MASTER THESIS

Proposal of an Adaptable and Scalable IoT Middleware for Hybrid Computational Models

Lucas Mauricio Castro e Martins

Brasília, December 2019

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROPOSAL OF AN ADAPTABLE AND SCALABLE IOT MIDDLEWARE FOR HYBRID COMPUTATIONAL MODELS

LUCAS MAURÍCIO CASTRO E MARTINS

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE.

APROVADA POR:

RAFAEL TIMÓTEO DE SOUSA JÚNIOR, Dr., ENE/UNB
(ORIENTADOR)

GEORGES DANIEL AMVAME NZE, Dr., ENE/UNB
(EXAMINADOR INTERNO)

ELDER OROSKI, Dr., UTFPR
(EXAMINADOR EXTERNO)

Brasília, 17 de dezembro de 2019.

FICHA CATALOGRÁFICA

MARTINS, LUCAS MAURICIO CASTRO E
Proposal of an Adaptable and Scalable IoT Middleware for Hybrid Computational Models [Distrito Federal] 2019.
xvi, 88 p., 210 x 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2019).
Master Thesis - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Internet of Things	2. Adaptable IoT
3. IoT Middleware	4. Fog computing
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

MARTINS, L. M. C. E (2019). *Proposal of an Adaptable and Scalable IoT Middleware for Hybrid Computational Models*. Dissertação de Mestrado, Departamento de Engenharia Elétrica, Publicação PPGEE.DM 739/2019, Universidade de Brasília, Brasília, DF, 88 p.

Lucas Mauricio Castro e Martins
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

Dedication

Life is short, complex and surprising.
It is up to each one of us to contribute to make our days here on earth more enjoyable and fruitful.
Therefore, I dedicate this work to all who believe and commit themselves to respect to others and the greater good, above their own individualities.

Lucas Mauricio Castro e Martins

Acknowledgements

The academic journey is a responsibility of student, but it is never a lonely journey. My master's journey began in 2014 shortly after completing a *lato sensu* specialization in Software Engineering. This thesis is the end of this journey and I take my public written acknowledgment to people and institutions below, mentioning them in chronological order of events:

To professor Genaína Rodrigues for her guidance in my first steps in graduate school at UnB. I extend thanks to the PPGInf staff;

To my friend Mario Miura for the opportunity to let me join his team of a UnB-DPU project. Spoiler alert: this event was a game changer, rerouting me to graduation course. In addition, I have received all his support at very difficult times in my financial and professional life;

To my colleagues I worked with directly or indirectly on DGPU or GSI/PR projects;

To Cleinilton Fernandes for telling me about the UIoT project and introducing me to two characters cited as follows;

To Claudio Santoro and Caio Cesar Silva for passionately introducing me to the UIoT project and the IoT field itself;

To Caio Cesar Silva and Francisco de Caldas Filho for the tips, guidance, and advices they gave me that help me for a successfully PPGEE application;

To Julio Cesar de Almeida and Flávia Paiva, my chiefs in the STF during this period. Their support, patience, tips, and authorisation for my participation in academic events as well for the training license were key for finishing this work;

To my colleagues at the STF for their support and understanding at the moments I needed. Mainly, to Jacinto Cunha for all his support and backup all over the way, but mostly in the final phase of the research;

To fellow researchers from the LATITUDE laboratory, represented by prof. Fábio Mendonça and Kelly Santos, for their tips, support, and resources;

To the University of Brasilia for offering a high quality course here in Brasilia, for the structure, and for the support given;

To the PPGEE and PPEE staff for all their support and patience;

To the co-authors who collaborated and published with me up to now;

To the professors João Paulo Lustosa, William Giozza and Robson Albuquerque for their valuable tips and guidance on scientific methods, academic career, and writing of academic articles;

To the bright young students and researchers who work/worked in the UIoT project for their dedication, help, and assistance to me and to the project. I am grateful to all the many members

who have been with us for no matter how long. At the risk of injustice, I dare to mention Dayanne da Cunha, Bruno J. Praciano, Cassio Fabius, Lincoln Barbosa, João de Menezes, Pedro da Costa, and Daniel Prado because their key participation on my thesis' results;

To Gabriel Alves for his support throughout the course and his huge patience during the most critical days of my immersion in this work;

To professor Robson Albuquerque, again, for the confidence in my work and the efforts to open my eyes to take better actions towards a succeed course conclusion;

To Juliana Stela, Célia Araújo, Francisco de Caldas Filho, Polyane Wercelens, Klayton de Castro, and Cohen Gorman, who joined the "task force" to help me get this work done;

To the professors Elder Oroski and Georges Nze for accepting to be part of the board of examiners and for their contributions to this work;

To Francisco Caldas Filho, my great partner in the research and actions in the UIoT laboratory, for his tips, guidance, support, and work produced;

To professor Rafael de Sousa Jr., my advisor, for the teachings, guidance, leadership, advices, kicks in the butt, opportunities, and confidence. His effort to combine science and innovation projects is an inspiration as well as his open-mindedness regarding his students is a lesson for me;

To my brothers and parents, whether by blood or by marriage, for their support, words of encouragement and confidence (with this family confidence in me, it's virtually impossible to keep one's feet on the ground);

To my mom, Verediana Castro, that always maintained a learning-prone environment that promoted my learning and development;

To Juliana Stela and Lucas Mateus, my beautiful and beloved children, for their tips, questions, logistical support, care, and conversations that brought fruitful information and curiosities that helped me to shape my work;

To Célia Araújo, my companion and soul mate, for her complicity, trust, support, patience, tips, and shielding;

To God, Allah, Obatala, or Science (whatever higher entity you believe in).

Wholeheartedly, thank you all!

I gratefully acknowledge the support from the Brazilian Union Public Defender (DPGU), grant 066/2016, and the Institutional Security Office of the Presidency of the Republic of Brazil (GSI/PR) Grant 002/2017.

The UIoT Lab has the support of the Federal District Research Support Foundation FAPDF (Projects UIoT 0193.001366/2016 and SSDDC 0193.001365/2016), and the LATITUDE/UnB Laboratory (Project SDN 23106.099441/2016-43). I thank both institutions for their support.

ABSTRACT

The United Nations estimates that the world population will reach almost 10 billion people by 2050. This increase in the world population, as well as its longevity, force optimizations in the productive chain that supplies food for many people. The IoT has shown that it can contribute in this scenario as researchers have presented several applications for the IoT. Such as: smart home, smart building, smart city, and industry 4.0. However, the benefits promised by IoT can only be realized by customers willing to meet prerequisites. Such as: quality Internet connectivity, and the willingness to use resources allocated to the cloud computing architecture. If it does not fit the above scenario, IoT can hardly be enjoyed. Currently, research in the most varied fields of information and communication technology seeks to solve or mitigate these restrictions. This work embraces the concept of multiple, and overlapping, IoTs and propose a middleware that is flexible and adaptable to various scenarios and configurations, becoming highly adaptable and scalable. This middleware supports the creation of its IoT instance and can natively, and transparently, interact with other instances. The architecture, components, and operation of middleware are presented, describing the influence received from Microservice Architecture, as well as edge computing and fog computing. It is demonstrated that the proposal met its objectives, allowing the creation of IoT networks in heterogeneous scenarios, maintaining its functionalities. As a result, its users have greater ownership and confidence in the data they handle on its IoT network. Throughout the research, some middleware and/or IoT aspects suggested that it can be explored to refine and/or evolve it. Such as: the need to apply intelligent agent techniques to improve middleware autonomy and performance; integrate with other work focusing on middleware and instance security, and apply methodologies and ontologies aimed at integrating with third party middleware.

Keywords: Internet of Things; Adaptable IoT; IoT Middleware; Cloud computing; Edge computing; Fog computing.

RESUMO

A Organização das Nações Unidas estima que a população terrestre chegue a quase 10 bilhões de pessoas em 2050. Esse aumento da população mundial, bem como da sua longevidade pressionam pela necessidade de otimizações da cadeia produtiva que suprem desde a alimentação até o lazer dos seres humanos. IoT tem dado a impressão de que pode contribuir neste cenário. Pesquisadores têm apresentado diversas aplicações de IoT, tais como, por exemplo, casas inteligentes, prédios inteligentes, cidades inteligentes e indústria 4.0. Porém, os benefícios prometidos por IoT só podem ser obtidos por clientes dispostos a atenderem pré-requisitos como conexão de qualidade com a Internet e à predisposição de utilizar recursos alocados na arquitetura de nuvem computacional. Caso não se enquadre no cenário acima, dificilmente pode-se usufruir de IoT. Atualmente, pesquisas nos mais variados campos de tecnologia da informação e comunicação procuram resolver ou mitigar essas restrições. Este trabalho abraça o conceito das múltiplas e sobrepostas IoTs e propõe um *middleware* que seja flexível e adaptável a diversos cenários e configurações, tornado-se altamente adaptável e escalável. Esse *middleware* suporta a criação da sua instância IoT e pode nativamente interagir com outras instâncias de forma transparente. São apresentados a arquitetura, os componentes e a forma de funcionamento do *middleware*, descrevendo a influência recebida da Arquitetura de Microsserviços, bem como de Computação na borda e Computação em Nevoeiro. É demonstrado que a proposta atendeu seus objetivos, permitindo a criação de redes IoT em cenários heterogêneos, mantendo as suas funcionalidades. Por consequência, seus usuários têm maior propriedade e confiança nos dados que manipulam na sua rede IoT. Ao longo da pesquisa, alguns aspectos do middleware e/ou de IoT se destacaram, sugerindo a possibilidade de serem melhor explorados para algum refinamento e/ou evolução. Destacam-se a necessidade de aplicar técnicas de agentes inteligentes para melhorar a autonomia e o desempenho dos *middlewares*; de integrar com outros trabalhos cujo foco é a segurança do *middleware* e da instância e de aplicar metodologias e ontologias voltadas à integração com *middlewares* de terceiros.

Palavras-chave: Internet das Coisas; IoT Adaptável; Middleware IoT; Computação em nuvem; Computação na borda; Computação em nevoeiro.

Contents

List of Figures

List of Tables

List of Acronyms

General Acronyms

Ad Hoc	Ad Hoc network
AI	Artificial Inteligence
API	Application Programming Interface
CBSE	Componet-Based Software Engineering
CPS	Cyber-Physical System
DBMS	Database Management Systems
DDD	Domain-Driven Design
DIY	Do-it-yoursef
DL	Deep Learning
ECG	Electrocardiogram
ESB	Enterprise Service Bus
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
ICT	Information and communications technology
IP	Internet Protocol
IPv6	Internet Protocol version 6
IQR	Interquartile range
JSON	JavaScript Object Notation
LAN	Local area network
M2M	Machine-to-machine
ML	Machine Learning
MAC	Media access control
MQTT	Message Queue Telemetry Transport
MSA	Microservices Architecture
NIST	National Institute of Standards and Technology
NoSQL	Non SQL (Structured Query Language) or Non Relational
PaaS	Platform as a Service
P2P	Peer-to-peer
QoS	Quality of Service
RAM	Random-access memory
REST	REpresentational State Transfer
RFC	Request for Comments
RFID	Radio-frequency identification
RTT	Round-trip time
SaaS	Software as a Service

SBC	Single-board computer
SPA	Single page application
SOA	Service-Oriented Architecture
TCP	Transmission Control Protocol
ubi-comp	Ubiquitous Computing
UDP	User Datagram Protocol
UI	User Interface
UPnP	Universal Plug and Play
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
vCPU	Virtual CPU (Central processing unit)
WAN	Wide area network
WSN	Wireless Sensor Network
XML	Extensible Markup Language

Internet of Things Related

IIoT	Industrial Internet of Things
IoP	Internet of People
IoT	Internet of Things
SIoT	Social Internet of Things

Related Softwares Proposed by Researchers

DIMS	Data Interface Management System
FaaS4IoT	Fog-as-a-Service for Internet of Things
HiCH	Hierarchical Fog-Assisted Computing Architecture for Healthcare IoT
RAISe	REST API Approach for IoT SErvices
UIoT	UnB Internet of Things
UIMS	User-friendly Interface Management System

Units of Measure

A	Ampere
GB	Gigabyte
GHz	Gigahertz
MB	Megabyte
Mbps	Megabits per second
ms	Millisecond
TB	Terabyte

1 INTRODUCTION

From the beginning, population growth has been perceived and reported, although this growth happens differently for each period, always related to historical events. According to Alves (2013), the world population was 226 million in year 1. In the late 15th century, the world population was already 438 million, in the late 16th, it was 556 million, in the late 18th, it had reached 1 billion and by the late 19th century the total was over 1.5 billion. The Industrial and Energy Revolution caused rapid demographic growth, reaching 6.1 billion inhabitants in the late 20th century. According to data made available by the United Nations, population growth has increased sharply and the projection for 2050 is that the population will reach 9.8 billion people, as shown in Figure 1.1. (UNITED NATIONS, 2019)

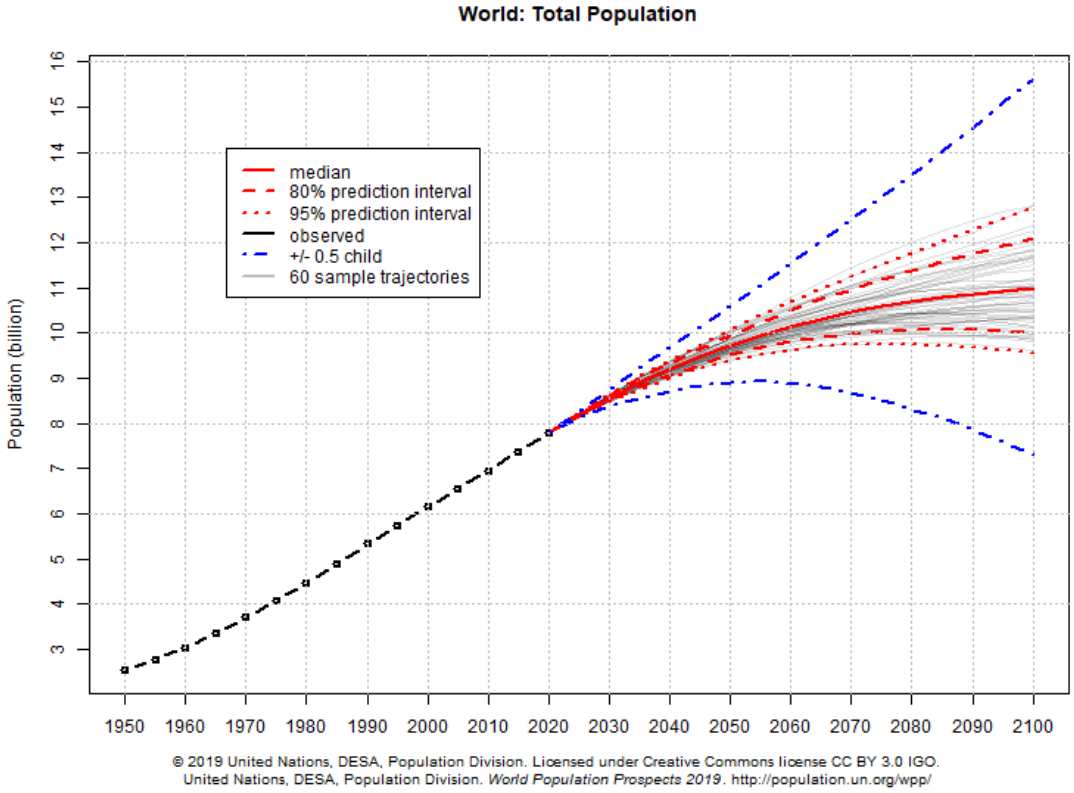


Figure 1.1: World population growth between 1950 and 2019 and its projection for 2050 (UNITED NATIONS, 2019)

With the exponential demographic growth in the twentieth century, the advance in technology, especially in the food, freshwater and health sectors, is indispensable in order to optimize and meet the demands of the population. Scientific and technological developments improved several branches that were directly related to population mortality. Among them the development of basic sanitation, improvement of diagnoses and cure for diseases, with diagnosis and treatment equipment and chemicals, development of techniques for the production of food on a large scale and with less waste and cost. This led to a decrease in infant mortality and an increase in life

expectancy. Even with natural disasters, urban violence, and epidemics, the world's population is still growing, albeit at a slow rate.

As a result, the technological revolutions contribute to the longevity of the population. According to United Nations (2019), life expectancy was almost 50 years old in the 1960s. It jumped to 72 years in 2019 and is expected to reach almost 77 years in 2050. Moreover, this factor now contributes to the population ageing: UN estimates that the percentage of people over 65 years of age will increase from 9% in 2019 to 16% in 2050. This suggests it is necessary to provide adapted assistance, especially for the older age group and for people with special needs.

In order to meet the demand of this large and growing population and to optimize the increasingly scarce physical space, further evolution is needed in the production, distribution and conservation of food, freshwater and medicine. This evolution implies a more sustainable use of the resources already accessed because it is no longer defensible that, as exhibited by Kamienski et al. (2019), agricultural activities continues to consume 70% of the freshwater available in the world. Thus, it is necessary to develop a means of improving the use of already occupied spaces and available resources, and to reuse the scrap produced, moving towards environmental sustainability.

The Internet of Things (IoT) has shown its potential to support this process by making environments and objects more intelligent. An intelligent environment has the ability to increase production potential and minimize losses. For example, a smart garden on the roof of a building can be used for food production with minimal human intervention. This type of system can make the necessary adjustments to light, nutrients and humidity to influence plant development.

One feature that has been leveraging IoT adoption and development is inherited from the Maker or Do-it-yourself (DIY) culture: It is possible to build an IoT solution using technologies and tools available in people's daily lives; especially open software, open hardware and cloud computing solutions. This has allowed onlookers and *makers* to rehearse the building of small room automation solutions. Although these solutions are quite simple, they mean some advance to the places where they are applied. Another interesting effect is that academia also reaps the same benefit; it becomes possible to start research on IoT solutions in isolation without relying on technology transfer from laboratories or manufacturers.

On the financial side, IoT also proves to be a very attractive solution because its components are often low cost solutions. On several occasions, makers have come up with a way to automate many processes and, more recently, IoT solutions that cost a few hundred dollars. Academics also benefit from the same situation because it is possible to start research on IoT solutions with few financial resources.

In addition to simulated or evaluative situations, IoT, Cyber-Physical Systems (CPS) and their variations are already used in real-world situations to solve real-world problems:

- Smart Grids have been tried since the late 2000s;

- Smart Cities have been studied since the early 2010s, and;
- Smart Watches recording the number of steps has become popular since 2015.

But, a distinction must be made: the automatic response to a simple and isolated stimulus does not fit well with the IoT proposal. IoT proposes to make the environment do its tasks with minimal human intervention possible and this implies the need to consider several variables, with stimuli from the most diverse sensors.

To differentiate the two cases, consider the following examples: if a room light comes on when a presence sensor detects a person's presence, it's a non-IoT-compliant automation solution. If you add a light sensor that gives the light the ability to check if the environment is already clear when a person's presence is detected, it is still an automation solution. If you add a person identification mechanism so that the light only activates according to the detected person's preference and this dataset and operations are shared with other devices, this would be a typical IoT scenario.

Achieving this kind of result requires the use of Artificial Intelligent (AI) mechanisms such as Machine Learning (ML) and Deep Learning (DL). However, both ML and DL become more efficient as they have more data to be used as input for training and model validation. This requires the gathering and handling of larger data volumes by IoT solutions.

Because IoT principles and objectives are well defined, there is some similarity in their adopted architectural standards as well as in some of the technologies used. However, there is no definition of the standards, norms and technologies that should be used or respected. Considering that the elementary building blocks of IoT are low cost and resource constrained sensors and actuators, and there is a need for these devices to communicate via the Internet, cloud computing is a consensus among the proposed IoT solutions. This is due to the ease of storing and processing large volumes of data with dynamic resource allocation.

The offered benefits notwithstanding, the use of cloud imposes some limitations such as reliance on stable, secure connectivity and within Transmission Control Protocol/Internet Protocol (TCP/IP) protocols to use the Internet as a means, and insecurity regarding the security and privacy of the collected and produced data.

Although there are discussions about where and how to use IoT and CPS, it is a fact that they are here to stay and will gain more and more space in people's lives. A curious factor about this is that people are increasingly immersed in IoT solutions without realizing it. A great example of this is the Home Media and Home Assistants platforms from companies like Amazon, Apple and Google. As a result, the discussion around these sensitive issues is misunderstood in order to gain their benefits and minimize their risks and side effects.

Given the limitations inherent in IoT solutions and concerns about the availability, reliability, and privacy of information, fog computing presents itself as an option to help address these issues properly. Fog computing is a computing model in which computational resources are deployed

in servers on the Internet and on the network itself (YI; LI; LI, 2015). Because it allows the IoT solution to remain low cost and decentralization of rules and information can be used as a tool to increase data security.

In this context, fog IoT middleware is proposed, utilizing fog's flexibility and portability, maintaining the low cost feature and giving its user ownership of their data, giving them the ability to centralize or distribute this data. The proposed solution must still have the ability to integrate with other cloud or fog solutions.

1.1 MOTIVATION

The centralization of device management and the processing of the data that is produced and used by them is a strategy that has some tradeoffs. On one hand, it facilitates system configuration and enhances resource utilization. On the other hand, it increases the volume of data being trafficked and, consequently, the effort employed in transforming this data into useful information.

This philosophy is employed by major Internet Age corporations like Google and Amazon because their business strategy is based on getting as much data as possible for profitable information generation. However, this scenario is not best suited for IoT solutions that have a specific purpose. For example, a smart vegetable garden in a neighborhood need not know the route someone is taking when walks to the market in another city.

In addition, distributed applications (such as IoT) are vulnerable to typical data communication problems such as channel availability, signal interference, communication latency, and throughput. Thus, high network latency in traditional IoT cloud architecture becomes a deterrent for applications that need near real-time responses. There are also scenarios in which some restriction on Internet connectivity makes the use of conventional IoT cloud architecture inefficient or even impracticable. This is especially true in places with poor telecommunication infrastructure that has an intermittent connection, a low speed signal or an unstable signal.

Just as there is no consensus on architecture and technologies, IoT is built on a wide variety of communication protocols and standards. In addition to the syntactic difficulty, there is a semantic difficulty: there is no consolidated ontology for dealing with data and the relationship between IoT devices. As a result, each manufacturer adopts the technology and ontology stack that suits them best, creating silos around their products and solutions. This factor makes it difficult and, in some cases, makes it impossible to integrate different products.

In addition to this, the Internet environment carries itself several inherent vulnerabilities to this type of network. Thus, the collaboration of some cloud providers and governments is not transparent as revealed by Snowden. Moreover, recent cases of data leaks on cloud platforms such as Amazon and Yahoo have drawn much negative attention to these services. This scenario creates distrust and insecurity, which hinders some people from choosing to use cloud-based IoT solutions. (CITIZENFOUR, 2014)

Fog computing arises as an interesting option for dealing with issues such as centralizing control and data, use in context with poor connectivity, the need for interoperability, and the concerns with the security of the information exposed in this work.

However, when using fog computing, it is necessary to create an ecosystem with distributed units that communicate between themselves. This implies the need to carry all or parts of the solution to other platforms options. This portability of middleware or its components to run on other systems is made difficult by the use of monolithic architectural patterns. To overcome this problem, it is also necessary to work in the field of software engineering to enable the adoption of this strategy.

1.2 OBJECTIVES

Given the difficulties of developing robust IoT applications and awareness of the necessary actions to be taken to ease the deploying and scaling of the IoT applications, the following research question arises: *do the flexibility and orchestration proposed by fog computing and distributed computing actually allow you to create a secure and efficient infrastructure that caters for standard IoT users as well those users with constrained connectivity and/or those with privacy concerns?*

With the purpose of answering this question, this master thesis pursues the hypothesis: *an adaptable and scalable IoT can provide data ownership to its owners, have lower network latency, be cheaper and more flexible for different usage scenarios compared to cloud-centric IoT.*

With this in mind, the main objective of this thesis is to **propose an adaptable and scalable IoT middleware for building IoT networks in both cloud-based infrastructure and scenarios that are restricted or unable to use this conventional cloud architecture**. This adaptable and scalable IoT middleware must allow its data and services to be used locally by its owner, and must also be able to be shared with third parties in accordance with established policies. The IoT network provided by this middleware must have access to all capabilities offered for a global IoT network.

To achieve this main goal, several secondary objectives have been defined:

- Refine and implement the IoT architecture in a distributed manner by reviewing middleware roles and components;
- Design ontologies for interoperability between services offered by devices;
- Design Microservices Architecture at the IoT gateway and middleware to support communication with devices with different requirements;
- Explore the use of containers and Microservices Architecture in middleware;

- Explore the use of miniaturized computers to build middleware into portable, easy-to-install equipment, and;
- Design mechanisms to enable the use of middleware in connectivity-constrained scenarios.

1.3 RESEARCH METHOD

This work has an empirical nature with the purpose of exploratory research and quantitative approach. Bibliographic research was conducted through an empirical and exploratory process aiming to raise the state of the art regarding the themes related to this work. The study covered an investigation into IoT and its application areas, as well as the elements that compose them. From this investigation, the scope of the work was delimited in the IoT middleware "minimization" and "cooperative". Minimization aims to enable it to work in simpler hardware configurations such as a single-board computer (SBC) called Raspberry Pi. The cooperation aims to make it expandable, resilient and, while maintaining its low cost, have a good processing and storage capacity.

To contemplate this scope, two questions were latent: Microservices and fog computing. The Microservices Architecture fits well with the issue of middleware minimization, allowing you to think about its refactoring while respecting its interfaces. Separating the components into smaller and single purpose pieces was the chosen strategy. Fog computing model brings all the baggage on the logic of interaction and cooperation between middleware instances, with a special focus on resource and data sharing among nodes participating in the network.

For the development and experimental implementation, scenarios and resources available in the LATITUDE laboratory were used, so that it focused on small available telemetry and automation solutions.

As the prototype concluded, laboratory experiments were performed to validate issues that drew more attention in the studies in the UIoT project. The empirical methodology was used to evaluate the results. The collected data were extracted from laboratory observation mechanisms. Thus, the proposal was evaluated and considered viable to answer the research question.

1.4 RESEARCH CONTRIBUTIONS

According to the objectives described, the main contributions of this work are:

- Definition of low-cost IoT middleware that enables IoT to be used off-grid and in restricted Internet connectivity scenarios
- Definition of an IoT middleware compatible with the fog computing paradigm, specifying its architecture and the rules of interaction of its components;

- Functional implementation of the IoT architecture proposal in order to validate said proposal, and;
- Platform consolidation to allow studies on peer-to-peer (P2P) and Oportunistic Networks.

1.4.1 Publications related to the thesis

The following list contains the publications we made related to this thesis. They are chronologically presented along with a brief summary of their content that highlights the contributions included on them.

