DESIGNING INTEROPERABLE, SCALABLE, AND MAINTAINABLE REMOTE PATIENT MONITORING IoT SYSTEMS

Pedro Lopes de Souza

Enschede, the Netherlands, 2024

Ph.D. – Thesis Series, No. XX-XXX

Cover Design: Izabel Lopes de Souza
Printing: Ipskamp, Enschede, the Netherlands

Graduation committee:

Chairman, secretary:	Prof. Dr. XXX (University of Twente)
Promotor:	Dr. L. Ferreira Pires (University of Twente)
Assistant Promotors:	Dr. J. L. R. Moreira (University of Twente)
Members:	Dr. R. R. Ciferri (Federal University of São Carlos)
	Dr. XXX (XXX)
	Dr. XXX (XXX)
	Dr. XXX (XXX)
	Dr. XXX (XXX)

Ph.D. Thesis Series No. XX-XXX
University of Twente
P.O. Box 217, 7500 AE
Enschede, the Netherlands

ISBN XXX-XX-XXX-XXXX-X
ISSN XXXX-XXXX (Ph.D. Thesis Series No. XX-XXX)
DOI XX.XXXX/X.XXXXXXXXXXXXX
http://dx.doi.org/XX.XXXX/X.XXXXXXXXXXXXX

DESIGNING INTEROPERABLE, SCALABLE AND MAINTAINABLE REMOTE PATIENT MONITORING IoT SYSTEMS

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de Rector Magnificus,
prof.dr. A. Tom Veldkamp,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op XXX XX XXX 2024 te XX.XX uur.

door
Pedro Lopes de Souza
geboren op 19 november 1985
te Campina Grande, Paraíba, Brazilië

Dit proefschrift is goedgekeurd door:
Dr. Luís Ferreira Pires (promotor)
Dr. João Luiz Rebelo Moreira (assistent-promotor)

DESIGNING INTEROPERABLE, SCALABLE AND MAINTAINABLE REMOTE PATIENT MONITORING IoT SYSTEMS

DISSERTATION

to obtain
the degree of doctor at the University of Twente,
on the authority of the Rector Magnificus,
Prof. Dr. A. Tom Veldkamp,
on account of the decision of the graduation committee
to be publicaly defended
on XXX the XXth of XXX 2024 at XX.XX.

by
Pedro Lopes de Souza
born on 19th November 1985
in Campina Grande, Paraíba, Brazil

This dissertation has been approved by:
Dr. Luís Ferreira Pires (promotor)
Dr. João Luiz Rebelo Moreira (assistant promotor)

TESE DE DOUTORADO

UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM
CIÊNCIA DA COMPUTAÇÃO

"DESIGNING INTEROPERABLE, SCALABLE AND MAINTAINABLE REMOTE PATIENT MONITORING IoT SYSTEMS"

AUTOR: PEDRO LOPES DE SOUZA
ORIENTADOR: DR. RICARDO RODRIGUES CIFERRI

SÃO CARLOS
06 /06/2024

CAIXA POSTAL 676
FONE/FAX: (16) 3351-8233
13565-905 - SÃO CARLOS - SP
BRASIL

Composição da Banca Examinadora:

Orientador:	Dr. Ricardo Rodrigues Ciferri, Universidade Federal de São Carlos (UFSCar), Brasil
Membros:	Dr. Luís Ferreira Pires, Universidade de Twente, Holanda
	Prof. Dra. Rossana Maria de Castro Andrade, Universidade Federal do Ceará (UFC), Brasil
	Dra. Renata Silva Souza Guizzardi, Universidade de Twente, Holanda
	Prof. Dr. João Paulo Andrade Almeida, Universidade Federal do Espírito Santo (UFES), Brasil

Abstract

Internet of Things (IoT) consists of networks of everyday objects, called 'things', equipped with embedded technology, sensors and actuators, capable of collecting and transmitting data through wireless network connections to the Internet. One of the big challenges in IoT is dealing with the huge amount of data produced by their things, in addition to dealing with the data heterogeneity, the varied capabilities of the things, the diverse offered services, and the different IoT platforms. For each domain and for each vendor, there is usually a specific and proprietary IoT platform, with no de facto standards it is currently being used or expected in the near future. Therefore, ensuring semantic interoperability when multiple and different IoT devices and system components must interoperate is one of the problems that needs extensive investigation in this area.

In addition to generating large volumes of data that require relatively complex processing, IoT systems have become increasingly complex and hard to maintain. Furthermore, they require high scalability, availability and data throughput in the context of critical applications such as in the Health domain.

IoT, Ubiquitous Computing, and Cloud Computing can be used in combination to build telemedicine systems to contribute to the Pervasive Healthcare's holistic vision of providing healthcare anywhere, anytime, to anyone. Such healthcare includes: prevention; short-term monitoring (*e.g.*, patient at home) and long-term monitoring (*e.g.*, elderly); incident detection and management; intervention in case of emergency; and treatment. Among the main tools of Pervasive Healthcare, Remote Patient Monitoring (RPM) stands out.

According to the World Health Organization (WHO), Noncommunicable diseases (NCD) are one of the biggest challenges in Health. The four main groups of NCDs, cardiovascular diseases, Diabetes, chronic respiratory diseases and Cancer cause about 70% of deaths in the

world. NCDs have a non-infectious origin, tend to be long-lasting and are the result of a combination of genetic, physiological, environmental and lifestyle factors. For a more effective treatment of NCD patients, WHO recommends a periodic monitoring of the health conditions of these patients. Some NCDs, such as Hypertension, Diabetes, Asthma and Obesity, allow for continuous RPM, enabling the patients and their health professionals to actively exert disease control.

In this context, this research investigated how to design interoperable, scalable and maintainable RPM IoT systems for continuous monitoring of patients suffering from NCDs that can be monitored remotely. The main contributions of this research are:

• A layered architecture to design RPM IoT systems for NCDs that can be monitored remotely.

• The design of three RPM IoT systems for monitoring Hypertension, which include:

- A sensor platform built in a wearable IoT device to capture the patient's vital signs;
- A mobile application running on the patient's mobile device to pre-process the clinical data;
- Two applications deployed in the cloud for processing, managing, and archiving clinical data;
- An IoT semantic model deployed in the cloud to enable interoperability of different devices and system components; and
- An IoT Microservices Architecture (MSA) deployed in the cloud to improve system, scalability, maintainability and data throughput.

• An approach to design interoperable, scalable and maintainable RPM IoT systems for NCDs that can be monitored remotely.

Acknowledgements

This PhD research started on June 2019 at the Ubiquitous Computing Group (GCU) of the Graduate Program of Computer Science (PPGCC) at the Federal University of São Carlos (UFSCar) in Brazil. In August 2022 I joined the 'International Academic Cooperation Agreement for Bi-National Supervision of PhD Students Aimed at Obtaining a Jointly Supervised Double PhD Degree', which was stablished between UFSCar and the University of Twente (UT) in the Netherlands. From November 2022 to October 2023, I continued my PhD research at the Group Semantics, Cybersecurity & Services (SCS) of the Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS) at UT within the scope of this international agreement. I returned to GCU/PPGCC/UFSCar in November 2023 to complete my PhD research and the writing of my thesis to be defended first at UFSCar in June 2024 and later at UT.

Completing a PhD thesis is a long process that requires a lot of persistence and dedication. There were many moments when I found myself lost, insecure and stuck in my research. It's a very unpleasant and discouraging set of emotions that drive you always from completing the journey. Thankfully, several people around me kept me motivated and pushed me forward until the end. And for that, they deserve my immense gratitude and appreciation. I firmly believe that completing a PhD thesis is a transformational process of growing as a person and as a scientist, and without these people, I would not have been able to achieve it.

I will start by thanking my parents, Wanderley e Inês, for always trusting me and supporting me during this journey. Coming from a family of academics gives you a lot of insights and benefits that most PhD students do not have. They help me a lot during my research by providing directions and ideas on how to proceed on many occasions. They also helped me write, revise my original monograph, and suggest improvements.

I am grateful to my sister Izabel and brother Adriano for all their love and care through these years of physical distance. They are the most brilliant and amazing people that I know. Their love and support helped me during dark periods such as the pandemic, and not even the distance kept us apart from being active in each other's lives..

My sincere appreciation to Prof. Dr. Ricardo Rodrigues Ciferri for supervising me, encouraging my research, and for his patience, motivation, and enthusiasm through all the development of this PhD project.

My gratitude to Prof. Dr. Luís Ferreira Pires for supervising me as well. His ideas and advice were fundamental to improving this work. He also painstakingly read and commented on every single paragraph of the original monograph. His comments played an essential role in improving the quality and accessibility of this dissertation. I have learned many things from him and deeply respect his commitment to research and students.

I want to demonstrate my appreciation to Prof. Dr João Luiz Rebelo Moreira for co-supervising me during my stay at the University of Twente. I also want to thank him for his intense participation in the publications resulting from this research. His PhD thesis was fundamental to give me insights on how to approach many problems during this thesis.

