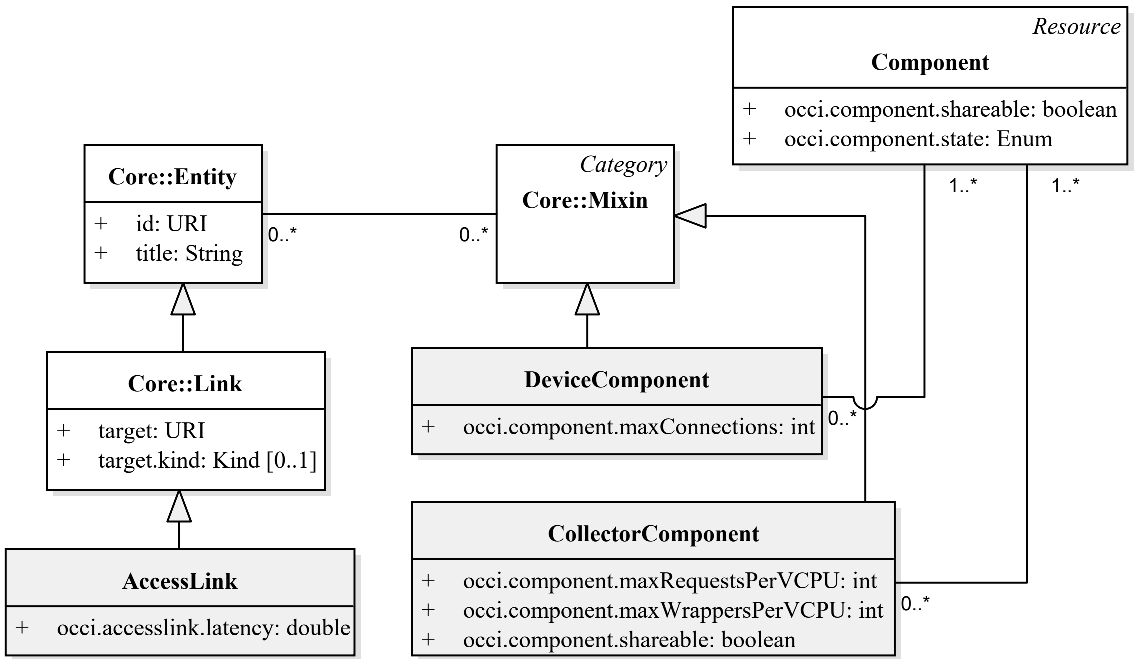

Resource
Component
+ occi.component.shareable: boolean
+ occi.component.state: Enum
Core::Entity
+ id: URI
+ title: String
Category
Core::Mixin
0..*
0..*
1..*
1..*
Core::Link
+ target: URI
+ target.kind: Kind [0..1]
DeviceComponent
+ occi.component.maxConnections: int
0..*
AccessLink
+ occi.accesslink.latency: double
CollectorComponent
+ occi.component.maxRequestsPerVCPU: int
+ occi.component.maxWrappersPerVCPU: int
+ occi.component.shareable: boolean
0..*

Figure 4.22: Extensions of the OCCI Platform for the Cloud of Things.

As a result, during the provisioning process, already deployed **`CollectorComponent`** and **`DeviceComponent`** entities should be considered. The already defined **`Component.state`** attribute specifies whether components are instantiated or not. Also, these components should be shareable among applications. However, a newly requested application might require IoT resources which are not all managed by existing **`CollectorComponent`**s. In such case, it is possible to utilize already deployed services and add missing resources to them. Therefore, the provisioning process should be aware of existing services and their maximal capacity.

We added the **`CollectorComponent Mixin`** attributes which enables to extend the capabilities of the existing **`Component`** resource in the OCCI platform. It is described in Table 4.7. The **`CollectorComponent`** includes the **`CollectorComponent.shareable`** boolean attribute to indicate whether an instance is shareable or not. Furthermore, the capacities of the component are given by **`CollectorComponent.maxRequestsPerVCPU`** and **`CollectorComponent.maxWrappersPerVCPU`**. They represent the maximal number of data streams it can collect from devices or dispatch to applications based on its allocated infrastructure resources. In some cases, a connected object might be linked to two different **`CollectorComponent`**s. Therefore, we needed to know how many connections it can accept. If it only accepts one connection, it means the connected object cannot be shared among data delivery components. We represented the **`DeviceComponent Mixin`** to describe IoT devices related **`Component`**s. It is described in Table 4.8.

Table 4.7: **`Attribute`**s Defined for the **`CollectorComponent Mixin`**

Attribute	Type	Mult.	Mutability	Description
occi.component.shareable	Boolean	0..1	Mutable	Indicates whether the component is shared among multiple applications or not.
occi.component.maxRequestsPerVCPU	Integer	1	Mutable	Maximal request rate supported per an allocated VCPU unit.
occi.component.maxWrappersPerVCPU	Integer	1	Mutable	Maximal managed connected objects wrappers per an allocated VCPU unit.

Table 4.8: **`Attribute`**s Defined for the **`DeviceComponent Mixin`**

Attribute	Type	Mult.	Mutability	Description
occi.component.maxConnections	Integer	0..1	Mutable	Maximal number of concurrent connections.

Mapping Replication

CoT request graphs describe needed IoT device components. However, if a deployed **`CollectorService`** component satisfies requested sensing and actuating resources, there is no need to redo the mapping of IoT device components. Let's consider two separate requests needing an X-GSN component delivering a temperature data stream as illustrated in 4.23. When the first request arrives, the provisioning process will search for a suitable X-GSN host and a connected device able to measure the temperature. Once the first request is satisfied, the cloud operator will have an interconnected temperature sensor and an X-GSN instance. However, when the second request is received, the orchestration process will redo the same steps even though it can directly reuse the already deployed X-GSN instance. Such operation is time and resources consuming which is inefficient. Therefore, it should not be replicated for each request.

To solve this problem, we attached in the proposed model each existing **`CollectorComponent`**s to managed sensing and actuating resources via an **`AccessLink`** entity instance. The **`AccessLink`** instance provides information about already provisioned resources. Therefore, the provisioning process checks directly if existing collector instances satisfy user request or not without scanning further existing components. In the previous example, with this change in the model, the orchestration process will be able to detect that the X-GSN instance is already attached to a temperature data stream and therefore select it directly instead of creating another instance. This not only speeds up the provisioning process but reduces the resources utilization.

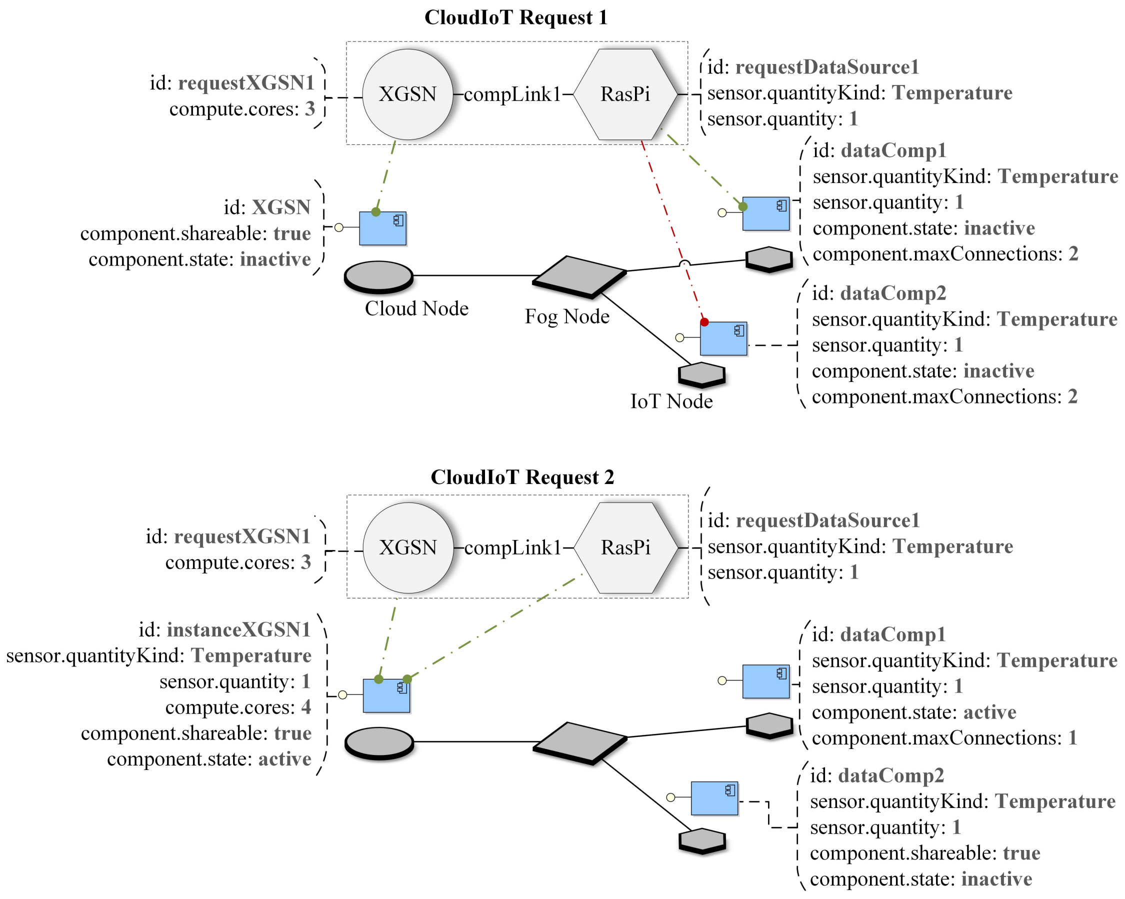

CloudIoT Request 1
id: requestXGSN1
compute.cores: 3
XGSN
compLink1
RasPi
id: requestDataSource1
sensor.quantityKind: Temperature
sensor.quantity: 1
id: dataComp1
sensor.quantityKind: Temperature
sensor.quantity: 1
component.state: inactive
component.maxConnections: 2
id: XGSN
component.shareable: true
component.state: inactive
Cloud Node
Fog Node
id: dataComp2
sensor.quantityKind: Temperature
sensor.quantity: 1
component.state: inactive
component.maxConnections: 2
IoT Node
CloudIoT Request 2
id: requestXGSN1
compute.cores: 3
XGSN
compLink1
RasPi
id: requestDataSource1
sensor.quantityKind: Temperature
sensor.quantity: 1
id: instanceXGSN1
sensor.quantityKind: Temperature
sensor.quantity: 1
compute.cores: 4
component.shareable: true
component.state: active
id: dataComp1
sensor.quantityKind: Temperature
sensor.quantity: 1
component.state: active
component.maxConnections: 1
id: dataComp2
sensor.quantityKind: Temperature
sensor.quantity: 1
component.shareable: true
component.state: inactive

Figure 4.23: Avoiding Mapping Replication Example Scenario.

4.6 Conclusion

Combining the cloud and the Internet of Things evolved over the years. At first, IoT middlewares managed data collection on premise while distant cloud components analysed gathered data on-demand. Then, IoT components were migrated to the cloud to benefit from its characteristics (e.g. elasticity). However, sensing and actuating resources were still managed separately from cloud resources. In fact, provisioning mechanisms were divided into two stages: (1) the cloud components orchestration, and (2) the IoT resources selection and integration. Such partitioning prevented a global optimization of both domains infrastructures. Therefore, the next logical evolution of such convergence is the unified management of Cloud Computing and IoT resources. The seamless integration of both domains is referred to as the Cloud of Things. In this chapter, we modelled the CoT resources on the IaaS and PaaS levels. Our objective is to enable a joint management of underlying resources. In this perspective, we studied existing standards for the cloud and the IoT to identify the most suitable one for modelling the CoT. We selected the OCCI standard due to its extensibility and comprehensive modelling of the cloud. We extended the existing OCCI model with respect to the CoT characteristics for IaaS and PaaS. Also, we showed through scenarios how the proposed model enables the management of CoT resources and allows CoT mapping processes to interpret resources description.