1. CALDAS FILHO, F. L. d.; MARTINS, L. M. C. e.; ARAÚJO, I. P.; MENDONÇA, F. L. L. d.; COSTA, J. P. C. L. d.; DE SOUSA JÚNIOR, R. T. Gerenciamento de Serviços IoT com Gateway Semântico [IoT Service Management with Semantic Gateway]. In: *Atas das Conferências IADIS Ibero-Americanas WWW/Internet 2017 e Computação Aplicada 2017*. Vilamoura, Algarve, Portugal: IADIS Press, 2017. p. 199–206. ISBN 978-989-8533-70-8.;
2. MARTINS, L. M. C. e.; CALDAS FILHO, F. L. d.; DE SOUSA JÚNIOR, R. T.; GIOZZA, W. F.; COSTA, J. P. C. L. d. Proposta de Adoção de Microsserviços em IoT [Proposal of IoT Microservice Adoption]. In: *Atas das Conferências IADIS Ibero-Americanas WWW/Internet 2017 e Computação Aplicada 2017*. Vilamoura, Algarve, Portugal: IADIS Press, 2017. p. 63–70. ISBN 978-989-8533-70-8.;
3. CALDAS FILHO, F. L. d.; MARTINS, L. M. C. e.; ARAÚJO, I. P.; MENDONÇA, F. L. L. d.; COSTA, J. P. C. L. da; DE SOUSA JÚNIOR, R. T. Design and Evaluation of a Semantic Gateway Prototype for IoT Networks. In: *Companion Proceedings of the 10th International Conference on Utility and Cloud Computing*. Austin, TX, USA: ACM, 2017. (UCC '17 Companion), p. 195–201. ISBN 978-1-4503-5195-9.;
4. MARTINS, L. M. C. e.; CALDAS FILHO, F. L. d.; DE SOUSA JÚNIOR, R. T.; GIOZZA, W. F.; COSTA, J. P. C. L. da. Increasing the Dependability of IoT Middleware with Cloud Computing and Microservices. In: *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*. Austin, TX, USA: ACM, 2017. (UCC '17 Companion), p. 203–208. ISBN 978-1-4503-5195-9.;
5. SPERLING, T. L. von; CALDAS FILHO, F. L. de; DE SOUSA JÚNIOR, R. T.; MARTINS, L. M. C. e; ROCHA, R. L. Tracking intruders in IoT networks by means of DNS traffic analysis. In: *2017 Workshop on Communication Networks and Power Systems (WCNPS)*. Brasília, DF, Brazil: IEEE, 2017. p. 1–4.;
6. RIBEIRO, C. F. C.; CALDAS FILHO, F. L. d.; MARTINS, L. M. C. e; ABBAS, C. J. B.; DE SOUSA JÚNIOR, R. T. Protocolos de Redundância de Gateway Aplicados em Redes IoT. In: *Anais do XXXVI Simpósio Brasileiro de Telecomunicações e Processamento de Sinais (SBrT 2018)*. Campina Grande, PB, Brazil: SBrT, 2018. p. 1065–1069.;

7. SPERLING, T. L. von; FRANÇA, B. de A.; CALDAS FILHO, F. L. de; MARTINS, L. M. C. e; ALBUQUERQUE, R. de O.; DE SOUSA JÚNIOR, R. T. Evaluation of an IoT device designed for transparent traffic analysis. In: *2018 Workshop on Communication Networks and Power Systems (WCNPS)*. Brasília, DF, Brazil: IEEE, 2018. p. 1–5.;

8. POLETTI, J. V.; MARTINS, L. M. C. e; ALMEIDA, S.; HOLANDA, M.; DE SOUSA JÚNIOR, R. T. A Real Data Analysis in an Internet of Things Environment. In: INSTICC. *Proceedings of the 4th International Conference on Internet of Things, Big Data and Security - Volume 1: IoTBDS,*. Heraklion, Crete, Greece: SciTePress, 2019. p. 438–445. ISBN 978-989-758-369-8.;

9. DUTRA, B. V.; ALENCASTRO, J. F. de; CALDAS FILHO, F. L. de; MARTINS, L. M. C. e; DE SOUSA JÚNIOR, R. T.; ALBUQUERQUE, R. de O. HIDS by signature for embedded devices in IoT networks. In: UNIVERSIDAD DE EXTREMADURA. *Actas de las V Jornadas Nacionales de Investigación en Ciberseguridad (JNIC 2019)*. Cáceres, Spain, 2019. p. 53–61. ISBN 978-84-09-12121-2.;

10. CALDAS FILHO, F. L. de; ROCHA, R. L.; ABBAS, C. J. B.; MARTINS, L. M. C. e; CANEDO, E. D.; DE SOUSA JÚNIOR, R. T. QoS Scheduling Algorithm for a Fog IoT Gateway. In: *4th Workshop on Communication Networks and Power Systems (WCNPS 2019)*. Brasília, DF, Brazil: IEEE, 2019. p. 122–127..

1.5 OUTLINE

Besides this introduction, this master thesis is divided as follows: Chapter 2 explores the background for IoT, Microservices and cloud, edge and fog computing. Chapter 3 proposes the IoT platform concept and a concept's implementation. Chapter 4 depicts the IoT middleware usage possibilities, using hypothetical scenarios. Chapter 5 presents the testing methodology for validating the proposal and the testing data results are shown and discussed in terms of proposal validation. Finally, Chapter 6 draws conclusions based on the information presented in this thesis.

2 BACKGROUND AND RELATED WORKS

In first instance, this Chapter presents all concepts and methods used to build the proposed adaptable and scalable IoT and the Chapter also to points it out from other works. First of all, Section 2.1 describes IoT itself, its architecture, its usage, a brief discussion about Social IoT and the University of Brasília IoT (UIoT) project. Next, Section 2.2 discusses computing paradigms of XXI Century such as cloud, fog and edge computing. Section 2.3 presents the Microservices Architecture and its implications. Finally, Section 2.4 presents the works that are somehow related with this research.

2.1 INTERNET OF THINGS

Telemetry and remote monitoring of resources and environments have several advantages such as increased security, resource savings, and fostering nature studies. Supported by advances in computing and communication technologies since the beginning of the XX century, industry and academia are devoting significant efforts to explore it expecting take advantage of what it can undertake.

The Wireless Sensor Network (WSN) represents a major breakthrough in this area. As stated by Gubbi et al. (2013), a WSN consists of grouping dedicated wireless sensors that are distributed in a spatial region for the purpose of reading (sensing) and writing physical data specific to that environment, as well as storing and manipulating this data in a centralized and grouped manner.Authors described that WSNs are widely used in environmental monitoring, infrastructure monitoring, traffic monitoring, and retail. They also say that with the popularization of the Internet since the late 1990s, telemetry techniques, including WSNs, began to use the Internet as a means of communication.

In 1999, Ashton (2009) coined the term IoT to say that "[w]e need to empower computers with their own means of gathering information, so they can see, hear and smell the world for themselves [...]" to use the data obtained to help us interact with this physical world. He pointed out that people are physical beings living in a physical world and that digital information must come into the physical world for real utility. Also in Ashton (2009), author further argued that there was a great separation between the physical and the digital world, with the aggravating fact that almost all information manipulated by computers and the Internet was produced and fed by humans in the most diverse possible interactions: typing, photography, audio recording, scanning a bar code.

In its initial view, IoT was thought of as using recent technologies at the time as radio-frequency identification (RFID) to link these physical devices in the context of supply chains (ASHTON, 2009; GUBBI et al., 2013). However, soon IoT became broader and it has being

applied in situations such as:

- Smart home/building: control of home equipment and environmental monitoring of enterprise's facilities - will allow better resource management; (GUBBI et al., 2013)
- Living assistance: healthcare monitoring system for sick person and home monitoring system for elderly care - reducing hospitalization costs through early intervention and treatment by moving from the clinic-centric treatment to patient-centric healthcare (GUBBI et al., 2013; FARAHANI et al., 2018). Azimi et al. (2017, p. 174:2) highlights that "[t]he main function of automated health monitoring systems is to detect medical emergencies and patient health deterioration early enough, as rapid response [...] is instrumental to implement effective countermeasures";
- Smart city: aiming to handle urban mobility issues, population's health and well-being, provision of services to the inhabitants; (GUBBI et al., 2013)
- Smart agriculture: farming has a broad set of applications that can be built with IoT. Such as, irrigation management, treatment of soil quality, pest control, harvest performance or measurement. It can contribute to increase the productivity and sustainability specially by using machine-to-machine (M2M) operations; (GUBBI et al., 2013; NÓBREGA et al., 2019)
- Smart metering: is the constant monitoring of services provided to its customers, both monitoring endpoints and distribution points that are of interest to the service provider; (GUBBI et al., 2013)
- Smart grid: coupled with monitoring, has the ability to make the electrical production and distribution system adapt to the needs of the moment; (GUBBI et al., 2013)
- Water networks: as with smart grids, water distribution can be monitored and adapted according to demand. In addition, IoT systems can be used to constantly check water quality; (GUBBI et al., 2013)
- Industry 4.0: called the Fourth Industrial Revolution, treats with the interaction of manufacturing systems with the physical environment to manage the production process and optimize the resources used, and it results of intersecting studies in smart factory, energy management and IoT (SHROUF; ORDIERES; MIRAGLIOTTA, 2014; ZHONG et al., 2017). Shrouf, Ordieres & Miragliotta (2014) recalls that a basic principle of Industry 4.0 is to monitor work in progress in real time to make smart decisions in cooperation with humans, as Zhong et al. (2017) points out.

2.1.1 IoT definitions

There is no consensus on the exact definition of IoT and this issue is natural considering that it has influences from academic concepts that were already in use such as telemetry and WSN. As

well as it has emerged from a variety of experiments in industry and academia. In the following paragraphs, it is presented, in chronological order, some coined definitions for IoT that were considered in this research.

- Ashton (2009, online): computing power can be used to better manage things by empowering these computers to acquire data from the physical world without the need for human intervention, so that "[w]e would know when things needed replacing, repairing or recalling, and whether they were fresh or past their best."

- Guillemin & Friess (2009, p. 4), cited by Perera et al. (2014, Sec. II): "The Internet of Things allows people and things to be connected Anytime, Anyplace, with Anything and Anyone, ideally using Any path/network and Any service."

- Atzori, Iera & Morabito (2010):

  > The Internet of Things (IoT) is a novel paradigm that is rapidly gaining ground in the scenario of modern wireless telecommunications. The basic idea of this concept is the pervasive presence around us of a variety of things or objects – such as Radio-Frequency IDentification (RFID) tags, sensors, actuators, mobile phones, etc. – which, through unique addressing schemes, are able to interact with each other and cooperate with their neighbors to reach common goals. (ATZORI; IERA; MORABITO, 2010, p. 2787)

- Atzori et al. (2012):

  > The Internet of Things (IoT) integrates a large number of technologies and envisions a variety of things or objects around us that, through unique addressing schemes and standard communication protocols, are able to interact with each others and cooperate with their neighbors to reach common goals. (ATZORI et al., 2012, p. 3594)

- Gubbi et al. (2013):

  > Interconnection of sensing and actuating devices providing the ability to share information across platforms through a unified framework, developing a common operating picture for enabling innovative applications. This is achieved by seamless ubiquitous sensing, data analytics and information representation with Cloud computing as the unifying framework. (GUBBI et al., 2013, p. 1647)

- Borgia (2014):

  > The Internet of Things (IoT) is a new paradigm that combines aspects and technologies coming from different approaches. Ubiquitous computing, pervasive computing, Internet Protocol, sensing technologies, communication technologies, and embedded devices are merged together in order to form a system where the

real and digital worlds meet and are continuously in symbiotic interaction. (BORGIA, 2014, p. 1)

Ubiquitous computing is an area of computer science focused on making the interaction of machines and humans more natural and seamless (WEISER; GOLD; BROWN, 1999).

- Silva et al. (2016a, Sec. 1): "all application comprising objects or devices that can interact with other objects and applications over the internet."
- Alaba et al. (2017, p. 11): "IoT is a realm where physical items are consistently integrated to form an information network with the specific end goal of providing advanced and smart services to users."
- Muccini & Moghaddam (2018, Sec. 4.1): "IoT is the internal/external communication of intelligent components via internet in order to improve the environment through proving smarter services."
- Roopa M.S. et al. (2019, p. 32): "Internet of Things (IoT) paradigm connects physical world and cyberspace via physical objects and facilitate the development of smart applications and infrastructures."

The definitions presented may differ from the technological resources envisioned at the time they were concepted, but they clearly establish the purpose of IoT: the technology integration with the environment in order to deliver intelligent and seamless service to the user.

In this work, it is embraced a broader definition, as stated in Caldas Filho et al. (2017a, p. 195) summarizing ideas from Borgia (2014), Gubbi et al. (2013): Internet of Things "is a paradigm for building computer systems distributed throughout the Internet, in which, the most diverse devices, objects and things will be connected and interacting with applications to extend various services to people". Thus, it is also contemplated that IoT can allow interaction at any time, anywhere, with anything or anyone, ideally using any path and any device.

Furthermore, it is highlighted some issues that are already consensus among the authors: interaction of objects with the Internet, data center resource utilization, and Ubiquitous Computing.

Objects must have some type of internet connectivity. As needed, it should be possible to access or allow them to access the Internet. Authors in Gubbi et al. (2013) go further and argue that with the adoption of technologies such as Internet Protocol version 6 (IPv6), all objects should be published on the Internet.

To remain simple, smart objects, sensors and actuators must use stronger and external computational resources to store data as well as to extract value from this raw data. For this, data center resources should be allocated, especially in cloud computing.

Support for Ubiquitous Computing, or simply ubi-comp, is a hallmark and very intrinsic to IoT. Several authors of IoT works discuss and design systems announcing features "seamlessly" and "automatically". Although they does not mention this field of computer science, their work

is clearly influenced by ubi-comp without. Weiser, Gold & Brown (1999, p. 694) explain that "the physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives, and connected through a continuous network."

2.1.2 IoT reference architecture

The works in Guth et al. (2018), Muccini & Moghaddam (2018), Mineraud et al. (2016) explore the architecture of IoT platforms, presenting their features and characteristics. They also presents their considerations on open issues and gaps in the field. It is important to highlight the IoT reference architecture depicting its components and their intercommunication described by Guth et al. (2018) and shown in Figure 2.1 with its components are sensors, actuators, devices, gateway, middleware, and applications. The IoT reference architecture and its components will be discussed in the following paragraphs to support the reading of Chapter 3.

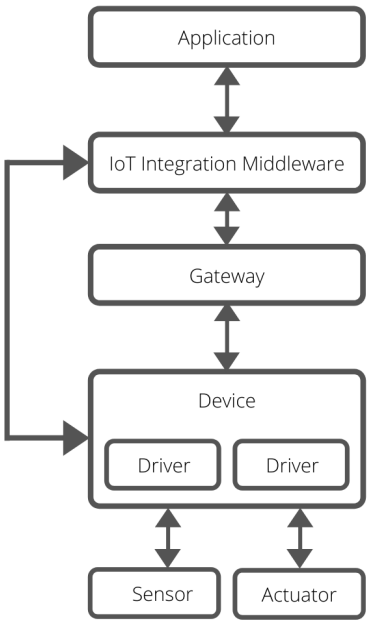


Figure 2.1: IoT reference architecture despicted by Guth et al. (2018, Sec. II)

Sensors are equipment deployed in the physical environment with the mission of reading a condition of that environment and converting it into electrical signals. They are connected or embedded to devices that are able to interpret and make sense of the electrical signals they receive from sensors. Sensors are commonly connected to device by wires, wireless or a cutting-edge shape.

Actuators are equipment installed in the physical environment capable of converting electrical signal into some action in the physical environment. In opposite way, they are capable of converting received electrical signal into some action in the physical environment. This action can be performed by creating some optical, sound, magnetic, electrical or mechanical effect according to the "command" received.

Devices, also called as objects, smart objects and smart devices, are hardware components that are capable of feeling or acting upon the physical environment in which they are inserted. Guth et al. (2018) also defines devices as hardware components that are attached to sensors and actuators by a software layer called Driver. Similar to the operating systems' "device drivers" described by Tanenbaum & Bos (2015), the drivers are the software layer capable of interpreting data exchanged with sensors and actuators. The set composed by device and sensor forms the IoT perception layer (MUCCINI; MOGHADDAM, 2018).

Guth et al. (2018) also point out that devices can be self-contained or connected to another system. In the former, it operates in isolation, forming a system that operates according to pre-defined behaviors. For example, an isolated automatic door that triggers its actuator to open the door as soon as its presence sensor detects a person approaching. In the latter, it operates interconnected to an IoT gateway or an IoT middleware. Authors in Shah-Mansouri & Wong (2018) mention that IoT devices can suffer from limited processing means.

An IoT Gateway is the component that helps devices to connect to further systems when they have any kind of limitation to reach the system. The IoT gateway is an optional component for devices that are not restricted to find their middleware or target system. To achieve this purpose, the gateway implements the necessary technologies to provide the interaction of the devices with the desired system. This gateway implementation includes protocols such as Message Queue Telemetry Transport (MQTT) and ZigBee, data representation formats such as JavaScript Object Notation (JSON), Extensible Markup Language (XML), and MessagePack, as well as target system-specific security procedures. (GUTH et al., 2018; CALDAS FILHO et al., 2017a).

Typically, IoT gateways are deployed on the client's local network. Considering its capillarity and proximity to the end user, it can be used to perform other tasks. For example, Caldas Filho (2019) proposes that it should optimize wide area network (WAN) communication from the local IoT network to the system's core over the Internet.

In a typically distributed systems that can be composed by heterogeneous components as IoT, it is necessary to include a "middleware" that can communicate and integrate with the various underlying networks, hardware, operating system, programming languages, protocols and patterns. The term middleware is used to define this kind of software that is at the center of the distributed system and the complexity around this kind of software is sometimes unseen because they operates over Internet protocols. (SOMMERVILLE, 2016; COULOURIS et al., 2012)

Sommerville (2016) also highlights that, in a broad sense, a middleware has two types of support: the interaction support and the provision of common services.

In the IoT reference architecture presented by Guth et al. (2018), the IoT middleware is called "IoT Integration Middleware". They described it as responsible for receive and evaluate devices data, as well it should send commands to the actuators. Mineraud et al. (2016) adds the concept of IoT platform that is defined as the middleware and infrastructure that enables smart object interaction.

IoT middleware has many features to make the IoT network work. Thus, they are complex software with robust architectures to be able to extract some useful information in the most diverse ways in which it receives the data.

IoT applications are software built to make use of the IoT network so that they can consume data from the middleware as well it can produce data to it. They are usually specialized systems that work in specific purposes. In other words, they are softwares that gain insight into the physical environment. Architecturally speaking, it is a freer layer that interacts with IoT middleware by Web services and is installed and used in many different ways. Actually, many of these IoT applications are in mobile architecture. (GUTH et al., 2018)

2.1.3 UnB IoT

The UIoT is an IoT project developed at the University of Brasilia since early of 2013. The term UIoT stands for UnB Internet of Things. Sometimes, it is also called Universal IoT because of its goal: propose an IoT architecture and middleware capable of tracking and reporting the current state of generic devices. Authors in Ferreira, Canedo & de Sousa Júnior (2013), Ferreira et al. (2014), Ferreira, Canedo & de Sousa Júnior (2014) gradually make progress, culminating in the first version of the middleware that was consolidated in the master thesis presented in Ferreira (2014). In the Ferreira's thesis, it is proposed an IoT middleware architecture to "track and notify current status of generic devices". In the proposal, it defines the use of Universal Plug and Play (UPnP) as well as specifies the physical and logical components of this architecture that allow interaction with these intelligent objects.

In 2016, it is presented the second version of the project. Initially, Silva et al. (2016a) refines and evolves this vision by consolidating its Application Programming Interface (API) and its architecture's protocols. Then, in Silva et al. (2016b), it is proposed to replace UPnP by a self-registration process in which the device itself, aware of its context, becomes responsible for the initiative of its participation in the network.

The UIoT middleware presented in Silva et al. (2016a) is a monolithic software composed of two modules, the REST API Approach for IoT SErvices (RAISe) and the User-friendly Interface Management System (UIMS), as shown in Figure 2.2.

RAISe is the middleware's core module and its features are: data persistence, provide API interface for access to IoT network data and services; authorization management of clients with network access, and; provide user interface for IoT network administrators. At both ends of its architecture are API interface and the database management systems (DBMS) connector which in turn is attached to the database. And, in between, it is a robust logical multilayered software, namely, Data Access Layer, Business Logic Layer, and Presentation Layer. In addition to providing the API, Presentation Layer also has an user interface (UI) that allows you to query data as well as manage middleware settings and operations.

UIMS is designed to provide data visualization on the IoT network, focusing on the end user.

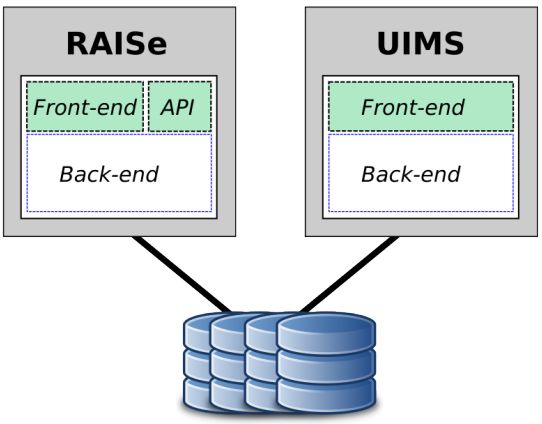


Figure 2.2: UIoT middleware architecture presented in Silva et al. (2016a)

To this end, its project comprises the use of graphical data display techniques, including time series.

Both modules used DBMS Couchbase to store their data. Couchbase is a NoSQL data repository that is a big data centric database.

Its design was focused on the cloud environment and caused middleware to take a lot of infrastructure resources to run satisfactorily. The middleware required the configuration described in Table 2.1 to service the approximately 30 devices that were part of its network at LATITUDE laboratory.

Table 2.1: UIoT system requirements

Software	**Requirements description**
Couchbase	16 vCPU, 16 GB RAM, and 1 TB of storage
RAISe	4 vCPU, 4 GB RAM, and 500 MB of storage
UIMS	2 vCPU, 2 GB RAM, and 500 MB of storage

2.2 CLOUD, FOG, AND EDGE COMPUTING

The need for resource optimization and the increased integration between systems led the information and communications technology (ICT) area to explore new computational models. This Section introduces cloud, fog, and edge computing models.

2.2.1 Cloud Computing

As stated by Mell & Grance (2011), cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction.

Computing resources have been shared since the 1960s when, to make the cost of operating large computers cheaper, their processing time was divided among several users. This sharing increased from the interconnection of these computers in communication networks, especially with the Internet.

In the early 2000s, e-commerce and online services were established as profitable business models and, as they were consolidated, it demanded more ICT infrastructure to support their business. So they evolved how to manage data centers so that services are delivered reliably and cost-effectively. Some companies that had a large structure to support their online businesses realized that they could "lease" some of their idle infrastructure resources. This model was named cloud computing.

This business model has been leveraged by technological advances such as increased Internet link throughput, data center virtualization, and the use of server allocation, configuration and management automation techniques and tools.

To regulate and encourage the use of cloud computing in the United States Government, the National Institute of Standards and Technology (NIST) produced some reports. In Mell & Grance (2011), authors define five essential characteristics all cloud service must exhibit: on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service. In an "on-demand self-service", the costumer performs by himself/herself all actions to provision computer capabilities without human interaction with the service provider. In a "broad network access", computer resources are accessible over the network through thin or thick client platforms. In a "resource pooling", service provider shares computer capabilities with multiple clients in order to resources utilization rates. In a "rapid elasticity", computer resources can be elastically scale by adding more resources and remove unneeded capability when demand decrease. In a "measured service", service providers should have metering capability in order to properly monitor and report the client resource usage and Quality of Service (QoS).

2.2.1.1 Hosting Models

Hosting models refer to the access and availability of the cloud computing environment. Computational clouds can be categorized as public, private, community, or hybrid (SOTOMAYOR et al., 2009; VOORSLUYS; BROBERG; BUYYA, 2011):

- Public Cloud: Infrastructure is maintained by a business, academic, government organization, or a combination of them. Users, knowing the location of the service, access this infrastructure using appropriate access control mechanisms.
- Private Cloud: Access to and use of infrastructure is unique to an organization and may be maintained by the organization itself, third parties, or a combination of them;
- Community Cloud: Infrastructure is shared by multiple organizations collaboratively and can be maintained by one or more of these organizations, and;

- Hybrid Cloud: Consists of two or more clouds of distinct hosting models.

2.2.1.2 Service Models

The service models that are offered in the cloud computing paradigm can be divided into categories according to the level of capacity abstraction provided and the service model of the providers. In the literature it is possible to find proposals with more than three categories. However, the most common is the division of services into Infrastructure as a Service, Platform as a Service, and Software as a Service, as they are described as it continues (VOORSLUYS; BROBERG; BUYYA, 2011):

- Infrastructure as a Service (IaaS): In IaaS the user controls the operating systems, storage resources and applications. Eventually, you can select network components. Thus, the consumer is offered computational resources essential for the construction of an on demand application environment, such as storage, network, among others. To enable interaction with computing devices, IaaS provides a single interface for infrastructure administration, storage and communication. In addition, IaaS provides support for adding new components in a simplified and transparent manner. However, cloud infrastructure administration or control is not the responsibility of the user;
- Platform as a Service (PaaS): In this model the user has control of the deployed applications and their settings. However, unlike IaaS, the user does not manage or control the underlying infrastructure such as operating systems, storage and network resources;
- Software as a Service (SaaS): the user does not manage or control the underlying infrastructure or individual application characteristics except very specific configurations.

2.2.2 Edge Computing

Some applications may require a very short response time. They often involve private data and can produce a large volume of data and can generate a heavy load on networks. Data is increasingly produced at the network edge making it more efficient to process the data also at the network edge. Thus, edge computing has emerged as a model which processing takes place at the ends of the network rather than at a central point (SHI et al., 2016).

In this model, the resources of an edge server are placed on the internet edge close to mobile devices, sensors, end user and emerging IoT. The edge is usually located just a step away from the end devices. Such devices are not only data consumers, but also serve as data producers (SHI; PALLIS; XU, 2019). Processing at the edge layers significantly reduces the communication volume between devices and the processing core, consequently network traffic, the resulting costs, latency, and improved quality of service (CHANG et al., 2014).

Figure 2.3 shows, in the edge computing paradigm, bi-directional computing streams, which

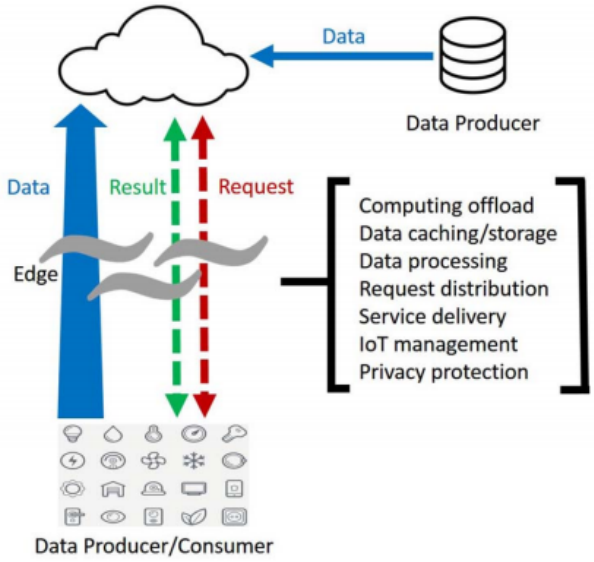


Figure 2.3: Edge computing paradigm in Shi et al. (2016)

devices can not only request cloud services and content, but can also perform cloud computing tasks. One of the bi-directional computing streams is from devices to the cloud (upstream) and the other is from cloud to devices (downstream). Edge can distribute request and delivery services from cloud to user, store data, cache data, perform compute offloads, and perform processing (SHI et al., 2016). Edge computing is related to grid computing, which tasks are divided into several machines. Most of the time this division is done manually with source code change. The edge computing replica code fragments, as well as relevant information, create a template, facilitating the distribution of tasks between devices (SCHENFELD, 2017).

2.2.3 Fog Computing

Fog is a paradigm that has emerged as a distributed computing infrastructure, which applications and services can be handled on cloud servers and on the network itself (YI; LI; LI, 2015). It provides computing, storage, and networking services between end devices and traditional cloud servers through a highly virtualized platform (BONOMI et al., 2012). Moreover, fog computing encourages data manipulation at the edges of a network, ie, instead of sending all the collected data to the cloud, it is suggested to process the data at the edges. This idea is also called edge analysis. Local data processing helps mitigate the weakness of cloud computing (PERERA et al., 2017).

Schenfeld (2017) states that the fog concept arises to meet three main objectives:

- Improve efficiency and reduce the amount of data that needs to be transmitted for processing, analysis and storage;
- Bring the information consumer closer to the data provider, and;

- Provide security and compliance for data transmission.

Fog computing extends the computing cloud, transferring resources, services, and data to the edge of the network. This avoids network bottlenecks, brings user content and computing closer, reduces network latency, and improves system performance and user experience. It also provides next-hop processing, empowering IoT, relieving the massive data network (NIKOLOUDAKIS et al., 2016).