I want to express my heartfelt gratitude and thanks to Ronitti Rodrigues and Akmal Alikhujaev. They are wonderful people and very skilful developers. Their research results are one of the foundation pillars of this thesis. Ronitti's MSc thesis was the starting point for the first design cycle of my research. We had the opportunity to work together and help each other, improving our work results and promoting many pleasant moments. Akmal graduation project was also the starting point for the last design cycle of this research. Supervising his work was a wonderful experience, and I learned much from him.

I am honoured to have Prof. Dr. Ricardo Rodrigues Ciferri from UFSCar, Prof. Dr. Luís Ferreira Pires and Prof. Dr. Renata Silva Souza Guizzardi from UT, Prof. Dr. Rossana de Castro Andrade from Universidade Federal do Ceará (UFC), and Prof. Dr. João Paulo Andrade Almeida from Universidade Federal do Espírito Santo (UFES) on the defence committee of my PhD thesis.

I thank the Semantics, Cybersecurity & Services (SCS) group for sponsoring half of my stay at the UT. Special thanks to Prof. Dr. Giancarlo Guizzardi and Prof. Dr. Luís Ferreira Pires for believing in my research potential and, therefore, seeking resources so I could stay at UT and keep developing my PhD research. Being able to stay longer abroad has been an exciting and rewarding experience.

I thank my fellow lab mates in São Carlos (Brazil) for the stimulating discussions, for the sleepless nights we were working together before

deadlines, and for all the fun we have had in the last years. Also I thank my colleagues in University of Twente and SCS group for providing a good atmosphere and a wonderful experience living abroad.

Last but not least, I would like to thank I thank the Brazilian National Council of Technological and Scientific Development (CNPq) for sponsoring our research in the context of the Brazilian National Institute of Science and Technology in Medicine Assisted by Scientific Computing (INCT-MACC). I also thank the Brazilian Federal Agency for Support and Evaluation of Graduate Education (CAPES) for sponsoring 06 months of my stay at the UT.

Pedro Lopes de Souza
June, 2024 – São Carlos, Brazil

Contents

List of Figures

List of Tables

List of Acronyms

ACC	American College of Cardiology
AHA	American Heart Association
ANN	Artificial Neural Networks
API	Application Programming Interface
BADL	Basic Activities of Daily Living
BAN	Body Area Network
BDD	Behaviour Driven Development
BLE	Bluetooth Low Energy
BPMN	Business Process Model Notation
BPMT	Business Process Modelling Techniques
BVSD	Building and Validating System Design
CDMN	Clinical Data Measurement Devices
CDSS	Clinical Decision Support System
CIA	Confidentiality, Integrity and Availability
CPU	Central Processing Unit
CQ	Competency Question
CT	Communication Technologies
DAG	Directed Acyclic Graph
DB	Database
DBP	Diastolic Blood Pressure
DCLC	Designing Cloud Layer Components
DDD	Domain-Driven Design
DE	Domain Expert
DevOps	Software Development and IT Operations
DF	Development Framework
DFLC	Designing Fog Layer Components

DNS	Domain Name System
DRO	Domain Reference Ontologies
DS	Design Science
DSDO	Designing System Domain Ontologies
DSLC	Designing Sensor Layer Components
DSM	Design Science Methodology
DT	Development Team
ECG	Electrocardiography
EDA	Event-Driven Architecture
EEG	Electroencephalography
eHealth	Electronic Health
EHAW	eHealth/Ageing-well
EHR	Electronic Health Record
e/mHealth	Electronic/Mobile Health
ESB	Enterprise Service Bus
ESR	Extracting System Requirements
ET	Encryption Techniques
ETSI	European Telecommunications Standards Institute
FIESTA–IoT	Federated Interoperable Semantic IoT/cloud Testbeds and Applications
FOAF	Friend of a Friend
EWS	Early Warning System
FHIR	Fast Healthcare Interoperability Resources
GOE	Guidelines for Ontology Engineering
HE	Hardware Engineer
HIPAA	Health Insurance Portability and Accountability Act
HL7	Health Level Seven
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IADL	Instrumental Activities of Daily Living
ICDx	International Classification of Diseases
ICNP	International Classification for Nursing Practice
ICT	Information and Communication Technology
ICU	Intensive Care Unit
IDE	Integrated Development Environment
IHTSO	International Health Terminology Standards Organisation

IIR	Infinite Impulse Response
IO	Input/Output
IoHT	Internet of Healthcare Things
IoMT	Internet of Medical Things
IoT	Internet of Things
IoT-LSS	IoT-based large scale SOA
ISP	Internet Service Provider
JWKS	JSON Web Key Set
JWT	JSON Web Token
KQ	Knowledge Question
LAN	Local Area Network
MA	Monolithic Architecture
MAp	Mobile Application
MBIoT-SMH	Microservice-Based IoT System for Monitoring Hypertension
MD	Mobile Device
MeDIC	Medical Data Interoperability through Collaboration
mHealth	Mobile Health
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport
MSA	Microservices Architecture
M2M	Machine-To-Machine
NCD	Noncommunicable Disease
NIC	National Intelligence Council
OBO	Open Biological and Biomedical Ontologies
ODIoT-SMH	Ontology-Driven IoT System for Monitoring Hypertension
OE	Ontology Engineer
OIDC	OpenID Connect
OLAP	Online Analytics Processing
OLED	OntoUML Lightweight Editor
OLT	Ontology Languages and Tools
OLTP	Online Transaction Processing
OpenEHR	Open Electronic Health Records
OS	Operating System
OTS	Off-The-Shelf

OWL	Web Ontology Language
PAG	Physical Activity Group
PAS	Physical Activity Session
PCU	Processing and Communication Units
PLT	Programming Languages and Tools
PPG	Photoplethysmography
PTT	Performance Testing Tools
QoL	Quality of Life
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
REST	Representational State Transfer
RPM	Remote Patient Monitoring
RPP	Reactive Programming Paradigm
RPS	Requests Per Second
RTOS	Real-Time Operating System
SA	System Artifact
SABiO	Systematic Approach for Building Ontologies
SADT	Structured Analysis and Design Technique
SAIF	Service-Aware Interoperability Framework
SAREF	Smart Appliances REFerence
SAREF4EHAW	SAREF extension for the eHealth/Ageing-well domain
SB	System Behaviours
SBIoT-MPH	System Based on Internet of Things for Monitoring Patients with Hypertension
SBP	Systolic Blood Pressure
SCL	System Cloud Layer
SD	Standard Deviation
SDD	System Domain Documents
SDI	Sensor Data Ingestion
SDK	Software Development Kit
SE	Software Engineer
SE_d	Software Engineering for Development
SE_o	Software Engineering for Operations
SE_q	Software Engineering for Quality Assurance
SeAc	Sensors and Actuators
SF	System Features

SFL	System Fog Layer
SI	Semantic Interoperability
SLR	Systematic Literature Review
SM	Semantic Model
SMo	Semantic Module
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
SNOMED CT	Systematized Nomenclature of Medicine Clinical Terms
SO	System Ontology
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOSA	Sensor, Observation, Sample, and Actuator
SP	Sensor Platform
SPARQL	SPARQL Protocol and RDF Query
SS	System Scenarios
SSL	System Sensor Layer
SSN	Semantic Sensor Network
SSO	Single-Sign-On
ST	Simulation Tools
SUS	"Sistema Único de Saúde"
SWL	Semantic Web Language
SWRL	Semantic Web Rule Language
TIA	Technology Integration Adapters
*TQ_*	Technical Research Question
TS	Triplestore
TT	Testing Tools
UFO	Unified Foundational Ontology
UI	User Interface
UL	Ubiquitous Language
ULT	Ubiquitous Language Terminology
UML	Unified Modelling Language
URI	Uniform Resource Identifier
USNL	User Stories in Natural Language
UX	User Experience
WBAN	Wireless Body Area Network
WBSN	Wireless Body Sensor Network

WSN	Wireless Sensor Network
WHO	World Health Organization
W3C	World Wide Web Consortium
3DP	3-Dimensional Printer

Chapter 1

Introduction

The use of Information and Communication Technology (ICT) in Pervasive Healthcare, mainly the Internet of Things (IoT), Ubiquitous Computing and Cloud Computing technologies, has contributed to improve healthcare services, reduce medical errors, increase health professional productivity and decrease costs, as health information becomes available anytime and anywhere. However, for such benefits to be widely spread several technological problems surrounding healthcare services require better solutions in diverse areas, such as security, privacy, reliability, quality of service, performance, standardization, scalability, maintainability, and interoperability. Regarding the latter area, one of the main problems is to provide semantic interoperability across heterogeneous IoT systems, applications and devices.

According to the World Health Organization (WHO), Noncommunicable diseases (NCD) are one of the biggest challenges in Health, and for a more effective treatment of NCD patients WHO recommends periodic monitoring of the health conditions of these patients. Some NCDs, such as Hypertension, Diabetes, Asthma and Obesity, allow for continuous Remote Patient Monitoring (RPM), enabling the patients and their health professionals to actively exert disease control.

In this context, this research investigated how to design interoperable, scalable and maintainable RPM IoT systems for the continuous monitoring of patients suffering from NCDs that can be monitored remotely.