Chapter 5

Efficient Provisioning in the Cloud of Things

Contents

5.1 Introduction

In a partially integrated Cloud Computing and Internet of Things (IoT) environments, orchestrating an IoT application undergoes three separate stages: (1) cloud services provisioning, (2) connected objects selection, and (3) Virtual Objects (VOs) placement in cloud data centres. Even though these steps might be coordinated to achieve a more efficient provisioning, they are still executed in multiple phases which prevents a global optimization of resources. Therefore, the use of Cloud Computing and IoT resources becomes less efficient. Moreover, state of the art mechanisms for selecting connected objects are separated from the placement decisions which can be inefficient [68, 87, 103].

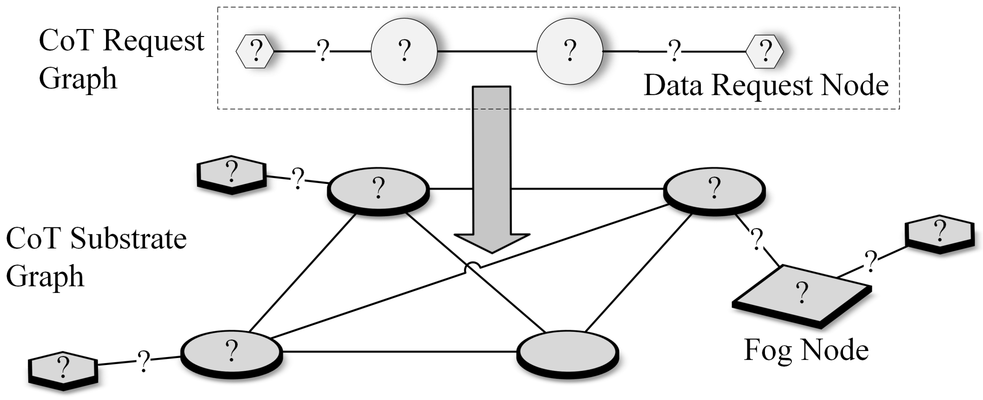


Figure 5.1: Cloud of Things Integral Mapping Scenario

The Cloud of Things (CoT) needs to support and facilitate end-to-end IoT applications delivery. However, such vision requires novel mechanisms for resource allocation and optimization. Such mechanism should consider the entire IoT application and CoT infrastructure in a single stage. Recently, several approaches were proposed to optimize bandwidth usage, energy consumption, and sensors lifetime. However, to the best of our knowledge, none of these approaches have tackled the problem of an IoT application deployment in a CoT infrastructure in a holistic manner.

In this chapter, we propose an efficient solution to address the mentioned challenge. We formulate the problem as a graph mapping problem similarly to Figure 5.1 illustration. We represent graphically the required cloud services, connected objects, and VOs. We then use a Linear Program (LP) to calculate the optimal solution which minimizes the cost of the deployment.

This chapter is organized as follows. In Section 5.2, we highlight the different challenges regarding the resource allocation problem of services/applications in a CoT infrastructure. In Section 5.3, we present the analytical model of the problem which optimizes the mapping process of the CoT request graph onto the CoT substrate. The model has two instances. The first considers infrastructure level resources while the other addresses platform level resources. In section 5.4, we present our simulation environment and discuss the obtained results. Finally, the chapter is concluded in Section 5.5.

5.2 Problem Statement

An abstract CoT request is composed of a set of nodes related to either the cloud computing environment or to the IoT environment as depicted in Figure 5.2.a. Let us consider that data request nodes indicate how many temperature sensing services the CoT customer wants to deploy, while the cloud service indicates the number of requested vCPUs to process the data. In a traditional Cloud Computing provisioning model, one request node is mapped to one substrate node that satisfies the requested resources. Consequently, the cloud service in Figure 5.2.a is mapped to a single substrate node. However, such mapping is not necessarily efficient for IoT resources. In fact, the requested temperature sensing services can be allocated from different substrate nodes. As we notice, the data request node in Figure 5.2.a is mapped to two different substrate nodes.

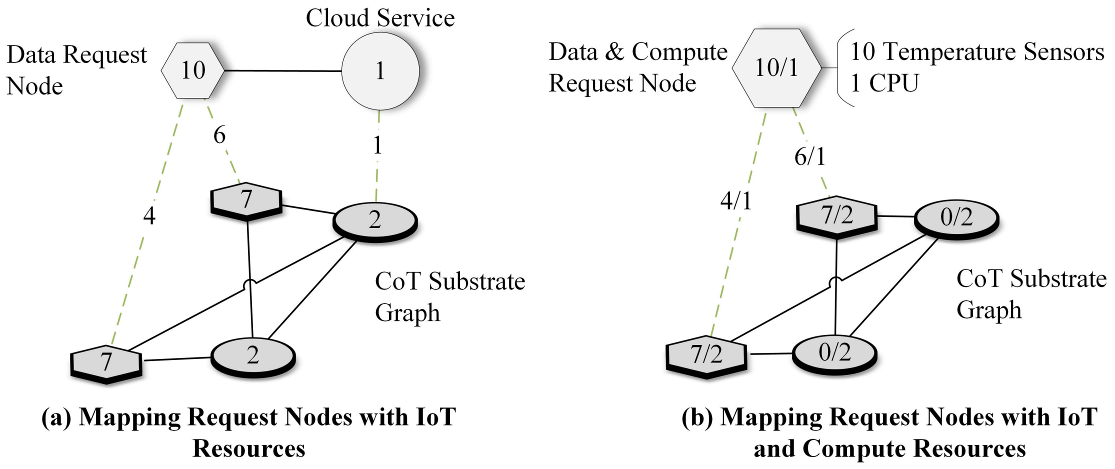

Cloud Service
Data Request Node
10
1
6
1
4
7
2
CoT Substrate Graph
7
2
(a) Mapping Request Nodes with IoT Resources
Data & Compute Request Node
10/1
10 Temperature Sensors
1 CPU
6/1
4/1
7/2
0/2
CoT Substrate Graph
7/2
0/2
(b) Mapping Request Nodes with IoT and Compute Resources

Figure 5.2: Distribution of Request Nodes with IoT Resources on Multiple Substrate Nodes

Some request nodes might require IoT resources alongside with compute and storage resources as depicted in Figure 5.2.b. Indeed, there exist some high-powered connected objects able to offer compute and storage resources beside sensing and actuating. For example, a gateway offering temperature sensing services might have also available virtualized computing and storing resources. Such gateway is able to host a service requiring a limited amount of the compute resource and providing temperature sensing data. In such case, we can also collect temperature sensing data from different substrate nodes. However, we must make sure that these nodes can also provide the requested vCPU resources. In contrast to IoT resources, Virtual Central Processing Units (vCPUs) cannot be divided. In Figure 5.2.b, a request node with 10 temperature sensors and 1 vCPU can be mapped to multiple substrate nodes. The IoT resources are divided between these substrate nodes, while vCPUs need to be satisfied entirely at each host as illustrated in Figure 5.2.b.

The provisioning process of a CoT request onto a CoT substrate should take into consideration the possibility of dividing a request node into a set of IoT resources on multiple substrate nodes. Otherwise, the CoT customer has to define himself multiple data request nodes and assign to each one a portion of needed IoT resources. In this case, the request might get rejected because its data request node requires to interact with too many connected objects. In addition, such approach is not efficient regarding the complexity of the process since it increases the CoT request size and therefore the problem size. Furthermore, it requires a prior knowledge of the infrastructure topology and available resources.

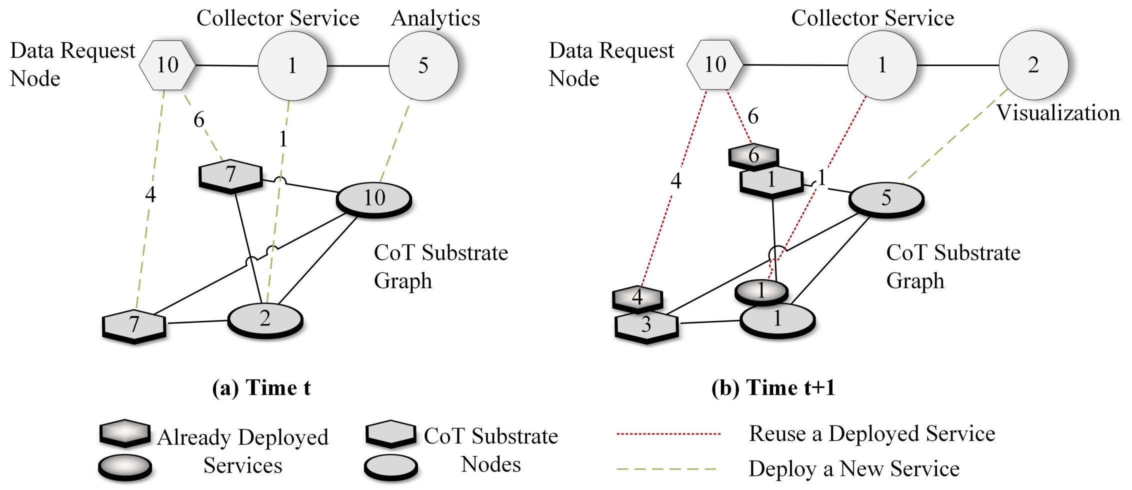


Figure 5.3: Re-using IoT Resources and Corresponding Deployed Delivery Services

As previously stated in Chapter 3, IoT resources might be shared among applications. It is therefore necessary that the provisioning process should be able to verify whether delivery services such as VOs has been already deployed and check if it is possible to reuse them for new incoming requests. As illustrated in Figure 5.3.b (i.e. time $t + 1$) the services deployed in 5.3.a (i.e. time t) can be reused for the request at time $t + 1$. If sharing IoT resources was not possible, the second request at time $t + 1$ would not have been satisfied because there are not enough resources available in the infrastructure at that time.

We argue therefore that deploying an IoT application in a CoT infrastructure is different from the traditional Cloud Computing deployment. There is a need to consider the possibility of mapping one request node to multiple nodes while verifying the availability of each resource type differently. This is due to the fact that some resources are partitioned across a set of nodes, while others should be entirely made available in each selected node. Moreover, the provisioning process in the CoT environment should consider previously allocated resources and reuse them in future deployments if possible to minimize the resources utilization and therefore the deployment cost.

5.3 Proposed Resources Provisioning Model

A unified representation of Cloud Computing and IoT allows a holistic view of the infrastructure. Such a unified model of available resources will eventually facilitate the one stage coordinated provisioning. To highlight the proposed approach, let's consider a substrate node element n_i^s which represents either a cloud data centre, a fog node, or a connected object. Each node n_i^s is attached to a set of resources r_i^s which identify its nature and geographical location g_i. In this model, on one hand, we identify the compute (c_i), storage (t_i), sensing (S_i), and actuating (A_i) resources. On the other hand, we identify the communication (network) resources that are represented as the available bandwidth $b_{i,j}^s$ between any two nodes n_i^s and n_j^s. Cloud data centres possess large (virtually unlimited) amounts of compute, storage, and network resources.

Fog nodes are characterized by the same type of cloud resources but in contrast to cloud data centres, these resources are limited. Not to mention that Fog nodes have also relatively lower latencies to connected objects since they are geographically closer to them. IoT devices do have communication capabilities and therefore network resources to connect to the network. High powered connected objects might be deployed to provide on-device compute and storage resources. It's worth noting that all connected objects have processing and storage capabilities even limited, however, low-powered connected objects reserve them for their sensing and actuating operations. Such resources are not available for external use. In the latter case, the node is represented without compute and storage resources in our model. With the same logic, a request node (i.e. virtual node) n^v represents an element which needs to be mapped and deployed into a substrate node.

A request node in the CoT Infrastructure as a Service (IaaS) level might represent a Virtual Machine (VM), a sensor, or an actuator. A VM request node is not attached to any sensing and actuating resources. Furthermore, the same request node in the CoT Platform as a Service (PaaS) level might represent a cloud service, a sensing service, data stream, etc. However, in the CoT PaaS level, infrastructure level resources are also represented as software components might be requested with minimal hardware requirements. We define the set of available atomic services at each substrate node n_i^s as $\Delta_i^s = \{\delta_{i,1}^s, \delta_{i,2}^s, ...\}$. Similarly, the requested service by a requested node n_j^v is noted as δ_j^v. $\delta^s.d$ and $\delta^s.s$ indicate respectively whether a substrate service δ^s is deployed or not, and is shareable or not. δ^s and δ^v represent the service ID, hence, we can check if two services are the same by the simple comparison $\delta^s - \delta^v = 0$.