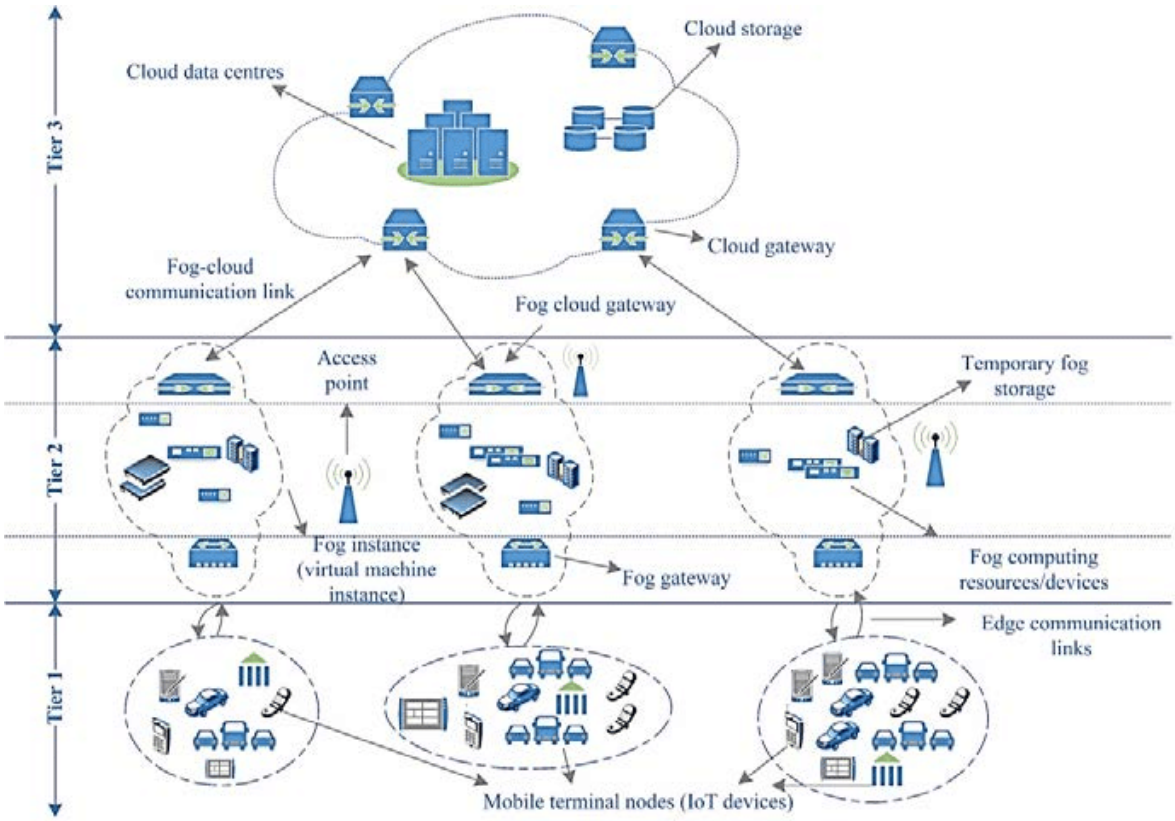


Figure 2.4: Fog computing paradigm in Taneja & Davy (2016)

As shown in Figure 2.4, Taneja & Davy (2016) structured the fog computing architecture in three layers, to facilitate the visualization of its components and the identification of its roles. The first layer is the end devices and terminals layer and this layer the data generation and consumption. The second layer is the edge/fog layer, consisting in network infrastructure as well as the intelligence for edge/fog resource management near/into the network. Then, the third layer is the cloud computing layer, consisting in the cloud resources.

2.3 MICROSERVICES

The term Microservices, also called Microservices Architecture or MSA, refers to an architectural style in which a single application is built by the composition of (sometimes thousands) microservices. Each microservice runs on its own process and communicates over a lightweight

network infrastructure and protocols.

A microservice is a small service with a single responsibility and it has its own and independent infrastructure. “Small” means that the service must have a single responsibility. “Autonomous” means the service must have its own and independent infrastructure.

The works in Alshuqayran, Ali & Evans (2016), Pahl & Jamshidi (2016), Thones (2015) help to visualize the current scenario and the paths that Microservices tends to follow. The first two review the literature on microservices and the third brings an interview that eases the first contact with the area.

Lewis & Fowler (2014), Newman (2015) report that the Microservices architectural style was not “invented”, but emerged from the experience of different software teams with diverse techniques, patterns, practices, and tools. Both works also report that the Microservices architectural style was not invented, but arose from the experience of different software teams with diverse cultures, techniques, patterns, practices and tools. The term “Microservices” was first used in 2011 in a software architecture workshop, and until then, several terms were used to describe this approach.

Microservices Architecture arose in contrast to the architectural style massively used hitherto in the corporate environment: Monolith. Dragoni et al. (2016) says “[a] monolith is a software application whose modules cannot be executed independently”. This means the entire software is built in a common technology set over the same set of application servers and database servers. Thus, any addition or change of language, framework or database affects the entire application (NEWMAN, 2015). The way to scale the application on both architectures is another major difference highlighted in the Microservices literature. As described in Figure 2.5, with Microservices you can scale only the services that are most in demand. Whereas, in Monolith, the entire application needs to be scaled.

In these experiences, Newman (2015) states that organizations realized that they could deliver software faster when they adopted fine-grained architectures. Thones (2015) adds that many organizations started deploying large systems and databases and empirically came to microservices from the breakdown of these large systems. As emblematic examples, he cites Netflix and Amazon.

Newman (2015), describes the following Microservices influences:

- Domain-Driven Design (DDD), showed the importance of representing businesses in the way it organizes applications. Thones (2015) points out that DDD has also proposed how to divide a big problem into small domains that can be dealt with, using concepts such as a strategic model, limited context, subdomains, and domains;
- Continuous delivery is a practice in which software can be deployed in production after any change to its source code. Either for building a new functionality or for correcting an error. This idea showed how to make the software deployment process more effective and

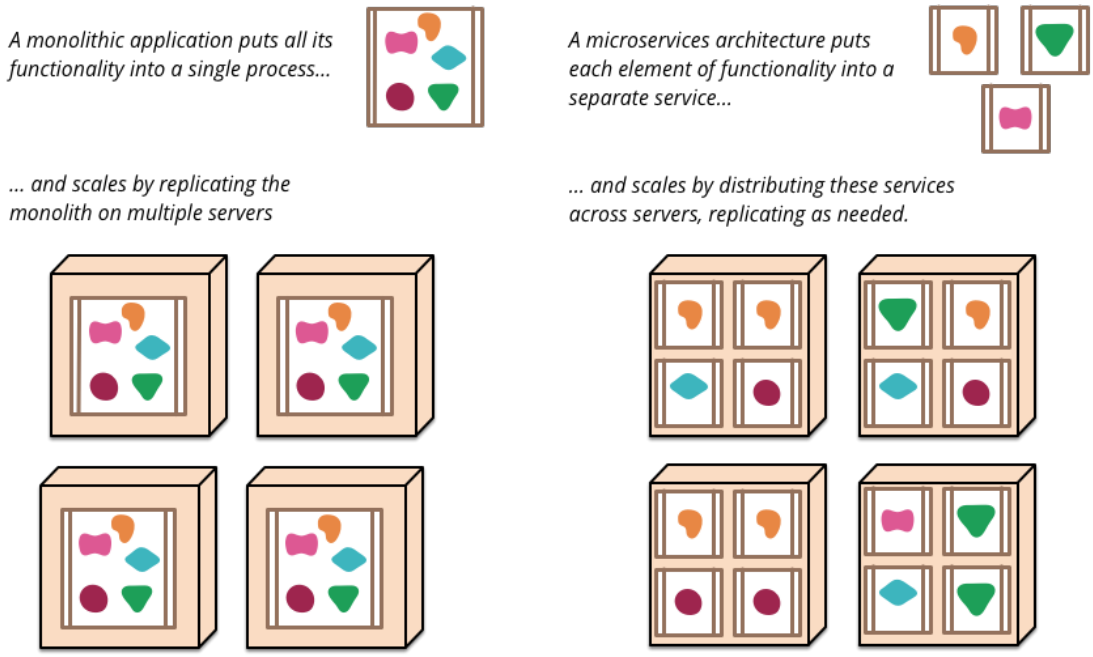


Figure 2.5: Lewis & Fowler (2014)'s sketch of Monoliths vs Microservices

efficient by allowing each submitted code change to be treated as a candidate version of this software;

- Use of the Web as it was designed, following its concepts and standards. In this sense, it relies on REpresentational State Transfer (REST), an architectural style proposed by Fielding (2000), in which Web applications must identify their resources through URI, use uniform interface and self-describing messages and make non-persistent state communications (state interactions must occur through links). Several of the proposed pillars in the REST are used to support microservices;

- Hexagonal architecture is an architectural pattern aimed at allowing a software or a component to be equally workable by users, programs, automated tests and scripts using appropriate adapters. Its goal is to protect the program logic from the idiosyncrasies of each form of use;

- On-demand virtualization enables machine provisioning and scaling as needed. Combined with infrastructure automation, it allows you to manipulate these machines to scale the application;

- Structuring software developers into small, self-contained teams responsible for the entire life cycle of their services. This strategy is used by companies like Amazon and Google, and;

- Design of fault-resistant software, anti-fragile software. In this principle, the software must be able to adapt to chaotic scenarios and continue to function in the way it was designed. Or-

ganizations that adhere to this philosophy cause errors in their infrastructure when verifying that the software behaves properly.

Dragoni et al. (2016) considers Microservices to be the second generation of service-oriented architectures evolving to Service-Oriented Architecture (SOA). The authors also highlight the concepts that are obtained from SOA. These being: split application implementation into services and service choreography over orchestration. They also highlight the fundamental differences that separate SOA and Microservices. Such as: the size of the service should be small, having only one responsibility; the delimited context, imported from DDD, which combines related functionalities in a single organizational capacity, and; operational independence between the services in which each part of the service must be built and performed independently of the others.

In Lewis & Fowler (2014), authors emphasize that the channel is another big difference between the two generations of service-oriented architectures. SOA relies on middlewares as Enterprise Service Bus (ESB) to act as a service integration bus. ESBs are products with various features and facilities. Such as: message routing, data transformation, protocol transformation, and business rule enforcement. However, for microservices, the channel is solely a means of transport, and Lewis & Fowler (2014) explain this approach with the expression "smart endpoints and dumb pipes".

To meet an application's growing demand for resources, it is desirable that it be scalable. The Microservices architectural style provides facilities for increasing the capacity of an application: it is enough add new instances of its microservices to be made available, and when they are no longer needed, these instances can be shut down. Importantly, as each microservice is independent, they can be augmented individually without impacting other microservices. This feature helps you benefit from the virtually infinite processing power of cloud computing, as stated in Botta et al. (2014). To accomplish this, it is especially important to use fully automated deployment machinery and, in a broad sense, infrastructure automation comprising a whole movement of best practices in operations automation coupled with the programmable infrastructure, also called "infrastructure as code" (THONES, 2015).

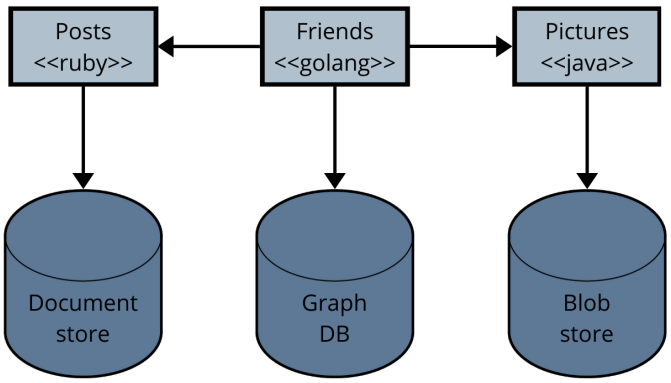


Figure 2.6: Newman (2015)'s example of how Microservices can painlessly lead to a heterogeneous architecture

2.3.1 Containerization and Docker

Cloud computing has spurred demand for the deployment of computing environments. Many solutions are available for this purpose and often focus on the concept of containers. I.E units that group all dependencies to run an application from one host to a guest computing environment. Newman (2015) advocates the adoption of infrastructure automation practices and, through containers, it is possible to implement such practices. Thus, the use of Docker containers has been adopted to simplify software deployment (TRUYEN et al., 2018).

A container is a standard unit of software that packages code and all its dependencies so that the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable software package that includes everything needed to run an application. These being: code, system tools, libraries, and system settings. (DOCKER INC., 2019).

A Docker-based approach works similarly to a virtual machine image in addressing the dependency issue, providing a binary image on which all software has already been installed, configured, and tested (BOETTIGER, 2015). Using containers provides an execution environment where a context is created and the execution of software processes is isolated. Thus, software processes assume they are running on a virtual machine dedicated to them.

It should be noted that it has been common to use the term "Docker" as a synonym for containers, but there are other tools with this same purpose. This association is because the Docker tool was one of the first in the market and is used by the vast majority of practitioners.

Docker's approach has to offer a simpler configuration and parameter consistency solution for applications through the use of images. Using containers allows software and its dependencies, such as databases and supporting software, to run on multiple computing platforms, significantly reducing the time and effort required to set up the environment. Therefore, software deployment and configuration becomes simpler as you just set the general parameters of execution of the account instead of configuring details of the execution of each software or any of its dependencies (KIM et al., 2017) .

2.4 RELATED WORKS

This Section briefly describes works related to the proposed research. To delimit the analysis, it was considered only projects that explicitly address the use of any technology, for example fog computing or hierarchical relationship, or any issue mentioned in this paper, for example off-grid working. Then, the differences between this proposal and these works is pointed out and discussed.

There are some researches about **edge and fog computing** in the IoT universe. In general and as it is stated in Section 2.2, studies on the use of these technologies were started to save data

traffic between local area networks (LANs) and the cloud. Next, it is highlighted and discussed some of these works.

Schenfeld (2017) presents the Fog-as-a-Service for Internet of Things (FaaS4IoT) and the proposal for an IoT architecture with fog and edge computing for smart cities domain. Thus, the author allows devices to be able to act within their context even if they are temporarily disconnected from the internet.

Chang, Srirama & Buyya (2017) points out that fog computing is a response to cloud-centric IoT's limitations. But it also has its own limitations such as being tied to the provider's platform or working only on cellular networks. To address these issues, they propose Indie Fog, a platform that enables fog computing within the user's own network. The platform provides an IoT application execution environment, performing infrastructure actions and leaving its users to worry only about these applications.

As in this research, these three works address the use of fog computing in the context of IoT in order to decrease application latency and make it more reliable for scenarios that rely on real-time or near real-time processing. However, in addition to using fog computing, this work combines the use of fog computing with other technologies and strategies, as well as giving the user the flexibility to configure as needed for their reality.

Nonetheless, it was noted that Chang, Srirama & Buyya (2017) has a similar feature to this work: it is flexible about deploying modes to configure its fog environment, allowing it to connect to other fogs in a software-defined mode. But this proposed work goes ahead and takes away the obligation for cloud connection, achieving even more restricted usage scenarios.

Hierarchical relationship between IoT instances is evaluated in some works. Azimi et al. (2017), Chekired, Khoukhi & Mouftah (2018), Shah-Mansouri & Wong (2018) were set apart and discussed as it follows.

Work in Azimi et al. (2017) Hierarchical Fog-Assisted Computing Architecture for Healthcare IoT (HiCH) that is a hierarchical fog-enabled IoT system for remote health monitoring. As a fog-enabled solution, they split features between the system core, in the cloud, and the nodes located near the patients. They evaluate their proposal in a scenario in which HiCH monitors ECG signals to detect patient's arrhythmia. In the results, the paper demonstrates the traffic reduction and the reduction in response time to a sensor measurement, so that the system can react faster.

The authors of Chekired, Khoukhi & Mouftah (2018) point out that in the industrial scenario, the information collected must provide immediate feedback to the control system and response mechanisms. Their paper proposes a fog architecture for IIoT based on hierarchical multi-tiers servers. They also address system performance issues through a probabilistic algorithm for resource allocation and utilization of fog nodes and through a priority queuing model that separates priority and non-priority data.

The work in Shah-Mansouri & Wong (2018) argue that there is an competition among IoT users in order to use processing resources. Which in turn implies in different and unfair perfor-

mance, delaying their tasks offloading. Thus, they propose a game theoretic approach and then they formulate a computation offloading game to validade their proposal.

The three works propose solutions for handling with systems that are sensitive to communication latency, namely, health assistance, industrial automation and computational offloading tasks. As in this research, their papers suggest using distributed processing between a remote cloud module and modules running within the local network. Following the fog computing model, it was used the strategy of splitting features and coordination of this distributed processing. However, they merely address the issue of hierarchy between IoT instances. Beyond this scenario, this proposal goes further and allows the middleware to work in ways other than hierarchical.

Providing IoT with properties and techniques of **social networks** is a novel strategy that aims to counteract the limitations of rigid and centralized networks. Below it is highlighted some of these works and the relationship with this work proposal.

Authors in Conti, Passarella & Das (2017), Conti & Passarella (2018) opinion papers point out that three factors that are happening on the Internet today: increased communication at the edge of the Internet, rather than at its core; the internet is becoming increasingly data-centric, and; the expansion of the integration of the Internet with the physical world. Considering these factors, they present the Internet of People (IoP) which is defined as the new generation of people's communication. In the proposed IoP, communication and interaction takes place around the user and their mobile device, so that this device is the gateway to user interaction with the network. It also plays an active role in content production and decision making, both locally and collaboratively.

The works bring interesting discussions about "Next Generation Internet". However, this work differs particularly in two aspects: the first is that the middleware was designed to be configured in scenarios other than Ad Hoc-like defined by them. This is an important aspect of the proposal, but the work proposed in this thesis aims also to support those users with more traditional needs such as the hierarchical relationship. The second is that the proposed work is not limited and can be adapted for use on user's mobile device, but it is not limited to this scenario because some scenarios do not occur close to the user, such as in the forest monitoring.

According to Abdul et al. (2018), the actual large number of devices (and estimated for near future) combined with the flexibility of interconnections between them results in scalability and navigability issues. Their work proposes the "small world SIoT paradigm" in which they merge properties of small world networks concept with SIoT in order to reduce network complexity and improve its performance. They define small world network as a kind of small social network.

The work has good contributions and its proposed Small World SIoT is worth evaluate it in the SIoT context. However, they focus only on the social networking scenario and, as stated in the analysis of previous works, there is still a need to provide services in off-grid mode, in hierarchical scenarios and also for cases not centered on their user behavior.

3 PROPOSAL OF AN ADAPTABLE AND SCALABLE IOT MIDDLEWARE

In this Chapter, it is proposed the adaptable and scalable universal IoT middleware which is a flexible IoT middleware that is intended to allow IoT networks creation and that can be used in various scenarios and yet perform well and offer ownership of data to its owners. This Chapter is organized as it follows: in Section 3.1, it is presented an overview of an IoT instance and its components as they were used them in this work. In Section 3.2, the concept and features of the IoT middleware is discussed, since this concept is a key idea to build the adaptable and scalable IoT middleware. In Section 3.3, it was presented the middleware design, its architecture and assembling of the components of the hardware. Finally, in Section 3.4, it is highlighted the middleware set up factors.

3.1 THE IOT INSTANCE

Authors in Ferreira & de Sousa Júnior (2017) point out that there is not a single and homogeneous IoT, but various "instances" of IoT, as shown in Figure 3.1. In this sense, they meant these IoT "incarnations", as they also called it, coexist in order that each incarnation could be an overlay network on the Internet infrastructure. An IoT instance is an IoT system that includes at least one device, middleware, and application.

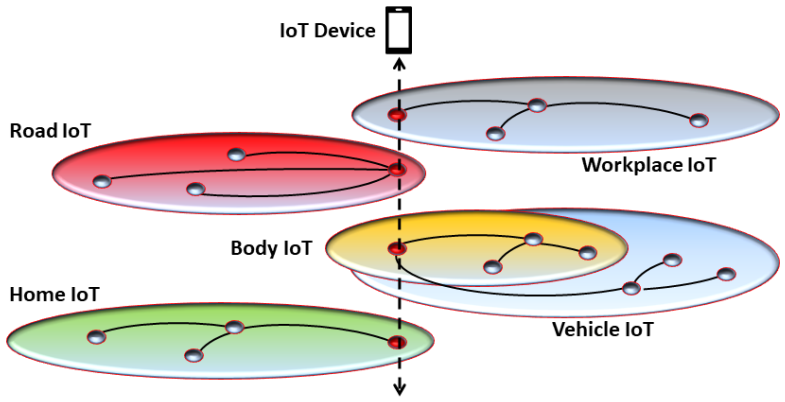
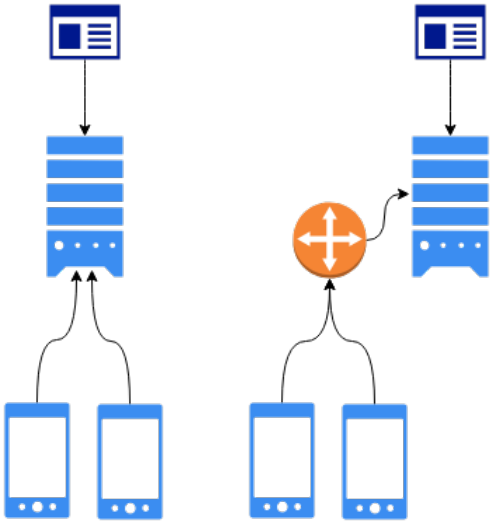


Figure 3.1: Ferreira & de Sousa Júnior (2017) sketch of different IoT instances that a moving device can be part of

In this work, the considered instance is according to the reference architecture presented in Section 2.1, composed of devices, gateways, middleware and applications.



(a) IoT architecture without a gateway

(b) IoT architecture with a gateway

Figure 3.2: IoT architecture

Note that the proposed IoT platform can be used both in the scenario that devices are able to access it directly, as shown in Figure 3.2(a), and where devices need an IoT gateway to access middleware, as shown in Figure 3.2(b).

3.1.1 Supported IoT Entities

The IoT devices supported in this work can be either smart devices that have the ability to interact directly with the middleware, as well as the simplest common devices that don't have this capability. Regardless, they must provide sensing or actuation services so that the IoT instance can interact in some way with these services.

In this thesis proposal, the UIoT Gateway is an IoT semantic gateway as described in Caldas Filho (2019). Thus, it is responsible for allowing limited devices to reach the Internet and IoT middleware. The UIoT Gateway must also provide means for performing fog-style operations in the network edge.

In addition to the native UIoT gateway, middleware allows interaction with third party IoT gateways as long as they respect UIoT interaction processes.

The IoT applications considered in this research are the applications described in Section 2.1 that make use of the services available by the IoT instance. These applications could be part of the UIoT as well as a third party application that interact with UIoT by the provided API. It should be highlighted that, regardless their nature, in both cases, the application should only have access

to information that is allowed for them to be used.

It is important to emphasize that the proposed middleware does not bring new architectural requirements to these applications. Accordingly, it is not imposed or constrained on the architecture of applications, they simply need to respect the middleware API.

3.2 THE PROPOSED IOT MIDDLEWARE

The IoT middleware proposed in this master thesis in an extension of the UIoT middleware presented in Section 2.1.3. UIoT is described in Silva et al. (2016a, Sec. 3.1) as a general purpose middleware that has a flexible and open architecture allowing the most diverse IoT devices to participate in its network. The Adaptable and Scalable UnB IoT middleware is a type of IoT middleware with the functions described in Section 2.1. Additionally, it is able to be executed in different ways such as in an "isolated" way as well it can be used in a "cooperative" way. In this sense, the term isolateddenotes two aspects: one refers to its connectivity configuration and the other is about its relation with other middlewares. The former means a middleware must be able to work in a constrained environment in terms of connectivity and it implies a middleware instance should work off-grid. Therefore, isolated from the Internet, it also should be able to handle with unstable connection. The latter means a middleware instance can be used without another instance and it implies the middleware has all resources in terms of storage, processing and networking, and capabilities it needs to play its role. In this way, it is the only IoT instance.

Despite of having isolation capability, the proposed IoT middleware can interact with other instances to provide and to consume provided services. It was designed for being used in various configurations and scenarios as it will be described in more details in Chapter 4. But, for instance, it is stated it works isolated with a single running instance or it can iterate to other instances and its way of using. Its relationship to other instances depends on security issues as well as its policies and settings.

The interaction between middleware instances is critical for allowing the expansion and optimization of available resources, as well as giving the system more flexibility. This aspect deserves more attention and will be discussed in more detail in this proposal.

3.2.1 IoT Middleware Features

An IoT middleware usually should be ableto store data streams from devices, manage devices, allow data visualization, secure data and services, and interact with devices and applications. Furthermore, it is proposed that the IoT middleware should also be able to be adaptable, distributable, scalable, and resource-sharing prone. These features will be discussed in the following paragraphs.

Ordinary IoT devices are usually resource-constrained in terms of storage, processing, power

supply, and connectivity. In a broad sense, each device has a constraint of sorts. Because, even though those devices have more processing or storage resources, they have a limited view of their own context. In this sense, the lack of information limits their context-building and their analysis skill. Thus, the proposed IoT middleware should offer data storage for its devices and allow them using these data as well other users and clients should be able to visualize that data.

IoT devices and applications are usually restricted in one function and they have fine granularity focusing on only one responsibility. Hence, their interaction with other devices and applications is limited to the context of performing their responsibility. This behavior makes room for some components responsible for making these entities work together or coordinate. The proposed IoT middleware defines the operation of its IoT network and, to that end, manages the participation of devices and applications in it. Importantly, middleware management is not invasive on the device or application, so it comes down to managing device behavior within the IoT network allowing, or not allowing, that device's or application's operations.

As a manager of the IoT network, the proposed IoT middleware knows the status of each participant in it. This makes it able to facilitate interaction with services provided by each of these members. This way, the proposed IoT middleware interacts with the provided services, devices, and applications of the IoT instance, as well as can broker the interaction between them. As the IoT middleware has the IoT network big picture, including its surroundings, it is the most appropriate agent to secure the network, controlling admission and discovery of services and devices, as well as monitoring these participants.

So far, these features are common to other IoT middleware as described in Section 2.1. But this master thesis proposal is to present an adaptable and scalable IoT middleware so, in this sense, it is highlighted its further features in the following paragraphs.

Unlike monolithic software and being fog-backed, the proposed IoT middleware should allow its components could be distributed to other nodes so that these nodes operate in an integrated and seamless manner as a single instance middleware.

The proposed IoT middleware must adapt to varied scenarios in terms of features and configurations so that it can work normally under a variety of different conditions. This adaptive feature allows to create a cloud-based IoT instance with LAN resources and even without a stable Internet connection.

In addition to the resource scaling already offered by the operating system or platform on which middleware is deployed, the proposed IoT middleware can scale the working capacity of its IoT network. This feature can be achieved by scale-out and cooperation. The first is the distribution of its components in other nodes, with the possibility of replicating any of these components, performing horizontal scaling. The other is cooperation with other instances to leverage idle resources they are willing to share.

The proposed IoT middleware must provide interfaces for sharing services with other nodes or instances. In any middleware instance, whether it is installed on a computer, a server, or a virtual

cloud server, these features can be made available through a uniform interface, so that only the policies of each instance and the trust relationship that determines how this resource sharing will occur.

3.2.2 Relationship between IoT Instances

The proposed IoT middleware provides the ability to interact with other IoT instances, so the relationship between them is the mechanism that gives the system the flexibility to suit the IoT instance usage requirements.

The interaction between instances is a relationship that must be observed individually in both directions and that results in the desired behavior for both. This means that it must be defined in each instance according to its interests, policies and configurations. Importantly, the relationship between two middlewares means that both agree with their role in the relationship and that its role varies according to the types of relationships established between them.

This setting must be made on each instance according to local policies and the configuration made by its administrator. This configuration consists of two definitions: the definition of the policies and parameters used by the instance for its operation and the definition of its relationship with other instances. This approach was chosen because it is a task with some degree of decision and willingness of the owner of each instance.

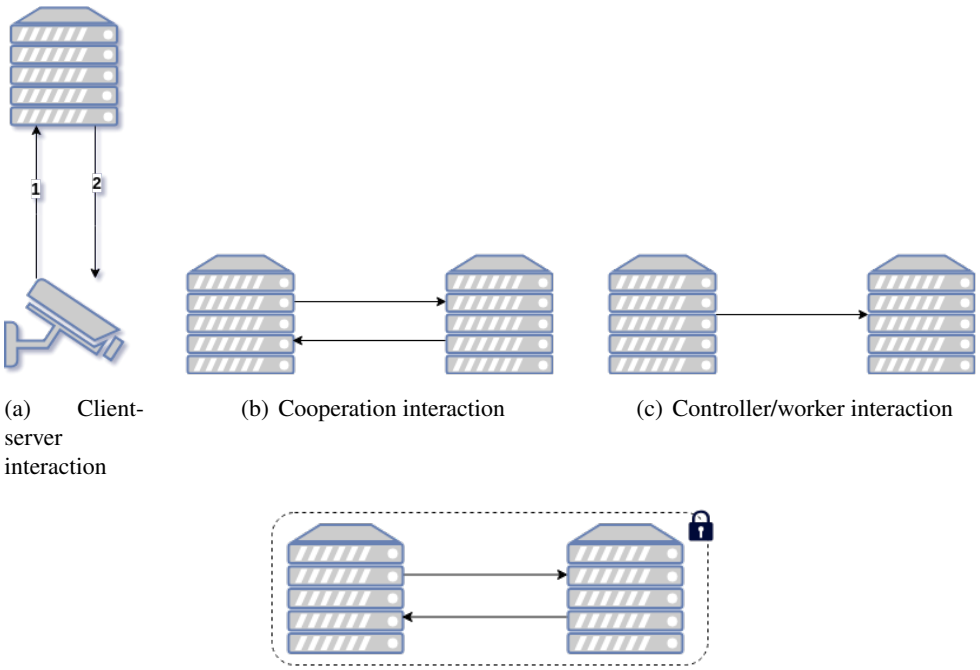
Thus, it is necessary to define the types of possible relationships between the instances and what each type of these entails for both parties involved. As described in Figure 3.3, the proposed IoT platform supports the following relationships: client-server, controller-worker, cooperation, and federation.