This chapter is organized as follows: Section 1.1 summarizes the relevant background for this research; Section 1.2 introduces the motivation for this research; Section 1.3 describes the main research objectives; Section 1.4 presents the research questions that have been answered to achieve these objectives; Section 1.5 deals with the adopted research methodology; Section 1.6 depicts the scope of this research; and Section 1.7 outlines the structure of this thesis.

1.1 Background

For some decades now, most countries have been facing the same healthcare issues: high costs to care for an increasing number of elderly people; a rapid increase in chronic diseases related to the 'modern lifestyle'; a growing demand for new health treatments and technologies; and a relative decrease in the number of health professionals in relation to the increase in population [1]. In this scenario, a major question is: 'How to offer better health services to a growing number of users, using limited financial and human resources?'.

Although ICT has made health processes more efficient, improving the quality and access to health services, several problems still require solutions, among which the following stand out: the significant number of medical errors that result in deaths and sequelae in patients; the high stress of health professionals, mainly caused by the overload of care; and the partial coverage of health services, mainly in rural and poor areas. These problems, combined with the issues mentioned above, create a number of challenges for policymakers, service providers, hospitals, insurers and patients [2].

Pervasive Healthcare has been considered a possible answer to that question, which according to [3] can be defined as: "healthcare to anyone, anytime, and anywhere by removing locational, time and other restraints while increasing both the coverage and quality of healthcare". To achieve such goals, Pervasive Healthcare uses ICT to enable patients to play a more active role in the management of their health. This includes support for performing self-management, self-monitoring, self-care, preventive efforts, cooperation between patient and healthcare institutions and between home and hospital, remote consultation, and monitoring [4].

ICT can help in the implementation of a Pervasive Healthcare model, thus contributing to the reduction of health costs, the expansion of service coverage areas, the reduction of medical errors and the increase in the productivity of health professionals. Such benefits are achieved, as health information is made available anytime and anywhere, via the use of mobile devices and wired and wireless communication networks [3]. This model must be supported by a pervasive, ubiquitous and interoperable computational structure, aiming to increase the health, quality of life and well-being of each individual. Such healthcare supports: prevention; short-term monitoring (*e.g.*, patient at home) and long-term monitoring (*e.g.*, elderly); incident detection and management; intervention in case of emergency; and treatment. Among the main tools of Pervasive Healthcare, Remote Patient Monitoring (RPM) stands out [5].

NCDs, also known as chronic diseases, have a non-infectious origin, tend to be long-lasting and are the result of a combination of genetic,

physiological, environmental and lifestyle factors [6]. The main NCDs, such as cardiovascular diseases, Cancer, chronic respiratory diseases and Diabetes, result in long-term consequences for people's health and usually require continuous treatment and care [6,7].

NCDs are currently considered the main causes of death and disability worldwide, becoming one of the biggest health challenges and causing concern due to its widespread and high social costs [8]. Every year, around 41 million people die worldwide from some type of NCD, corresponding to 74% of all deaths globally, with more than 3/4 of global NCD deaths occurring in low- and middle-income countries [6,9]. Cardiovascular diseases are the main cause of deaths (17.9 million), followed by Cancer (9.3 million), chronic respiratory diseases (4.1 million), and Diabetes (2.0 million) [6,9].

Hypertension is one of the most prevalent cardiovascular diseases, affecting 22% of the world's adult population over 18 years of age, and being the cause of death for approximately 9.4 million people every year [10]. In addition, this disease is a strong risk factor for other diseases acquired during life, such as coronary heart disease, left ventricular hypertrophy, cardiac arrhythmias including atrial fibrillation, strokes, and renal failure [11,12]. Hypertension is considered a silent killer with symptoms not visible for many years [10].

Diagnosis and monitoring of a hypertensive patient are necessary to maintain blood pressure within levels considered normal, whose values in adults are 120 mmHg for SBP and 80 mmHg for DBP [10,12]. However, the treatment of these patients is not always simple, as for a more effective diagnosis and medication, it is necessary to consider the risk factors and correlate them with disease progression data. In addition, conventional blood pressure measurements are usually performed only at the time of a consultation with a health professional, and the storage of these data is not structured so that they cannot be always queried when needed [13]. For a more effective treatment of hypertensive patients, periodic monitoring of the health conditions of these patients is necessary. To this end, vital signals, such as blood pressure, heart rate, and body temperature, must be periodically collected, *e.g.*, via sensors, and the corresponding clinical data must be stored, treated, and made available for analysis.

IoT, Ubiquitous Computing, and Cloud Computing can be applied to build telemedicine systems aiming at providing Pervasive Healthcare, by employing and combining technologies, such as wireless and sensors networks, and mobile devices, to the development of mobile applications and smart environments [14]. In this way, it is possible to increase the quality of health services and possibly decrease costs, allowing for less direct but still effective interaction between patient and health professionals, and providing ubiquitous access to health services [15].

Furthermore, data provided by different sensors may allow for better measurements that can be used to create more complete reports and, as a result, more accurate medical diagnoses, and more effective treatments [16].

IoT technologies are heterogeneous in terms of hardware and software at different levels, with different data formats and semantics, different devices from different manufacturers, different wired and wireless networking technologies, and different communication protocols. Usually, IoT devices produce large amounts of data, which can be first sent to a gateway to be pre-processed, for example, in mobile devices, and then forwarded to the cloud for further processing [17].

IoT heterogeneity can lead to incorrect or inefficient use, in addition to making it difficult for users to access the offered services. To cope with these different levels of heterogeneity, one should strive for semantic interoperability, which aims at providing a common data interpretation for the meaningful data exchange between different systems, applications and services. This encompasses the meaning of the data and their relationship, and requires common vocabularies to describe the data and ensure that data are unambiguously understood by the devices [18]. Many semantic models and approaches were proposed recently for IoT semantic interoperability, most of them employing ontologies, middleware, Semantic Web, and knowledge management systems [19].

In addition to generating large volumes of data that require relatively complex processing, IoT systems have become increasingly complex and hard to maintain. Furthermore, they require high scalability, availability and data throughput in the context of critical applications such as in the Health domain. Therefore, both academic researchers and software industry professionals have been proposing the use of Microservices Architecture (MSA) to overcome these challenges [20].

In short, microservices are independently deployable software artifacts, loosely coupled and modelled around a specific aspect of the business domain. MSA emerged as a specialization of Service Oriented Architecture (SOA), and was created mainly to address scalability and maintainability problems of Monolithic Architecture (MA). Although SOA and MSA revolve around the concept of loosely coupled services that encapsulate business functions, they diverge widely in governance patterns [21].

1.2 Motivation

Ubiquitous Computing, Pervasive Computing and IoT have been applied for at least two decades in several domains, such as Home, Transport, Logistics, Automation and Industrial Manufacturing, Process and Business

Management, Agriculture and Health, significantly transforming some of these domains, as it enables the connection of different types of devices and the efficient provision of quality services worldwide via the Internet [22]. In the Health domain, they gave rise to the concept of Pervasive Healthcare.

IoT systems have been used in several applications in the Health domain, mainly aiming to: reduce the waiting time in medical centres; monitor elderly; monitor patients with chronic diseases; track information to ensure that there are no errors in the delivery and administration of medication to patients; manage patient feeding; monitor patients with neurological disorders; and monitor patients performing physical activities [23,24].

IoT reduces the need for patients to travel from their homes and/or workplaces, mainly those with chronic diseases and the elderly. Based on the data provided by various IoT sensors, health professionals can monitor their patients remotely, anytime and anywhere, being able to perform diagnoses and prescribe treatments more quickly and efficiently, also from a distance [25]. It also allows patients to receive alerts at appropriate times to take their medications, including the ability to record dates and times when these medications were taken. This technological support provides health professionals with a more solid basis for making decisions regarding their diagnoses and the best therapeutic treatment for their patients [26].

One of the great challenges in IoT is how to deal with the huge amount of data produced by the IoT devices, in addition to the heterogeneity of data, the diverse capabilities of things and the services offered. For each domain and each vendor, there is usually a specific and proprietary IoT platform, with no de facto standards currently being used or expected in the near future. Therefore, ensuring the semantic interoperability of IoT systems between different types of platforms is one of the problems that requires intensive investigation.

Semantic interoperability refers to the ability of different systems to understand the meaning and context of information they exchange. This encompasses the meaning of the data and the relationship between the data. Ontologies, architectures, platforms, and knowledge management systems make it possible to achieve semantic interoperability. In IoT, semantic interoperability requires a common description of data and representation structures that characterize the IoT devices, their capabilities, and the data they produce. This requires developing a vocabulary to describe the structure and exchange of data, and ensure that are unambiguously understood by the IoT devices and by the software components, and can also be interpreted by machines [19].

Since IoT applications are generally built from a set of small and independent services, creating value-added services would require freely

combining services from different providers to take full advantage of the IoT's heterogeneity. Therefore, even if the direction is different, many of the requirements in MSA are similar to those in IoT, such as: lightweight communication; independent deployable software; a minimum of centralized management; and independent development techniques and technologies [27].