Let us consider a request graph (N^v, E^v) and a substrate graph (N^s, E^s). The request graph needs to be mapped onto the substrate graph. N^v and N^s correspond to the sets of requested nodes and infrastructure nodes respectively. A request graph has m^v nodes, while the substrate graph has a cardinality m^s. A substrate node $n_i^s \in N^s$ offers a set of available resources $r_i^s = \{c_i^s, t_i^s, S_i^s, A_i^s\}$, while a request node requires a set of resources $r_i^v = \{c_i^v, t_i^v, S_i^v, A_i^v\}$. Unlike compute, storage, and network resources, a node can have multiple types of sensing and actuating resources. For example, a connected object might offer several temperature and humidity sensors. Therefore, we represent sensing and actuating resources at a given node n_i as a set of sensors S_i and actuators A_i. We consider that the CoT infrastructure includes k types of sensors and p types of actuators. Hence, the sensing and actuation resources of a substrate node n_i^s can be represented as $S_i^s = \{s_{i,1}^s, ..., s_{i,k}^s\}$ and $A_i^s = \{a_{i,1}^s, ..., a_{i,p}^s\}$ respectively. $s_{i,1}^s$ is the number of sensors of type 1 available at the substrate node n_i^s.

For example, a Raspberry PI node having only one compute resource and two sensors of type 1 has $r^s = \{1, 0, \{2, 0, ..., 0\}, \{0, ..., 0\}\}$. Sensing and actuating resources are represented similarly for request nodes. Such grouped representation of IoT resources decreases the request and infrastructure sizes in terms of nodes, and therefore reduces the problem size. Furthermore, it allows the representation of complex infrastructure nodes providing large amounts of IoT resources such as gateways. E^s and E^v represent requested network edges and substrate nodes interconnections. A substrate edge $e_{i,j}^s$ connects two substrate nodes n_i and n_j. It has an available bandwidth $b_{i,j}^s$ and a latency $l_{i,j}^s$. Similarly, a request edge $e_{i,j}^v$ links two request nodes n_i^v and n_j^v. Such request edge requires a minimal available bandwidth $b_{i,j}^v$ and a maximal latency $l_{i,j}^v$. A summary of variables is presented in Table 5.2.

The defined variables can be also identified from the previous defined model which helps parsing described CoT requests and substrates using our OCCI model for IaaS and PaaS. We show in Table 5.1 the equivalence between the OCCI model and the analytical model.

Table 5.1: Equivalence between the Analytical Model and the CoT OCCI model

Parameter	**OCCI CoT Equivalence**
n_i^s, n_j^v	The `Node` Class
$e_{i,j}^s$, $e_{i,j}^v$	The `NetworkLink` Class
$l_{i,j}^s$, $l_{i,j}^v$	The `NetworkLink` attribute `occi.networklink.latency`
$b_{i,j}^s$, $b_{i,j}^s$	The `NetworkLink` attribute `occi.networklink.bandwidth`
$s_{i,h}^s$, $s_{j,h}^v$	The `Sensor` attribute `occi.sensor.quantity`
$a_{i,h}^s$, $a_{j,h}^v$	The `Actuator` attribute `occi.actuator.quantity`
h	The `Actuator` or `Sensor` attribute `quantityKind`
g_i	The `Location` Class
$\delta_{i,j}^s$, δ_i^v	The `Component` class
$\delta_{i,j}^s.t$, δ_i^v	The `Component` attribute `occi.component.state`
$\delta_{i,j}^s.s$, δ_i^v	The `Component` attribute `occi.component.shareable`

5.3.1 Domain Variable

Once the request and substrate graphs are modelled using previous notations, the request graph should be mapped onto the substrate graph. Such mapping aims to optimize a given objective function such as minimizing the resources utilization. We consider $\alpha_{i,j} \in [0, 1]$ as the domain variable which indicates whether a request node n_i^v is mapped onto a substrate node n_j^s or not. However, different types of resources

Table 5.2: Notation Table

Symbol	**Definition**
(N^s, E^s)	CoT Substrate Graph.
(N^v, E^v)	CoT Request Graph.
N^s	Set of substrate nodes. $n_i^s \in N^s$. $\|N^s\| = m^s$.
N^v	Set of request nodes. $n_i^v \in N^v$. $\|N^v\| = m^v$.
r_i^s, r_j^v	Set of available resources at n_i^s, Set of requested resources for n_j^v.
c_i^s, c_j^v	Available amount of compute at n_i^s, Requested amount of compute for n_j^v.
t_i^s, t_j^v	Available amount of storage at n_i^s, Requested amount of storage for n_j^v.
S_i^s, S_j^v	Set of available sensors at n_i^s, Set of requested sensors for n_j^v.
A_i^s, A_j^v	Set of available actuators at n_i^s, Set of requested actuators for n_j^v.
Δ_i^s, δ_j^v	Set of available services in substrate node n_i^s, Requested service for n_j^v.
$s_{i,h}^s$, $s_{j,h}^v$	Available amount of sensor type h at n_i^s, The requested amount for n_j^v.
$a_{i,h}^s$, $a_{j,h}^v$	Available amount of actuator type h at n_i^s, The requested amount for n_j^v.
k	The total number of sensor types in the CoT infrastructure.
p	The total number of actuator types in the CoT infrastructure.
E^s	Set of edges between substrate nodes.
E^v	Set of edges between request nodes.
$e_{i,j}^s$	Network edge between two substrate nodes n_i^s and n_j^s.
$b_{i,j}^s$	Network bandwidth between two substrate nodes n_i^s and n_j^s.
$l_{i,j}^s$	Network latency between two substrate nodes n_i^s and n_j^s.
$e_{i,j}^v$	Network edge between two request nodes n_i^v and n_j^v.
$b_{i,j}^v$	Minimal required network bandwidth between two request nodes n_i^v and n_j^v.
$l_{i,j}^v$	Maximal tolerated network latency between two request nodes n_i^v and n_j^v.
$l_{i,j}^u$	Acceptable latency between the users group u_i^v and the request node n_j^v.
$\alpha_{i,j}$	The fraction of request node n_i^v mapped to substrate node n_j^s.
$\beta_{i,j}$	$\lceil \alpha_{i,j} \rceil$.

Table 5.3: Prices Definitions Table

Symbol	**Definition**
ϕ_i^c	The price of a compute unit at substrate node n_i^s.
ϕ_i^t	The price of a storage unit at substrate node n_i^s.
$\phi_{i,h}^s$	The price of a sensor $s_{i,h}^s$ of type h at substrate node n_i^s.
$\phi_{i,h}^a$	The price of an actuator $a_{i,h}^s$ of type h at substrate node n_i^s.
$\phi_{i,j}^e$	The price of a data unit on edge $e_{i,j}$ connecting nodes n_i^s and n_j^s.
$\phi_{i,h}^\delta$	The price of h^{th} service $\delta_{i,h}^s$ at substrate node n_i^s.

require different mapping strategies. For example, several gateways can be used to satisfy a request node operating several temperature and humidity sensors. Unlike the classical Cloud Computing mapping [23, 147], a CoT request node might be mapped to multiple substrate nodes as illustrated in Figure 5.4. We can notice that the sensing resources are here retrieved from different substrate nodes while the compute resource is retrieved entirely from each used substrate node.

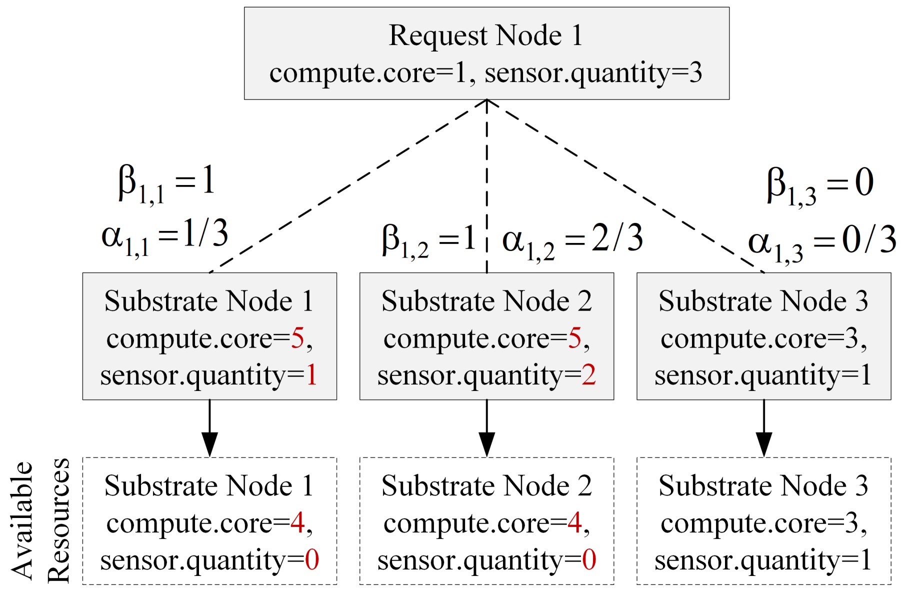


Figure 5.4: The Mapping of a Request Node with Sensing Resources.

It is important to notice that some resources such as compute cannot be fragmented. As a result, the constraints for verifying resources availability when mapping request nodes should take into consideration such specificity in the mapping strategies. Constraints (5.1) and (5.2) illustrate two constraint models. The first model is used for resources which must be made available as a whole in the substrate node, even though a fraction of the request node is mapped to it. This first model will be used to verify the availability of the compute resources for example. The second model is more appropriate for resources which can be fragmented across several substrate nodes. In this case, we only verify the availability of the required fraction at the substrate host. The constraint (5.2) is used for sensing and actuating resources.

$$\sum_i \beta_{i,j} x_i^v \leq x_j^s \quad \forall j : 1 \rightarrow m^s \tag{5.1}$$

$$\sum_i \alpha_{i,j} x_i^v \leq x_j^s \quad \forall j : 1 \rightarrow m^s \tag{5.2}$$

with:

$$\beta_{i,j} = \begin{cases} 1 & \text{if } \alpha_{i,j} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

The relation between $\beta_{i,j}$ and $\alpha_{i,j}$ can be expressed through the following constraints:

$$\beta_{i,j} \geq \alpha_{i,j} \quad \forall i : 1 \rightarrow m^v, j : 1 \rightarrow m^s$$
$$\beta_{i,j} \in \{0, 1\} \quad \forall i : 1 \rightarrow m^v, j : 1 \rightarrow m^s$$

Since $\alpha_{i,j}$ is a numeric variable (i.e. $\alpha_{i,j} \in [0, 1]$), cloud services will be mapped also onto multiple data centres. In such case, once the mapping is finished, we obtain a set of values indicating the fraction of the cloud service mapped to each data centre. We select the data centre with the larger fragment. We verify during the mapping that only data centres with sufficiently available compute, storage, and network resources are selected.