The client-server relationship is one in which an instance acts as a service provider for the client. This relationship is most common when thinking of cloud-based middleware with clients on LANs. In this relationship, there is no hierarchical relationship between the components, but the server has autonomy to define how their services will be made available. This type of relationship occurs when one instance consumes data or actuator services from another instance.

The controller-worker relationship is one in which an instance acts to control the relationship with the ability to determine rules for the worker. This relationship is most common when thinking of cloud-based middleware with clients on local networks. In this case, there is a hierarchical relationship between the components, so that the controller has the mission to define the behavior of workers and the network as a whole.

The cooperation relationship is one in which the two participants communicate “voluntarily” to share information or services. In this relationship, neither party has the ability to command the other, but resources are provided according to the rules of the sharing instance. However, interaction mechanisms can be used to suggest behaviors.

Resource sharing can occur widely. For example, an instance “A” could share its data connec-



(a) Client-server interaction

(b) Cooperation interaction

(c) Controller/worker interaction

(d) Federation interaction

Figure 3.3: Middleware's types of supported interactions

tion and act as an instance "B" gateway. In this case, instance "B" may use instance "A" as the path to reach the network or some resource while that network connectivity is available.

The federation relationship is one in which an instance joins with other instances to contribute to the formation of a new instance by combining the joined instances. The federated instance can be viewed as a single instance by its clients and members, but in fact its services are provided by the instance members.

For example, the joining of IoT instances in P2P networks that is proposed in Mendonça (2019). In this case, an instance "A" can join instance "B" to form a new federation or it can join an existing federation. Within the P2P protocol adopted in that work, the devices of both instances assume a new address in the federation and, from there, they can produce and consume other P2P instance services.

It is important to highlight that the relationship between middlewares is not directly related to the topology of the networks in which they are deployed. Conceptually speaking, there is also no limitation on the technology used for the connection, as long as the communication protocols are supported by both entities.

The only relationship requirement is that there must be connectivity between the two communication agents and this connectivity must be through some protocols that both parties support.

3.2.3 Trust between Instances

Regardless of the purpose of the interaction between instances, the relationship of trust between them is the cornerstone of both interaction operation and supporting security requirements.

Trust is a transitive feature which varies according to each of its components: subject and object. This implies that the relationship between agents "A" and "B" must be observed in both directions: "A" has its evaluation on "B" and "B" has its on "A". It is important to remember that an agent's trust in another is an inner attribute that is not reported and neither reachable by external agents.

3.3 IOT PLATFORM DESIGN

The use of centralized and cloud solutions brings its own limitations such as relying on Internet connectivity and accepting the risks of sharing data in someone else's cloud. Aware of this kind of limitation, this work proposed the evolution of UIoT middleware to use distributed and adaptive architecture, considering current computational models trends such as edge, fog and cloud computing. As described in Section 2.2, fog computing architecture is based on the composition of multiple nodes on the local network that can act in conjunction with cloud services. It is also anticipated that nodes can work together to share resources and give to the solution more capacity and flexibility.

Another issue is that because cloud-ready software is concerned with being scalable, it is often designed to be greedy in terms of resource usage. It is exposed by its operating requirements of some IoT platforms as, for instance, the UIoT operating requirements which is unveiled in Table 2.1. Although easily deployable in a cloud server, this configuration does not allow middleware deployment on SBCs such as Raspberry Pi 3 and Orange Pi that have 1 GB and 2 GB of RAM respectively.

3.3.1 Middleware Architecture

The architecture design of this work came from the UIoT architecture described in Silva et al. (2016a), Ferreira & de Sousa Júnior (2017), whereas it is a mature and feature-rich platform for managing IoT devices. It is a monolithic, cloud-ready, big-data-ready architecture and it relies on providing REST services to communicate with devices.

3.3.1.1 Core Drivers

To meet the intended features that were elicited in Section 3.2.1, this work needed to review some five key points of the UIoT middleware architecture. These points are discussed as it continues.

The first key factor is the structural aspect: the proposed IoT platform separates the core from other system modules. This separation purports to encapsulate the modules responsible for data and external dependency decisions. This was the first structural change from UIoT because it concentrated almost all of its functionality in just two modules and DBMS was closely coupled with the operation of both modules, as shown in Figure 2.2.

The second factor is about implementation encapsulation: the proposed IoT adopts the strategy of encapsulating the implementation of its software modules so that their interaction occurs only through publicly defined interfaces. This feature is an inheritance from the Microservices Architecture that aims to enable the implementation and deployment of modules to change without impacting their consumers.

The third factor is about the communication channel: the proposed IoT uses the network as the default communication channel, especially the HTTP protocol, in almost all communications between its modules and its modules with external actors. This strategy aims to maintain a standardized and uniform shape to enable integration with more tools as well as avoiding the adoption of tools for building service buses. The use communication over network protocols should be even when middleware is on a single host. It should be noted that, in this case, their communications occur in a private local operating system protocols. Nonetheless, both RAISe and UIoT's UIMS have monolithic architecture, so that the internal communication of their modules takes place directly between their implementation layers.

The fourth factor is about deployment: middleware must have an adaptable and pluggable framework to enable it to be used in a variety of scenarios. This factor is influenced by the strategy of avoiding making multiple implementations for different scenarios, so that it was used the reverse strategy of having one code. UIoT is designed to run in the cloud, with the ability to scale resources as needed, but it cannot be used in lower-resource environments as it requires resources like those described in Table 2.1.

The fifth factor is about technological decoupling: the proposed IoT fosters the separation between the functional and non-functional side, another feature inherited from the Microservices Architecture. This means that the relationship between modules should occur only by the defined interfaces and that no interface should expose technological aspects of its implementation. For example, database technology must be transparent and indifferent to middleware customers. The main expected benefit is the possibility of exchanging the technologies used without impacting customers.

3.3.1.2 Refactoring for the Microservices Architecture

The first phase of building the proposed IoT middleware was to refactory the UIoT middleware from its monolithic style to Microservices Architecture (MARTINS et al., 2017b). In the refactoring, all UI functionality that was in RAISe were moved to a module with the UI handling single responsibility. Following this refactoring, the new middleware was adapted to be

compatible with the system features provided by Raspberry Pi.

Microservices Architecture has influenced a number of decisions about the design and implementation of the proposed IoT middleware, as described in Martins et al. (2017b), Martins et al. (2017a). Specially because maintaining and developing UIoT middleware has shown that its monolithic architecture would make it difficult to deploy the system on small architectures. It will be highlighted some of these influences here to make it easier to understand how the proposed middleware works in light of this new architecture.

MSA microservices are similar to the standalone, reusable components of the Component-Based Software Engineering (CBSE). Both strategies bring the idea of reducing the problem to be solved in smaller, easier to implement parts, as well as the encapsulation of the implementation that must be protected by its interface. However, MSA adds issues such as the use of a uniform and standardized network-based interface and the infrastructure independence of each component. Therefore, it was defined by using REST APIs as the standard way of communication between components and between components and clients. The only exception is communication with devices that is augmented by specific technologies and protocols.

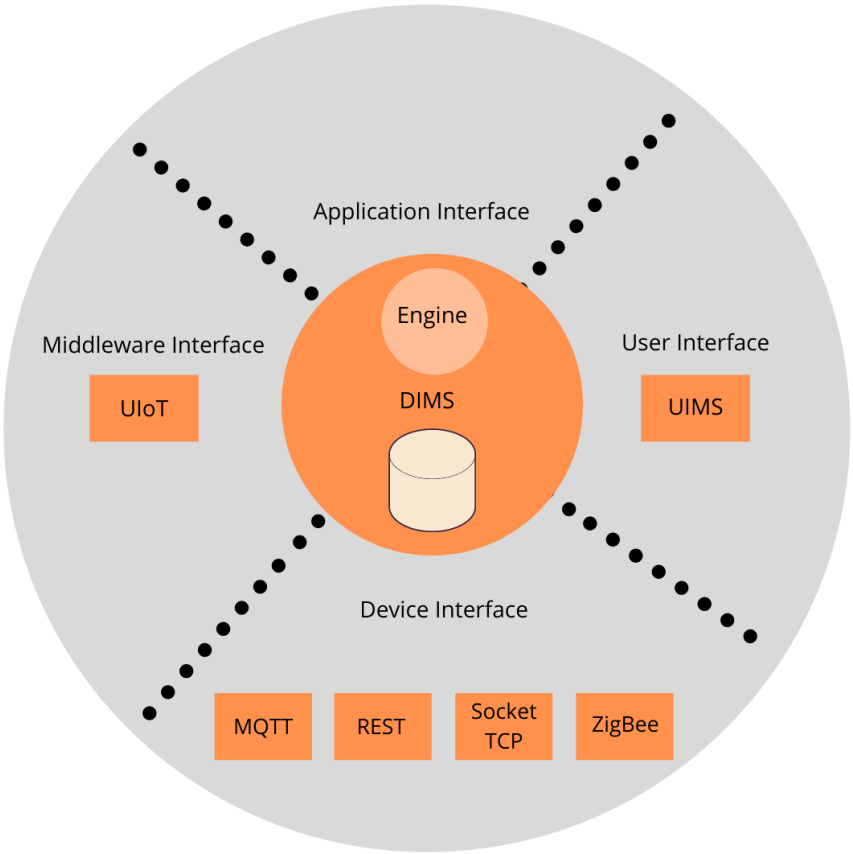


Figure 3.4: Proposed IoT middleware architecture

Granularity of the microservices is a topic under discussion in industry and academia, as it is presented in Shadija, Rezai & Hill (2017), Harper et al. (2016). In this work, microservices were defined according to their purpose, so that each middleware component described in Figure 3.4

represents a microservice. The only exception is the Engine that has more than one microservice, but this decision follows the same strategy of defining service according to its purpose.

The microservices infrastructure is designed to preserve its internal structure, so that no component depends on internal parts of another component. As a result, the data persistence of each component is completely separate from the others, so that no component is designed to access data from another component outside its access interface.

3.3.1.3 Middleware Architecture Components

The UIoT middleware architecture was the starting point and also a cornerstone for the design of the new architecture. Suitability for the five factors presented earlier in this Subsection would need to retain existing features in the original version, such as cloud-ready and big-data-ready, as well as universally address IoT scenarios.

Thus, the proposed IoT middleware's architecture was designed to have a core as a subsystem surrounded by layers of interfaces that allow the outside world to interact with the core, as illustrated by Figure 3.4. It was designed to separate into smaller parts that could operate together and this separation was performed on components and subsystems. For the concept of components, it is used the definition of Sommerville (2016, p. 467-470,760) who stated it as "[a] deployable, independent unit of software that is completely defined and accessed through a set of interfaces."

3.3.2 Middleware Abstract Interfaces

The abstract middleware interfaces are conceptual layers in charge of intermediating the interaction of external agents, whether human or machine, with the middleware core. The proposed IoT middleware has four interface modules, each with its own purpose and since the purposes of the interfaces are quite different, each module has its own architecture, specification and implementation, so the structure and modularization of the software to meet this layer depends on specific characteristics linked to its purpose. These modules are described in this Subsection.

3.3.2.1 Device Interface

Device Interface is an abstraction that defines the responsibility of making interfaces available to devices that integrate with the IoT network. Its responsibilities include providing an interface to receive device requests as well as sending data and commands to devices.

The Device Interface must implement the communication technologies and protocols necessary to enable communication with the target devices. Because this implementation is tied to the physical layer, technologies suche as Bluetooth and ZigBee require appropriate hardware to provide the communication means. This also occurs with the implementation of some software-defined technologies, such as MQTT that need ancillary software to play the role of broker.

3.3.2.2 Application Interface

Application Interface is responsible for providing the interface to applications that interact with the IoT network. This interface differs from Device Interface because it is designed to handle a larger dataset, allowing external applications to use data from the IoT network as well as to feed the network with data they produce.

An application can interact with the IoT instance provided by the middleware only after it is registered with this instance and given an authorization key. In this registration process, administrators may limit the services to which the application may have access.

Communication with applications in the Application Interface is performed by REST API defined in this thesis.

3.3.2.3 Middleware Interface

Middleware Interface is responsible for making the interface available to other middleware that interacts with the instance. This interface provides services for middleware instance management and configuration, allowing other middleware to use data from the IoT network as well as to feed the network with data it produces. The Middleware Interface also has services to enable management of middleware configuration and operating rules.

A middleware can interact with the IoT instance provided by the middleware only after it is registered with this instance and given an authorization key. In this registration process, administrators may limit the services to which the application may have access.

Communication with the middleware in the Middleware Interface is performed by REST API defined in this work.

3.3.2.4 User Interface

User Interface is responsible for providing the interface for human interaction with the instance. Unlike other interfaces that provide services, User Interface provides web applications so that users have friendly interaction with the instance, its data and services.

By default, these web applications have a user interface that runs in the browser as a single page application (SPA) which stores and consumes data from a set of APIs. In this design, servers are spared the processing and rendering of the interface and are only concerned with processing data manipulation services. This is important because in some cases the application is deployed on equipment with low processing resources compared to a cloud server such as Raspberry Pi.

3.3.3 Middleware Components

As explained in the previous subsection, the middleware has conceptual abstractions that need to be implemented. The proposed IoT middleware is made up of various modules to support those abstractions. Because they were designed based on the Microservices Architecture, each component has its own implementation and infrastructure. This Subsection introduces the software specification of each of these components.

3.3.3.1 Device Interface Component

Device Interface Component is the software that implements the Device Interface and is responsible for providing interfaces to devices on the IoT network. To simplify the architecture, this component implements the same interfaces provided to devices by the UIoT Gateway instances. It is important to note that the Device Interface Component's interfaces and UIoT Gateway's interfaces are similar, but both software have very distinct roles: the former is the device door for the middleware and the latter is a typical IoT gateway as defined in Section 2.1.

The modules that implement the interface of each communication technology have two abstract objects: the listener and the writer. The listener implements the ability to receive device communications on the supported channel, either by request-response or stream, and writer implements the active sending of communications to devices.

As described in Figure 3.5, the information received by the listener of each channel is made available to be stored in DIMS by the Engine.

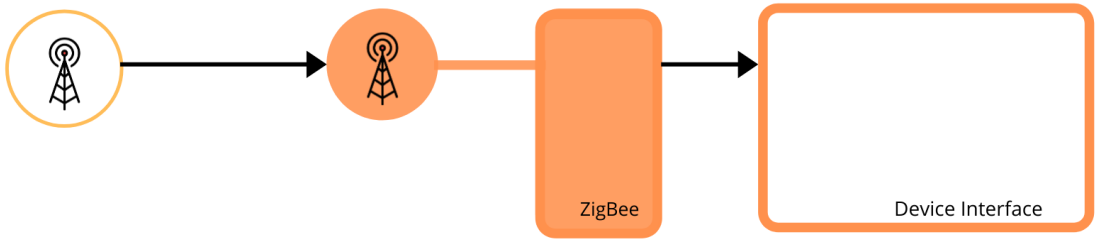


Figure 3.5: ZigBee channel communication example

The proposed middleware in this work implements interfaces for communications in the MQTT and ZigBee protocols, as well as in the Socket TCP, Socket UDP and REST standards.

The Device Interface Component provides API for client, services, data, and formulas interaction, as it is described below:

- `/client`: this is the Client API endpoint and it contains the services responsible for handling device registration data, supporting data registration by HTTP POST and data retrieving by HTTP GET. Its services are part of the self-registration process required by UIoT middleware;
- `/service`: this is the Service API endpoint and it contains the services responsible for

handling service registration data, supporting data registration by HTTP POST and data retrieving by HTTP GET. Its services are part of the self-registration process too;

- `/data`: this is the Data API endpoint and it contains the services responsible for handling data registration provided by device's services, supporting data registration by HTTP POST and data retrieving by HTTP GET. These services are central to IoT as they are used to record and query the data that is provided by IoT device services, and;
- `/formula`: this is the Formula API endpoint family and it contains the services responsible for handling service formulas, supporting data registration by HTTP POST, data retrieving by HTTP GET and data deletion by HTTP DELETE. These services allow you to expand service and device functionality, anonymize data, and summarize information to save bandwidth and/or disk space.

3.3.3.2 Application Interface Component

The Application Interface Component is the software unit that implements the Application Interface and is responsible for providing interfaces for applications wishing to interact with the IoT instance. Instance client applications interact with middleware through a set of REST APIs.

These applications must be previously authorized to interact with the middleware. In the authorization process, it should be informed what types of access (read and write), data classification and access time.

A subscription is when a customer subscribes to receive publisher-produced content. The middleware has this set of services to allow this interaction:

The Data API is made available for both this application to provide data for that instance and for querying data contained in that instance. The middleware has this set of services to allow this interaction:

The Application Interface Component provides API for application and service registration, its subscriptions, and data interaction, as it is described below:

- `/registration`: this is the Registration API endpoint and it contains the services responsible for handling application registration data, supporting data registration by HTTP POST and data retrieving by HTTP GET. Its services are part of the self-registration process required by UIoT middleware;
- `/service`: this is the Service API endpoint and it contains the services responsible for handling service registration data, supporting data registration by HTTP POST and data retrieving by HTTP GET. Its services are part of the self-registration process too;
- `/subscription`: this is the Subscription API endpoint family and it contains the services responsible for handling application's subscriptions for service data, supporting data

registration by HTTP POST, data retrieving by HTTP GET, data updating by HTTP PUT and data deletion by HTTP DELETE, and;

- `/data`: this is the Data API endpoint and it contains the services responsible for handling data registration provided by application's services, supporting data registration by HTTP POST and data retrieving by HTTP GET. These services are central to IoT as they are used to record and query the data that is provided by IoT application services.

3.3.3.3 Middleware Interface Component

The Middleware Interface Component is the software that implements the Middleware Interface. This mechanism that allows the proposed IoT middleware to interact with other instances as described in Section 3.2. The instances that interact with middleware through a set of REST APIs.

Since the instance has a valid token registration of another instance, it is able to interact with it. So it can manage its subscriptions, use shared resources, and send and retrieve data.

A subscription is when a customer subscribes to receive content produced by the publisher. The middleware has this set of services to allow this interaction:

The Data API is available for both this instance to provide data for that instance and for the data query contained in that instance. The middleware has this set of services to allow this interaction:

These instances must be previously authorized to interact with the middleware.

An instance needs to register itself in order to be able to interact with that instance. The middleware has this set of services to allow this interaction:

The Middleware Interface Component provides API for middleware registration, its subscriptions, and data interaction, as it is described below:

- `/registration`: this is the Registration API endpoint and it contains the services responsible for handling middleware registration data, supporting data registration by HTTP POST and data retrieving by HTTP GET. Its services are part of the self-registration process required by UIoT middleware;
- `/subscription`: this is the Subscription API endpoint family and it contains the services responsible for handling middleware's subscriptions for service data, supporting data registration by HTTP POST, data retrieving by HTTP GET, data updating by HTTP PUT and data deletion by HTTP DELETE, and;
- `/data`: this is the Data API endpoint and it contains the services responsible for handling data provided by managed middleware services, supporting data registration by HTTP POST and data retrieving by HTTP GET. These services are central to IoT as they are used to record and query the data that is provided by all kind of IoT client services.

3.3.3.4 DIMS

Data Interface Management System (DIMS) is a middleware core component that is responsible for storing and retrieving data handled by the middleware, and it abstracts the database into a REST interface so that components and clients are not coupled with the technologies and structures used for data storage. DIMS is a built-in middleware component, meaning that it is for use exclusively by the other internal components of the middleware architecture, so it has no interface outside the middleware and its API is protected from being accessed for them.

It should be noted that DIMS is solely responsible for storing data, with all its consistency and availability. However, it does not make use of this data, leaving this task to other middleware components.

In the current version of DIMS, MongoDB was chosen as the instrument of data persistence. MongoDB is a NoSQL data repository that adopts a document-oriented data model and natively allows JSON document handling (ACQUAVIVA et al., 2019). However, as noted, this issue is irrelevant to the other middleware components because they only manipulate data through the API provided by DIMS.

DIMS is responsible for the data sent by sensors and actuators, so it manages these which are the main data of the IoT architecture. But it is not synonymous with databases for the other middleware components, whereas each component must be responsible for its own data repository.

The DIMS provides API for client, services and data interaction, as it is described below:

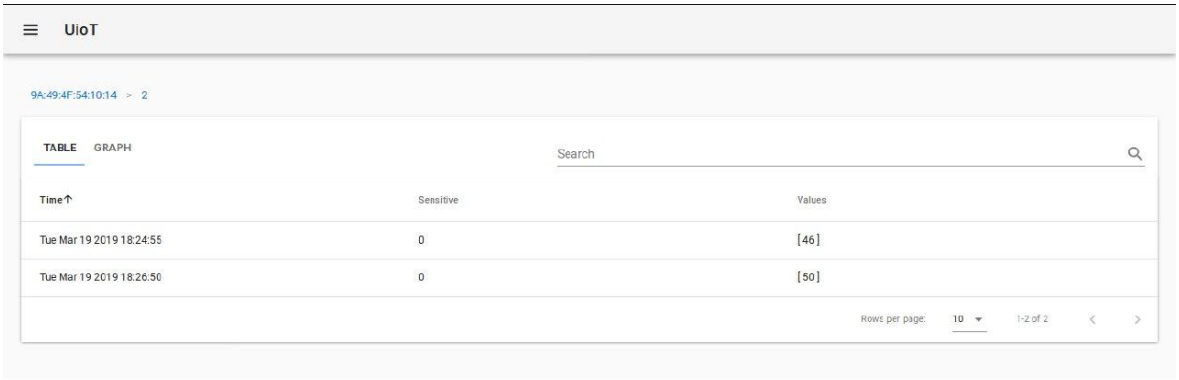
- `/clients`: this is the Client API endpoint and it contains the services responsible for handling client registration data, supporting data registration by HTTP POST and data retrieving by HTTP GET;
- `/services`: this is the Service API endpoint and it contains the services responsible for handling service registration data, supporting data registration by HTTP POST and data retrieving by HTTP GET, and;
- `/data`: this is the Data API endpoint and it contains the services responsible for handling managed services data, supporting data registration by HTTP POST and data retrieving by HTTP GET.

3.3.3.5 User Interface Component

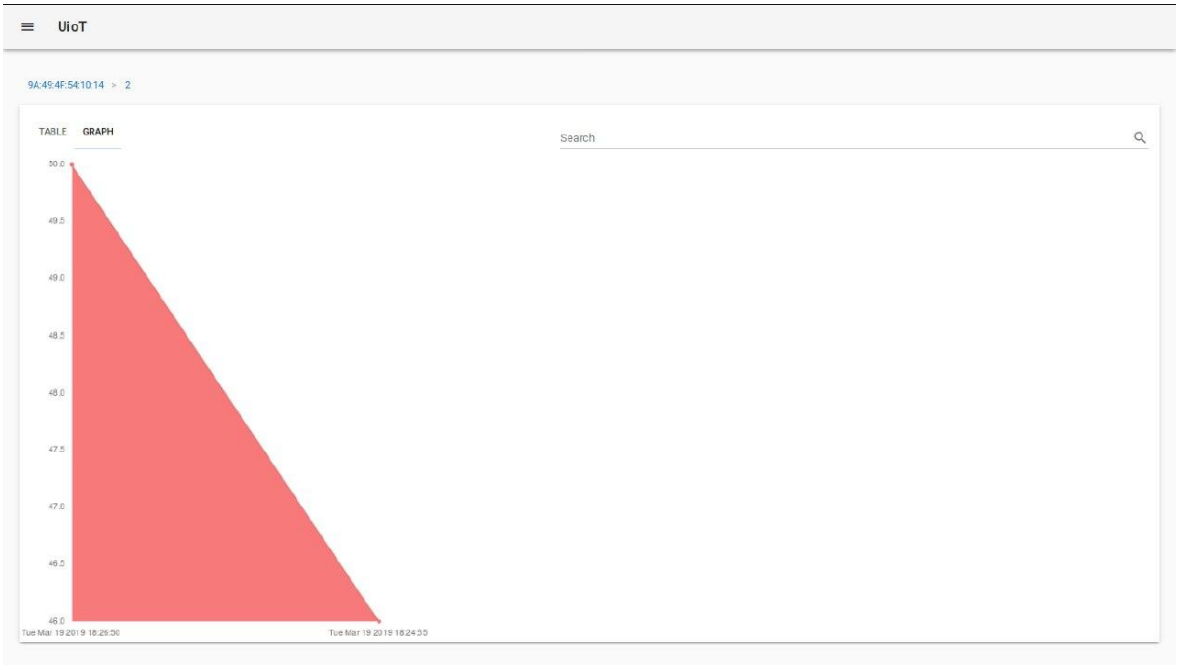
The User Interface Component is the component that implements the User Interface. It is responsible for the backend implementation of interfaces that are made available to users. Middleware provides two user interfaces: UIMS and Admin UI.

The User-friendly Interface Management System (UIMS) is one middleware's User Interface, so it is the software of the User Interface in which a person interacts with the middleware and its

data. Its main features are data visualization, rule registration and configuration of how middleware works.



(a) View of a list of an IoT device data in Admin UI



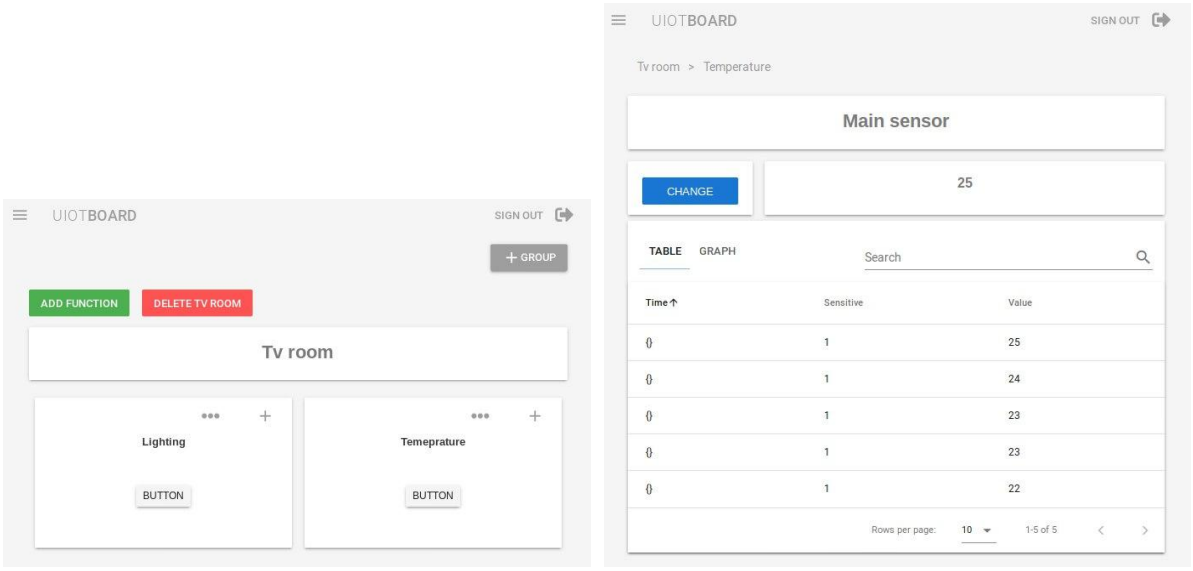
(b) View of a graphic with an IoT device data in Admin UI

Figure 3.6: Admin UI screens

One role of UIMS is to enable the IoT middleware user to view, in a friendly manner, the data that is generated on the IoT network. Therefore, it has queries to display data about clients and services, as well as the data they produced. Figure 3.6 illustrates the display of this data in list and graph format for ease of interpretation.

Figure 3.7(b) illustrates the configuration of how middleware works. In this functionality, the user has the option to define the role of middleware, its relationship with other middleware, as well as the address of its components.