Although MSA and IoT have the same objective, i.e., building applications from different services, they come from different paths. MSA originates from software companies, where their monolithic applications proved to be difficult to scale and maintain, and the idea was to divide them into smaller, modular pieces. In IoT, small services are already provided to the extent that they align with the capabilities of the embedded devices they represent, and the challenge is to build value-added applications from these heterogeneous services, which requires individual services to be designed in order to allow a high degree of interoperability [27].

Therefore, the design of applications in IoT can be considered bottom up while in MSA top down, as the latter divides an application and does not deal with multiple providers. However, since software companies define some guidelines for MSA on how the individual distributed services need to be designed to work together properly, and since IoT services should align with this design, the use of MSA in IoT, in addition to its benefits for maintenance and scalability of IoT applications, brings also benefits to the interoperability of IoT services, thus facilitating the creation of value-added applications [27].

1.2.1 Monitoring of NCD Patients

NCDs usually arise due to unhealthy lifestyle habits and adverse physical and social environments. Therefore, risk factors such as poor diets, physical inactivity, tobacco use, excessive alcohol use and stress must be monitored and controlled. The development of NCDs occurs slowly in some patients and quickly in others, and the symptoms are usually perceived in the already advanced stages of these diseases. WHO estimates that it is possible to prevent up to 80% of cases of premature onset of cardiovascular disease, diabetes and stroke [6].

This WHO goal can be achieved with two actions [28]: the reduction of risk factors related to lifestyle; and act as soon as the first signs appear, such as high blood pressure, high blood glucose levels and 'bad' cholesterol. However, non-adherence to treatment is a common problem for patients with NCDs, as they are often unable to follow therapeutic orders, whether these be medication prescription compliance, physical activity, or lifestyle changes. Even if the high costs of treatment are not a

problem, issues such as fear, forgetfulness and lack of information directly affect patient adherence to these treatments.

The continuous monitoring of patients with NCDs that can be monitored remotely, such as Hypertension, Diabetes, Asthma and Obesity, helps achieve this WHO goal, as it allows possible causes to be identified that lead a patient not adequately complying with the prescribed treatment [29]. The use of IoT, Ubiquitous Computing and Cloud Computing technologies in this monitoring enables the provision and exchange of a large amount of diverse information: physiological signals can be continuously collected, for example, via wearable sensors, processed into clinical data, and sent to the health professional for analysis; the patient can then quickly receive, for example, on their mobile device, information from the health professional regarding medications and/or lifestyle adjustments to be taken; the patient can automatically receive alerts to remember the time of medication and/or to perform physical activities; and automatic alerts can also be sent to both, the health professional and the patient, when critical situations occur.

1.3 Objectives

The main objective of this research has been to investigate how to design interoperable, scalable and maintainable RPM IoT systems for the continuous monitoring of patients suffering from NCDs that can be monitored remotely. This investigation was conducted by designing three RPM IoT systems for an NCD, namely Hypertension.

In this regard, we investigate the use of: sensor platforms to capture patient's vital signs and to convert them into the corresponding clinical data; mobile applications to pre-process the clinical data for properly presenting them to the patient; cloud services to treat the clinical data for properly presenting them to the health professionals; ontologies for semantic interoperability in IoT systems; and MSA to improve scalability, maintainability and data throughput of IoT systems.

Based on the experience gained from this investigation, another objective of this research is to propose an approach to be used mainly in the development of IoT systems for monitoring patients suffering from NCDs that can be monitored remotely, such as Hypertension, Diabetes, Asthma and Obesity.

1.4 Research Questions

Design Science (DS) "is the design and investigation of artifacts in context" [30]. Artifact is a generic term to denote things ranging from conceptual structures, organizations, business processes, methods, and techniques, to hardware and software systems and components. An artifact is designed for interacting with a problem context in order to improve it. In DS, a research problem can be classified as: a design problem, which demands a change in the real world, requires an analysis of the stakeholder goals and can be formulated via a Technical Research Question (TQ); or a Knowledge Question (KQ), which demands knowledge about the real world without requiring a change in it. According to these DS concepts, the major design problem of this research can be formulated by the following main TQ:

TQ: *How to improve the design of RPM IoT systems with semantic interoperability and MSA in the domain of NCDs that can be monitored remotely?*

In order to obtain knowledge to answer this TQ, comprehensive and systematic literature reviews were carried out to explore, understand, and summarize the current main issues and solutions for semantic interoperability and microservices in IoT. To guide these literature reviews, the following KQs were defined:

KQ_1: *What are the main IoT application domains?*

KQ_2: *What are the main IoT architectures and platforms?*

KQ_3: *How do these IoT architectures and platforms address semantic interoperability in the IoT application domains?*

KQ_4: *How do these IoT systems address microservices in the IoT application domains?*

The results obtained with these literature reviews helped decompose the major design problem of this research into simpler subproblems, which could be formulated by the following TQs:

TQ_1: *How to populate the layers of the chosen architectural model with components of RPM IoT systems?*

Several IoT architectural models have been proposed in the literature, but apparently these models have not converged to a common reference

model, mainly due to different specific requirements. In the Health domain, Cloud-Fog based architectures have been applied in the development of IoT systems, and a three-layer architecture (Sensor, Fog, Cloud) have been proposed for monitoring patients and elderly. As the design problem of this research targets RPM, we choose this three-layer architecture to structure an IoT system for this domain, and TQ_1 must be answered in order to populate each of these layers with IoT system components.

TQ_2: *How to collect, process, and analyse clinical data of RPM IoT systems according to the chosen architectural model?*

IoT platforms allow different types of IoT devices to be connected, and for perform different types of operations, such as accessing, managing, and processing their data. IoT platforms can take actions related to the connected IoT devices, playing an important role in supporting the design and implementation of IoT systems. As a result, several IoT platforms have been developed, most for general purposes and financed by open international projects or by private industry. Since the current scenario of IoT platforms is broad, heterogeneous, and as for IoT architectures is characterized by a lack of standardization, TQ_2 must be answered in order to develop a platform to acquire physiological signals from patients and to treat the corresponding clinical data in the appropriate architecture layers.

TQ_3: *How to design a semantic model for RPM IoT systems according to the chosen architectural model?*

IoT is intrinsically heterogeneous in terms of both hardware and software at different levels, encompassing different data formats and semantics, different devices from different manufacturers, different wired and wireless networking technologies, and different communication protocols. We must deal with this heterogeneity to achieve interoperability, and semantic interoperability is a possible solution, since it aims at providing meaningful data exchange between different applications and services. As many models for IoT semantic interoperability have been proposed since the second half of the last decade, most of them based on ontology, middleware and Semantic Web, TQ_3 must be answered in order to develop a semantic model, based on devices ontologies and diseases ontologies, to be deployed in the appropriate architecture layers.

TQ_4: *How to design an MSA for RPM IoT systems according to the chosen architectural model?*

In addition to generating large volumes of data that require relatively complex processing, IoT systems have become increasingly complex and hard to maintain. Furthermore, they require high scalability, availability and data throughput in the context of critical applications such as in the Health domain. As both academic researchers and software industry professionals have been proposing the use of microservices to overcome these challenges, TQ_4 should be answered in order to develop an MSA to be deployed in the appropriate architecture layers.

1.5 Research Methodology

This research has followed the Design Science Methodology (DSM) [30] by employing the Design Science framework shown in *Figure 1-1*, which describes the interaction between an *artifact* and a *problem context* for producing *effects*. In this work, the artifact is an RPM IoT system for monitoring hypertensive patients; the problem context consists of several stakeholders, such as health professionals, patients and physical activity practitioners, and the existing systems for monitoring and preventing NCDs; and the effect is the appropriate monitoring of hypertensive patients, and the improvements of this RPM IoT system with respect to similar ones, satisfying in this way the WHO recommendation to provide comprehensive healthcare to NCDs patients.

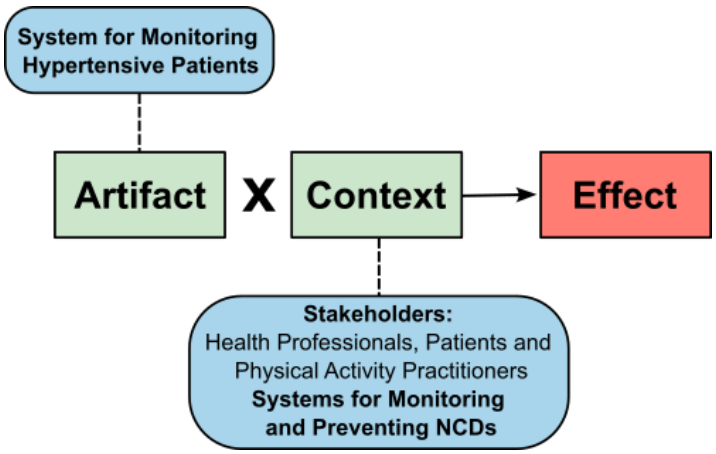


Figure 1-1 Design Science Framework

The Design Science framework prescribes an Engineering Cycle with five phases as illustrated in *Figure 1-2*.: (1) *Problem Investigation*: identify which phenomena must be improved and why; (2) *Treatment Design*: design one or more artifacts that could treat the problem; (3) *Treatment Validation*:

assess if these designs properly treat the problem; (4) *Treatment Implementation*: apply the treatment in the original problem context; and (5) *Treatment Evaluation*: evaluate how successful the implementation has been.