5.3.2 Cloud of Things Infrastructure

We formulate our placement problem similarly to the Cloud Computing VM placement problem formulation presented in [148]. The latter formulation linearises the traditional quadratic problem for VM placement [149]. Considering a binary mapping variable $x_{i,j}$ that indicates whether a request node i should be hosted on a substrate j, it is possible to introduce a new variable $y_{i,j,v,w}$ such that $y_{i,j,v,w} = x_{i,j}x_{v,w}$. Such variable represents edges mapping while hiding the quadratic expression as follows:

$$\min_x F(x) = \gamma F^{ct}(x) + \omega F^n(x) \quad \textbf{s.t. } \gamma + \omega = 1 \tag{5.4}$$

with:

$$F^{ct}(x) = \sum_i \sum_j x_{i,j}(\phi_i^c c_i^v + \phi_i^t t_i^v) \quad \forall i : 1 \rightarrow m^v, \quad \forall j : 1 \rightarrow m^s \tag{5.4a}$$

$$F^n(x) = \sum_i \sum_j \sum_v \sum_w y_{i,j,v,w} \phi_{i,j}^e b_{i,v}^v \quad \forall i, v : 1 \rightarrow m^v, \quad \forall j, w : 1 \rightarrow m^s \tag{5.4b}$$

subjected to:

$$\sum_i \sum_v y_{i,j,v,w} b_{i,v}^v \leq b_{j,w}^s \quad \forall j, w : 1 \rightarrow m^s \tag{5.4c}$$

$$\sum_j \sum_w y_{i,j,v,w} l_{j,w}^s \leq l_{i,v}^v \quad \forall i, v : 1 \rightarrow m^v \tag{5.4d}$$

$$\sum_i x_{i,j} c_i^v \leq c_j^s \quad \forall j : 1 \rightarrow m^s \tag{5.4e}$$

$$\sum_i x_{i,j} t_i^v \leq t_j^s \quad \forall j : 1 \rightarrow m^s \tag{5.4f}$$

$$\sum_i x_{i,j} = 1 \quad \forall j : 1 \rightarrow m^s \tag{5.4g}$$

$$\sum_i x_{i,j} l_{i,d(u_v^v)}^s \leq l_{i,j}^u \quad \forall j : 1 \rightarrow m^s, \quad \forall v : 1 \rightarrow z \tag{5.4h}$$

$$x_{i,j} \in \{0, 1\} \quad \forall i : 1 \rightarrow m^v, \forall j : 1 \rightarrow m^s \tag{5.4i}$$

$$\sum_j y_{i,j,v,w} = x_{v,w} \quad \forall i,v : 1 \to m^v, \quad \forall j : 1 \to m^s \tag{5.4j}$$

$$y_{i,j,v,w} = y_{v,w,i,j} \quad \forall i,v : 1 \to m^v, \quad \forall j,w : 1 \to m^s \tag{5.4k}$$

$$0 \leq y_{i,j,v,w} \leq 1 \quad \forall i,v : 1 \to m^v, \quad \forall j,w : 1 \to m^s \tag{5.4l}$$

The functions $F^{ct}(x)$ (5.4a) consists of compute and storage allocated resources while $F^n(x)$ (5.4a) corresponds to the network utilization price. ϕ^c_i, ϕ^t_i, and $\phi^n_{i,j}$ are compute, storage and network unit prices as defined in Table 5.3. Firstly, constraints (5.4c), (5.4e), and (5.4f) make sure that requested nodes are mapped to substrate nodes with sufficient resources. Secondly, constraints (5.4d) and (5.4h) preserve the Quality of Service (QoS) required by the CoT request graph in terms of communication latency. Finally, constraints (5.4g), (5.4i), (5.4j), (5.4k) and (5.4l) verify that all requested resources are mapped, define the relation between $x_{i,j}$ and $y_{i,j,v,w}$, and specify the boundaries of domain variables.

We adapt and extend this model to include sensor and actuator resources. Moreover, there is no need to consider the possibility of sharing IoT resources for the IaaS. In fact, at the IaaS level, customers have full control over the resources and their configuration. Therefore, it would not be possible to share IoT resources among multiple applications as it limits the ability of the customer to configure them as pleased. Under these assumptions, the model is formulated as follows:

$$\min_{\alpha} F(\alpha) = \gamma F^{ct}(\alpha) + \sigma F^o(\alpha) + \omega F^n(\alpha) \quad \textbf{s.t. } \gamma + \sigma + \omega = 1 \tag{5.5}$$

with:

$$F^{ct}(\alpha) = \sum_i \sum_j \beta_{i,j}(\phi^c_i c^v_i + \phi^t_i t^v_i) \quad \forall i : 1 \to p, \quad \forall j : 1 \to m \tag{5.5a}$$

$$F^o(\alpha) = \sum_i \sum_j \sum_h \alpha_{i,j}(s^v_{i,h}\phi^s_{j,h} + a^v_{i,h}\phi^a_{j,h})$$
$$\forall i : 1 \to m^v, \quad \forall j : 1 \to m^s, \quad \forall h : 1 \to z \tag{5.5b}$$

$$F^n(\alpha) = \sum_i \sum_j \sum_v \sum_w y_{i,j,v,w} b^v_{i,v} \phi^n_{j,w} \quad \forall i,v : 1 \to p, \quad \forall j,w : 1 \to m \tag{5.5c}$$

subjected to:

$$\sum_i \sum_v y_{i,j,v,w} b^v_{i,v} \leq b^s_{j,w} \quad \forall j,w : 1 \to m^s \tag{5.5d}$$

$$\sum_j \sum_w y_{i,j,v,w} l^s_{j,w} \leq l^v_{i,v} \quad \forall i,v : 1 \to m^v \tag{5.5e}$$

$$\sum_i \beta_{i,j} c^v_i \leq c^s_j \quad \forall j : 1 \to m^s \tag{5.5f}$$

$$\sum_i \beta_{i,j} t^v_i \leq t^s_j \quad \forall j : 1 \to m^s \tag{5.5g}$$

$$\sum_i \alpha_{i,j} s^v_{i,h} \leq s^s_{j,h} \quad \forall j : 1 \to m^s, \quad \forall h : 1 \to z \tag{5.5h}$$

$$\sum_i \alpha_{i,j} a^v_{i,h} \leq a^s_{j,h} \quad \forall j : 1 \rightarrow m^s, \quad \forall h : 1 \rightarrow z \tag{5.5i}$$

$$\sum_i \alpha_{i,j} = 1 \quad \forall j : 1 \rightarrow m^s \tag{5.5j}$$

$$\alpha_{i,j} \in [0, 1] \quad \forall i : 1 \rightarrow m^v, \forall j : 1 \rightarrow m^s \tag{5.5k}$$

$$\sum_j y_{i,j,v,w} = \alpha_{v,w} \quad \forall i, v : 1 \rightarrow m^v, \quad \forall j : 1 \rightarrow m^s \tag{5.5l}$$

$$y_{i,j,v,w} = y_{v,w,i,j} \quad \forall i, v : 1 \rightarrow m^v, \quad \forall j, w : 1 \rightarrow m^s \tag{5.5m}$$

$$0 \leq y_{i,j,v,w} \leq 1 \quad \forall i, v : 1 \rightarrow m^v, \quad \forall j, w : 1 \rightarrow m^s \tag{5.5n}$$

$$\beta_{i,j} \geq \alpha_{i,j} \quad \forall i : 1 \rightarrow m^v, j : 1 \rightarrow m^s \tag{5.5o}$$

$$\beta_{i,j} \in \{0, 1\} \quad \forall i : 1 \rightarrow m^v, j : 1 \rightarrow m^s \tag{5.5p}$$

F^{ct} (5.5a), F^o (5.5b), and F^n (5.5c) correspond to the compute, IoT, and network resources costs respectively. Firstly, constraints (5.5f), (5.5g), (5.5h), and (5.5i) make sure that requested nodes are mapped to substrate nodes with sufficient resources. Secondly, constraints (5.5e) preserve the QoS requested by the CoT request graph in terms of communication latency. Also, constraint (5.5j) verify that all requested resources are mapped while constraints (5.5k) and (5.5n) verify that domain variables remain in their value range. Constraints (5.5l) and (5.5m) define the relation between $\beta_{i,j}$ and $y_{i,j,v,w}$ while constraints (5.5o) and (5.5p) define the relation between $\alpha_{i,j}$ and $\beta_{i,j}$.

5.3.3 Cloud of Things Platform

For the CoT PaaS mapping we have to consider the availability of requested services at each substrate node. However, we need to account for previously deployed services as well. Therefore, a substrate node has additional constraints. The constraint aims for checking whether a requested service is available at the substrate node or not, whether this service is deployed or not, and whether this service is shareable or not. A mapping is possible only in two cases when the service is available at the substrate node: (1) the service is not deployed, or (2) the service is deployed and shareable. This constraint is represented as follows:

$$\alpha_{i,j} \delta^s_{i,h}.d(\delta^s_{i,h} - \delta^v_j)(1 - \delta^s_{i,h}.s) = 0 \quad \forall i : 1 \rightarrow m^s, \forall j : 1 \rightarrow m^v, \forall h : 1 \rightarrow |\Delta^s_i| \tag{5.6}$$

with:

$$\delta^s.d = \begin{cases} 1 & \text{if the service is deployed} \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

$$\delta^s.s = \begin{cases} 1 & \text{if the service is shareable} \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

Therefore the model for the CoT PaaS model becomes as follows:

$$\min_{\alpha} F(\alpha) = \gamma F^{ct}(\alpha) + \sigma F^{o}(\alpha) + \omega F^{n}(\alpha) \quad \textbf{s.t. } \gamma + \sigma + \omega = 1 \tag{5.9}$$

with:

$$F^{ct}(\alpha) = \sum_i \sum_j \beta_{i,j}(\phi_i^c c_i^v + \phi_i^t t_i^v) \quad \forall i : 1 \rightarrow p, \quad \forall j : 1 \rightarrow m \tag{5.9a}$$

$$F^{o}(\alpha) = \sum_i \sum_j \sum_h \alpha_{i,j}(s_{i,h}^v \phi_{j,h}^s + a_{i,h}^v \phi_{j,h}^a)$$
$$\forall i : 1 \rightarrow m^v, \quad \forall j : 1 \rightarrow m^s, \quad \forall h : 1 \rightarrow z \tag{5.9b}$$

$$F^{n}(\alpha) = \sum_i \sum_j \sum_v \sum_w y_{i,j,v,w} b_{i,v}^v \phi_{j,w}^n \quad \forall i, v : 1 \rightarrow p, \quad \forall j, w : 1 \rightarrow m \tag{5.9c}$$

subjected to:

$$\alpha_{i,j} \delta_{i,h}^s . d(\delta_{i,h}^s - \delta_j^v)(1 - \delta_{i,h}^s . s) = 0$$
$$\forall i : 1 \rightarrow m^s, \forall j : 1 \rightarrow m^v, \forall h : 1 \rightarrow |\Delta_i^s| \tag{5.9d}$$

$$\sum_i \sum_v y_{i,j,v,w} b_{i,v}^v \leq b_{j,w}^s \quad \forall j, w : 1 \rightarrow m^s \tag{5.9e}$$

$$\sum_j \sum_w y_{i,j,v,w} l_{j,w}^s \leq l_{i,v}^v \quad \forall i, v : 1 \rightarrow m^v \tag{5.9f}$$

$$\sum_i \beta_{i,j} c_i^v \leq c_j^s \quad \forall j : 1 \rightarrow m^s \tag{5.9g}$$

$$\sum_i \beta_{i,j} t_i^v \leq t_j^s \quad \forall j : 1 \rightarrow m^s \tag{5.9h}$$

$$\sum_i \alpha_{i,j} s_{i,h}^v \leq s_{j,h}^s \quad \forall j : 1 \rightarrow m^s, \quad \forall h : 1 \rightarrow z \tag{5.9i}$$

$$\sum_i \alpha_{i,j} a_{i,h}^v \leq a_{j,h}^s \quad \forall j : 1 \rightarrow m^s, \quad \forall h : 1 \rightarrow z \tag{5.9j}$$

$$\sum_i \alpha_{i,j} = 1 \quad \forall j : 1 \rightarrow m^s \tag{5.9k}$$

$$\alpha_{i,j} \in [0,1] \quad \forall i : 1 \rightarrow m^v, \forall j : 1 \rightarrow m^s \tag{5.9l}$$

$$\sum_j y_{i,j,v,w} = \alpha_{v,w} \quad \forall i, v : 1 \rightarrow m^v, \quad \forall j : 1 \rightarrow m^s \tag{5.9m}$$

$$y_{i,j,v,w} = y_{v,w,i,j} \quad \forall i, v : 1 \rightarrow m^v, \quad \forall j, w : 1 \rightarrow m^s \tag{5.9n}$$

$$0 \leq y_{i,j,v,w} \leq 1 \quad \forall i, v : 1 \rightarrow m^v, \quad \forall j, w : 1 \rightarrow m^s \tag{5.9o}$$

$$\beta_{i,j} \geq \alpha_{i,j} \quad \forall i : 1 \rightarrow m^v, j : 1 \rightarrow m^s \tag{5.9p}$$

$$\beta_{i,j} \in \{0,1\} \quad \forall i : 1 \rightarrow m^v, j : 1 \rightarrow m^s \tag{5.9q}$$

5.4 Implementation and Evaluation

We use IBM CPLEX to implement the proposed model in JAVA. The simulation compares between a global one stage mapping in the CoT and a two-stage mapping of Cloud Computing and IoT resources. We generate CoT request graphs with $m^s = 5 \rightarrow 20$ requesting a total number $u = 20 \rightarrow 320$ of connected objects. We only consider sensors during the simulation. The created CoT graphs are then divided into separate cloud and IoT graphs which are also mapped with the proposed model. This is possible since the model takes into consideration cloud and IoT resources simultaneously. This way, we simulate a two-stage mapping. We measure the gain of both orchestration approaches as well as different compute, IoT, and network costs. Results are shown in Figures 5.5 and 5.6.