For UIMS, a new set of APIs have been created in the User Interface Component to support the creation of data display grouping functionality and services, as it is described below:



(a) A room's available services in UIMS

(b) A device's service data in UIMS

Figure 3.7: UIMS screens

- `/group`: this is the Group API endpoint and it contains the services responsible for handling groups data, supporting data registration by HTTP POST, data retrieving by HTTP GET and data deletion by HTTP DELETE;
- `/func`: this is the Function API endpoint and it contains the services responsible for handling service registration data, supporting data registration by HTTP POST, data retrieving by HTTP GET and data deletion by HTTP DELETE, and;
- `/service`: this is the Service API endpoint and it contains the service's service responsible for handling group's services for service data, supporting data registration by HTTP POST and data deletion by HTTP DELETE.

As described earlier, the UIMS audience is the average user who wants to consume IoT instance resources. On the other hand, Admin UI is the application intended for the administrator of this instance. It has access to the telemetry details of the network and instance components.

3.3.3.6 Engine

The Engine is a core component of middleware that is responsible for controlling transactions and enforcing transformation and data distribution rules. The Engine is also an internal component of middleware. It is an active component that works according to pre-established middleware rules.

The Engine monitors the data received from customers and sends the data to its destination based on the rules that are entered. This mechanism can route raw data, as well as perform the necessary transformations in the data to deliver it to its recipients. For example, a particular

customer might request that only the average of the last 30 minute measurements be sent to.

3.3.4 Middleware Ontology

It was necessary to change the UIoT middleware model presented in Silva et al. (2016a, Sec. 3.1) to simplify device specification and to include new concepts related to instance interaction. Figure 3.8 presents a macro view of the proposed model that highlights the Client, Service, Data, and Rule entities.

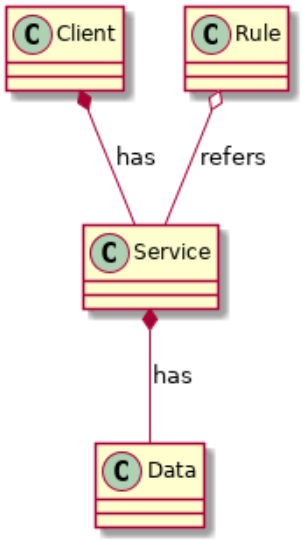


Figure 3.8: Middleware macro-entities

3.3.4.1 Client Entity

Client represents the main actor of interaction with the middleware, including devices, applications, middleware and users, bringing together the actors that interact from the outside into the middleware instance. Figure 3.9 illustrates these entities and their attributes.

The Device entity represents every type of physical device considered in the context of IoT, from small sensors and actuators to more complex devices such as smartphones. This entity includes specific attributes to trace the hardware characteristics of the device, such as MAC address, serial, and processor.

The Application entity represents applications that can be integrated with middleware to provide some service or functionality that is not provided natively by it. These applications are elements outside the middleware that are allowed to interact with its data and services.

The Middleware entity represents the middleware that relates to the instance of this middleware. This entity has the specific attributes of the relationship qualification between middleware such as trustLevel and interationType.

The User entity represents the people who will interact directly with IoT middleware. This

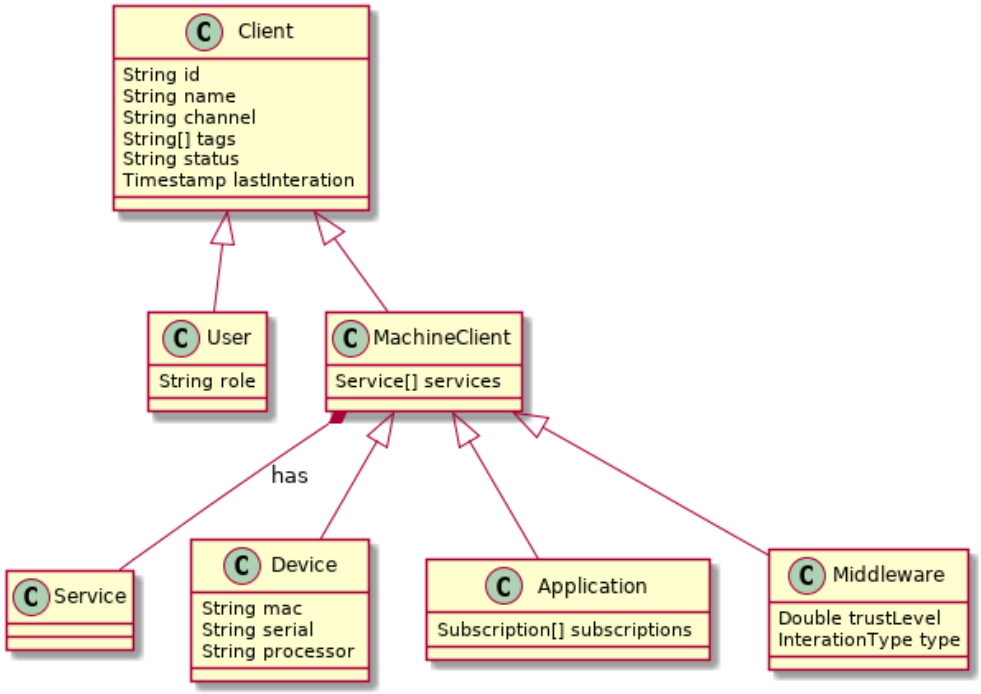


Figure 3.9: Client entitiy

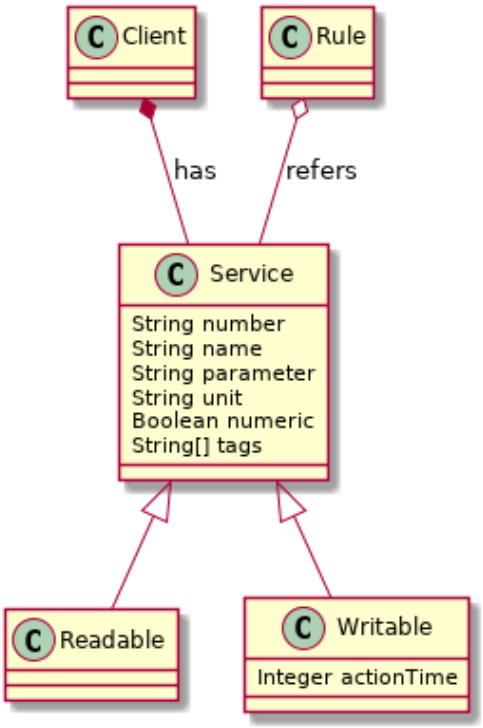


Figure 3.10: Service entitiy

entity has user-specific attributes like role.

3.3.4.2 Service Entity

The Service entity represents the services that customers provide that are the main concepts of interaction with the middleware, whether these services read or write data. For example, in the case of devices, their services are sensing and acting on the environment. Figure 3.10 illustrates these entities and their attributes.

3.3.4.3 Data Entity

The Data entity represents the data handled by customers in its services. For example, in the case of devices, their services are sensing and acting on the environment. Figure 3.11 illustrates these entities and their attributes.

This entity specializes in two others: DataRead and DataCommand. DataRead represents the data read from devices that do environment sensing, so this data represents readings taken from the environment.

DataCommand contemplates commands sent to actuators. Importantly, an actuator deals with the physical environment and this means that a command does not precisely mean that the action has been performed or the state of the service has changed.

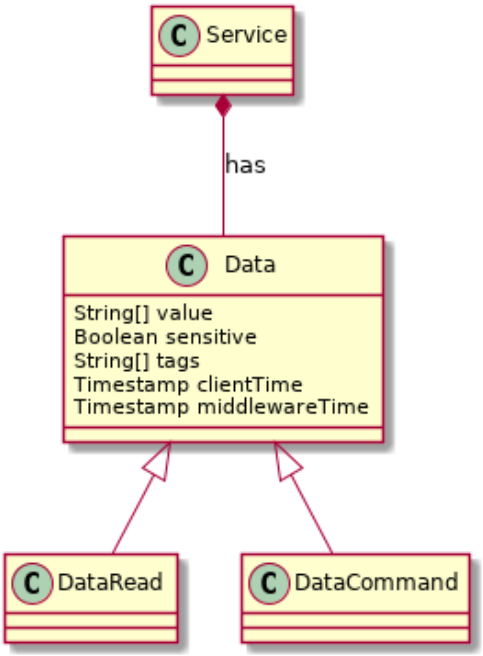


Figure 3.11: Data entitiy

3.3.5 Middleware Components Deployment Procedures

Middleware components are designed to run both via Docker Engine and directly on a Linux-based operating system.

In the former way, running it in Docker containers is interesting because you get the benefits of Docker's abstraction from the details of the deployment, so you can easily deploy it to more than one computing environment. This way, each component has its own settings to run on Docker images, in addition to its `Dockerfile` and `docker-compose.yml` files. But before deploying the images via Docker, you should configure the container execution parameters, such as indicating other components or features that the software needs to run.

In the latter way, it is necessary to install the Python support, the component and the packages it depends on on the host operating system, and set the permissions required to run it. It is also recommended that you configure the automatic boot of the component along with the operating system boot to prevent the component from being unavailable when the system restarts.

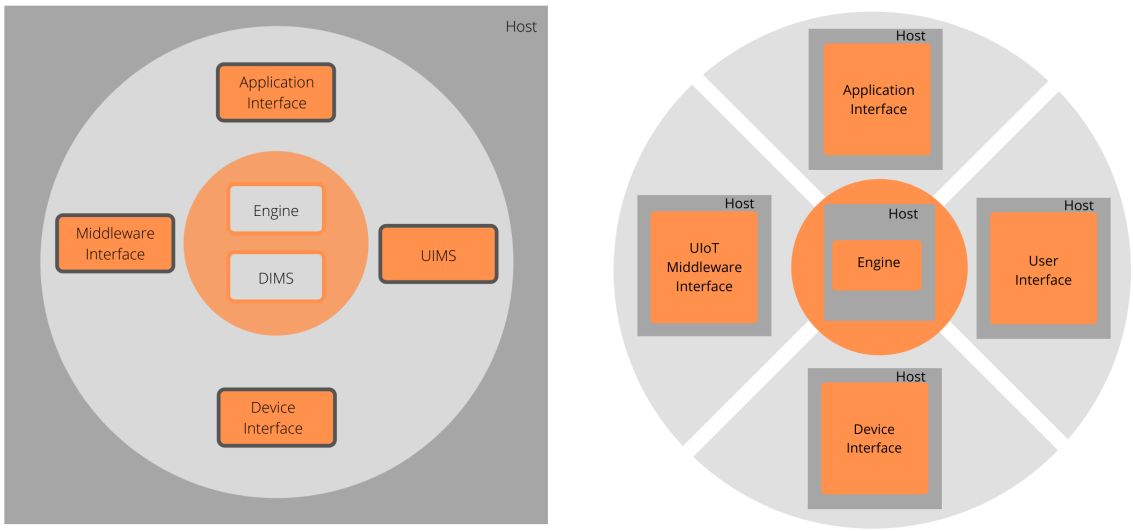
It is important to note that the development process for each middleware component is carried out on a computer with the necessary tools for development, debugging, and testing, as with any software development process. After completing development of the component version, the final steps are to generate and to test the component package and the component Docker image containing all the features that are required for its operation. Afterwards, the component is ready to be deployed in the operating environment.

3.4 MIDDLEWARE SET UP FACTORS

To give the desired flexibility, the proposed IoT design was premised that middleware should be adaptable to be used in a variety of scenarios to avoid the environment and limitations imposed by other middleware. This Section describes three aspects that are supported by the proposed IoT middleware and which should be considered when configuring and deploying the IoT network. It should be noted that the three aspects represent different dimensions and are used in a complementary way, allowing their use in a combined way.

3.4.1 Deployment Scheme Factor

As for its deployment, Figure 3.12 outlines the two possibilities for middleware deployment. In the first scenario, it can be deployed as a single part containing all the components necessary for it to function in a single bundle, so that resizing must occur in that single bundle. In the second scenario, middleware can be deployed in a distributed manner in which each node is responsible for some of the middleware functionality, so that resource resizing should be done node by node as needed.



(a) All of the middleware components are depolyed in one single host

(b) Each middleware component is deployed in its own host

Figure 3.12: Single vs distributed deployment

3.4.2 Supported Computation Model

As for the computation model on which the solution is deployed, Figure 3.13 describes the three possibilities envisaged: local, fog computing, and cloud computing.

In the local model, middleware and all its components should be deployed on local network resources. This computational model is tailored for situations in which there is any security issue about using cloud computing resources as well as in regions with poor internet infrastructure. Such as, remote facility monitoring, farming, and forest monitoring.

In the cloud computing model, middleware should be installed in a cloud environment to manage the IoT network and it does not matter how it is actually installed in the cloud environment.

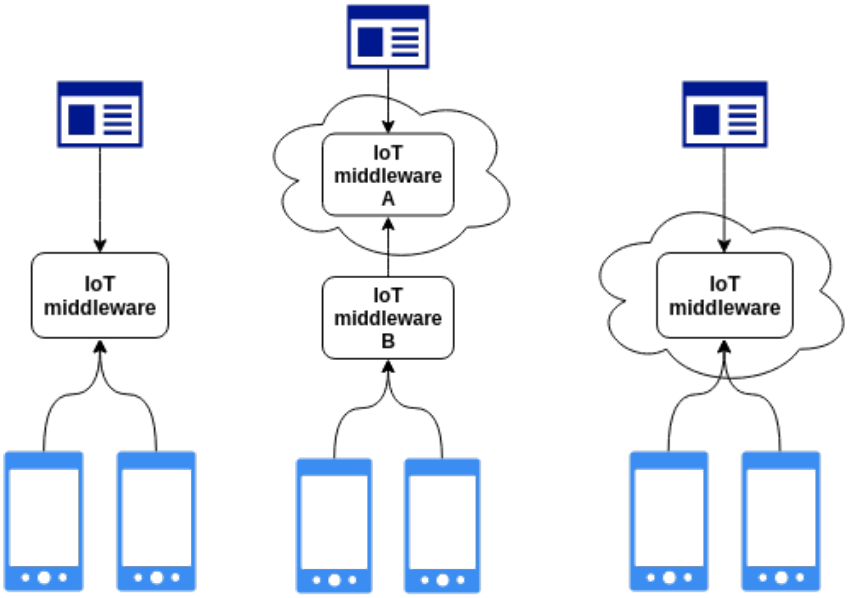
In fog computing model, middleware should be configured to be used in collaboration with other instances, either on the local network or on the Internet. In this model, each instance behaves like a fog node and can have the most diverse roles and types of interactions with the other nodes. It is important to remember that the fog computing mode is commonly combined with cloud resources.

Fog architecture is supported because it provides for a better use of available resources flexibly and which can give cloud characteristics to the local environment.

3.4.3 Social Operation Mode

Regarding its mode of operation, Figure 3.14 describes the four possibilities provided for middleware: standalone, hierarchical, cooperative, and federated.

In the standalone scenario, it can be used in isolation without interacting with other instances



(a) Local (b) Fog computing (c) Cloud computing

Figure 3.13: Local, fog computing, and cloud computing supported computational models

to receive or send resources. In this case, the entire IoT network boils down to a single operating instance.

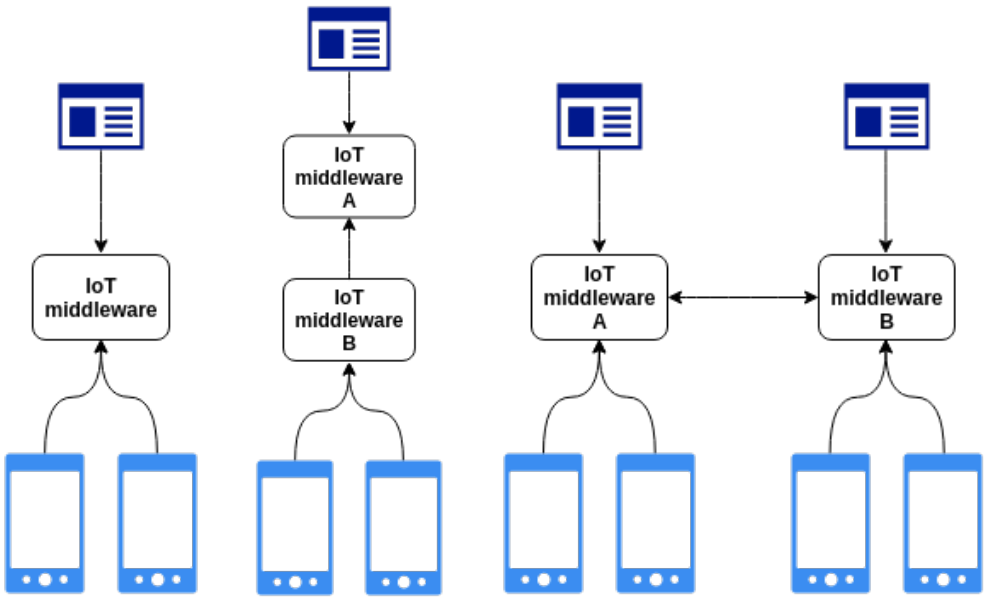
In the hierarchical scenario, command and control relationships are established between the instances, so that there is a centralized structure that determines the behavior of the other instances. In this case, the IoT network is composed of several operating instances.

In the cooperative scenario, the instance can be used cooperatively with other instances for resource sharing without a hierarchical relationship between them. In this case, cooperation with other middleware does not change the nature of the instance, but only expands its capabilities.

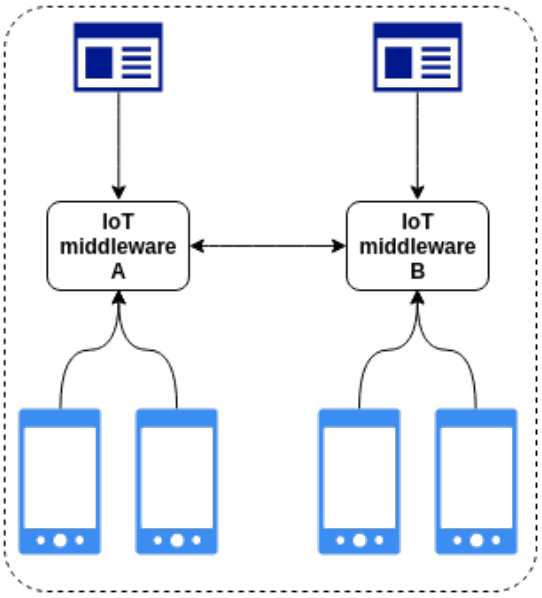
In the federated scenario, IoT instances join or form in a federation, so federation members understand that they are part of a single IoT instance. Federation members trust each other, but their internal guidelines guide their participation in the federation.

In all four cases, the interaction with other middleware is only for the exchange of data and device services, without any hierarchy or resource distribution relationship. In this case, middleware behaves as any application should behave.

It should be noted that an instance can connect to more than one instance simultaneously and with different roles.



(a) Standalone mode (b) Hierarchical mode (c) Cooperative mode



(d) Federated mode

Figure 3.14: Standalone, hierarchical, cooperative and federated mode

4 MIDDLEWARE USAGE SCENARIO

This Chapter illustrates the proposed IoT middleware usage possibilities, using hypothetical scenarios. Although they are hypothetical scenarios, they are based on situations with the real dimension.

4.1 STANDALONE OFF-GRID LOCAL SCENARIO

Forest monitoring is an important issue to support studies of that environment and the impact of human interference in it. In the former case, it is necessary to investigate its fauna and flora *in natura* regarding to animal group behaviour and plant growth. In the latter case, the impact of human activities in the environment such as deforestation and burning. This monitoring can occur a few meters from a residential neighborhood of Brasilia, as well as in the middle of the Amazon rainforest far from any man-made infrastructure.

The forest monitoring environment would have a WSN consisting of 30 ZigBee devices, all of them communicating with middleware via ZigBee. Each sensing device would be equipped with three services: monitoring temperature, humidity and the presence of fire in the environment. Temperature and humidity services scan the environment every 5 minutes and send to middleware while the fire sensor also reads every minute, but only communicates with the middleware if it detects fire. During the dry season in the Cerrado, there is a high risk of natural or man-made fires. Therefore, in this period it is necessary to configure a short time interval between the sensor measurements.

Without any data preprocessing the middleware will receive 720 temperature and humidity sensor measurements per hour and up to 1,800 fire detection notifications.

To meet the scenario as shown in Figure 4.1, the IoT instance could be used in off-grid mode, communicating with wireless sensors via the ZigBee protocol and storing their data locally. This way, data managed by the middleware is available for local consultation by users as well as other systems. For this, the proposed IoT middleware could be installed on ordinary computer or even an SBC computer, provided that it is powered by battery or some renewable energy such as solar energy. Its entire installation could be performed on a single host so that the processing and storage capacity is directly related to the capacity of the host on which it is installed.

In this case, in off-grid and standalone mode, there is no possibility to scale resources because, unlike the cloud environment, there is no other available resource. Thus, after a while in operation, it is likely that the host storage capacity limit will be reached, making system use unviable. IoT middleware can be configured to automatically discard older data to work around this problem. For instance, it can be configured to automatically apply aggregation and disposal on data in order

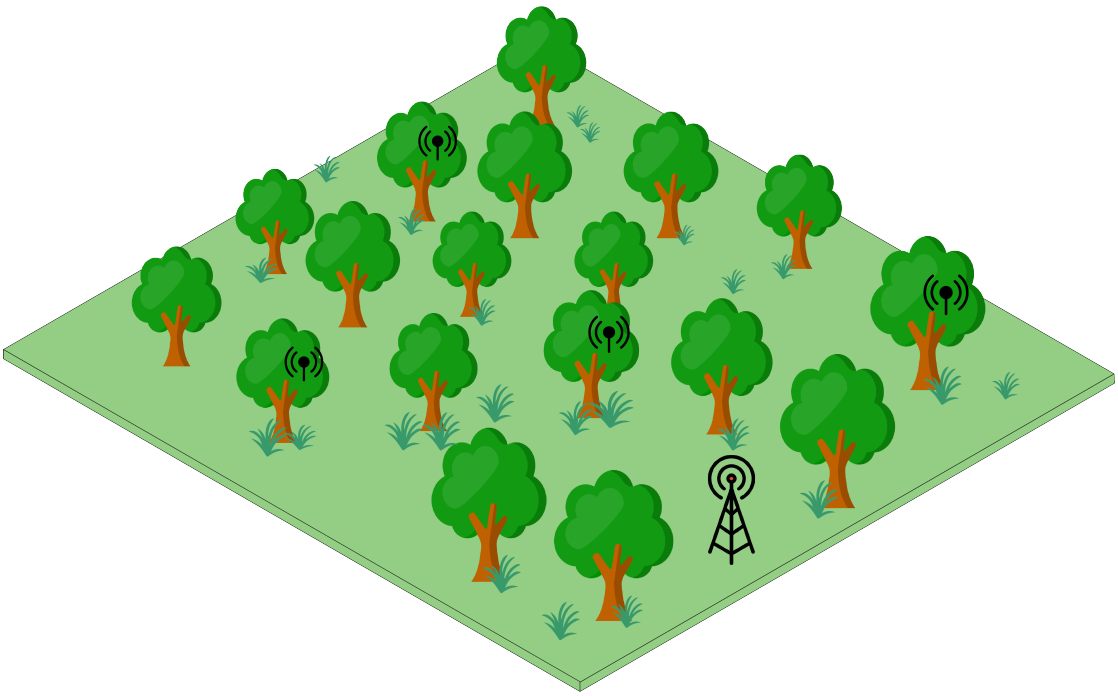


Figure 4.1: Local standalone middleware usage: the middleware and its sensors

to, in the former, produce new content stem from the aggregation of raw data, in the latter, discard older data. The aggregation rules could be according to the three rules exemplified below:

1. Aggregate readings for each device by averaging its readings every hour;
2. Discard raw data that is older than 48 hours age, and;
3. Discard aggregate data that is older than 7 days age.

It is important to remember that middleware could be operated directly by a user who can query or back up their data.

4.2 HIERARCHICAL EDGE SCENARIO

IoT applications for automation or monitoring of the agricultural sector need to be installed in wide physical spaces and with very precarious communication infrastructure and high cost of use.

In the depict scenario presented here, the farming environment would have 50 devices of the same type to perform irrigation and soil moisture monitoring, all communicating with the middleware at the network edge via a ZigBee network. These devices would be equipped with two services: check soil moisture and trigger the irrigation mechanism of an area. The moisture service reads every 5 minutes and sends it to middleware and, if it detects that the soil is below

the set threshold, triggers the irrigator. The irrigation service notifies middleware every time it is triggered and always works with a standard amount of pouring water.

Without any data preprocessing in the devices, the middleware will receive 600 humidity sensor measurements per hour and from zero to 600 irrigation trigger notifications.

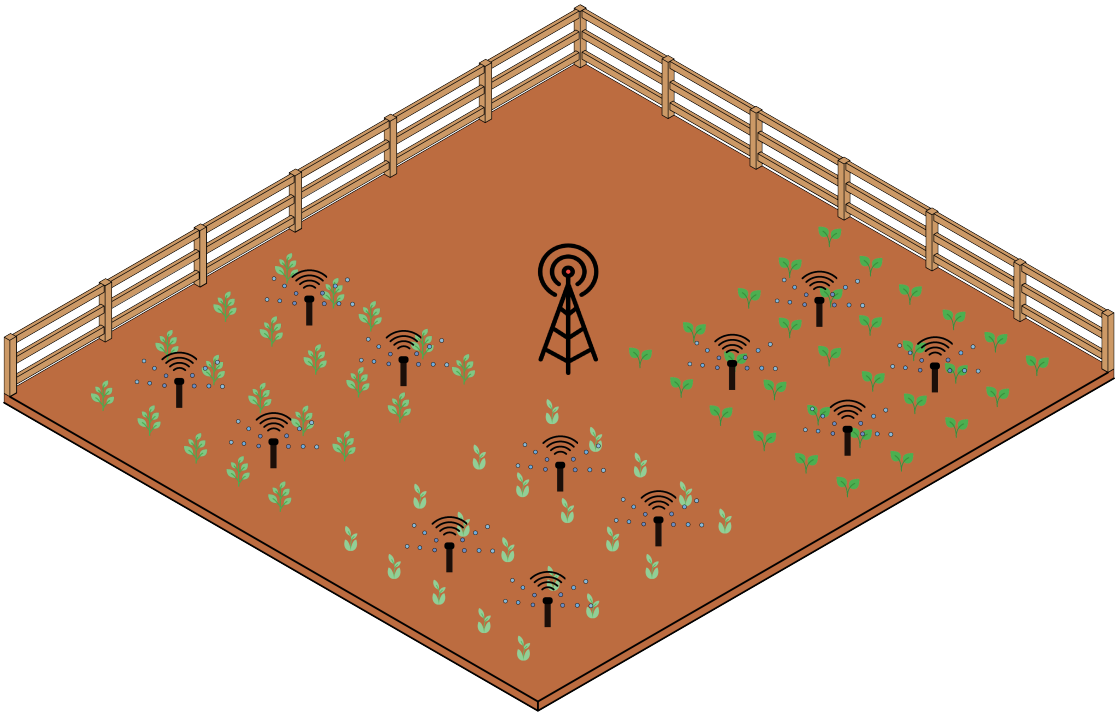


Figure 4.2: Edge and hierarchical middleware usage

To address this scenario, as shown in Figure 4.2, the IoT instance could consist of two nodes: one at the edge of the internal network and one in the cloud. The local node could be installed on a common computer as well as an SBC that would be responsible for preprocessing raw data on the edge of the local network and then sending that preprocessed data to the central node in the cloud. In this case, to save bandwidth, middleware edge can be configured with the following three rules:

1. Aggregate soil moisture readings from each device by averaging these readings every hour, reducing from 12 to 1 readings per device to be sent to the cloud every hour;
2. Prioritize moisture reading below a threshold that signals irrigation mechanism malfunctions over a period of time, and;
3. Aggregate notifications for the number of times a sprinkler has been triggered per day. Thus, the possible 14,400 notifications are reduced to a single information with the number of times the irrigation occurred.

It is important to remember that in this case too, the middleware edge user can operate it directly on the local network.

4.3 HIERARCHICAL FOG SCENARIO

Building automation scenarios are complex systems that are comprised of several sensors and actuators that in an integrated manner can give users the feeling of seemless use of the environment. To exemplify, consider building automation for an organization consisting of 3 five-floors buildings and a total flow of 4,000 people per day. This building automation comprises climate control and ambient lighting provided with various devices, as features described in Table 4.1, all of which communicating via the Wi-Fi network.