This research was structured according to a Design Cycle, which encompasses the three first phases of the Engineering Cycle. The result of a Design Cycle is a validated treatment that can trigger a new interaction through this cycle, or can trigger the next phase of the Engineering Cycle. In the last case, the validated treatment is implemented in a 'real world' context, where this implementation is used and evaluated. The last phase of the Engineering Cycle can trigger a new interaction through this cycle if deemed necessary. Each one of the interactions performed in this research is discussed below.

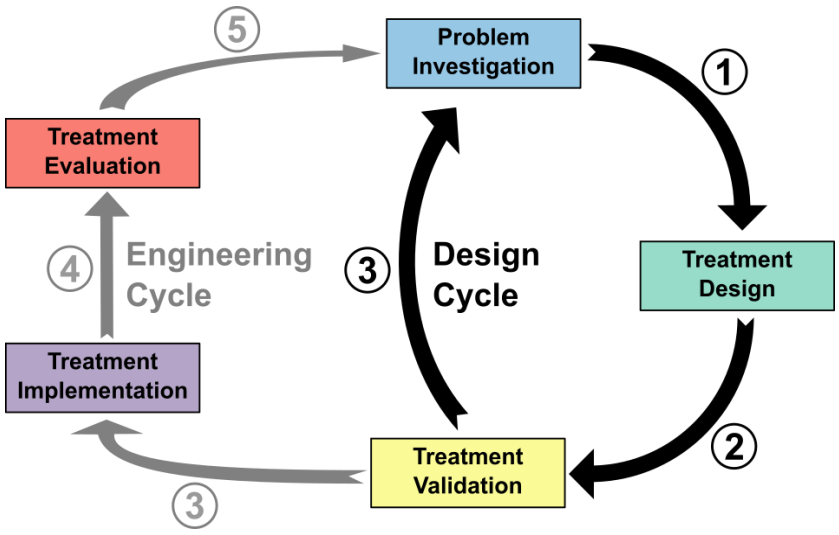


Figure 1-2 Design and Engineering Cycles

1.5.1 Iteration 1: System Based on Internet of Things for Monitoring Patients with Hypertension (SBIoT-MPH)

This research started with a first Design Cycle where the artifact *System Based on Internet of Things for Monitoring Patients with Hypertension (SBIoT-MPH)* [31] was developed.

In the problem investigation phase of this first cycle, a comprehensive literature review on Pervasive Healthcare was carried out. Next, this investigation focused on RPM systems for NCDs, and how to improve these systems using IoT, Ubiquitous Computing and Cloud Computing technologies in a cost-effective way.

The treatment design phase started by choosing Hypertension as the NCD to be treated, since it is one of the most prevalent NCDs that can be

remotely and effectively monitored. Next, the IoT three-layer architecture (Sensor, Fog, and Cloud) was chosen to structure the SBIoT-MPH artifact, and the necessary components to populate each of its layers were designed. A major concern in the SBIoT-MPH design was employing off-the-shelf (OTS) components to keep the system affordable.

To assess whether SBIoT-MPH meets the specified requirements, the treatment validation phase started by simulating an application scenario of this artifact, where a hypertensive patient has been monitored remotely by their cardiologist. In view of the potential large number of SBIoT-MPH users and the potential communication bottleneck between system components, an experiment was carried out to evaluate the choices of communication technologies that could be used in this system. Next, to assess the accuracy of blood pressure readings taken by the Sensor Layer component of SBIoT-MPH, we carried out an experiment to compare these readings with the readings taken by a digital blood pressure monitor. Finally, the Fog Layer and Cloud Layer components were tested individually, and the SBIoT-MPH was tested as a whole.

1.5.2 Iteration 2: Ontology-Driven IoT System for Monitoring Hypertension (ODIoT-SMH)

The validated SBIoT-MPH was the output of the first Design Cycle, which triggered the second Design Cycle of this research. In this cycle, the *Ontology-Driven IoT System for Monitoring Hypertension (ODIoT-SMH)* artifact [32] was developed. While this artifact reuses the SBIoT-MPH components, a new component and database have been introduced in the Cloud Layer to handle semantic interoperability.

In the problem investigation phase of this second cycle, two Systematic Literature Review (SLR) were performed. The first one addressed IoT and its use in Health, ontologies and their use in IoT and Health, and Cloud Computing and Big Data and their uses to support IoT, while the second one [19] included IoT application domains, IoT architectures and platforms, and semantic interoperability across IoT domains, architectures, and platforms.

The design treatment phase of this second cycle started based on the knowledge acquired with the SBIoT-MPH development and on the results from the problem investigation phase. When designing the ODIoT-SMH artifact, we sought to keep the original structure of SBIoT-MPH and reuse its components as much as possible. To this end, a semantic model for this new artifact was conceived to be fully deployed in the Cloud Layer, which was implemented through a semantic module introduced in this layer, communicating with the ODIoT-SMH applications of that layer, and with a Resource Description Framework (RDF) database (triplestore) [33] and

ontologies, which were also introduced in the Cloud Layer. As ontology engineering is essential for developing high-quality and reusable ontologies, we apply some IoT-specific best practices to reuse and extend existing ontologies for ODIoT-SMH.

To assess whether ODIoT-SMH properly treats the problem, we started the treatment validation phase of this second cycle by simulating an application scenario of this artifact for the remote monitoring of a hypertensive patient and their supervision by a cardiologist. To evaluate the scalability of this artifact, we simulated an application scenario considering different groups of geographically distributed patients and their cardiologists, also geographically distributed. Finally, the Fog Layer and Cloud Layer components were tested individually, and the ODIoT-SMH was also tested as a whole.

1.5.3 Iteration 3: Microservice-Based IoT System for Monitoring Hypertension (MBIoT-SMH)

The validated SBIoT-MPH also triggered the third Design Cycle of this research. In this cycle, the artifact *Microservice-Based IoT System for Monitoring Hypertension (MBIoT-SMH)* was developed.

The problem investigation phase of this third cycle started with a collaborative session to analyse the results of the SBIoT-MPH validation phase. Although we concluded that this system works well, we have faced the following challenges after deploying it: (1) as the number of requirements grows, so does the complexity of the system, requiring a new architectural approach to manage this complexity; and (2) since the sensor data read/write speeds (throughput) that could be supported were low, they were unlikely to deliver a comfortable User Experience (UX). Next, we carried out a comprehensive literature review on microservices, IoT systems and how they can be combined in an architecture. Taking into account the aforementioned challenges and the results of this literature review, we decided to redesign this system using MSA [21] to increase scalability, maintainability and improve sensor data throughput.

The design treatment phase of this third cycle started based on the knowledge acquired with the SBIoT-MPH development and on the results from the problem investigation phase. When designing the MBIoT-SMH artifact, we reused the Sensor Layer and Fog Layer components of SBIoT-MPH and built the Cloud Layer components using MSA. To do that, we first applied Domain-Driven Design (DDD) [34] to define domain models and identify bounded contexts. Then, we designed the MSA following its general principle 'code that changes together must stay together', and the Database-Per-Service pattern introduced in [21], which means that each

service should have its own database, unless it is unnecessary for the service, and other services will not be able to access it.

To assess whether MBIoT-SMH satisfies the specified requirements, the treatment validation phase started by simulating an application scenario of this artifact, where a hypertensive patient is monitored remotely by their cardiologist. We carried out an experiment to verify if MSA improves scalability when compared with a monolithic one, by running SBIoT-MPH and MBIoT-SMH in isolation with their own infrastructure to ensure that other processes do not cause congestion of computing resources. Then, we measure and analyse the throughput of both systems considering the number of concurrent users and the number of requests per second. Finally, the Fog Layer and Cloud Layer components were tested individually, and the MBIoT-SMH was tested as a whole.

1.6 Scope

This thesis aims to improve the design of RPM IoT systems with semantic interoperability and microservices, by proposing a semantic model and an MSA, both to be deployed in the Cloud Layer of these systems. More specifically, it focuses on developing RPM IoT systems for NCDs that can be monitored remotely, thus fulfilling the WHO recommendation to provide comprehensive healthcare for patients with this type of NCD.

In order to conceive an approach to the design of such systems, the scope of this thesis covers the development of artifacts, according to DSM and using the Design Cycle, for the remote monitoring of patients with Hypertension. It also points out how the developed artifacts can be adapted and/or extended for remote monitoring of patients with other types of NCDs, such as Diabetes, Asthma and Obesity, as well to allow remote monitoring of practitioners of physical activities aimed at preventing NCDs, which is another WHO recommendation.

The scope of this thesis also covers analyses during Design Cycle interactions aiming to:

- Evaluate the performance of the artifact, in terms of response time and data throughput, considering the communication technologies used.
- Evaluate the accuracy of clinical data measurements made by the artifact by comparing them with those made manually using a clinical device.
- Evaluate the scalability of the artifact, considering different groups of geographically distributed patients and their health professionals.