5.4.1 Evaluation Results

We notice the benefit of mapping the CoT graph with a global representation of the infrastructure. This difference is mainly due to the cost of data units exchanged between cloud and IoT nodes. During a two-stage mapping, such information cannot be accounted for. Either nodes in both graphs are mapped separately, or one of the two mappings uses the results of the other to coordinate its nodes placement. However, in the latter cases, at least one of the mappings will aim for less expensive hosts while ignoring the network cost. Hence, when the number of connected objects increase, the transmitted data between the cloud data centres and IoT devices increases, leading to additional costs. Our model captures the network state between cloud and IoT resources and therefore is able to adapt the orchestration of resources accordingly to minimize the overall cost. When the number of nodes increases, the gain decreases because the one stage mapping process selects more costly nodes when the network links bandwidth is not sufficient. However, the two-stage mapping does not consider these constraints because it is not aware of the bandwidth consumption between IoT and cloud resources. Hence, it always maps to the less costly substrate nodes.

5.5 Conclusions

With the growing market of the Internet of Things, a huge number of physical devices is expected to be deployed worldwide which will generate a very large amount of data that could constitute a huge value added if correctly managed. To manage these IoT devices and the corresponding generated data, it is necessary to create new efficient efficient CoT services/applications deployment and resources allocation.

Many research efforts focused on creating IoT platforms to address the challenges of IoT following a data-centric approach (e.g. OpenIoT and FI-WARE). In this research work, we proposed an alternative solution that provides a coordinated one-stage provisioning of IoT application in the CoT infrastructure. We argue that this approach while introducing some level of complexity (holistic management of the environment), it also enables more refined optimization of the resource allocation of both Cloud Computing and IoT resources along with the optimization of the network resources. We formulated the problem as an optimization problem identifying the objective function and the constraints. We specified the problem as a linear programming problem and solve it (in a specified use case) using IBM CPLEX. The results show that a holistic provisioning process proves to be $10\% - 20\%$ more efficient than two separate orchestration processes for cloud and IoT resources respectively.

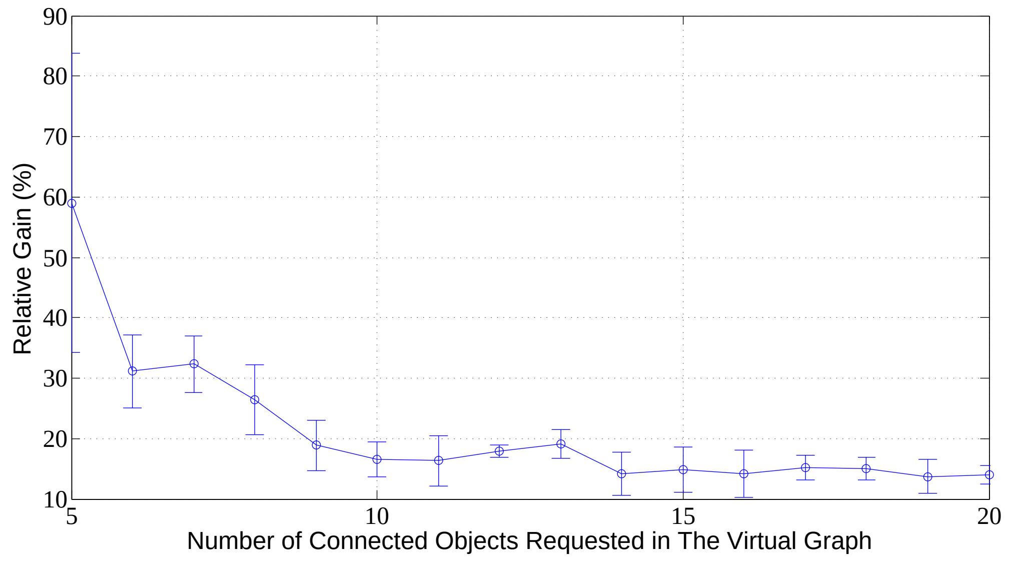


Figure 5.5: Relative Cost Gain of a One Stage Mapping over a Two Stages Mapping in CoT.

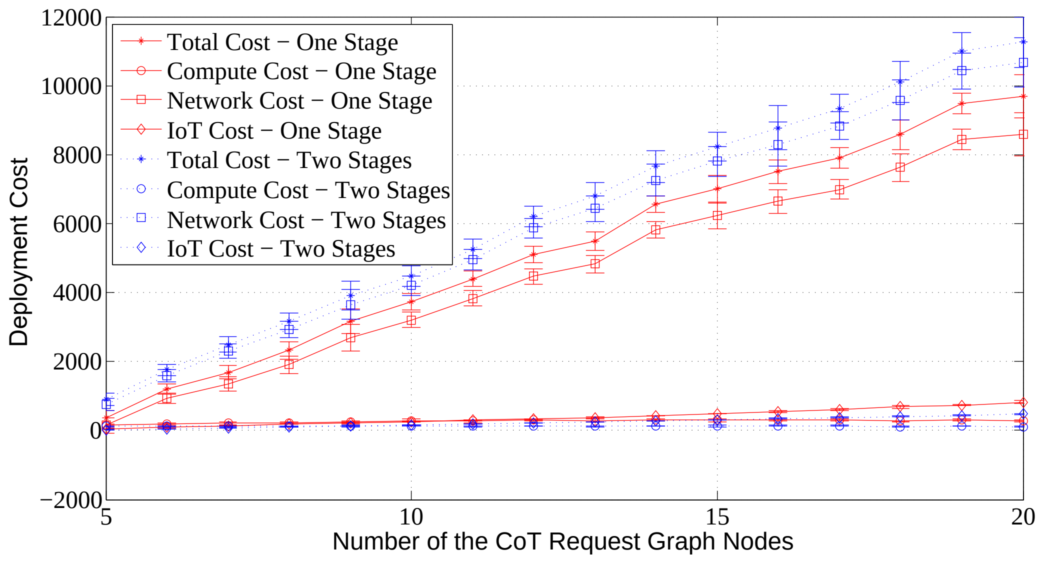


Figure 5.6: Cost of a One Stage Mapping Compared to a Two Stages Mapping.

Chapter 6

General Conclusion and Perspectives

The integration of Cloud Computing and the Internet of Things (IoT) realizes the Cloud of Things (CoT). Such integration makes possible the autonomous delivery of end-to-end IoT applications. In this thesis, we addressed the resources modelling and provisioning aspects in a CoT environment. First, we considered a partially integrated Cloud Computing and IoT infrastructures. In this context, we aimed at provisioning Virtual Objects (VOs) responsible for connecting a set of IoT devices to their corresponding cloud applications. Unlike existing approaches, we considered that connected objects and corresponding VOs could be shared among multiple applications to reduce the total amount of needed connected objects for a set of applications requirements. Such consideration increases the Quality of Service (QoS) constraints on each VO to deploy. To simplify the problem, we group VOs with identical constraints together and map grouped VOs instead of individual ones. We formulate the latter problem as two Linear Program (LP)s which aim to reduce the computing and networking cost of deployed VOs. The first deals with provisioning VOs for a set of application in an empty CoT environment, while the other considers that the CoT infrastructure contains previously deployed VO that can be reused. We introduce within the objective function a QoS index to minimize the communication latency between connected objects and cloud applications as well. As we notice, grouping VOs reduces the problem size and speeds up the execution of the LPs. Simulation results show that the shared approach is less costly than an unshared one. We show how the QoS index reduces the overall communication latency and achieves a better QoS concerning latency for all applications. However, employing the QoS index increases the cost as it forces a trade-off between cost and QoS.

Second, we surveyed existing standards for modelling the IoT and Cloud Computing. We synthesized the efforts to model the resources for IoT and cloud infrastructures to select a mature standard for modelling the CoT. We singled out the Open Cloud Computing Interface (OCCI) model. This model provides a complete representation of the cloud and has been adopted widely by researchers and existing cloud platforms such as OpenNebula. Then, we identified the basic resources which need to be added to the OCCI model to enable a representation of the CoT environment at the Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) levels. Such model constitutes the first step towards the auto-deployment of IoT applications in a CoT infrastructure. We present some example scenarios to illustrate how the model can be used to represent resources in the CoT. Furthermore, the model can be used to describe CoT request graphs and the CoT infrastructure. Such graphical representation of resources allows provisioning processes to parse directly incoming CoT request graphs and map them to the infrastructure.

Finally, we provided a model to specify analytically a CoT request and infrastructure as graphs. Such representation allows to map the CoT request on the infrastructure similarly to the virtual network embedding problem. We formulated an objective function with constraints to perform the optimal mapping of the CoT request onto the infrastructure. Such request contains interconnected cloud services and connected objects. Hence, the model can map cloud services to cloud data centres and perform a selection of IoT devices simultaneously. Simulations show that the provisioning of the CoT request in a single-stage outperforms the mapping in two separate stages considering cloud and IoT resources separately.

Several directions have been identified for future works:

In chapter 3, we do not take into consideration the selection of connected objects and consider they are already picked. However, as discussed in Chapter 3, combining the selection of connected objects and the deployment of VOs knowing the applications placement helps to enhance further the QoS. In fact, sharing VOs reduces the cloud resources utilization which minimizes the customers overall cost and increases the CoT provider profits as he becomes able to serve additional customers. However, sharing VOs among applications should consider the QoS requested by all applications. In our approach, such constraints cannot be altered which results in the rejection of some requests if the constraints are not met. Therefore, the provisioning process should be able to decide on sharing or not VO during the provisioning process to be able to satisfy applications constraints when it is not possible using a shared approach. Such flexibility in the model avoids rejecting incoming requests while minimizing as much as possible the resource utilization. Furthermore, sharing VOs can be tuned with the selection of connected objects to optimize the execution time while maintaining the cost reduction. As we notice, when the number of VO clusters increases, the execution time of the provisioning process increases. However, the number of clusters increases when connected objects do not have many similar sets of applications in common which can be controlled by the IoT selection process. A selection process can be employed to optimize the set of connected objects for each application in such a way as to minimize the number of clusters formed but maintain the total number of connected objects used to satisfy applications.

In chapter 4, the proposed model is a backbone for representing a CoT infrastructure. However, appropriate `Mixin`s should be defined by the CoT provider to personalize the model based on the properties of available components in his infrastructure (e.g. OpenStack data centre, Raspberry PI gateway). Works in [140, 141] can be used as an inspiration for extending our model. The objective of such model is to enable the automatic deployment of end-to-end IoT applications. Several cloud orchestration software exists which rely on the OCCI model. These models can be used as a base for developing a CoT orchestrator. Such model can also be linked with the node-RED project which helps define a workflow that can be then deployed automatically within the CoT infrastructure. IBM uses this project to facilitate the creation of workflows and enables their deployment in its cloud infrastructure (Bluemix). It also allows the deployment of IoT applications. However, such deployment is not yet fully automated as the connected objects are not provisioned on the fly. Intel is going in the same direction by adding an IoT module in its platform architecture. Such trends have been applied with OCCI PaaS for Cloud Computing to enable autonomous management and deployment of applications. The same approach can be applied for IoT applications. However, such approach is challenging in the CoT environment as self-configuration and self-adaptation mechanisms should consider different nature of hardware components which do not permit the same flexibility and scalability as cloud data centres.

Bibliography

[1] E. H. Cherkaoui, E. Rachkidi, M. Santos, P. A. L. Rego, J. Baliosian, and J. N. De, "SLA4CLOUD : Measurement and SLA Management of Heterogeneous Cloud Infrastructures Testbeds," in *3th International Workshop on ADVANCEs in ICT Infrastructures and Services*, pp. 1–6, 2014.

[2] T. Moreira, E. Rachkidi, L. M. Gardini, and R. Braga, "An Enhanced Architecture for LARIISA : An Intelligent System for Decision Making and Service Provision for e-Health using the cloud," in *4th International Workshop on ADVANCEs in ICT Infrastructures and Services*, 2015.