Table 4.1: Building automation scenario example

Device description	**Qty**	**Service**	**Event**	**Qty msgs**
Access turnstile to identify people entering and leaving buildings	8	Access counter	by access	10,000
Monitoring the presence of people in the environment	800	Presence sensor	every 30s	2,304,000
Environmental climate monitoring	800	Temperature sensor	every 30s	2,304,000
		Humidity sensor	every 60s	1,152,000
Room door situation monitoring	720	Door open sensor	by event	57,600
Central air conditioning controller	800	Vent direction controller	by event	9,600
Room Lighting Controller	720	Lighting trigger	by event	3,600

Caption: Qty – Quantity of devices, Qty msg – Quantity of messages by day.

For example, when people move around the floor toward an empty meeting room, the system can antecipate it and turn on lighting and climate as it detects that they are heading to the empty room. This decision can be made priorly or reactively.

Considering all 3,848 IoT devices in this system, a daily volume of 5,840,800 direct messages is generated. These messages are properly processed according to their context so that the sensor of building A does not interfere with the behavior of building B actuators. It is also important that messages are handled quickly to ensure that actions are timely.

To address this case, the IoT network could be segmented into subnets managed by middleware nodes other than the proposed IoT middleware installed on local network computers. These fog nodes could be used in a sectorized manner to address part of the organization's IoT network operating demand, as shown in Figure 4.3.

Dividing the network into areas under the responsibility of the nearest middleware allows each node to be responsible only for the devices in its area, delivering the data summarily to the central node in the cloud. Thus, each building could be divided into two areas, each under the responsibility of a node.

In addition to the preprocessing and optimizations exemplified in Section 4.2, the sector node can deliver data to the central node using data compression techniques, as described by Caldas Filho (2019).

4.4 DISTRIBUTED SCENARIO

Instances created by the proposed IoT middleware can work in a distributed way in terms of its deployment nodes. But also a node of the proposed IoT middleware can be configured to deployed in a distributed manner relative to the installation of its components in each one of the described scenarios in Sections 4.1, 4.2 and 4.3. This distribution can occur on both instances installed in a cloud environment and those installed directly on local computers.

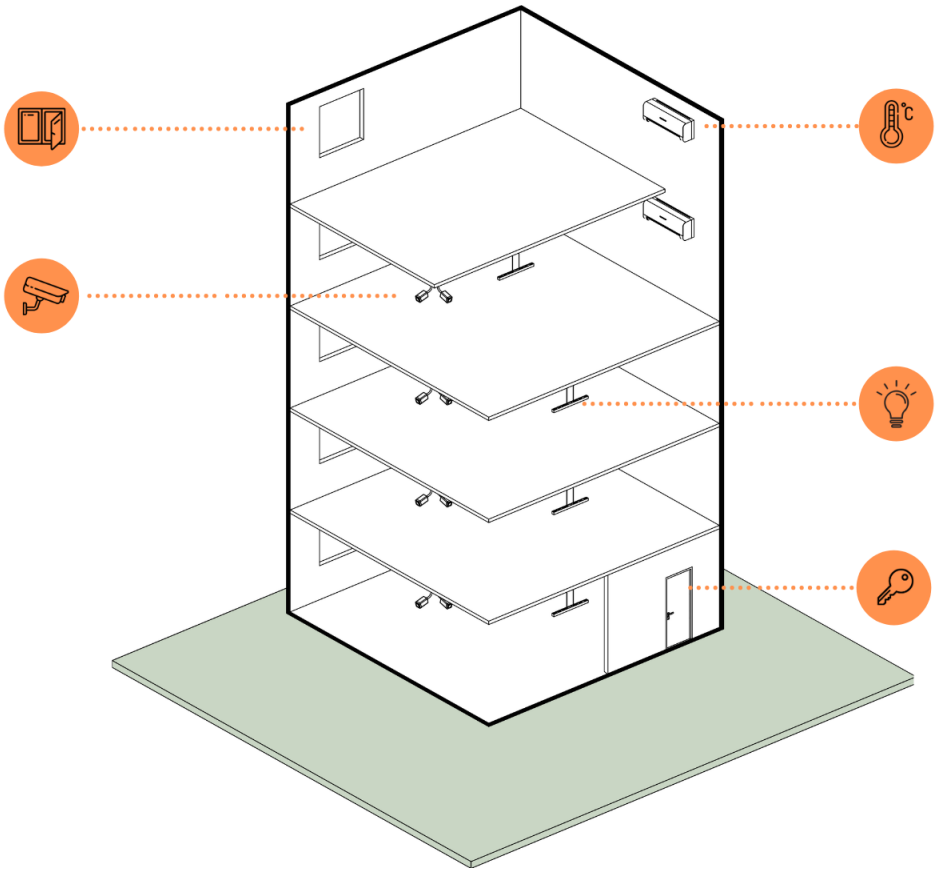


Figure 4.3: Fog and hierarchical middleware usage

The ability to distribute middleware components on local network computers optimizes the utilization of the organization's already allocated resources, as stated by Chiang & Zhang (2016). Authors in Qanbari et al. (2016) highlight that this design gives the middleware elasticity, allowing it to horizontal scaling by adding new nodes to its infrastructure. They also point out that this approach improves its dependability because avoids having a single point of failure.

5 EXPERIMENTS AND RESULTS

This Chapter aims to describe the experiments performed and their results in order to validate the purpose of this master thesis. It was designed three test scenarios to evaluate the proposed IoT middleware. The Chapter is organized as it follows: Section 5.1 presents an overview of the tests environment and general information that are useful for the testing evaluation. Then it is described the three test scenarios: the cloud computing scenario in Section 5.2, the local scenario in Section 5.3, and the fog computing scenario in Section 5.4. Finally, Section 5.5 discusses the tests results.

5.1 TESTING GUIDELINES

Using a real IoT environment as a testbed for evaluating an application or middleware has basically two issues. In the first, it is very costly, or even unfeasible, to build such real testing scenario because IoT is typically built with a lot of devices and softwares over a network which works together. The former, it comprises several and hardly manageable variables, such as: clock synchronizing, hardware and network failures, or physical interference in the environment, thus there's a minimum chance for conducting repeatable experiments. (DASTJERDI; BUYYA, 2016)

As suggested in Li (2018), the experiments presented in this thesis will be carried out in terms of round-trip time (RTT), given by Equation 5.1. $t_{request}$ is the data transmission time for the request, from the client to the server. $t_{processing}$ is the computation time the server needs to process the request. $t_{response}$ is the data transmission time for the request response, from the server to the client.

$$RTT = t_{request} + t_{processing} + t_{response} \tag{5.1}$$

Accordingly, the chosen test scenario was submitted to a controlled environment, composed by middleware installed in dedicated devices and software simulated sensors. Software simulated devices were used in the experiments to obtain a better control of the environment behavior, so that the focus of the experiment was the validation of the middleware. These virtual devices were programmed to behave in a manner compatible with the physical device of the tested scenario.

Hardware and servers have been standardized to improve control of the test environment, as well as the software controlled scenario. Therefore, Table 5.1 describes a list of resources that were used in the experiments described in this Chapter. In the description of each experiment, the resources used will be referenced as described in the Resource column of this table.

It was designed a hypothetical case based on scenarios described in other works as well as

Table 5.1: Testing resources used in the thesis experiments

Resource	Description
Cloud Server	Heroku cloud platform equipped with a 1-4 vCPU not dedicated with 1 CPU share, and 512 MB RAM.
Personal Computer	Dell Latitude 5480 equipped with a 2.5 GHz Intel Core i5 7200U 64 bits CPU, 16 GB RAM, 256 GB SSD, and wired Gigabit Ethernet connection.
Raspberry Pi	Raspberry Pi 3 Model B+ equipped with a Quad Core 1.2 GHz Broadcom BCM2837 64 bits CPU, 1 GB RAM, 32 GB Micro SD class 10, and wired Gigabit Ethernet LAN over USB 2.0 (max 300 Mbps) powered by source up to 2.5 A.

simulated scenarios researched in LATITUDE laboratory. So this work chose a smart farming scenario to explore the behavior of an IoT system by introducing a second variable that changes the context of devices. Then, the chosen test case was submitted to three computational models, generating three test scenarios, namely: local, cloud and fog computation models.

5.1.1 Scenario Description

In detail, the chosen scenario is a smart farming that contains soil moisture sensors, atmospheric pressure sensor and an actuator to irrigate the soil.

The system's objective is to keep the soil at a certain moisture level to favor some species plantation growth and also to optimize the use of water consumed to perform irrigation. To simplify the scenario, it was defined that this humidity is influenced by two factors: irrigation and rain. Irrigation is the direct intervention of the system that drips a certain volume of water into the soil of the specified area. Rainfall is a natural event that is unrelated to human intervention, but the rainfall high probability when the atmospheric pressure drops below 1,013 hectopascals.

To achieve the goal, including to reduce water consumption, two irrigation drive protocols are adopted. They are described as it continues:

Protocol #1 when the soil moisture is below 30%, and;

Protocol #2 when soil moisture is between 31 and 60% and atmospheric pressure is below 1,013 hectopascals.

Protocol #1 has the most priority and aims to prevent soil moisture from falling below the minimum safety level expected by the plantation. Thus, it is triggered regardless of weather conditions in relation to rain. In turn, protocol #2 aims to keep the soil moisture at a good level, but is only triggered if atmospheric pressure does not indicate that rain should occur soon.

The simulated plantation field is made up of 25 area sectors, each equipped with one device. In addition to these, there is an atmospheric pressure sensor for the entire field, covering all its sectors.

The virtual device was programmed with the two services described in the scenario: soil moisture sensor and irrigation trigger. The sensor service has been programmed to send two soil moisture measurements per minute and the values of its virtual measurement are randomly chosen on a scale of 10 to 90 percent in each measurement. The irrigation service is triggered if the reading is below 30% and it has been programmed to notify the result of its action every time it is triggered as measured by the sensor.

5.1.2 Simulated Conditions

The virtual devices were programmed to generate data as scripts that simulate possible combinations of crops events. So, four scenarios were developed by combining the two variables. Script I simulates a dry soil with no rain forecast. Script II simulates a dry soil with rain forecast at the end of the day. Script III simulates a moist soil and without the forecast of rain. Script IV simulates a rainy day on which the soil is soaked.

Virtual devices were programmed to generate random data within a range of values that characterizes the scenario programmed in the aforementioned scripts, rather than programming them to generate fixed or random data. This decision aims to prevent the experiment from being impacted by any traffic optimization mechanism based on repeated value transmission.

Table 5.2: Test script breakdown

Sc	It	Soil moisture sensor ranging values	Atmospheric pressure sensor ranging values
I	#1	Dry soil (humidity from 25 to 28%)	No rain forecast (pressure from 300 to 450)
	#2	Very dry soil (humidity from 10 to 13%)	
	#3	Dry soil (humidity from 25 to 28%)	
	#4	Slightly dry soil (humidity from 33 to 37%)	
	#5	Dry soil (humidity from 25 to 28%)	
II	#1	Dry soil (humidity from 25 to 28%)	No rain forecast (pressure from 300 to 450)
	#2	Very dry soil (humidity from 10 to 13%)	With rain forecast (pressure from 550 to 750)
	#3	Dry soil (humidity from 25 to 28%)	
	#4	Slightly dry soil (humidity from 33 to 37%)	
	#5	Dry soil (humidity from 25 to 28%)	No rain forecast (pressure from 300 to 450)
III	#1	Moist soil (humidity from 60 to 63%)	No rain forecast (pressure from 300 to 450)
	#2	Soil a little damp (humidity from 45 to 48%)	
	#3	Slightly dry soil (humidity from 33 to 37%)	
	#4	Moist soil (humidity from 60 to 63%)	
	#5	Soil a little damp (humidity from 45 to 48%)	
IV	#1	Soil a little damp (humidity from 45 to 48%)	With rain forecast (pressure from 550 to 750)
	#2	Moist soil (humidity from 60 to 63%)	
	#3	Soggy soil (humidity from 88 to 91%)	
	#4	Moist soil (humidity from 60 to 63%)	
	#5	Soil a little damp (humidity from 45 to 48%)	

Caption: Sc – Script identification, It – Iteration number.

Thus, combining the controlled-generated data and the random-generated data strategies, the simulated sensor generates a dataset with a small range of values within the expected range. Moreover, the script iterates different (and sequential) conditions in order to improve this data variation. For example, to simulate dry soil condition, the sensor generates a set of readings ranging from 25 to 28% humidity. Table 5.2 describes the strategies defines in each script.

The work in Smith et al. (2012) presents a measurement of soil moisture from 38 sites in Australia since 2001 until its publication in 2012. These measurements are carried out continuously, with an interval of 20 minutes. The data set presented by the authors shows that the variation in soil moisture is small between each measurement: the variation is less than 1% in almost all measurements, with the exception of a few episodes. However, for stress testing purposes in this experiments scenarios, they were set to send their measurements every minute.

5.1.3 Analysis Process

The analysis process used in this work is based on the framework proposed in Praciano et al. (2018). But it was adapted to be performed in the steps presented as it follows: data collection, data pre-processing, application of the data cleaning model, and data visualization with discussions.

Data was collected in the tests performed. The tests were performed to simulate real situations of an IoT application, as shown in the Subsection 5.1.2. The simulations were programmed to use random data, but with variation within some logic that simulates a real smart device, as shown in the Subsection 5.1.1. Each simulation was also performed in fixed time boxes, as described in each test description.

Data pre-processing consists in a initial data analysis with the purpose of identifying data that does not fit the sample. Thus, data obtained in the experiment were tabulated in its raw state to conduct their initial analysis. These data are presented in the description of the experiment to support its pre-analysis.

Hence it is performed some method of data sanitization, including the application two techniques to improve data quality, so that it was possible to make a better analysis of the system's behavior. The first is the removal of variations caused by some specificity of its functioning, based on outcomes from data pre-processing. After removing samples based on behavior analysis, the Turkey Fences method is applied for identifying outliers in a data set relied on statistical calculations. Authors in Schwertman, Owens & Adnan (2004) state that Turkey Fences uses the interquartile range (IQR), which is the difference between values of the third and the first quartiles (Equation 5.2), to define the "fences" that it will be used to wipe outliers. This master thesis uses the inner fence, which formulas are described in Equations 5.3 and 5.4. Then, an outlier is any value beyond either lower or upper limit.

$$IQR = Q3 - Q1 \tag{5.2}$$

$$lower\ inner\ fence = Q1 - 1.5 \times IQR \tag{5.3}$$

$$upper\ inner\ fence = Q3 + 1.5 \times IQR \tag{5.4}$$

Finally, the resulting data set is presented, as well as a discussion of the data within the scenario of the experiment.

Throughout the experiment, when necessary, data and information relevant to the context of the experiment and/or its analysis are presented in the subsection and the context that describes it.

5.2 TESTING THE TYPICAL CLOUD SCENARIO

This first set of tests explores the scenario in which an IoT middleware is installed in a cloud environment and the devices are installed on the local network. The experiment was performed to validate the operation of the proposed middleware in the usual cloud-IoT mode.

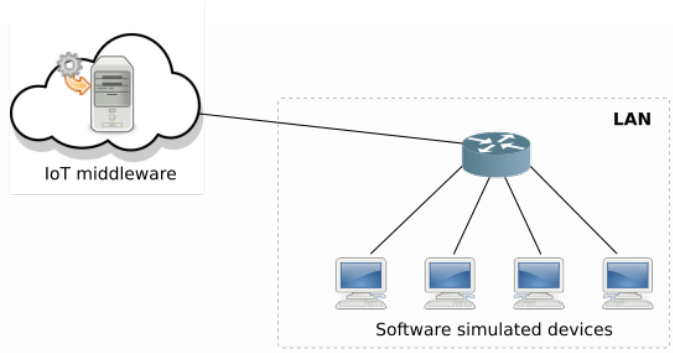
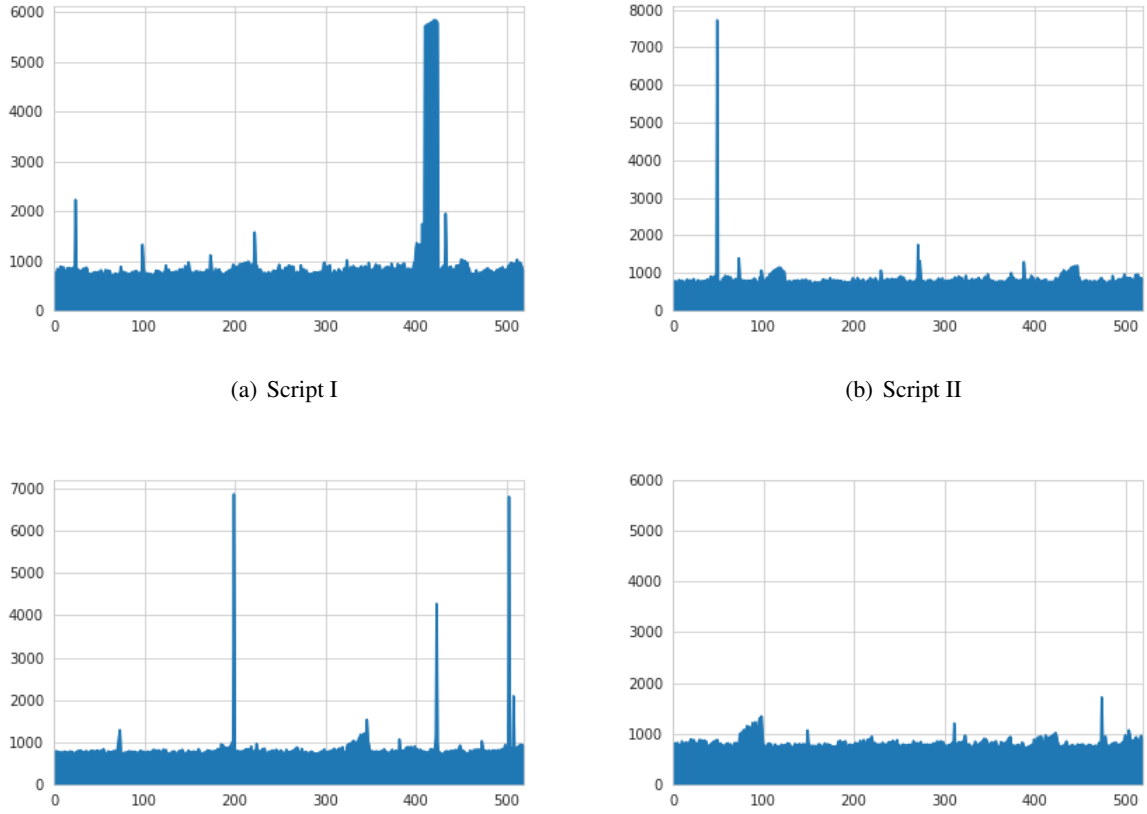


Figure 5.1: Cloud middleware testing architecture

The experiment environment depicted in Figure 5.1 was set up as follows:

- 25 software-simulated devices with soil moisture sensors running on a Personal Computer;
- 1 software-simulated device with atmosphere pressure sensor running on a Personal Computer, and;
- A single IoT middleware node with all its components running on Cloud Server.

Each simulation was performed in a fixed time box of 20 minutes. The values measured in the performed simulation are described in Table 5.3 and illustrated in Figure 5.2. The 520 requests occurred over an average time ranging from 817 to 971 ms, with standard deviation ranging from 101 to 841 milliseconds, and with coefficients of variation ranging from 12.37% to 86.64%. These results will be discussed as it follows.



(a) Script I

(b) Script II

(c) Script III

(d) Script IV

Figure 5.2: Results of cloud middleware testing with raw data (time in milliseconds)

Table 5.3: Cloud computing middleware testing response time in milliseconds with raw data

	Script I	**Script II**	**Script III**	**Script IV**
Mean	971.08	837.39	836.57	817.33
Standard deviation	841.35	320.46	416.49	101.09
Coefficient of variation	86.64%	38.27%	49.78%	12.37%
Minimum	670.95	688.64	679.17	685.77
25%	759.69	764.01	758.31	762.25
50%	804.13	798.22	783.07	790.37
75%	875.21	851.31	818.07	839.11
Maximum	5,846.71	7,728.12	6,869.96	1,716.95

This raw simulation data presented some performance peaks of their execution and these behavior occurred at different moments of each time the test was executed. In the Script I execution, depicted in Figure 5.2(a), peaks greater than 5,500 ms occurred around 400th and 430th requests. In the Script II execution, depicted in Figure 5.2(b), a few peaks above 7,500 ms occurred around 50th request. In the Script III execution, depicted in Figure 5.2(c), peaks over 6,500 ms occurred in two different times: about 200th and about 500th requests, and peak over 4,000 ms occurred about 430th requests. In the Script IV execution, depicted in Figure 5.2(d), there is only a few peaks smaller than 2,000 ms: some about 70th to 100th requests and some about 470th request.

There is no feature in the system and in its environment that requires these performance peaks of up to 7 times the average and that occur at different times of the tests. Moreover, the cloud service connection is accomplished by a series of components, such as: the local network has two network switches, Internet connection is via a non-dedicated link, and the service is running on a cloud platform given by a cloud provider. Given the large number of stochastic variables involved in the tests and the system aspects, it rules out the possibility that these peaks represent only variance of the result values. Therefore, these peaks values are considered outliers and should be discarded from the analysis.

Table 5.4: Results for the cloud middleware for sanitized data (time in milliseconds)

	Script I	**Script II**	**Script III**	**Script IV**
Mean	813.27	800.23	781.81	791.06
Standard deviation	72.63	54.61	40.84	47.85
Coefficient of variation	8.93%	6.82%	5.22%	5.46%
Minimum	677.20	688.64	679.17	685.77
25%	758.77	762.80	755.65	759.54
50%	800.22	789.45	776.27	785.58
75%	856.36	828.54	803.53	820.22
Maximum	1,029.36	979.67	896.06	952.36

The new resulting system performance data set is presented in Table 5.4 and illustrated in Figure 5.3. The average time for the requests ranges from 781 to 813 ms, with standard deviation ranging from 40 to 72 milliseconds. The coefficients of variation, 8.93% for Script I, 6.82% for Script II, 5.22% for Script III, and 5.46% for Script IV, indicate low variance in the data set which can also be observed in the distribution of samples in Figure 5.3, showing that most results are grouped close to the median (visually, its the denser area in the axes of bullet columns for each script), within the limits of the 2nd and 3rd quartiles.

It is interesting to highlight that in this experiment the proposed middleware was performed to simulate a cloud-based IoT middleware. In this scenario, the presented results were consistent, showing a variation total between 677 and 1,029 ms. Nevertheless, it was not possible to properly compare the results presented in this thesis with other works because of the particularities of each work and their experiments. A common issue is that all the found works carried out the experiments on local networks, simulating the cloud computing environment. This factor masks the results because they do not consider the latency presented by the Internet connection and the

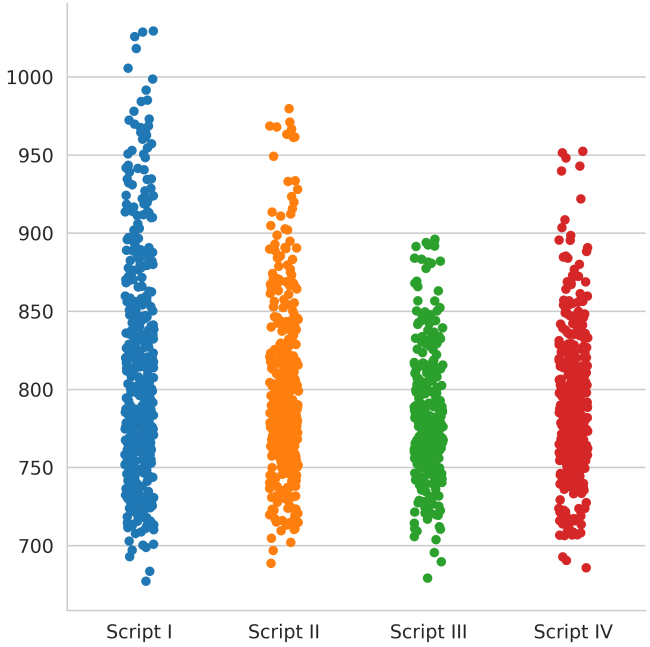


Figure 5.3: Distribution of the results for the cloud middleware for sanitized data (time in milliseconds)

cloud platform. For instance, Azimi et al. (2017) presents an average time of 125 ms with a standard deviation of 17 ms in an experiment scenario based in Wi-Fi communications and considering both gateway-middleware and middleware-gateway transmission times, but not including the computation time for data analytics ($t_{processing} = 0$), although the middleware needs to decide actions to be taken based on the information received from the sensor (Observe-Decide-Act control strategy).

Thus, considering the results of our experiment and the difference between testing methods, it can be concluded that in our scenario the proposed middleware was able to support the proposed scenario.

5.3 TESTING THE LOCAL STANDALONE SCENARIO

In this set of tests, the chosen test scenario was configured in the local standalone off-grid scenario format, similar to the scenario illustrated in Section 4.1. The experiment was performed to validate the operation of middleware in local standalone form on an SBC host. In other words, all the sensors and the middleware are in the same and local network. The experiment environment depicted in Figure 5.4 was set up as follows:

- 25 software-simulated devices with soil moisture sensors running on a Personal Computer;
- 1 software-simulated device with atmosphere pressure sensor running on a Personal Computer, and;

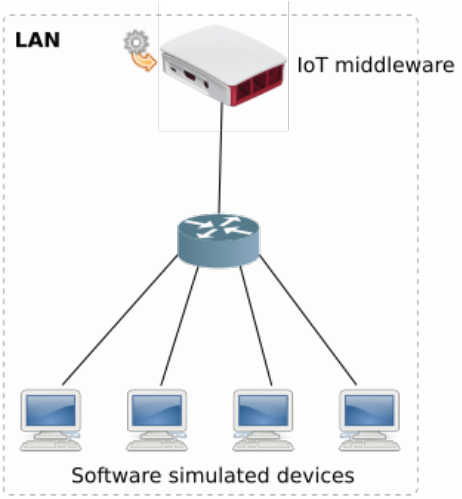


Figure 5.4: Local standalone middleware testing architecture

- 1 IoT middleware node with all its components running on a Raspberry Pi.

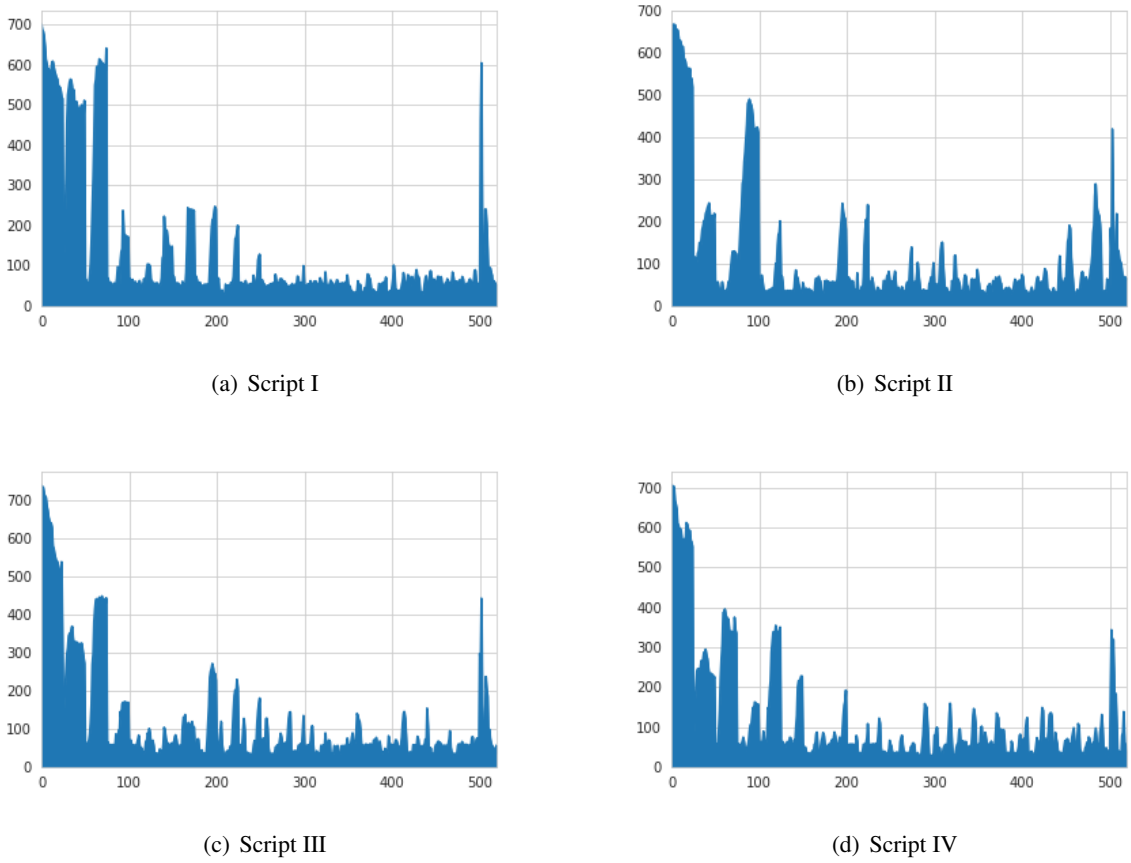
Each simulation was also performed in a fixed time box of 20 minutes. The raw values measured in the performed simulation are described in Table 5.5 and illustrated in Figure 5.5. The 520 requests execution time presented a high coefficient of variation in order that all of them is above 100%. These data will be discussed as it follows.