Finally, based on the experience gained from developing these artifacts, the scope of this thesis covers proposing an approach for designing interoperable, scalable and maintainable RPM IoT systems for NCDs that can be monitored remotely.

Issues of user privacy and security fall outside the scope of this thesis, as countries around the world have their own sets of regulations to ensure that health information is kept confidential and secure, such as the *General Data Protection Regulation* in the European Union [35] and the *Health Insurance Portability and Accountability Act (HIPAA)* in the United States [36].

1.7 Structure

The remaining of this thesis is structured as follows:

• Chapter 2 provides an overview of *Pervasive Healthcare*, *Internet of Things (IoT)*, *Ontology* and *Microservices*, as these subjects were the relevant background for this research. Readers familiar with these subjects may skip this chapter.

• Chapter 3 presents the *System Based on Internet of Things for Monitoring Patients with Hypertension (SBIoT-MPH)* artifact, which was developed in the first Design Cycle of this research, by describing its architecture, the components of its layers, and the performed analyses.

• Chapter 4 presents the *Ontology-Driven IoT System for Monitoring Hypertension (ODIOT-SMH)* artifact, which was developed in the second Design Cycle of this research, by describing its semantic model, the components of the semantic module deployed in the Cloud Layer, the designed ontologies, and the performed analyses.

• Chapter 5 presents the *Microservice-Based IoT System for Monitoring Hypertension (MBIoT-SMH)* artifact, which was developed in the third Design Cycle of this research, by describing its Domain Model and MSA, the MSA components deployed in the Cloud Layer, and a comparative analysis between SBIoT-MPH and MBIoT-SMH.

• Chapter 6 presents an approach for designing interoperable, scalable, and maintainable RPM IoT systems, which we draw from the experience of designing these three artifacts, describing its main activities, i.e.: the extracting of system requirements; the design of system domain ontologies; the design of the Sensor Layer components, focusing on its sensor platform; the design of the Fog Layer components, focusing on its mobile application; the design of the Cloud Layer components, focusing on its semantic model, and MSA; and the building and validation of the entire system.

• Chapter 7 deals with related works, involving RPM IoT systems for Hypertension, interoperable IoT systems in Health, and scalable and maintainable IoT systems in Health, and performs comparative analyses of these works with the work presented in this thesis.

• Chapter 8 concludes this thesis, highlighting its contribution and pointing to future research.

Chapter 2

Pervasive Healthcare, IoT, Ontology and Microservices

This chapter presents the relevant background for this research and is organized as follows: Section 2.1 introduces the concept of Pervasive Healthcare, presents some challenges in healthcare and how the use of ICT can help face them. It then addresses Remote Patient Monitoring (RPM), one of the main instruments for achieving Pervasive Healthcare. Finally, it discusses Noncommunicable diseases (NCD), especially the ones that can profit from remote monitoring, with a focus on Hypertension; Section 2.2 presents the main concepts of IoT, as well as Pervasive, Mobile, Ubiquitous and Cloud Computing. It then deals with Electronic(e) and Mobile(m) Health, describing the main technologies used in these domains, and introduces the concept of Internet of Healthcare Things (IoHT); Section 2.3 introduces the concept of Ontology, presenting its classification and some main issues. It then addresses IoT ontologies, above all those involving the description of sensors and actuators. Finally, it deals with Health ontologies, mostly those related to e/m-Health; and Section 2.4 introduces the concept of Microservice, and discusses the advantages and disadvantages of using a Monolithic Architecture (MA) and a Microservices Architecture (MSA) in the development of distributed software applications. It then discusses the relation between Service Oriented Architecture (SOA) and MSA. Finally, it presents two strategies for decomposing a system into a set of services to create an MSA.

2.1 Pervasive Healthcare

A major challenge in healthcare is how to improve health services, using limited human and financial resources, for an increasingly growing number of people. One way to face this challenge is to use a distributed healthcare

model, which permeates citizens' daily lives, in order to produce faster responses and help patients manage their own health.

The objective of Pervasive Healthcare is to enable this distributed model through the intensive use of ICT, making healthcare available anywhere, anytime and to anyone [3]. To this end, Pervasive Healthcare must be supported by a pervasive, ubiquitous and interoperable computational infrastructure, aiming to improve the health, well-being and Quality of Life (QoL) [37] of each individual, whether healthy or not.

Pervasive Healthcare can support the following activities: prevention; maintenance and checkups; treatment; incidence detection and management; emergency intervention and transport; and short-term (*e.g.*, patient at home), long-term (*e.g.*, elderly) and personalized monitoring [4]. All these activities can be supported through the combination of Ubiquitous Computing, Cloud Computing and IoT technologies to enable effective, efficient and reliable access to healthcare services and providers and clinical information anytime, anywhere. Regarding the monitoring requirement, these technologies have been successfully used to build RPM systems [15].

2.1.1 Remote Patient Monitoring (RPM)

A few years ago, patients requiring prolonged treatments generally had to remain in confined environments, such as hospitals and nursing homes. Today, thanks to advances in ICT, several computational systems to support healthcare have been developed, including those for RPM.

RPM systems aim to automate processes in healthcare, usually performing continuous monitoring of patients' vital signs, enabling them to receive care from health professionals and services from healthcare providers, regardless of their location. In this way, patients suffering from chronic and prolonged diseases who require continuous care, mainly the elderly, can live in the comfort of their homes and with a higher Quality of Life (QoL) [2].

Figure 2-1 illustrates a typical RPM scenario, in which patient's vital signs are capture by heterogeneous sensors in a Wireless Body Area Network (WBAN). These signals are transmitted via wireless communication technologies (*e.g.*, Bluetooth, ZigBee, Wi-Fi) to the patient's mobile device for pre-processing, and the corresponding clinical data is forwarded to the cloud via Internet. These clinical data are processed and stored in the cloud, making them available for consultation. Based on the history of clinical data collected, analyses can be performed manually or automatically and, in the event of an emergency, alerts can be sent to the patient, health professionals, healthcare service providers, or even ambulances to request for immediate assistance.

Depending on their use, complexity and kind of intervention by health professionals, RPM systems can be classified into four main types [39]:

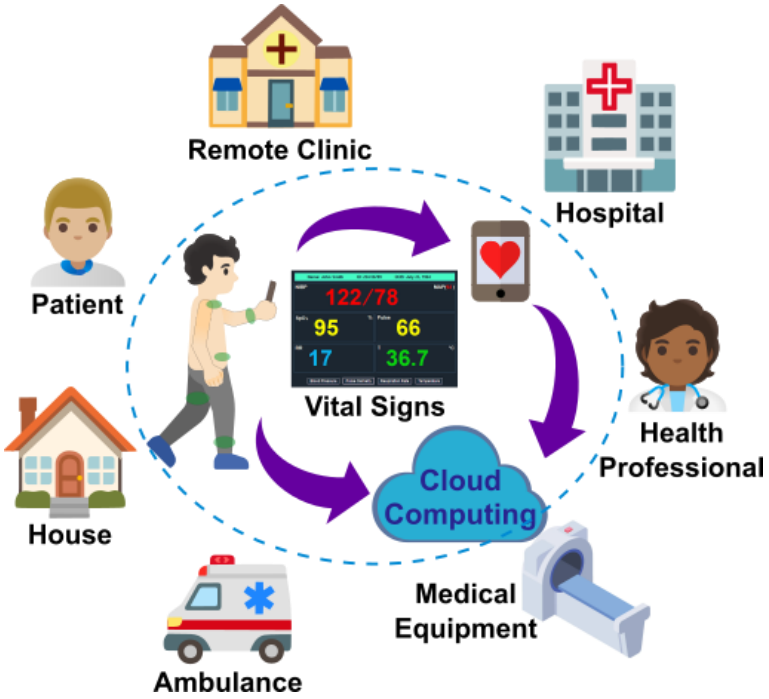


Figure 2-1 Remote Patient Monitoring (adapted from [38])

(a) *Self-monitoring*, when the system does not involve an external caregiver and the patient is responsible for their well-being and, consequently, the system depends on the patient's self-care, self-discipline and self-confidence. This type of system generally employs technologies to integrate invasive and non-invasive sensors, mobile devices and applications that allow the patient to collect and monitor their own clinical data;

(b) *Assisted monitoring*, when the system allows the patient to request assistance from a health professional through messages, calls, emails and emergency alerts. This type of system generally employs technologies to integrate WBANs, mobile devices and a remote server;

(c) *Supervised monitoring*, when the system allows the periodic sending and storage of the patient's clinical data in the cloud, so that they can be consulted by health professionals, family members and duly authorized people. Usually, health professionals can intervene in emergency cases and even configure parameters in WBANs; and

(d) *Continuous monitoring*, when the system, in addition to the functionalities described in the previous types, employs a real-time data

sharing approach between the patient and health professionals, using intelligent techniques to correlate data from different sensors, aiming to offer assistance or preventive actions to the patient.