[3] E. Rachkidi, E. H. Cherkaoui, M. Ait-idir, N. Agoulmine, N. C. Taher, M. Santos, and S. Fernandes, "Towards Efficient Automatic Scaling and Adaptive cost-optimized eHealth Services in Cloud," in *2015 IEEE Global Communications Conference: Selected Areas in Communications: E-Health (GC' 15 - SAC - E-Health)*, pp. 1–6, IEEE, dec 2015.

[4] E. Rachkidi, E. H. Cherkaoui, M. Ait-idir, N. Agoulmine, N. C. Taher, M. Santos, and S. Fernandes, "Cooperative dynamic eHealth service placement in Mobile Cloud Computing," in *2015 17th International Conference on E-health Networking, Application & Services (HealthCom)*, (Boston, USA), pp. 627–632, IEEE, oct 2015.

[5] E. Rachkidi, N. Agoulmine, D. Belaïd, and N. Chendeb, "Towards an Efficient Service Provisioning in Cloud of Things (CoT)," in *2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, dec 2016.

[6] T. Moreira, H. Martin, E. Rachkidi, and N. Agoulmine, "An experiment on deploying a privacy-aware sensing as a service in the Sensor-Cloud," in *5th International Workshop on ADVANCEs in ICT Infrastructures and Services*, pp. 1–8, 2017.

[7] E. Rachkidi, N. Agoulmine, N. Chendeb, and D. Belaïd, "Resources Optimization and Efficient Distribution of Shared Virtual Sensors in Sensor-Cloud," in *2017 IEEE International Communications Conference (ICC)*, pp. 1–6, 2017.

[8] E. Rachkidi, N. Agoulmine, J. Baliosian, and J. Bustos, "VNET : Towards End-to-End Network Cloudification," in *5th International Workshop on ADVANCEs in ICT Infrastructures and Services*, pp. 1–5, 2017.

[9] E. Rachkidi, D. Belaïd, N. Agoulmine, and N. Chendeb, "Cloud of Things Modeling for Efficient and Coordinated Resources Provisioning," in *25th International Conference on COOPERATIVE INFORMATION SYSTEMS*, 2017.

[10] K. Ashton, "That 'Internet of Things' Thing," *RFiD Journal*, p. 4986, 2009.

[11] O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, I. S. Jubert, M. Mazura, M. Harrison, M. Eisenhauer, P. Doody, F. Peter, G. Patrick, G. Sergio, B. Harald, Sundmaeker Alessandro, J. Ignacio Soler, M. Margaretha, H. Mark, E. Markus, and D. Pat, "Internet of Things Strategic Research Roadmap," tech. rep., 2009.

[12] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, pp. 2787–2805, oct 2010.

[13] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, pp. 1497–1516, sep 2012.

[14] E. Borgia, "The Internet of Things vision: Key features, applications and open issues," *Computer Communications*, vol. 54, pp. 1–31, dec 2014.

[15] J. A. Stankovic, "Research Directions for the Internet of Things," *IEEE Internet of Things Journal*, vol. 1, pp. 3–9, feb 2014.

[16] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys & Tutorials*, vol. 17, pp. 2347–2376, jan 2015.

[17] D. Evans, "The Internet of Things - How the Next Evolution of the Internet is Changing Everything," Tech. Rep. April, 2011.

[18] A. Zaslavsky, C. Perera, and D. Georgakopoulos, "Sensing as a Service and Big Data," *Proceedings of the International Conference on Advances in Cloud Computing (ACC-2012)*, pp. 21–29, jan 2013.

[19] P. M. Mell and T. Grance, "The NIST definition of cloud computing," tech. rep., National Institute of Standards and Technology, Gaithersburg, MD, dec 2011.

[20] P. P. Ray, "A survey of IoT cloud platforms," *Future Computing and Informatics Journal*, pp. 1–12, mar 2017.

[21] E. Cavalcante, J. Pereira, M. P. Alves, P. Maia, R. Moura, T. Batista, F. C. Delicato, and P. F. Pires, "On the interplay of Internet of Things and Cloud Computing: A systematic mapping study," *Computer Communications*, vol. 89-90, pp. 17–33, sep 2016.

[22] S. Distefano, G. Merlino, and A. Puliafito, "Enabling the Cloud of Things," in *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pp. 858–863, IEEE, jul 2012.

[23] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[24] T. Metsch, A. Edmonds, and B. Parák, "Open Cloud Computing Interface - Infrastructure," tech. rep., Open Grid Forum, 2016.

[25] T. Metsch and M. Mohamed, "Open Cloud Computing Interface - Platform," tech. rep., Open Grid Forum, 2016.

[26] M. Mohamed, D. Belaïd, and S. Tata, "Extending OCCI for autonomic management in the cloud," *Journal of Systems and Software*, vol. 122, pp. 416–429, dec 2016.

[27] 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, mar 2016.

[28] M. Aazam, I. Khan, A. A. Alsaffar, and E. N. Huh, "Cloud of Things: Integrating Internet of Things and cloud computing and the issues involved," in *Proceedings of 2014 11th International Bhurban Conference on Applied Sciences and Technology, IBCAST 2014*, pp. 414–419, 2014.

[29] M. Yuriyama and T. Kushida, "Sensor-Cloud Infrastructure - Physical Sensor Management with Virtualized Sensors on Cloud Computing," in *2010 13th International Conference on Network-Based Information Systems*, pp. 1–8, IEEE, sep 2010.

[30] S. Yi, C. Li, and Q. Li, "A Survey of Fog Computing," in *Proceedings of the 2015 Workshop on Mobile Big Data - Mobidata '15*, (New York, New York, USA), pp. 37–42, ACM Press, 2015.

[31] G. Suciu, V. Suciu, A. Martian, R. Craciunescu, A. Vulpe, I. Marcu, S. Halunga, and O. Fratu, "Big Data, Internet of Things and Cloud Convergence - An Architecture for Secure E-Health Applications," *Journal of Medical Systems*, vol. 39, p. 141, nov 2015.

[32] J. Cubo, A. Nieto, and E. Pimentel, "A Cloud-Based Internet of Things Platform for Ambient Assisted Living," *Sensors*, vol. 14, pp. 14070–14105, aug 2014.

[33] L. Sanchez, L. Muñoz, J. A. Galache, P. Sotres, J. R. Santana, V. Gutierrez, R. Ramdhany, A. Gluhak, S. Krco, E. Theodoridis, and D. Pfisterer, "SmartSantander: IoT experimentation over a smart city testbed," *Computer Networks*, vol. 61, pp. 217–238, mar 2014.

[34] 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, pp. 1645–1660, sep 2013.

[35] M. Serrano, H. N. M. Quoc, D. Le Phuoc, M. Hauswirth, J. Soldatos, N. Kefalakis, P. P. Jayaraman, and A. Zaslavsky, “Defining the Stack for Service Delivery Models and Interoperability in the Internet of Things: A Practical Case With OpenIoT-VDK,” *IEEE Journal on Selected Areas in Communications*, vol. 33, pp. 676–689, apr 2015.

[36] N. Koshizuka and K. Sakamura, “Ubiquitous ID: Standards for Ubiquitous Computing and the Internet of Things,” *IEEE Pervasive Computing*, vol. 9, no. 4, pp. 98–101, 2010.

[37] S. N. Han, I. Khan, G. M. Lee, N. Crespi, and R. H. Glitho, “Service composition for IP smart object using realtime Web protocols: Concept and research challenges,” *Computer Standards & Interfaces*, vol. 43, pp. 79–90, jan 2016.

[38] N. C. Kushalnagar, G. M. C. Montenegro, and C. A. Schumacher, “RFC4919: IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals,” 2007.

[39] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, “Transmission of IPv6 Packets over IEEE 802.15.4 Networks, RFC 4944,” *Network Working Group*, pp. 1–30, 2007.

[40] R. Want, “An Introduction to RFID Technology,” *IEEE Pervasive Computing*, vol. 5, pp. 25–33, jan 2006.

[41] R. Want, “Near field communication,” *IEEE Pervasive Computing*, vol. 10, no. 3, pp. 4–7, 2011.

[42] R. S. Kshetrimayum, “An introduction to UWB communication systems,” *IEEE Potentials*, vol. 28, no. 2, pp. 9–13, 2009.

[43] T. Adame, A. Bel, B. Bellalta, J. Barcelo, and M. Oliver, “IEEE 802.11AH: the WiFi approach for M2M communications,” *IEEE Wireless Communications*, vol. 21, pp. 144–152, dec 2014.

[44] E. Khorov, A. Lyakhov, A. Krotov, and A. Guschin, “A survey on IEEE 802.11ah: An enabling networking technology for smart cities,” *Computer Communications*, vol. 58, pp. 53–69, mar 2015.

[45] L. Vangelista, A. Zanella, and M. Zorzi, “Long-Range IoT Technologies: The Dawn of LoRa,” in *Future Access Enablers of Ubiquitous and Intelligent Infrastructures* (V. Atanasovski and A. Leon-Garcia, eds.), vol. 159 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pp. 51–58, Cham: Springer International Publishing, 2015.

[46] J. Gozalvez, "New 3GPP Standard for IoT [Mobile Radio]," *IEEE Vehicular Technology Magazine*, vol. 11, pp. 14–20, mar 2016.

[47] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, and P. Polakos, "Wireless Sensor Network Virtualization: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 18, pp. 553–576, jan 2016.

[48] S. Kabadayi, A. Pridgen, and C. Julien, "Virtual Sensors: Abstracting Data from Physical Sensors," in *2006 International Symposium on a World of Wireless, Mobile and Multimedia Networks(WoWMoM'06)*, vol. 2006, pp. 587–592, IEEE, 2006.

[49] S. Madria, V. Kumar, and R. Dalvi, "Sensor Cloud: A Cloud of Virtual Sensors," *IEEE Software*, vol. 31, pp. 70–77, mar 2014.

[50] M. Fazio and A. Puliafito, "Cloud4sens: a cloud-based architecture for sensor controlling and monitoring," *IEEE Communications Magazine*, vol. 53, pp. 41–47, mar 2015.

[51] W. Dargie and C. Poellabauer, *Fundamentals of Wireless Sensor Networks.* 2010.

[52] R. Chu, L. Gu, Y. Liu, M. Li, and X. Lu, "SenSmart: Adaptive Stack Management for Multitasking Sensor Networks," *IEEE Transactions on Computers*, vol. 62, pp. 137–150, jan 2013.

[53] 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*, pp. 1–19, jan 2016.

[54] M. Gigli and S. Koo, "Internet of Things: Services and Applications Categorization," *Advances in Internet of Things*, vol. 01, no. 02, pp. 27–31, 2011.

[55] G. Aceto, A. Botta, W. de Donato, and A. Pescapè, "Cloud monitoring: A survey," *Computer Networks*, vol. 57, pp. 2093–2115, jun 2013.

[56] 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 - MCC '12*, (New York, New York, USA), p. 13, ACM Press, 2012.

[57] S. Sarkar, S. Chatterjee, and S. Misra, "Assessment of the Suitability of Fog Computing in the Context of Internet of Things," *IEEE Transactions on Cloud Computing*, vol. 7161, no. c, pp. 1–1, 2015.

[58] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, pp. 14–23, oct 2009.

[59] U. Shaukat, E. Ahmed, Z. Anwar, and F. Xia, "Cloudlet deployment in local wireless networks: Motivation, architectures, applications, and open challenges," *Journal of Network and Computer Applications*, vol. 62, pp. 18–40, feb 2016.

[60] R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shiraz, A. Yousafzai, and F. Xia, "A survey on virtual machine migration and server consolidation frameworks for cloud data centers," *Journal of Network and Computer Applications*, vol. 52, pp. 11–25, jun 2015.

[61] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe, "Mobile fog," in *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing - MCC '13*, (New York, New York, USA), p. 15, ACM Press, 2013.

[62] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog Computing: Principles, architectures, and applications," in *Internet of Things: Principles and Paradigms*, pp. 61–75, 2016.

[63] OpenFog Consortium Architecture Working Group, "OpenFog Reference Architecture for Fog Computing," no. February, pp. 1–162, 2017.

[64] A. Brogi and S. Forti, "QoS-aware Deployment of IoT Applications Through the Fog," *IEEE Internet of Things Journal*, pp. 1–1, 2017.