Table 5.5: Local standalone middleware testing response time in milliseconds with raw data

	Script I	**Script II**	**Script III**	**Script IV**
Mean	133.07	111.12	121.35	118.73
Standard deviation	166.87	139.72	143.40	137.72
Coefficient of variation	125.40%	125.74%	118.17%	115.99%
Minimum	27.40	27.12	27.08	26.99
25%	52.40	37.17	53.58	49.13
50%	58.73	57.00	59.60	60.04
75%	98.25	102.67	110.38	119.20
Maximum	700.53	666.90	739.51	706.15

These local tests results illustrated in Figure 5.5 pointed out that the middleware takes a while to warm up after its initialization and this behavior needs to be discussed. This issue causes the system to handle the first 60 or 70 requests in twice as much time (some almost three times) than the other following requests. Thus, the system takes between 520 and 720 ms to answer the first 30 requests, i.e., 5 to 7 times the average response time for the other following requests. After this initial period, the system performs consistently well for the remainder of its execution. This can be explained by remembering that the loading of objects in memory is performed on demand after the initial basic loading of the system. Considering that the IoT middleware is used continuously and is not typically (re)initialized during its use, this behavior does not invalidate the observed performance for the purpose of this thesis. Thus, for the purpose of the analysis of our results the first 100 requests were discarded and then the IQR method was applied to remove outliers from

the resulting data.



(a) Script I

(b) Script II

(c) Script III

(d) Script IV

Figure 5.5: Results of local standalone middleware testing with raw data (time in milliseconds)

The new resulting system performance data set is presented in Table 5.6 and this data is grouped by occurrences in Figure 5.6. The average time for the requests ranges from 51.29 to 57.83 ms, with standard deviation ranging from 11.75 to 23.64 milliseconds. The coefficients of variation, 21.32% for Script I, 33.89% for Script II, 29.50% for Script III, and 40.88% for Script IV, show that low variance is observed in the data set as can also be visualized in Figure 5.6, since most of the data is grouped close to the median of the columns for each script, within the limits of the 2nd and 3rd quartiles.

In this experiment, the proposed middleware was performed to simulate local middleware installed in an SBC. In this scenario, the results presented were consistent, showing a total variation between 27 and 134 ms. This result shows lower performance than that presented by Silva et al. (2016a). In that experiment, the response time varied between 7 and 26 ms. However, it is necessary to highlight some differences between the two experiments, given that in the first experiment, the middleware was deployed in a server with higher computational resources (32 GB of memory and an Intel Xeon 2.79 GHz CPU) compared to this thesis' experiment with SBC resources (1 GB of memory and Quad Core 1.2 GHz Broadcom BCM2837 64 bits CPU – see Table 5.1 for SBC full specifications). The authors used this configuration in their work because they were

Table 5.6: Results for the local standalone middleware for sanitized data (time in milliseconds)

	Script I	**Script II**	**Script III**	**Script IV**
Mean	55.10	51.29	54.88	57.83
Standard deviation	11.75	17.38	16.19	23.64
Coefficient of variation	21.32%	33.89%	29.50%	40.88%
Minimum	27.40	27.20	27.08	27.64
25%	50.39	35.58	40.17	36.82
50%	55.55	52.29	56.27	55.82
75%	59.81	60.09	60.12	65.10
Maximum	90.60	110.70	106.30	134.47

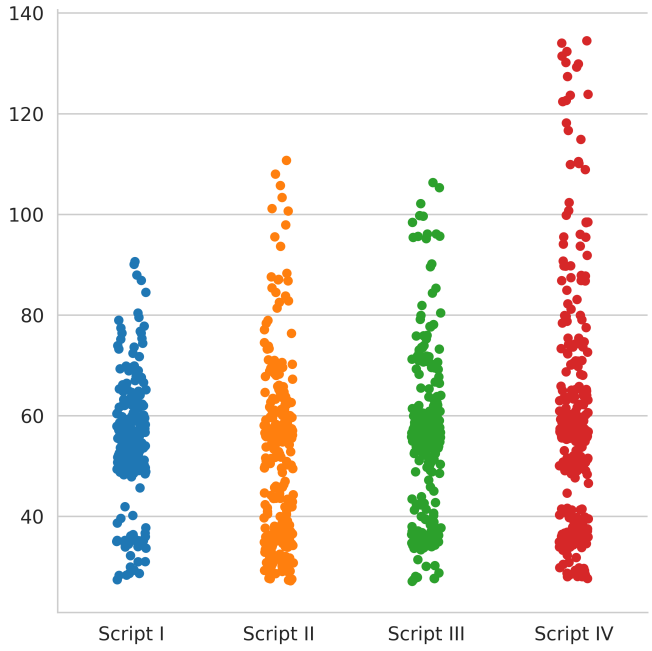


Figure 5.6: Distribution of the results for the local standalone middleware for sanitized data (time in milliseconds)

simulating the behavior of middleware in a computational cloud. Considering this difference, it can be concluded that the proposed middleware was able to support the proposed scenario, even if installed on a Raspberry Pi.

5.4 TESTING THE FOG HIERARCHICAL SCENARIO

In this set of tests, the chosen test scenario was configured in the fog computing model, similar to the scenario illustrated in Section 4.3. The experiment was performed to validate the operation of middleware-fog on an SBC host communicating with a master middleware on cloud infrastructure. Hence, all the sensors and the local middleware are in the same and local network and other IoT middleware runs on a cloud server. The experiment environment depicted in Figure 5.7 was set up as follows:

- 25 software-simulated devices with soil moisture sensors running on a Personal Computer;
- 1 software-simulated device with atmosphere pressure sensor running on a Personal Computer;
- 1 IoT middleware node with all its components running on a Raspberry Pi, and;
- 1 IoT middleware node, with all its components, running on a cloud server.

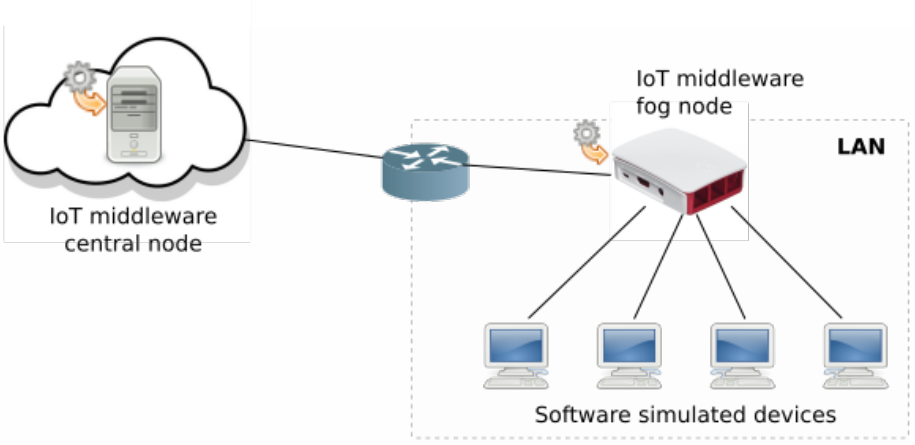


Figure 5.7: Edge hierarchical middleware testing architecture

Each simulation was also performed in a fixed time box of 20 minutes. The raw values measured in the performed simulation are described in Table 5.7 and illustrated in Figure 5.8. The 520 requests execution time presented a low coefficient of variation in order that all of them is below 30%. These data will be discussed as it follows.

These local tests results illustrated in Figure 5.8 pointed out the system also takes a while to warm up after its initialization and this behavior, related to loading of objects in memory, is not

Table 5.7: Fog computing middleware testing response time in milliseconds with raw data

	Script I	**Script II**	**Script III**	**Script IV**
Mean	1,070.27	1,066.81	1,041.89	1,048.33
Standard deviation	300.10	260.31	283.01	306.39
Coefficient of variation	28.04%	24.40%	27.16%	29.23%
Minimum	864.72	846.16	857.61	853.22
25%	927.65	947.34	926.89	928.29
50%	963.51	983.76	959.62	954.38
75%	1,063.19	1,053.06	1,009.24	1,013.07
Maximum	2,497.77	2,265.11	2,809.78	2,576.85

considered for discussion since typically the middleware does not (re)initialize during its norma operation. In this case, a scenario similar to the scenario presented in Section 5.3 occurs because the fog module has the same configuration profile that the local standalone middleware of that experiment. Thus, the first 100 requests were discarded and then the IQR method was applied to remove outliers.

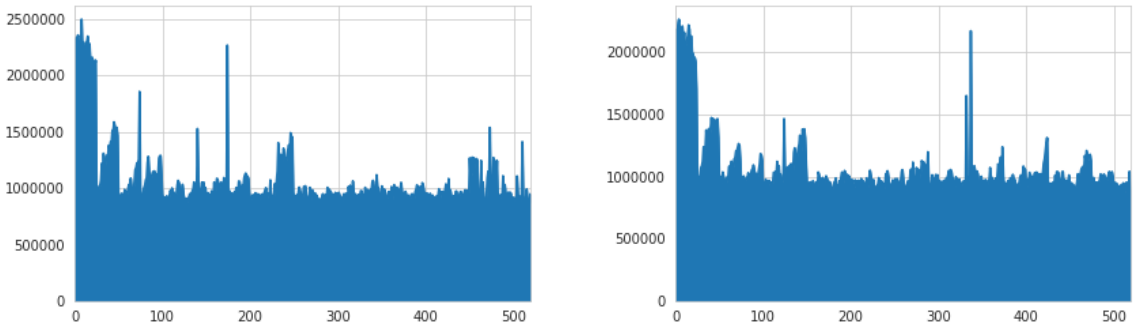
The new resulting system performance data set is presented in Table 5.8 and this data is grouped by occurrence in Figure 5.9. The average time for the requests ranges from 947.96 to 955.61 ms, with standard deviation ranging from 43.70 to 55.62 milliseconds. The coefficients of variation, 5.82% for Script I, 5.43% for Script II, 4.59% for Script III, and 4.65% for Script IV, show that low variance is observed in the data set as can also be visualized in the Figure 5.9, i.e., most of the data is grouped close to the median, within the limits of the 2nd and 3rd quartiles.

Table 5.8: Results for the fog computing middleware for sanitized data (time in milliseconds)

	Script I	**Script II**	**Script III**	**Script IV**
Mean	955.61	975.57	950.71	947.96
Standard deviation	55.62	53.01	43.70	44.04
Coefficient of variation	5.82%	5.43%	4.59%	4.65%
Minimum	864.71	846.16	857.61	853.22
25%	917.00	939.56	920.60	920.90
50%	942.06	967.61	945.38	941.72
75%	979.39	1,009.29	976.56	968.92
Maximum	1,129.90	1,128.53	1,078.50	1,074.43

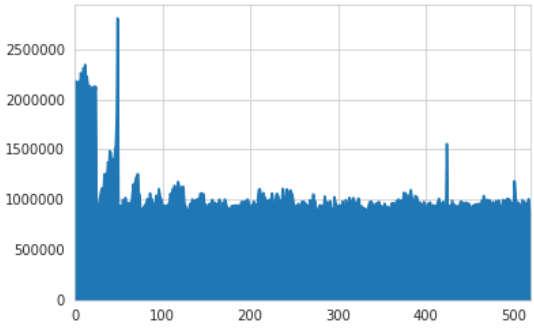
In this experiment, the proposed middleware was performed to simulate fog computing middleware with one node installed in an SBC and the other in a cloud computing infrastructure. In this scenario, the results presented were consistent, showing a variation total between 846 and 1,129 ms.

For comparison purposes, results from two other studies are presented as follows. The experiments in Schenfeld (2017) results in a 147 ms RTT for 10 devices connected to the middleware and this time goes up to 855 ms when the middleware serves 50 devices. The experiment in Li (2018) results in an 18 ms RTT for HTTP requests with 1,000 bytes of payload and 22 ms for 3,000 bytes of payload. The presented results in these works are better than the results obtained in

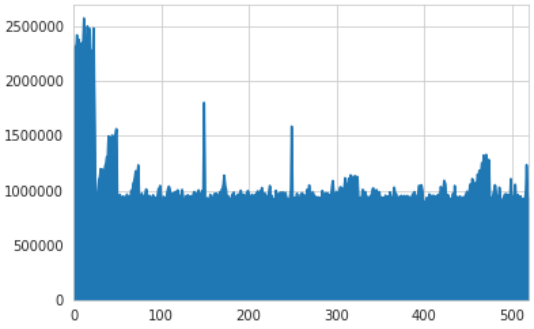


(a) Script I

(b) Script II



(c) Script III



(d) Script IV

Figure 5.8: Results of fog computing middleware testing with raw data (time in milliseconds times a thousand)

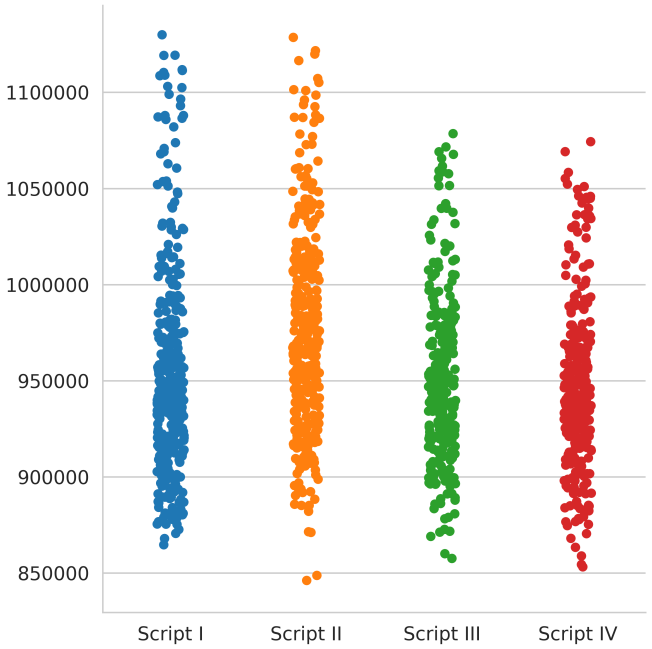


Figure 5.9: Distribution of the results for the fog computing middleware for sanitized data (time in milliseconds times a thousand)

the tests carried out in this thesis. However, it is not possible to make a comparison under similar conditions because, although their proposals are for fog computing, both simulations are carried out on a local network, without using cloud computing.

5.5 ANALYSIS AND DISCUSSION

In this Chapter, the proposed middleware was submitted to three different operating scenarios: local standalone in Section 5.3, cloud computing in Section 5.2, and fog computing in Section 5.4. The results presented demonstrated that the middleware was able to adapt to the proposed scenarios.

The results also confirmed what was expected about the difference between the computational models described in the fog, edge, and cloud computing models literature and other researches as cited in this work:

- When the proposed middleware is used locally, it responds with much higher performance than the cloud model (for instance, the Script I: 55.10 ms for the local tests and 813.27 ms for the cloud computing tests);
- When the proposed IoT middleware is used in fog computing mode, it shows up to 21% more time in RTT than the cloud model because it needs to process data before sending it to the central middleware, and;
- With the exception of a few peaks, the three computing models tests have low-variance in their performance (the biggest variance was 40.88% in the local standalone tests).

It is important to note that the experiments consider a system with no optimization on any of its components. That is, local servers, network components, the Internet connection, and cloud servers were used in their default configuration.

The local configuration tests illustrated in Figure 5.5 showed the occurrence of mini performance spikes around the 500th request, with responses between 380 and 480 ms. This behavior raised the suspicion that it was some sign of behavior that might occur outside the view given by the experiment. So an extra experiment was performed to validate this question, as represented in Figure 5.10. In this new execution, 1,300 requests were made to the middleware, in a mean time of 81.5 ms with standard deviation of 134.8 milliseconds. It demonstrated that the system remains consistent even after the 500th run, experiencing only a few peaks with 200 ms long. It also confirmed the need for warming up the system, similar to the original experiments.

A closing remark is necessary regarding these experiments: despite the intersection that exists between the different studies on the subject, there is no standardization on how to implement the solution, much less when it comes to validating it. This divergence hindered the comparison of the results obtained in this thesis with the other works. For instance:

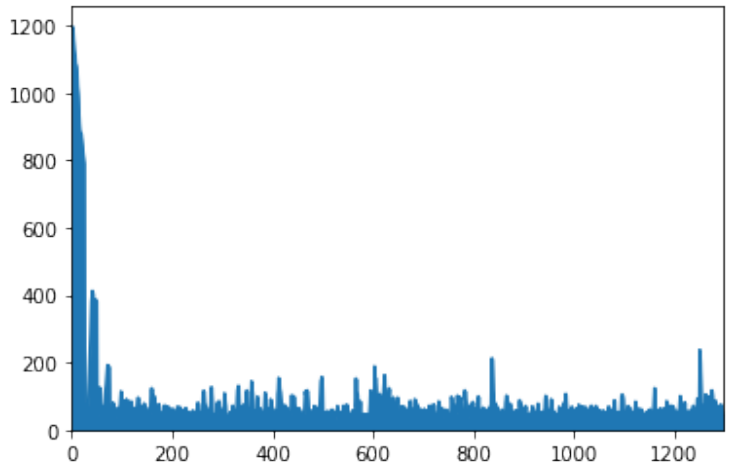


Figure 5.10: Long running tests results for the local standalone middleware testing (time in milliseconds)

- Works like Chekired, Khoukhi & Mouftah (2018), Shah-Mansouri & Wong (2018) are more concerned with validating some technical aspects of their proposal and do not yet have practical usage scenarios to be used as a comparison parameter, and;
- Works like Abdul et al. (2018), Chang, Srirama & Buyya (2017), Conti & Passarella (2018) are more focused on validating the proposal modeling and still do not have practical usage scenarios to compare.

6 CONCLUSION

The IoT paradigm and its variations have shown their utility in various application areas. As described in Section 2.1, reports describe real benefits that users and organizations have achieved with resource optimization in, for instance, smart cities, smart homes/buildings, agriculture and industry. Furthermore, several researches have shown progress in various aspects of the study of IoT and intelligent things.

Although, adaptability and flexibility of the IoT middleware was desired in this situation, what it is saw in practice is that each manufacturer has generated its IoT silo, making integration between different brands unfeasible or inefficient.

This master thesis proposes a middleware that supports the creation of an IoT instances that can be adpted to different computing scenarios and transparently interact with other instances. This thesis presents the architecture, components, and operation of the proposed middleware. It also describes how the design takes influences from the Microservices Architecture, as well as edge computing and fog computing models. To evaluate how the proposal was accomplished, its architecture was detailed and validated as well as its components and its way of operation.

This research took as starting point the concept that existing IoT middleware, such as the UIoT middleware developed at the University of Brasilia, could be adapted for use in restricted environments and constrained computing and communication resources. Then the proposal counted on concepts and technologies that have been explored in IoT such as edge and fog computing, as well as P2P connections between middlewares. In this sense, this work takes advantage of the benefits of the "IoT incarnations" definition, eventually overthrowing the unique IoT myth.

Initially, the conceptual proposition of a new architecture was presented for IoT middleware, evolving the existing strengths in the platform previously developed in the UnB UIoT project. The redesign of this architecture was based on concepts and techniques of the distributed computing discipline, paying special attention to modern models such as cloud, fog and edge computing models. Then, aspects of these concepts and fundamentals are gradually explored in the research, development and experimentation cycles. This process was performed until the validation or refutation of the proposed questions was reached.

Once this architecture was validated, the UIoT middleware was refactored to the Microservices Architecture, and then the middleware was ported to simpler architectures such as the SBC architecture. Having completed this first phase of the research, it was possible to evaluate a fundamental point of the work: its ability to be adapted to different deployment environments.

Then, the concept of inter-instance communication was explored so that instances could work together. This strategy was important to take better advantage of the edge and fog computing models. Finally, elements of the edge and fog computation model were adopted to allow coordinated action.

Then this master thesis is devoted to show the proposed middleware adaptability by deploying it in different computational models (such as local standalone, fog and cloud computing). It executed successfully all demanded features. Regarding its performance, RTT data was collected to explore the client view more realistically than previous works. Under harder conditions, the performance results were numerically worse than previous works, but the discussion clarifies the more realistic aspects of our proposal validation compared to mentioned previous works.

However, we reiterate that there is no way to make a fair comparison with other studies since the conditions under which the tests are performed are different (highlighting the use of cloud resources, even when the system is designed for the fog and/or cloud computing models).

Concomitantly with the development of the central idea of this thesis, several researches related to the IoT theme and its subareas were carried out, as it was shown in Subsection 1.4.1.

Throughout this research, it was demonstrated in papers published that it was possible to refactor the IoT ecosystem to be deployed and used in more restricted computing environments beyond the cloud computing model that is widely used in early IoT solutions. So this work can be a starting point for applying IoT in more restricted scenarios, including those with no fixed computing environments, unconventional connectivity conditions, and access restrictions and information classification.

6.1 FUTURE WORKS

Throughout this research, some middleware and/or IoT aspects suggested directions be explored to refine and evolve this work.

The use of intelligent agent techniques and algorithms can allow a better utilization of internal network resources when the IoT instance is configured in fog computing model. Thus, it is expected that nodes can automatically find and use available resources more efficiently, in particular by improving the energy efficiency of the solution. For example, it is envisaged to make the IoT middleware able to clone itself, in order to adapt its operation or to transfer its experience from one context to another one or to other nodes.

In this work, simple protocols and implementations were used to solve security and trust issues. However, it is interesting to investigate consolidated concepts in this area to evaluate if they are more appropriate than the solutions used in this work.

It is worth remembering that, in parallel to this work, LATITUDE laboratory's researches explored the topic IoT security and present conclusions that are in the context of IoT, according to publications Dutra et al. (2019), Kfouri et al. (2019), Gonçalves et al. (2019). These researches are related to this work, but they are not directly linked to its core proposal and, as such, have not yet been integrated into it. Thus, it is suggested to integrate the advances described in these works into the solution proposed in this master thesis.

This research proposes the adoption of the processing resource sharing envisioned and defined in the fog computing literature. This is a feature that can greatly adapt and expand IoT solutions, but third-party code execution is a complex issue that involves, among other things, several compatibility and security issues. Thus, this issue has been superficially addressed in this paper and it deserves further analysis.

Protecting data itself through the use of data labeling techniques is a subject that IoT explores. It is worth checking if this technique can really bring benefits to the thesis proposal.

The integration between instances was broadly approached in this work. However, to narrow the scope, only compatible instances were used in this research. So that, ontology differences between IoT instances is an open issue in the proposed middleware. A branch of study that can also be performed is to use interoperability techniques such as those proposed in Ganzha et al. (2017).

So, we can also conclude that this thesis opens new research directions to be explored.

Bibliography

ABDUL, R.; PAUL, A.; GUL M., J.; HONG, W.-H.; SEO, H. Exploiting Small World Problems in a SIoT Environment. *Energies*, MDPI, v. 11, n. 8, August 2018. ISSN 1996-1073.

ACQUAVIVA, A.; APILETTI, D.; ATTANASIO, A.; BARALIS, E.; BOTTACCIOLI, L.; CERQUITELLI, T.; CHIUSANO, S.; MACII, E.; PATTI, E. Forecasting Heating Consumption in Buildings: A Scalable Full-Stack Distributed Engine. *Electronics*, v. 8, n. 5, April 2019. ISSN 2079-9292.

ALABA, F. A.; OTHMAN, M.; HASHEM, I. A. T.; ALOTAIBI, F. Internet of Things security: A survey. *Journal of Network and Computer Applications*, v. 88, p. 10–28, June 2017. ISSN 1084-8045.

ALSHUQAYRAN, N.; ALI, N.; EVANS, R. A Systematic Mapping Study in Microservice Architecture. In: *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. Macau, China: IEEE, 2016. p. 44–51. ISBN 978-1-5090-4781-9.

ALVES, J. E. D. *Human swarm?* 2013. Accessed: Nov 22, 2019. [Online]. Available: <https://www.ufjf.br/ladem/2013/09/28/enxame-humano-artigo-de-jose-eustaquio-diniz-alves/>.

ASHTON, K. *That 'Internet of Things' Thing*. 2009. Accessed: Apr 05, 2019. [Online]. Available: <https://www.rfidjournal.com/articles/view?4986>.

ATZORI, L.; IERA, A.; MORABITO, G. The Internet of Things: A Survey. *Computer Networks*, v. 54, n. 15, p. 2787–2805, October 2010. ISSN 1389-1286.

ATZORI, L.; IERA, A.; MORABITO, G.; NITTI, M. The Social Internet of Things (SIoT) – When social networks meet the Internet of Things: Concept, architecture and network characterization. *Computer Networks*, v. 56, n. 16, p. 3594–3608, November 2012. ISSN 1389-1286.

AZIMI, I.; ANZANPOUR, A.; RAHMANI, A. M.; PAHIKKALA, T.; LEVORATO, M.; LILJEBERG, P.; DUTT, N. HiCH: Hierarchical Fog-Assisted Computing Architecture for Healthcare IoT. *ACM Transactions on Embedded Computing Systems*, ACM, v. 16, n. 5s, p. 174:1–174:20, September 2017. ISSN 1539-9087.

BOETTIGER, C. An Introduction to Docker for Reproducible Research. *ACM SIGOPS Operating Systems Review*, ACM, v. 49, n. 1, p. 71–79, January 2015. ISSN 0163-5980.

BONOMI, F.; MILITO, R.; ZHU, J.; ADDEPALLI, S. Fog Computing and Its Role in the Internet of Things. In: ACM. *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. Helsinki, Finland, 2012. p. 13–16.

BORGIA, E. The Internet of Things vision: Key features, applications and open issues. *Computer Communications*, v. 54, p. 1–31, December 2014. ISSN 01403664.

BOTTA, A.; DONATO, W. de; PERSICO, V.; PESCAPE, A. On the Integration of Cloud Computing and Internet of Things. In: . Barcelona, Spain: IEEE, 2014. p. 23–30. ISBN 978-1-4799-4357-9.

CALDAS FILHO, F. L. d. *Proposta de um Gateway IoT em Computação Fog com Técnicas de Aceleração WAN [Proposal of an IoT Gateway in Fog Computing with WAN Acceleration Techniques]*. Master Thesis — University of Brasília, Brasília, DF, Brazil, July 2019.

CALDAS FILHO, F. L. d.; MARTINS, L. M. C. e.; ARAÚJO, I. P.; MENDONÇA, F. L. L. d.; COSTA, J. P. C. L. da; DE SOUSA JÚNIOR, R. T. Design and Evaluation of a Semantic Gateway Prototype for IoT Networks. In: *Companion Proceedings of the 10th International Conference on Utility and Cloud Computing*. Austin, TX, USA: ACM, 2017. (UCC '17 Companion), p. 195–201. ISBN 978-1-4503-5195-9.

CALDAS FILHO, F. L. d.; MARTINS, L. M. C. e.; ARAÚJO, I. P.; MENDONÇA, F. L. L. d.; COSTA, J. P. C. L. d.; DE SOUSA JÚNIOR, R. T. Gerenciamento de Serviços IoT com Gateway Semântico [IoT Service Management with Semantic Gateway]. In: *Atas das Conferências IADIS Ibero-Americanas WWW/Internet 2017 e Computação Aplicada 2017*. Vilamoura, Algarve, Portugal: IADIS Press, 2017. p. 199–206. ISBN 978-989-8533-70-8.

CALDAS FILHO, F. L. de; ROCHA, R. L.; ABBAS, C. J. B.; MARTINS, L. M. C. e; CANEDO, E. D.; DE SOUSA JÚNIOR, R. T. QoS Scheduling Algorithm for a Fog IoT Gateway. In: *4th Workshop on Communication Networks and Power Systems (WCNPS 2019)*. Brasília, DF, Brazil: IEEE, 2019. p. 122–127.