2.1.2 Noncommunicable Diseases (NCD)

According to the WHO, NCDs tend to be of long duration and are the result of a combination of genetic, physiological, environmental and behavioural factors. Every year, NCDs kill 41 million people, which is equivalent to 74% of all annual deaths in the world, with 77% of annual NCD deaths (31.57 million) occurring in low and middle-income countries [6]. The four main groups of NCDs, namely cardiovascular diseases, Cancer, chronic respiratory diseases and Diabetes, account for over 81% of annual NCD deaths (33.2 million) and usually require ongoing treatments and care, causing disabilities and limitations in work and leisure activities [6,7].

Although often associated with the elderly, people of all age groups, regions and countries are affected by NCDs, since evidence shows that around 41% of annual NCD deaths (17 million) occur before the age of 70 years. Children, adults and the elderly are all vulnerable to the NCD risk factors, which include unhealthy diets, physical inactivity, exposure to tobacco smoke, harmful use of alcohol, and air pollution [6].

NCDs are driven by the 'modern way of life', which includes the rapid and unplanned urbanization, the globalization of unhealthy lifestyles and an aging population. Unhealthy diets and physical inactivity can increase blood pressure, blood glucose, blood lipids and weight. These metabolic risk factors can lead to cardiovascular diseases, which is the NCD that causes the most premature deaths. Among these metabolic risk factors, the main one is high blood pressure, to which 19% of global annual deaths are attributed, followed by high blood glucose, overweight and obesity [6].

To reduce the NCD impact on society, a comprehensive approach is needed that requires the participation of all sectors involved (*e.g.*, health, education, planning, finance, transport, agriculture) in order to reduce NCD risk factors and promote interventions to prevent and control NCDs. To reduce these risk factors, it is necessary to monitor them, as well as the progress and trends of NCDs, to obtain subsidies aimed at establishing adequate control and management policies [6].

NCD management includes the detection, screening and treatment of these diseases, as well as providing access to palliative care for people in need. Interventions with a high impact on NCDs can be carried out in primary healthcare to reinforce early detection and treatment, as evidence

shows that such interventions are excellent economic investments because they can reduce the need for more expensive treatments [6].

The *2030 Agenda for Sustainable Development* [40] recognizes NCDs as a major challenge for sustainable development, and government heads committed to developing by 2030 national responses to reduce by one third the premature mortality caused by NCDs through the prevention and treatment of these diseases. WHO plays a key leadership role in coordinating and promoting the global fight against NCDs for achieving the Sustainable Development Goals.

In 2019, WHO extended the *Global Action Plan for the Prevention and Control of Noncommunicable Diseases* [41] until 2030, which was originally established for the period 2013 to 2020, and called for the development of an *Implementation Roadmap* 2023-2030 [42] to accelerate the progress in the prevention and control of NCDs. This roadmap supports actions to achieve a set of nine global goals with the greatest impact on the prevention and management of NCDs.

A major problem in the treatment of NCDs is the lack of patient adherence, as patients are often unable to follow therapeutic orders, whether medication prescriptions, physiotherapy or lifestyle changes. One way to deal with these issues is to design and apply an adherence program like the one proposed in [43], which contains the following basic actions:
(a) *Enabling*, which involves preparing the patient to properly follow the medical prescription, for example, providing information on how the medication should be taken, and creating a caring relationship with the patient;
(b) *Reinforcing*, which involves providing feedback to the patient, for example, about their health status, goals achieved and to be achieved;
(c) *Prompting*, which involves efforts to remind the patient, for example, to take the prescribed medication doses at the correct time; and
(d) *Problem solving*, which involves identifying specific problems and proposing solutions for cases where the previous actions were not sufficient to promote patient adherence to treatment.

The use of Ubiquitous Computing, Cloud Computing and IoT technologies can assist in each one of these actions. The patient can receive, on their mobile or wearable device, information from health professionals regarding medications and/or lifestyle adjustments to be taken. It allows the patient to quickly receive feedback from health professionals and/or automatically generated, based on the analysis of data relating to their health status. Enables the creation of alerts to remind the patient when to take medication and/or perform physical activities. Continuous monitoring allows the provision of a large amount and

diversity of information, allowing the identification of possible causes that lead the patient to not adequately follow the prescribed treatment [26].

Wearable sensors also play a very important role in monitoring NCDs. The continuous collection and analysis of various types of physiological data related to these diseases allows health professionals to obtain early diagnoses and quickly carry out prevention and treatment actions, which can reduce the probability of these diseases getting worse and deaths.

2.1.3 Hypertension

Hypertension is one of the most prevalent NCD, affecting 22% of the world's adult population over 18 years of age, and being the cause of death for approximately 9.4 million people every year [10]. Hypertension is a characterized by high and persistent levels of blood pressure in the arteries, with values equal to or greater than 140/90 mmHg, which cause the heart to exert greater effort than normal to pump blood throughout the body. Symptoms typically appear in situations where blood pressure has already reached very high values and, in this case, chest pain, headache, ringing in the ears, nosebleeds, dizziness and blurred vision may occur.

Often asymptomatic, Hypertension tends to evolve and cause structural and/or functional changes in organs, such as the heart, brain, kidneys and blood vessels. Thus, it becomes one of the main risk factors for the occurrence of complications, such as stroke, myocardial infarction, aneurysm and kidney and heart failure [11,12]. The American Heart Association (AHA) classifies blood pressure into five categories, as shown in *Table 2-1* [44].

Table 2-1 Blood Pressure Categories (from [44])

Category	Systolic mmHg		Diastolic mmHg
Normal	Less than 120	And	Less than 80
Elevated	120-129	And	Less than 80
Hypertension Stage 1	130-139	Or	80-89
Hypertension Stage 2	140 or Higher	Or	90 or Higher
Hypertension Crisis	Higher than 180	and/or	Higher than 120

Based on this blood pressure classification, the AHA recommends that patients:

(a) *Normal*, continue to maintain healthy lifestyle habits, such as regular physical activities and maintaining a balanced diet;

(b) *Elevated*, adopt recommended measures, such as adopting healthy lifestyle habits, as they are at risk of developing Hypertension;

(c) *Hypertension Stage 1*, see a physician for follow-up, as in addition to prescribing changes in habits, the patient may consider the use of medications;

(d) *Hypertension Stage 2*, see your physician, as it will probably be necessary to use medication to control Hypertension; and
(e) *Hypertension Crisis*, if blood pressure readings reach this level and do not return to normal within five minutes, it is necessary to seek immediate medical attention.

Since 1980, the American College of Cardiology (ACC) and AHA have been translating scientific evidence into clinical practice guidelines with recommendations to improve cardiovascular health. In 2013, ACC/AHA collaborated with other organizations to publish four guidelines on: cardiovascular risk assessment; lifestyle modifications to reduce cardiovascular risk; management of blood cholesterol in adults; and management of overweight and obesity in adults. In 2014, ACC/AHA collaborated with other organizations to define guidelines for the prevention, detection, evaluation, and management of high blood pressure in adults, and a Prevention Subcommittee was established to help develop a set of guidelines on cardiovascular disease prevention. These guidelines, which are based on systematic methods for evaluating and classifying evidence, provide a foundation for quality cardiovascular care [45].

2.2 Internet of Things (IoT)

The term Ubiquitous Computing was coined by Mark Weiser in the early 90s in his work entitled "The Computer for the 21st Century" [46]. In his holistic vision, Ubiquitous Computing would provoke a third revolutionary "wave" in the history of Computing, the first two provoked by mainframes and personal computers respectively, since it was expected to cause a drastic change in the way people usually interacted with computers and in the way computing devices had influenced their lives until then.

This third wave would be the result of the increasing use of everyday objects, such as personal objects (*e.g.*, watches, glasses, cards), home and office objects (*e.g.*, heaters, lamps, locks) and household appliances (*e.g.*, stoves, refrigerators, washing machines), equipped with processing, storage and communication capabilities, which would permeate the daily routines of its users.

Although Pervasive Computing, Mobile Computing and Ubiquitous Computing are often considered synonymous, some authors distinguish them conceptually [47]:
(a) *Pervasive Computing* deals with the integration of computing devices into the environment in which they are inserted in a way that is imperceptible to the user. These devices are capable of obtaining information related to

this environment and using it to dynamically generate computational models, being also capable of detecting and interacting with other computational devices for providing services in an 'intelligent' way to their users;

(b) *Mobile Computing* deals with the ability of mobile computing devices (*e.g.*, smartphones, tablets) to provide access to information and services anywhere and at any time. Although these devices typically have lower processing, memory and energy capacities than desktops and notebooks, they are lighter, more portable and have communication interfaces with wireless networks, thus enabling connectivity with various services and facilitating user mobility; and

(c) *Ubiquitous Computing* encompasses the two previous computations, that is, it deals with the mobility and integration of computing devices into the environment and other devices, permeating the daily lives of its users and adding other technologies to create distributed, intelligent and highly connected systems.

Depending on the levels of embeddedness and mobility, four dimensions can be defined for computing [48], as illustrated in *Figure 2-2*. Traditional business computing, such as supported by desktop computers, present low levels of embeddedness and mobility, while Ubiquitous Computing, such as supported by smartphones or tablets, presents high levels of both. This Ubiquitous Computing characteristic allow its devices to interact with the environment and offer services to its users in a ubiquitous way and according to their contexts.