[65] D. Milojičić, I. M. Llorente, and R. S. Montero, "OpenNebula: A Cloud Management Tool," *IEEE Internet Computing*, vol. 15, pp. 11–14, mar 2011.

[66] G. Tanganelli, C. Vallati, and E. Mingozzi, "Energy-Efficient QoS-aware Service Allocation for the Cloud of Things," in *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, vol. 2015-Febru, pp. 787–792, IEEE, dec 2014.

[67] S. Misra, S. Chatterjee, and M. S. Obaidat, "On Theoretical Modeling of Sensor Cloud: A Paradigm Shift From Wireless Sensor Network," *IEEE Systems Journal*, pp. 1–10, 2014.

[68] S. Chatterjee and S. Misra, "Optimal composition of a virtual sensor for efficient virtualization within sensor-cloud," *IEEE International Conference on Communications*, vol. 2015-Septe, pp. 448–453, 2015.

[69] S. Chatterjee, S. Sarkar, and S. Misra, "Energy-efficient data transmission in sensor-cloud," *2015 Applications and Innovations in Mobile Computing (AIMoC)*, pp. 68–73, 2015.

[70] L. Ben Saad and B. Tourancheau, "Lifetime optimization of sensor-cloud systems," in *2015 7th International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5, IEEE, jul 2015.

[71] J. Soldatos, N. Kefalakis, M. Hauswirth, M. Serrano, J.-p. Calbimonte, M. Riahi, K. Aberer, P. P. Jayaraman, A. Zaslavsky, I. P. Žarko, L. Skorin-Kapov, and R. Herzog, "OpenIoT: Open Source Internet-of-Things in the Cloud," in *Interoperability and Open-Source Solutions for the Internet of Things*, vol. 9001, pp. 13–25, 2015.

[72] S. Distefano, G. Merlino, and A. Puliafito, "A utility paradigm for IoT: The sensing Cloud," *Pervasive and Mobile Computing*, vol. 20, pp. 127–144, jul 2015.

[73] S. Bose and N. Mukherjee, "SensIaas: A Sensor-Cloud Infrastructure with Sensor Virtualization," in *2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud)*, pp. 232–239, IEEE, jun 2016.

[74] X. Sheng, J. Tang, X. Xiao, and G. Xue, "Sensing as a Service: Challenges, Solutions and Future Directions," *IEEE Sensors Journal*, vol. 13, pp. 3733–3741, oct 2013.

[75] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Sensing as a service model for smart cities supported by Internet of Things," *Transactions on Emerging Telecommunications Technologies*, vol. 25, pp. 81–93, jan 2014.

[76] J. Barbaran, M. Diaz, and B. Rubio, "A Virtual Channel-Based Framework for the Integration of Wireless Sensor Networks in the Cloud," in *2014 International Conference on Future Internet of Things and Cloud*, pp. 334–339, IEEE, aug 2014.

[77] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati, "Cloud of Things for Sensing-as-a-Service: Architecture, Algorithms, and Use Case," *IEEE Internet of Things Journal*, vol. 3, pp. 1099–1112, dec 2016.

[78] C. Doukas and F. Antonelli, "COMPOSE: Building smart context-aware mobile applications utilizing IoT technologies," in *Global Information Infrastructure Symposium - GIIS 2013*, pp. 1–6, IEEE, oct 2013.

[79] A. Puliafito, "SensorCloud: An Integrated System for Advanced Multi-risk Management," in *2014 IEEE 3rd Symposium on Network Cloud Computing and Applications (ncca 2014)*, pp. 1–8, IEEE, feb 2014.

[80] S. Distefano, G. Merlino, and A. Puliafito, "Towards the Cloud of Things Sensing and Actuation as a Service, a Key Enabler for a New Cloud Paradigm," in *2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pp. 60–67, IEEE, oct 2013.

[81] S. H. Kim and D. Kim, "Multi-tenancy Support with Organization Management in the Cloud of Things," in *2013 IEEE International Conference on Services Computing*, pp. 232–239, IEEE, jun 2013.

[82] B. Christophe, M. Boussard, M. Lu, A. Pastor, and V. Toubiana, "The web of things vision: Things as a service and interaction patterns," *Bell Labs Technical Journal*, vol. 16, pp. 55–61, jun 2011.

[83] N. Mitton, S. Papavassiliou, A. Puliafito, and K. S. Trivedi, "Combining Cloud and sensors in a smart city environment," *EURASIP Journal on Wireless Communications and Networking*, vol. 2012, p. 247, dec 2012.

[84] T. Dinh and Y. Kim, "An Efficient Interactive Model for On-Demand Sensing-As-A-Services of Sensor-Cloud," *Sensors*, vol. 16, p. 992, jun 2016.

[85] M. Lemos, C. de Carvalho, D. Lopes, R. Rabelo, and R. H. Filho, "Reducing Energy Consumption in Provisioning of Virtual Sensors by Similarity of Heterogenous Sensors," in *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pp. 415–422, IEEE, mar 2017.

[86] C. Perera, A. Zaslavsky, P. Christen, M. Compton, and D. Georgakopoulos, "Context-aware sensor search, selection and ranking model for internet of things middleware," *Proceedings - IEEE International Conference on Mobile Data Management*, vol. 1, pp. 314–322, 2013.

[87] C. Perera, A. Zaslavsky, C. H. Liu, M. Compton, P. Christen, and D. Georgakopoulos, "Sensor search techniques for sensing as a service architecture for the internet of things," *IEEE Sensors Journal*, vol. 14, no. 2, pp. 406–420, 2014.

[88] D. H. Phan, J. Suzuki, S. Omura, K. Oba, and A. Vasilakos, "Multiobjective Communication Optimization for Cloud-Integrated Body Sensor Networks," in *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 685–693, IEEE, may 2014.

[89] G. Skourletopoulos, C. X. Mavromoustakis, G. Mastorakis, J. N. Sahalos, J. M. Batalla, and C. Dobre, "Cost-benefit analysis game for efficient storage allocation in cloud-centric Internet of Things systems: A game theoretic perspective," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 1149–1154, IEEE, may 2017.

[90] S. Misra, H.-C. Chao, R. Tirkey, A. Mondal, S. Bera, and S. Chattopadhyay, "Optimal gateway selection in sensor-cloud framework for health monitoring," *IET Wireless Sensor Systems*, vol. 4, no. 2, pp. 61–68, 2014.

[91] S. Chatterjee and S. Misra, "QoS estimation and selection of CSP in oligopoly environment for Internet of Things," in *2016 IEEE Wireless Communications and Networking Conference*, vol. 2016-Septe, pp. 1–6, IEEE, apr 2016.

[92] J. Kim and J.-W. Lee, "OpenIoT: An open service framework for the Internet of Things," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pp. 89–93, IEEE, mar 2014.

[93] S. Sotiriadis, K. Stravoskoufos, and E. G. Petrakis, "Future Internet Systems Design and Implementation: Cloud and IoT Services Based on IoT-A and FIWARE," in *Designing, Developing, and Facilitating Smart Cities*, pp. 193–207, Cham: Springer International Publishing, 2017.

[94] T. Kramp, R. van Kranenburg, and S. Lange, *Enabling Things to Talk.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[95] Zhenyu Wu, T. Itala, Tingan Tang, Chunhong Zhang, Yang Ji, M. Hamalainen, and Yunjie Liu, "Gateway as a service: A cloud computing framework for web of things," in *2012 19th International Conference on Telecommunications (ICT)*, no. Ict, pp. 1–6, IEEE, apr 2012.

[96] J. A. Galache, T. Yonezawa, L. Gurgen, D. Pavia, M. Grella, and H. Maeomichi, "ClouT: Leveraging Cloud Computing Techniques for Improving Management of Massive IoT Data," in *2014 IEEE 7th International Conference on Service-Oriented Computing and Applications*, pp. 324–327, IEEE, nov 2014.

[97] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: State-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[98] E. Y. Nakagawa, F. Oquendo, and J. C. Maldonado, "Reference Architectures," in *Software Architecture 1*, pp. 55–82, Chichester, UK: John Wiley & Sons, Ltd, may 2014.

[99] S. Madria, V. Kumar, and R. Dalvi, "Sensor cloud: A cloud of virtual sensors," *IEEE Software*, vol. 31, no. 2, pp. 70–77, 2014.

[100] F. Li, M. Vogler, M. Claessens, and S. Dustdar, "Towards Automated IoT Application Deployment by a Cloud-Based Approach," in *2013 IEEE 6th International Conference on Service-Oriented Computing and Applications*, no. January 2016, pp. 61–68, IEEE, dec 2013.

[101] K. T. Tran, *Efficient complex service deployment in cloud infrastructure.* PhD thesis, Universite d'Evry Val d'Essonne, 2013.

[102] S. Li, L. D. Xu, and S. Zhao, "The internet of things: a survey," *Information Systems Frontiers*, vol. 17, no. 2, pp. 243–259, 2015.

[103] Y. Zhou, S. De, W. Wang, and K. Moessner, "Search Techniques for the Web of Things: A Taxonomy and Survey," *Sensors*, vol. 16, p. 600, apr 2016.

[104] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati, "Cloud of Things for Sensing as a Service: Sensing Resource Discovery and Virtualization," in *2015 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–7, IEEE, dec 2015.

[105] K. B. and M. H.T., "Sensing services in cloud-centric Internet of Things: A survey, taxonomy and challenges," *2015 IEEE International Conference on Communication Workshop, ICCW 2015*, pp. 1865–1870, 2015.

[106] Y. Xu and S. Helal, "An Optimization Framework for Cloud-Sensor Systems," in *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, vol. 2015-Febru, pp. 38–45, IEEE, dec 2014.

[107] M. Botts, G. Percivall, C. Reed, and J. Davidson, "OGC® Sensor Web Enablement: Overview and High Level Architecture," in *GeoSensor Networks*, pp. 175–190, 2008.

[108] A. Broring, J. Echterhoff, S. Jirka, I. Simonis, T. Everding, C. Stasch, S. Liang, and R. Lemmens, "New Generation Sensor Web Enablement," *Sensors*, vol. 11, pp. 2652–2699, mar 2011.

[109] N. Chen, L. Di, G. Yu, and M. Min, "A flexible geospatial sensor observation service for diverse sensor data based on Web service," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 64, pp. 234–242, mar 2009.

[110] M. E. Poorazizi, S. H. L. Liang, and A. J. S. Hunter, "Testing of sensor observation services," in *Proceedings of the First ACM SIGSPATIAL Workshop on Sensor Web Enablement - SWE '12*, no. c, (New York, New York, USA), pp. 32–38, ACM Press, 2012.

[111] S. Liang, C.-y. Huang, and T. Khalafbeigi, "OGC SensorThings API Part I:Sensing," tech. rep., OGC Implementation Standard, 2016.

[112] D. Guinard and V. Trifa, "Towards the Web of Things : Web Mashups for Embedded Devices," *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009)*, pp. 1–8, 2009.

[113] D. Guinard, V. Trifa, F. Mattern, and E. Wilde, "From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices," in *Architecting the Internet of Things*, pp. 97–129, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[114] D. Driscoll and A. Mensch, "Devices Profile for Web Services Version 1.1," Tech. Rep. July, jul 2009.

[115] G. Moritz, E. Zeeb, S. Prüter, F. Golatowski, D. Timmermann, and R. Stoll, "Devices profile for web services and the REST," *IEEE International Conference on Industrial Informatics (INDIN)*, pp. 584–591, 2010.

[116] C. Lerche, N. Laum, G. Moritz, E. Zeeb, F. Golatowski, and D. Timmermann, "Implementing powerful Web Services for highly resource-constrained devices," in *2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pp. 332–335, IEEE, mar 2011.

[117] I. K. Samaras, G. D. Hassapis, and J. V. Gialelis, “A modified DPWS protocol stack for 6LoWPAN-based wireless sensor networks,” *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 209–217, 2013.

[118] G. Moritz, F. Golatowski, C. Lerche, and D. Timmermann, “Beyond 6LoWPAN: Web Services in Wireless Sensor Networks,” *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 1795–1805, 2013.