CHANG, C.; SRIRAMA, S. N.; BUYYA, R. Indie Fog: An Efficient Fog-Computing Infrastructure for the Internet of Things. *Computer*, v. 50, n. 9, p. 92–98, September 2017. ISSN 1558-0814.

CHANG, H.; HARI, A.; MUKHERJEE, S.; LAKSHMAN, T. Bringing the cloud to the edge. In: IEEE. *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. Toronto, ON, Canada, 2014. p. 346–351.

CHEKIRED, D. A.; KHOUKHI, L.; MOUFTAH, H. T. Industrial IoT Data Scheduling Based on Hierarchical Fog Computing: A Key for Enabling Smart Factory. *IEEE Transactions on Industrial Informatics*, v. 14, n. 10, p. 4590–4602, October 2018. ISSN 1941-0050.

CHIANG, M.; ZHANG, T. Fog and IoT: An Overview of Research Opportunities. *IEEE Internet of Things Journal*, v. 3, n. 6, p. 854–864, December 2016.

CITIZENFOUR. Direction: Laura Poitras. Production: Mathilde Bonnefoy, Laura Poitras, and Dirk Wilutzky. Starring: Edward Snowden; Glenn Greenwald; William Binney; Jacob Appelbaum; and Ewen MacAskill. United States and Germany: Radius-TWC, 2014. 1 film (113 min), son., color., 35 mm.

CONTI, M.; PASSARELLA, A. The internet of people: A human and data-centric paradigm for the next generation internet. *Computer Communications*, v. 131, p. 51–65, October 2018. ISSN 0140-3664. COMCOM 40 years.

CONTI, M.; PASSARELLA, A.; DAS, S. K. The Internet of People (IoP): A new wave in pervasive mobile computing. *Pervasive and Mobile Computing*, v. 41, p. 1–27, October 2017. ISSN 1574-1192.

COULOURIS, G.; DOLLIMORE, J.; KINDBERT, T.; BLAIR, G. *Distributed Systems: Concepts and Design*. 5th. ed. Boston, MA, USA: Pearson Education, 2012. ISBN 978-0-13-214301-1.

DASTJERDI, A. V.; BUYYA, R. Fog Computing: Helping the Internet of Things Realize Its Potential. *Computer*, v. 49, n. 8, p. 112–116, August 2016.

DOCKER INC. *Docker*. 2019. [Online]. Available: <https://www.docker.com/>.

DRAGONI, N.; GIALLORENZO, S.; LAFUENTE, A. L.; MAZZARA, M.; MONTESI, F.; MUSTAFIN, R.; SAFINA, L. Microservices: yesterday, today, and tomorrow. *arXiv:1606.04036 [cs]*, June 2016. ArXiv: 1606.04036. [Online]. Available: <http://arxiv.org/abs/1606.04036>.

DUTRA, B. V.; ALENCASTRO, J. F. de; CALDAS FILHO, F. L. de; MARTINS, L. M. C. e; DE SOUSA JÚNIOR, R. T.; ALBUQUERQUE, R. de O. HIDS by signature for embedded devices in IoT networks. In: UNIVERSIDAD DE EXTREMADURA. *Actas de las V Jornadas Nacionales de Investigación en Ciberseguridad (JNIC 2019)*. Cáceres, Spain, 2019. p. 53–61. ISBN 978-84-09-12121-2.

FARAHANI, B.; FIROUZI, F.; CHANG, V.; BADAROGLU, M.; CONSTANT, N.; MANKODIYA, K. Towards fog-driven IoT eHealth: Promises and challenges of IoT in medicine and healthcare. *Future Generation Computer Systems*, v. 78, p. 659–676, 2018. ISSN 0167-739X.

FERREIRA, H. G. C. *Arquitetura de Middleware para Internet das Coisas [Internet of Things Middleware Architecture]*. Master Thesis — University of Brasília, Brasília, DF, Brazil, 2014.

FERREIRA, H. G. C.; CANEDO, E. D.; DE SOUSA JÚNIOR, R. T. IoT Architecture to Enable Intercommunication Through REST API and UPnP Using IP, ZigBee and Arduino. In: *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. Lyon, France: IEEE, 2013. p. 53–60.

FERREIRA, H. G. C.; CANEDO, E. D.; DE SOUSA JÚNIOR, R. T. A ubiquitous communication architecture integrating transparent UPnP and REST APIs. *International Journal of Embedded Systems*, Inderscience, v. 6, n. 2/3, p. 188, 2014. ISSN 1741-1068, 1741-1076.

FERREIRA, H. G. C.; DE SOUSA JÚNIOR, R. T. Security analysis of a proposed internet of things middleware. *Cluster Computing*, v. 20, n. 1, p. 651–660, March 2017. ISSN 1386-7857, 1573-7543.

FERREIRA, H. G. C.; DE SOUSA JÚNIOR, R. T.; DEUS, F. E. G. d.; CANEDO, E. D. Proposal of a Secure, Deployable and Transparent Middleware for Internet of Things. In: *2014 9th Iberian Conference on Information Systems and Technologies (CISTI)*. Barcelona, Spain: IEEE, 2014. p. 1–4.

FIELDING, R. T. *Architectural styles and the design of network-based software architectures*. PhD Thesis — University of California, Irvine, Irvine, CA, USA, 2000. Accessed: Apr 11, 2014. [Online]. Available: <https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.

GANZHA, M.; PAPRZYCKI, M.; PAWŁOWSKI, W.; SZMEJA, P.; WASIELEWSKA, K.; PALAU, C. E. From implicit semantics towards ontologies — practical considerations from the INTER-IoT perspective. In: *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*. Las Vegas, NV, USA: IEEE, 2017. p. 59–64. ISSN 2331-9860.

GONÇALVES, D. G. V.; CALDAS FILHO, F. L. de; MARTINS, L. M. C. e; KFOURI, G. de O.; DUTRA, B. V.; ALBUQUERQUE, R. de O.; DE SOUSA JÚNIOR, R. T. IPS architecture for IoT networks overlapped in SDN. In: *2019 Workshop on Communication Networks and Power Systems (WCNPS)*. Brasília, DF, Brazil: IEEE, 2019.

GUBBI, J.; BUYYA, R.; MARUSIC, S.; PALANISWAMI, M. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, v. 29, n. 7, p. 1645–1660, 2013. ISSN 0167-739X.

GUILLEMIN, P.; FRIESS, P. *Internet of Things: Strategic Research Roadmap*. Brussels, Belgium, 2009. 50 p. [Online]. Available: <http://www.internet-of-things-research.eu/pdf/IoT_Cluster_Strategic_Research_Agenda_2009.pdf>.

GUTH, J.; BREITENBÜCHER, U.; FALKENTHAL, M.; FREMANTLE, P.; KOPP, O.; LEYMANN, F.; REINFURT, L. A Detailed Analysis of IoT Platform Architectures: Concepts, Similarities, and Differences. In: ____. *Internet of Things*. Singapore, Singapore: Springer Singapore, 2018. p. 81–101. ISBN 978-981-10-5861-5.

HARPER, K. E.; GOOIJER, T. de; SCHMITT, J. O.; COX, D. Microdatabases for the Industrial Internet. *Databases - Computers and Society arXiv:1601.04036*, January 2016. ArXiv: 1601.04036. [Online]. Available: <http://arxiv.org/abs/1601.04036>.

KAMIENSKI, C.; SOININEN, J.-P.; TAUMBERGER, M.; DANTAS, R.; TOSCANO, A.; CINOTTI, T. S.; MAIA, R. F.; NETO, A. T. Smart water management platform: Iot-based precision irrigation for agriculture. *Sensors*, MDPI, v. 19, n. 2, January 2019. ISSN 1424-8220.

KFOURI, G. de O.; GONÇALVES, D. G. V.; DUTRA, B. V.; ALENCASTRO, J. F. de; CALDAS FILHO, F. L. de; MARTINS, L. M. C. e; PRACIANO, B. J. G.; ALBUQUERQUE, R. de O.; DE SOUSA JÚNIOR, R. T. Design of a Distributed HIDS for IoT Backbone Components. In: *Communication Papers of the 2019 Federated Conference on Computer Science and Information Systems*. Leipzig, Germany: PTI, 2019. p. 81–88. ISBN 978-83-955416-3-6. ISSN 2300-5963.

KIM, B.; ALI, T.; LIJERON, C.; AFGAN, E.; KRAMPIS, K. Bio-Docklets: virtualization containers for single-step execution of NGS pipelines. *GigaScience*, 2017.

LEWIS, J.; FOWLER, M. *Microservices*. 2014. Accessed: Nov 23, 2019. [Online]. Available: <http://martinfowler.com/articles/microservices.html>.

LI, Y. An Integrated Platform for the Internet of Things Based on an Open Source Ecosystem. *Future Internet*, MDPI, v. 10, n. 11, 2018. ISSN 1999-5903.

MARTINS, L. M. C. e.; CALDAS FILHO, F. L. d.; DE SOUSA JÚNIOR, R. T.; GIOZZA, W. F.; COSTA, J. P. C. L. da. Increasing the Dependability of IoT Middleware with Cloud Computing and Microservices. In: *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*. Austin, TX, USA: ACM, 2017. (UCC '17 Companion), p. 203–208. ISBN 978-1-4503-5195-9.

MARTINS, L. M. C. e.; CALDAS FILHO, F. L. d.; DE SOUSA JÚNIOR, R. T.; GIOZZA, W. F.; COSTA, J. P. C. L. d. Proposta de Adoção de Microsserviços em IoT [Proposal of IoT Microservice Adoption]. In: *Atas das Conferências IADIS Ibero-Americanas WWW/Internet 2017 e Computação Aplicada 2017*. Vilamoura, Algarve, Portugal: IADIS Press, 2017. p. 63–70. ISBN 978-989-8533-70-8.

MELL, P.; GRANCE, T. *The NIST Definition of Cloud Computing*. Gaithersburg, MD, USA, 2011. 7 p.

MENDONÇA, F. L. L. *Proposição de um Modelo de Interoperação Peer-to-Peer para Internet das Coisas - P2PIoT [Proposition of a Peer-To-Peer Interoperation Model for IoT]*. PhD Thesis — University of Brasília, Brasília, DF, Brazil, July 2019.

MINERAUD, J.; MAZHELIS, O.; SU, X.; TARKOMA, S. A gap analysis of Internet-of-Things platforms. *Computer Communications*, v. 89-90, p. 5–16, 2016. ISSN 0140-3664.

MUCCINI, H.; MOGHADDAM, M. T. Iot architectural styles. In: CUESTA, C. E.; GARLAN, D.; PÉREZ, J. (Ed.). *Software Architecture*. Cham, Switzerland: Springer International Publishing, 2018. p. 68–85. ISBN 978-3-030-00761-4.

NEWMAN, S. *Building Microservices*. 1st. ed. Sebastopol, CA, USA: O'Reilly, 2015. ISBN 978-1-4919-5035-7.

NIKOLOUDAKIS, Y.; PANAGIOTAKIS, S.; MARKAKIS, E.; PALLIS, E.; MASTORAKIS, G.; MAVROMOUSTAKIS, C. X.; DOBRE, C. A Fog-Based Emergency System for Smart Enhanced Living Environments. *IEEE Cloud Computing*, v. 3, n. 6, p. 54–62, November 2016. ISSN 2372-2568.

NÓBREGA, L.; GONÇALVES, P.; PEDREIRAS, P.; PEREIRA, J. An IoT-Based Solution for Intelligent Farming. *Sensors*, v. 19, n. 3, 2019. ISSN 1424-8220.

PAHL, C.; JAMSHIDI, P. Microservices: A Systematic Mapping Study. In: . Roma: SciTePress, 2016. v. 1, p. 137–146. ISBN 978-989-758-182-3.

PERERA, C.; QIN, Y.; ESTRELLA, J. C.; REIFF-MARGANIEC, S.; VASILAKOS, A. V. Fog computing for sustainable smart cities: A survey. *ACM Computing Surveys (CSUR)*, ACM, v. 50, n. 3, p. 32, 2017.

PERERA, C.; ZASLAVSKY, A.; CHRISTEN, P.; GEORGAKOPOULOS, D. Context Aware Computing for The Internet of Things: A Survey. *IEEE Communications Surveys Tutorials*, IEEE, v. 16, n. 1, p. 414–454, 2014. ISSN 1553-877X.

POLETTI, J. V.; MARTINS, L. M. C. e; ALMEIDA, S.; HOLANDA, M.; DE SOUSA JÚNIOR, R. T. A Real Data Analysis in an Internet of Things Environment. In: INSTICC. *Proceedings of the 4th International Conference on Internet of Things, Big Data and Security - Volume 1: IoTBDS,*. Heraklion, Crete, Greece: SciTePress, 2019. p. 438–445. ISBN 978-989-758-369-8.

PRACIANO, B. J. G.; DA COSTA, J. P. C. L.; MARANHÃO, J. P. A.; MENDONÇA, F. L. L. de; DE SOUSA JÚNIOR, R. T.; PRETTZ, J. B. Spatio-Temporal Trend Analysis of the Brazilian Elections Based on Twitter Data. In: *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. Singapore, Singapore: IEEE, 2018. p. 1355–1360. ISSN 2375-9259.

QANBARI, S.; PEZESHKI, S.; RAISI, R.; MAHDIZADEH, S.; RAHIMZADEH, R.; BEHINAEIN, N.; MAHMOUDI, F.; AYOUBZADEH, S.; FAZLALI, P.; ROSHANI, K.; YAGHINI, A.; AMIRI, M.; FARIVARMOHEB, A.; ZAMANI, A.; DUSTDAR, S. Iot design patterns: Computational constructs to design, build and engineer edge applications. In: *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*. Berlin, Germany: IEEE, 2016. p. 277–282.

RIBEIRO, C. F. C.; CALDAS FILHO, F. L. d.; MARTINS, L. M. C. e; ABBAS, C. J. B.; DE SOUSA JÚNIOR, R. T. Protocolos de Redundância de Gateway Aplicados em Redes IoT. In: *Anais do XXXVI Simpósio Brasileiro de Telecomunicações e Processamento de Sinais (SBrT 2018)*. Campina Grande, PB, Brazil: SBrT, 2018. p. 1065–1069.

ROOPA M.S.; PATTAR, S.; BUYYA, R.; K.R., V.; IYENGAR, S.; PATNAIK, L. Social Internet of Things (SIoT): Foundations, thrust areas, systematic review and future directions. *Computer Communications*, v. 139, p. 32–57, 2019. ISSN 0140-3664.

SCHENFELD, M. C. *Fog e Edge Computing : Uma Arquitetura Híbrida em um Ambiente de Internet das Coisas [Fog and Edge Computing: A Hybrid Architecture in an Internet of Things Environment]*. Master Thesis — Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, RS, Brazil, 2017.

SCHWERTMAN, N. C.; OWENS, M. A.; ADNAN, R. A simple more general boxplot method for identifying outliers. *Computational Statistics & Data Analysis*, Elsevier, v. 47, n. 1, p. 165–174, 2004. ISSN 0167-9473.

SHADIJA, D.; REZAI, M.; HILL, R. Microservices: Granularity vs. Performance. In: *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*. Austin, TX, USA: ACM, 2017. (UCC '17 Companion), p. 215–220. ISBN 978-1-4503-5195-9.

SHAH-MANSOURI, H.; WONG, V. W. S. Hierarchical Fog-Cloud Computing for IoT Systems: A Computation Offloading Game. *IEEE Internet of Things Journal*, v. 5, n. 4, p. 3246–3257, August 2018. ISSN 2372-2541.

SHI, W.; CAO, J.; ZHANG, Q.; LI, Y.; XU, L. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, IEEE, v. 3, n. 5, p. 637–646, October 2016. ISSN 2372-2541.

SHI, W.; PALLIS, G.; XU, Z. Edge Computing. *Proceedings of the IEEE*, IEEE, v. 107, n. 8, p. 1474–1481, August 2019.

SHROUF, F.; ORDIERES, J.; MIRAGLIOTTA, G. Smart factories in Industry 4.0: A review of the concept and of energy management approached in production based on the Internet of Things paradigm. In: *2014 IEEE International Conference on Industrial Engineering and Engineering Management*. Bandar Sunway, Malaysia: IEEE, 2014. p. 697–701. ISSN 2157-362X.

SILVA, C. C. d. M.; FERREIRA, H. G. C.; DE SOUSA JÚNIOR, R. T.; BUIATI, F.; VILLALBA, L. J. G. Design and Evaluation of a Services Interface for the Internet of Things. *Wireless Personal Communications*, January 2016. ISSN 0929-6212, 1572-834X.

SILVA, C. C. de M.; CALDAS, F. L. de; MACHADO, F. D.; MENDONÇA, F. L. L.; DE SOUSA JÚNIOR, R. T. Proposta de auto-registro de serviços pelos dispositivos em ambientes de IoT. In: *Anais do XXXIV Simpósio Brasileiro de Telecomunicações e Processamento de Sinais (SBrT 2016)*. Santarém, PA, Brazil: SBrT, 2016.

SMITH, A. B.; WALKER, J. P.; WESTERN, A. W.; YOUNG, R. I.; ELLETT, K. M.; PIPUNIC, R. C.; GRAYSON, R. B.; SIRIWARDENA, L.; CHIEW, F. H. S.; RICHTER, H. The Murrumbidgee Soil Moisture Monitoring Network Data Set. *Water Resources Research*, Wiley, v. 48, n. 7, 2012.

SOMMERVILLE, I. *Software Engineering*. 10th. ed. Harlow, UK: Pearson Education, 2016. ISBN 978-1-292-09613-1.

SOTOMAYOR, B.; MONTERO, R. S.; LLORENTE, I. M.; FOSTER, I. Virtual Infrastructure Management in Private and Hybrid Clouds. *IEEE Internet Computing*, v. 13, p. 5, 2009.

SPERLING, T. L. von; CALDAS FILHO, F. L. de; DE SOUSA JÚNIOR, R. T.; MARTINS, L. M. C. e; ROCHA, R. L. Tracking intruders in IoT networks by means of DNS traffic analysis. In: *2017 Workshop on Communication Networks and Power Systems (WCNPS)*. Brasília, DF, Brazil: IEEE, 2017. p. 1–4.

SPERLING, T. L. von; FRANÇA, B. de A.; CALDAS FILHO, F. L. de; MARTINS, L. M. C. e; ALBUQUERQUE, R. de O.; DE SOUSA JÚNIOR, R. T. Evaluation of an IoT device designed for transparent traffic analysis. In: *2018 Workshop on Communication Networks and Power Systems (WCNPS)*. Brasília, DF, Brazil: IEEE, 2018. p. 1–5.

TANEJA, M.; DAVY, A. Resource Aware Placement of Data Analytics Platform in Fog Computing. *Procedia Computer Science*, v. 97, p. 153–156, 12 2016.

TANENBAUM, A. S.; BOS, H. *Modern Operating Systems*. 4th. ed. Amsterdam, The Netherlands: Pearson Education, 2015. ISBN 978-0-13-359162-0.

THONES, J. Microservices. *IEEE Software*, v. 32, n. 1, p. 113–116, January 2015. ISSN 0740-7459.

TRUYEN, E.; BRUZEK, M.; LANDUYT, D. V.; LAGAISSE, B.; JOOSEN, W. Evaluation of container orchestration systems for deploying and managing NoSQL database clusters. In: *Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. San Francisco, CA, USA: IEEE, 2018. p. 17–20.

UNITED NATIONS (Ed.). *World Population Prospects 2019*. 2019. (Population Division). Accessed: Nov 22, 2019. [Online]. Available: <https://population.un.org/wpp/>.

VOORSLUYS, W.; BROBERG, J.; BUYYA, R. Cloud computing: Principles and paradigms. *Ch. Introduction to Cloud Computing*, p. 1–44, 2011.

WEISER, M.; GOLD, R.; BROWN, J. S. The origins of ubiquitous computing research at PARC in the late 1980s. *IBM Systems Journal*, IBM, n. 4, p. 693–696, 1999. ISSN 0018-8670.

YI, S.; LI, C.; LI, Q. A Survey of Fog Computing: Concepts, Applications and Issues. In: ACM. *Proceedings of the 2015 Workshop on Mobile Big Data*. Hangzhou, China: ACM, 2015. (Mobidata '15), p. 37–42.

ZHONG, R. Y.; XU, X.; KLOTZ, E.; NEWMAN, S. T. Intelligent Manufacturing in the Context of Industry 4.0: A Review. *Engineering*, v. 3, n. 5, p. 616–630, 2017. ISSN 2095-8099.

APPENDIX

I. MIDDLEWARE STANDARD APIS

Several middleware operations are performed through REST APIs that are made available to internal components and, in some cases, to external clients. Interaction via REST API is performed since the first version of UIoT middleware described in Ferreira, Canedo & de Sousa Júnior (2013). However, in this paper, the REST API was changed to fit the concept modeling described in Section 3.3.4. This appendix describes which defaults are used by middleware.

Operations are provided according to the methods, headers, and return codes specified in the HTTP protocol as described in Request for Comments (RFC)-7540. Therefore, query operations are performed using the GET method, POST register operations, and DELETE delete operations.

Respecting the definition of REST, some operations are performed on resources or resource groups. Inclusion is performed on resource groups, so the resource group address must be specified in the URI. This operation has the semantics that a new item will be added to the resource group at a position to be defined by the API. The deletion is performed on a specified resource, so the resource address must be specified in the URI. The query can be performed on both resource and resource groups, in which case the operation is a resource query and the second is a resource collection query, in both cases the URI must specify where the operation is will be held. The collection query allows data filtering through the use of query params.

Table I.1 shows a portion of services as an example of what will be described in Appendix II.

Table I.1: Description of the Proposed IoT Middleware API

URI	**HTTP method**	**Definition**
`/client`	POST	Send a new client information to be registered in the UIoT instance.
`/list/client`	GET	List a set of clients using multiple parameters to filter the result data set. If no query parameters is given, then the server will return all clients registered.
`/service`	POST	Send a new service information from a device to be registered in the UIoT instance.
`/data`	POST	Send data of a service to be registered in the UIoT instance.
`/list/data`	GET	List a set of data using multiple parameters to filter the result data set. If no query parameters is given, then the server will return all data registered.

The middleware API uses the codes described in Table I.2 to indicate the result of processing of the request.

As defined in the REST architecture, the services provided are stateless and the use of caching mechanisms and transmission optimization are transparent to the client and also their interface is standardized and uniformly decoupled from the display that potential clients make (FERREIRA,

Table I.2: Proposed IoT Middleware API response codes

Code	Allowed methods	Definition	Description
200	GET	Ok.	Client request was received, accepted and processed correctly and the request response is being sent in the response.
204	DELETE	No content.	Client request was received, accepted and processed correctly and the request response has no content to display.
400	POST	Bad request.	Client request was received but not accepted because it was performed in an invalid format and can not be processed by the server.
403	all	Forbidden.	Client request was received and, although the client is identified, the server refuses to authorize it.
404	GET, DELETE	Not found.	Client request was received but cannot be processed because the resource was not found at the specified address.
405	all	Method Not allowed.	Client request was received, but will not be processed because the server does not support the method required on the specified resource.

2014). Some services have prerequisites to run, and meeting these prerequisites is still done statelessly. For example, a service can only be registered after the customer has already been registered.

Data is transferred in JSON format, both on data input and on output. Listing I.1 shows a sample of a JSON data code.

```
{
  "clientId": "de780f5d-3960-4e9e-ae4a-ed4242b429ea",
  "serviceNumber": 3,
  "sensitive": 1,
  "value": [
    "27.3"
  ],
  "clientTime": "2019-02-06T19:17:49BRST",
  "tags": [
    "example-tag"
  ]
}
```

Listing I.1: Exemplo de corpo de uma requisição em formato JSON

II. API SUMMARY

II.1 DEVICE INTERFACE COMPONENT API

The REST API services available in Device Interface Component are described as follows:

- HTTP POST in `/client` to send a new client information to be registered in the UIoT instance;
- HTTP GET in `/client` to list a set of clients using multiple parameters to filter the result data set. If no query parameters is given, then the server will return all clients registered;
- HTTP POST in `/service` to send a new service information from a device to be registered in the UIoT instance;
- HTTP GET in `/service` to list a set of services using multiple parameters to filter the result data set. If no query parameters is given, then the server will return all services registered.
- HTTP POST in `/data` to send data of a service to be registered in the UIoT instance;
- HTTP GET in `/data` to list a set of data using multiple parameters to filter the result data set. If no query parameters is given, then the server will return all data registered.
- HTTP POST in `/formula` to send data of a formula to be registered in the UIoT instance;
- HTTP GET in `/formula` to list a set of formulas using multiple parameters to filter the result data set. If no query parameters is given, then the server will return all formulas registered;
- HTTP GET in `/formula/{ID}` to get data of the formula identified in the UIoT instance by ID (the provided id value), and;
- HTTP DELETE in `/formula/{ID}` to delete from the UIoT instance the formula identified in the UIoT instance by ID (the provided id value).

II.2 APPLICATION INTERFACE COMPONENT API

The REST API services available in Application Interface Component are described as follows:

- HTTP POST in `/registration` to send the application information to be registered in the UIoT instance;

- HTTP GET in `/registration` to retrieve the application information stored in the instance. If the application is not registered yet, then the server will response 404 HTTP status code;
- HTTP POST in `/subscriptions` to ask for a subscription in a service or a set of services in that UIoT instance;
- HTTP GET in `/subscriptions` to list the subscriptions the application has in the UIoT instance;
- HTTP GET in `/subscriptions/{subscription_id}` to retrieve information of the subscription identified by the id informed in `subscription_id`;
- HTTP PUT in `/subscriptions/{subscription_id}` to overwrite the content of the subscription identified by the id informed in `subscription_id`;
- HTTP DELETE in `/subscriptions/{subscription_id}` to delete the subscription identified by the id informed in `subscription_id`;
- HTTP POST in `/data` to send data of a service to be registered in the UIoT instance, and;
- HTTP GET in `/data` to list a set of data using multiple parameters to filter the result data set. If no query parameters is given, then the server will return all data registered.

II.3 MIDDLEWARE INTERFACE COMPONENT API

An instance needs to register itself in order to be able to interact with that instance. The middleware has this set of services to allow this interaction:

- POST in `/registration` to send the instance information to be registered in the UIoT instance;
- GET in `/registration` to retrieve the instance registration information stored in the target instance. If the instance is not registered yet, then the server will response 404 HTTP status code.
- HTTP POST in `/subscriptions` to ask for a subscription in a service or a set of services in that UIoT instance;
- HTTP GET in `/subscriptions` to list the subscriptions the instance has in that UIoT instance;
- HTTP GET in `/subscriptions/{subscription_id}` to retrieve information of the subscription identified by the id informed in `subscription_id`;

- HTTP PUT in `/subscriptions/{subscription_id}` to overwrite the content of the subscription identified by the id informed in `subscription_id`;
- HTTP DELETE in `/subscriptions/{subscription_id}` to delete the subscription identified by the id informed in `subscription_id`.
- HTTP POST in `/data` to send data of its own service or a service kept in this UIoT instance, and;
- HTTP GET in `/data` to query data using multiple parameters to filter the result data set. If no query parameters is given, then the server will return a dataset according to its own policies.

II.4 DIMS API

The REST API services available in DIMS are described as follows:

- HTTP POST in `/clients` to send a new client information to be registered in the UIoT instance.
- HTTP GET in `/clients` to list a set of clients using multiple parameters to filter the result data set. If no query parameters is given, then the server will return all clients registered.
- HTTP POST in `/services` to send a new service information from a device to be registered in the UIoT instance;

  HTTP GET in `/services` to list a set of services using multiple parameters to filter the result data set. If no query parameters is given, then the server will return all services registered.
- HTTP POST in `/data` to send data of a service to be registered in the UIoT instance, and;
- HTTP GET in `/data` to list a set of data using multiple parameters to filter the result data set. If no query parameters is given, then the server will return all data registered.

II.5 USER INTERFACE COMPONENT API

For UIMS, a new set of Web services have been created to support the creation of data display grouping functionality and services.

- HTTP POST in `/group` to send a new group information to be registered in the instance's UIMS;

- HTTP GET in `/list/group` to list a set of clients using multiple parameters to filter the result data set. If no query parameters is given, then the server will return all clients registered;
- HTTP POST in `/delete/group` to delete a group identified in the request's body;
- HTTP POST in `/func` to send a new function to the identified group;
- HTTP POST in `/add/func` to send a new post of the identified group:
- HTTP GET in `/list/group` to list all functions available to the user;
- HTTP POST in `/delete/func` to remove the identified function from the identified group, both identifications should be provided in the request's body, and;
- HTTP POST in `/delete/service` to remove the identified service from the identified group, both identifications should be provided in the request's body.