[119] M. Compton, P. Barnaghi, L. Bermudez, R. Garcia-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, V. Huang, K. Janowicz, W. D. Kelsey, D. Le Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K. Page, A. Passant, A. Sheth, and K. Taylor, “The SSN ontology of the W3C semantic sensor network incubator group,” *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 17, pp. 25–32, dec 2012.

[120] C. A. Henson, J. K. Pschorr, A. P. Sheth, and K. Thirunarayan, “SemSOS: Semantic sensor Observation Service,” in *2009 International Symposium on Collaborative Technologies and Systems*, pp. 44–53, IEEE, 2009.

[121] S. J. D. Cox, “An explicit OWL representation of ISO/OGC observations and measurements,” *CEUR Workshop Proceedings*, vol. 1063, pp. 1–18, 2013.

[122] S. J. Cox, “Ontology for observations and sampling features, with alignments to existing models,” *Semantic Web*, vol. 8, pp. 453–470, dec 2016.

[123] D. Le-Phuoc, H. Q. Nguyen-Mau, J. X. Parreira, and M. Hauswirth, “A middleware framework for scalable management of linked streams,” *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 16, pp. 42–51, nov 2012.

[124] M. B. Alaya, S. Medjiah, T. Monteil, and K. Drira, “Toward semantic interoperability in oneM2M architecture,” *IEEE Communications Magazine*, vol. 53, pp. 35–41, dec 2015.

[125] M. Bermudez-Edo, T. Elsaleh, P. Barnaghi, and K. Taylor, “IoT-Lite: A Lightweight Semantic Model for the Internet of Things,” *UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld*, pp. 1–8, 2016.

[126] A. Gangemi, *Ontology design patterns for Semantic Web content*, vol. 3729, pp. 262–276. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.

[127] F. Scharffe, J. Euzenat, and D. Fensel, “Towards design patterns for ontology alignment,” in *Proceedings of the 2008 ACM symposium on Applied computing - SAC ’08*, (New York, New York, USA), p. 2321, ACM Press, 2008.

[128] N. Seydoux, K. Drira, N. Hernandez, and T. Monteil, “IoT-O, a Core-Domain IoT Ontology to Represent Connected Devices Networks,” in *Knowledge Engineering and Knowledge Management, 20th International Conference, EKAW 2016, Bologna, Italy, Proceedings* (E. Blomqvist, P. Ciancarini, F. Poggi, and

F. Vitali, eds.), vol. 10024 of *Lecture Notes in Computer Science*, pp. 561–576, Cham: Springer International Publishing, 2016.

[129] K. Janowicz and M. Compton, "The stimulus-sensor-observation ontology design pattern and its integration into the semantic sensor network ontology," *CEUR Workshop Proceedings*, vol. 668, 2010.

[130] M. Jansen, O. Koch, and M. Schellenbach, *SNPS: An OSGi-Based Middleware for Wireless Sensor Networks*, vol. 393 of *Communications in Computer and Information Science*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[131] J.-p. Calbimonte, S. Sarni, J. Eberle, and K. Aberer, "XGSN : An Open-source Semantic Sensing Middleware for the Web of Things," *7th International Workshop on Semantic Sensor Networks*, 2014.

[132] C. Perera, A. Zaslavsky, P. Christen, A. Salehi, and D. Georgakopoulos, "Connecting mobile things to global sensor network middleware using system-generated wrappers," in *Proceedings of the 11th ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE'12)*, (New York, New York, USA), p. 23, ACM Press, 2012.

[133] M. Waschke, *Cloud Standards: Agreements That Hold Together Clouds*. Berkeley, CA: Apress, 2012.

[134] T. Metsch, A. Edmonds, R. Nyrén, and R. Nyr, "Open Cloud Computing Interface - Core," tech. rep., Open Grid Forum, 2011.

[135] S. Yangui and S. Tata, "An OCCI Compliant Model for PaaS Resources Description and Provisioning," *The Computer Journal*, vol. 59, pp. 308–324, mar 2016.

[136] R. Boutaba and N. L. S. da Fonseca, *Cloud Services, Networking, and Management*, vol. 53. Hoboken, NJ: John Wiley & Sons, Inc, apr 2015.

[137] J. Parpaillon, P. Merle, O. Barais, M. Dutoo, and F. Paraiso, "OCCIware - A formal and tooled framework for managing everything as a service," *CEUR Workshop Proceedings*, vol. 1400, pp. 18–26, 2015.

[138] J. Durand, A. Otto, G. Pilz, and T. Rutt, "Cloud Application Management for Platforms Version 1.1," tech. rep., OASIS Committee Specification 01, 2014.

[139] J. Durand, A. Otto, G. Pilz, and T. Rutt, "Cloud Application Management for Platforms Version 1.2," tech. rep., OASIS Committee Specification Draft 01, 2017.

[140] A. Ciuffoletti, "OCCI-IOT: an API to deploy and operate an IoT infrastructure," *IEEE Internet of Things Journal*, vol. 4662, no. c, pp. 1–1, 2017.

[141] P. Merle, C. Gourdin, and N. Mitton, "Mobile Cloud Robotics as a Service with OCCIware," *Proceedings of the 2nd IEEE International Congress on Internet of Things, IEEE ICIOT 2017*, vol. 1, 2017.

[142] M. Weyrich and C. Ebert, "Reference Architectures for the Internet of Things," *IEEE Software*, vol. 33, pp. 112–116, jan 2016.

[143] P. Fremantle, "A Reference Architecture for the Internet of Things (White Paper).," Tech. Rep. October, WSO2, 2015.

[144] S.-W. Lin, B. Miller, J. Durand, G. Bleakley, A. Chigani, R. Martin, B. Murphy, and M. Crawford, "The Industrial Internet of Things Volume G1: Reference Architecture," tech. rep., 2017.

[145] J. Swetina, G. Lu, P. Jacobs, F. Ennesser, and J. Song, "Toward a standardized common M2M service layer platform: Introduction to oneM2M," *IEEE Wireless Communications*, vol. 21, no. 3, pp. 20–26, 2014.

[146] K. Aberer, M. Hauswirth, and A. Salehi, "A Middleware For Fast and Flexible Sensor Network Deployment," *Proceedings of the 32nd international conference on Very large data bases*, pp. 1199–1202, 2006.

[147] F. Stefanello, V. Aggarwal, L. S. Buriol, and M. G. C. Resende, "Hybrid Algorithms for Placement of Virtual Machines across Geo-Separated Data Centers," *European Journal of Operational Research*, 2016.

[148] F. Stefanello, V. Aggarwal, L. S. Buriol, J. F. Gonçalves, and M. G. Resende, "A Biased Random-key Genetic Algorithm for Placement of Virtual Machines across Geo-Separated Data Centers," in *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference - GECCO '15*, no. August, (New York, New York, USA), pp. 919–926, ACM Press, 2015.

[149] C. G. Lee and Z. Ma, "The generalized quadratic assignment problem," tech. rep., 2004.

Titre : Modélisation et Optimisation du Placement de Services Composés dans une Infrastructure Convergente de l'Informatique en Nuage et de l'Internet des Objets.

Keywords : Cloud Computing, Internet des Objects, Cloud des Objects, Optimisation des Resources, Modélisation des Resources.

Résumé : La convergence de l'Internet des Objets IdO (Internet of Things) et de l'Informatique en Nuage (Cloud Computing) est une approche prometteuse. D'une part, l'Informatique en Nuage fournit des ressources de calcul, de réseau, et de stockage théoriquement illimitées, et d'autre part, l'IdO permet l'interaction des services en nuage avec des objets du monde réel. Une convergence efficace de ces deux technologies aura un impact certainement important sur les innovations dans les domaines des services IT par l'introduction de nouveaux modèles de services d'IdO à la demande. Dans un tel contexte, les objects connectés sont vitualisés et offerts comme étant des services en nuage accessibles sur Internet depuis n'importe où et à n'importe quel moment. Ces services sont connus sous le nom d'Objets Virtuels (OVs). Ils cachent l'hétérogénéité de l'IdO et lient les objects connectés aux services en nuage traditionnels (i.e. services de stockage) pour fournir des applications IdO.

Dans cette thèse, nous considérons d'abord une intégration partielle de l'IdO et de l'Informatique en Nuage. Cette intégration fournit l'IdO au sein d'un seul niveau de service de l'Informatique en Nuage. Dans ce cas, les ressources de l'IdO et de l'Informatique en Nuage sont approvisionnées séparément. Nous nous concentrons dans ce travail sur l'orchestration des OVs dans une infrastructure en Nuage. Nous définissons un algorithme d'approvisionnement basé sur une stratégie de partage où chaque objet connecté est associé à un seul OV et peut être consommé par plusieurs applications. Nous proposons deux programmes linéaires pour effectuer l'approvisionnement des OVs. Le premier en cas où il n'y a pas des OVs précédemment déployés dans l'infrastructure, tandis que l'autre prend en compte le cas où il y a des OVs déjà déployés. Notre approche minimise les coûts opérationnels des OVs et la latence de communication par rapport aux approches qui considèrent une stratégie de non-partage.

La deuxième partie de cette thèse considère une intégration complète de l'IdO et de l'Informatique en Nuage. Nous appelons cette intégration le Nuage des Objects (NdO). Dans ce contexte, un client sera capable de demander un approvisionnement, un déploiement, et une mise à l'échelle automatique d'une application IdO de bout en bout à la volée avec un minimum d'efforts de gestion. En particulier, nous abordons l'aspect de l'approvisionnement. Nous définissons un modèle orienté ressources capable de décrire une demande d'une application IdO et une infrastructure NdO sur différents niveaux de service. Nous basons notre modèle sur les spécifications OCCI définies par l'OGF. En outre, nous définissons un algorithme d'approvisionnement en une étape coordonnée pour orchestrer une application IdO dans une infrastructure NdO. L'algorithme considère les ressources de l'IdO et de l'Informatique en Nuage simultanément. Les simulations montrent qu'un processus d'approvisionnement en une étape coordonnée est $10\% - 20\%$ plus efficace que deux processus d'orchestration des ressources de l'IdO et de l'Informatique en Nuage séparés.

Title : Modelling and Placement Optimization of Compound Services in a Converged Infrastructure of Cloud Computing and Internet of Things.

Keywords : Cloud Computing, Internet of Things, Cloud of Things, Resource Optimization, Resource Modeling.

Abstract : The convergence of the Internet of Things (IoT) and Cloud Computing technologies is a promising approach. On the one hand, Cloud Computing provides virtually unlimited computing, networking, and storage resources for constrained IoT devices. On the other hand, the IoT enables the interaction of cloud services with real world things. Such integration stimulates innovation in both areas and provides novel service delivery models such as the Sensing as a Service in different application domains (i.e. healthcare, transportation, smart-city, smart-building). In such convergence, things are abstracted and offered as cloud services accessible over the Internet from any place and at any time. Such abstractions are known as Virtual Objects (VOs) and connect things to traditional cloud services (e.g. data analytics, storage services) to deliver IoT applications.

In this thesis, we consider first a partial integration of the IoT and Cloud Computing. Such integration focuses on delivering the IoT within a single service level of Cloud Computing, namely: the application, the platform, or the infrastructure level. In this context, IoT and Cloud Computing resources are provisioned separately. We focus in this work on the orchestration of VOs in a cloud infrastructure. For this purpose, we define a provisioning algorithm based on a sharing strategy where each connected object is associated with a single VO and can be consumed by multiple applications. We propose two linear programs to perform the provisioning of VOs. The first considers no previously deployed VOs in the infrastructure, while the other takes into consideration pre-deployed VOs. Our approach minimizes VOs operational cost and communication latency in both cases compared to those with a non-sharing strategy.

The second part of this thesis considers a full integration of the IoT and Cloud Computing. We refer to such integration as the Cloud of Things (CoT). In this context, a customer should be able to request end-to-end IoT application provisioning, deployment, auto-scaling, and release on the fly with minimal management efforts. In this thesis, we address the provisioning aspect. We define a resource-oriented model able to describe an IoT application request and a CoT infrastructure on different service levels. We base our model on the OCCI specifications defined by the OGF. Furthermore, we define a single stage provisioning algorithm to orchestrate a described IoT application into a CoT infrastructure. The algorithm considers cloud and IoT resources simultaneously. Simulations show that a one-stage provisioning process is $10\% - 20\%$ more efficient than two separate orchestration processes for cloud and IoT resources.