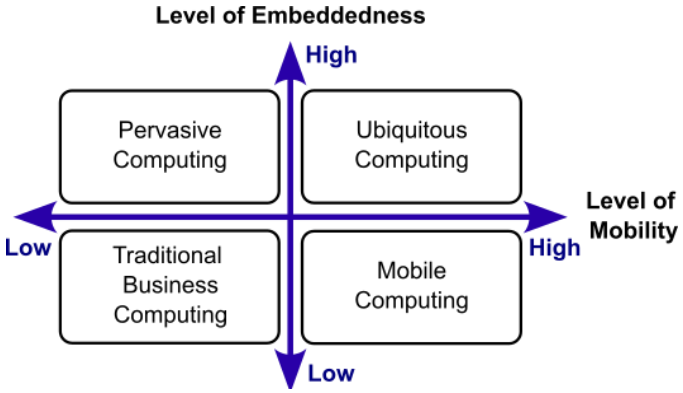


Figure 2-2 Computing Dimensions (from [48])

Ubiquitous Computing has been enriched by the large-scale adoption of technological advances in computing in the last two decades. Such advances allowed the development of increasingly smaller devices with greater processing, storage and communication capacity, which could be

embedded in various everyday objects and interconnected via the Internet more effectively and at a lower cost, giving rise to the IoT paradigm [49].

The term IoT was coined by Kevin Ashton in 1999. IoT consists of networks of everyday objects, called 'things', equipped with embedded technology, sensors and actuators, capable of collecting and transmitting data via wireless connections to the Internet [50]. This is a concept that refers to the digital interconnection of everyday objects with the Internet, which allows remote control of things that can be accessed as service providers. In this sense, with billions of sensors and actuators deployed and combined across multiple domain-specific platforms, Mark Weiser's vision of a hyper-connected world can be reached very soon.

IoT enables the development of applications in a wide variety of domains, among which we highlight Home, Transport, Logistics, Automation and Industrial Manufacturing, Process and Business Management, Agriculture and Health [51]. *Figure 2-3* illustrates a scheme for the interconnection of things, where the application domains were chosen based on the impact of the data generated, with users ranging from an individual to national organizations [52].

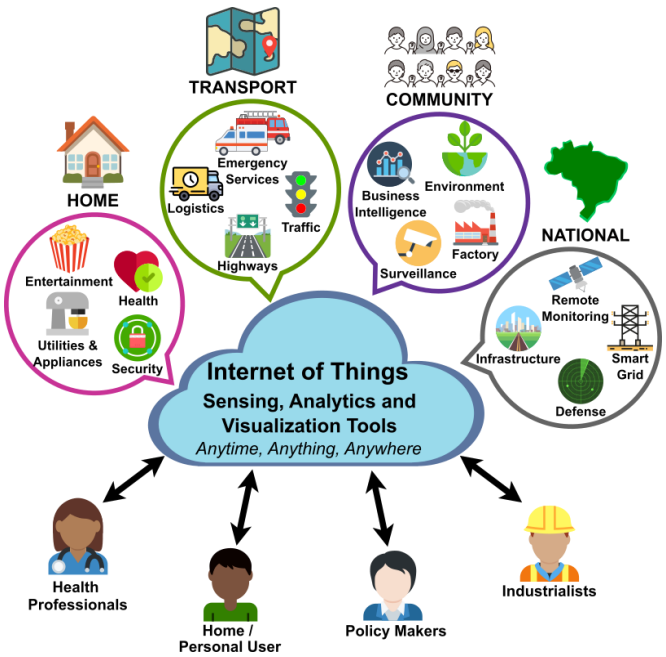


Figure 2-3 IoT Scheme, Application Domains and Users (adapted from [52])

The National Intelligence Council (NIC) included IoT in the list of six 'Disruptive Civil Technologies' [53]. The NIC predicted that 'by 2025, Internet nodes will reside in everyday things such as food packages, furniture and paper documents'. The NIC highlights future opportunities,

as 'popular demand, combined with technological advances, drives the widespread diffusion of IoT, which could, like the Internet, make an invaluable contribution to economic development'.

However, for IoT to become widely accepted, many problems need to be resolved, among which the following stand out: interoperability between interconnected objects; intelligence of these objects for their adaptation and autonomous behaviour; privacy; and security. IoT also generates challenges for communication networks, as things produce a large amount of data and generally have low computing, storage and energy capabilities. Cloud Computing provides the necessary infrastructure to process this enormous volume of data.

Cloud Computing refers to the provision of computing as a service, where shared hardware and software resources are accessible via any computer or mobile device connected to the Internet. Its cost-based model enables the provision of end-to-end services for companies and users to access applications on demand from anywhere.

2.2.1 IoT Architecture

An example of a high-level IoT architecture for a cloud-based solution, which is often used in several application domains, consists of three layers, as illustrated in *Figure 2-4*.

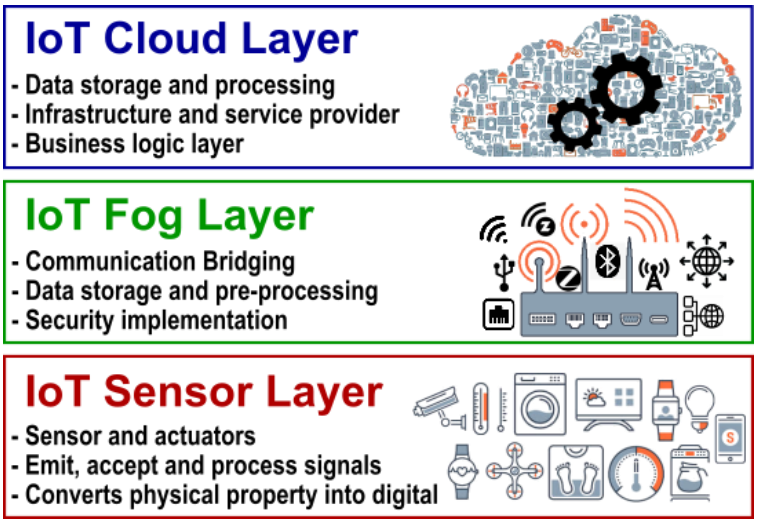


Figure 2-4 High-Level IoT Architecture Based in the Cloud

At the bottom layer are the things that generate data from different sources, possibly described using different semantics. In the middle layer is the gateway, which receives and eventually pre-processes the data coming from the bottom layer. At the top layer is the cloud, which receives,

processes and stores the data coming from the middle layer. According to the principle of interoperability, heterogeneous data can be interrelated and 'combined' at the top layer.

2.2.2 Electronic(e)/Mobile(m) Health and Internet of Healthcare Things (IoHT)

Electronic Health (eHealth) is the practice of healthcare supported by ICT, with web-based healthcare services being a potential area for IoT applications [54]. Mobile Health (mHealth) is part of eHealth that focuses on the use of connected mobile devices to provide healthcare services, such as patient monitoring, medication management, storage and retrieval of clinical data [55].

Among the main technologies used in e/mHealth, Wireless Sensor Networks (WSN) stand out, which are made up of distributed and interconnected sensor nodes that can cooperatively monitor physiological data, such as temperature, blood pressure, heart rate and movement. Wireless Body Area Networks (WBAN) are part of WSN and allow the collection of data streams relating to human physical, physiological and behavioural conditions, to process them in real time and/or store them in dedicated data repositories for later processing.

Data collection from heterogeneous sources is intense is e/mHealth, and its potential sources of information can be divided into: non-wearable sensors (*e.g.*, scale, oximeter, ECG meter, blood pressure monitor); wearable sensors (*e.g.*, pedometer, accelerometer, gyroscope, bracelet, in-ear devices); hospital information systems; and radiological information systems.

Wearable devices can also be classified into: sensors for home use (*e.g.*, glucose monitor); ambulatory sensors (*e.g.*, Holter monitor); implantable sensors for continuous monitoring; and sensors embedded in assistive devices. The data collected from the patient can be classified as: qualitative (*e.g.*, analysis of health status via questionnaires); and quantitative (*e.g.*, collection and analysis of information on physiological parameters).

In e/mHealth, data relating some parameter can be collected from different sources. For example, blood pressure can be measured in the clinic or at home using a portable physiological sensor and heart rate using a Holter monitor or a wearable sensor in a WBAN. As the context and accuracy of these measurements can be different, an IoT system must provide means to interrelate such data.

When applied to the healthcare domain, IoT can be called the Internet of Healthcare Things (IoHT) [56], whose main goal is to facilitate remote

data exchange for physical processes such as patient monitoring, patient progress, treatment, observation and consultation.

Figure 2-5 illustrates a high-level IoHT architecture that enables real-time monitoring and consultation and consists of three main layers: the Sensor Layer contains heterogeneous sensors such as electroencephalography (EEG), electrocardiography (ECG), photoplethysmography (PPG), motion and temperature, which are tied to the patient's body and send data to the next layer; the Personal Server Layer contains on-body and off-body coordinators, which receive data from sensors, eventually pre-process these data and transmit them to the Internet; and the Medical Server Layer contains telemedicine and medical servers, which receive data from the Internet, processes and stores these data to be accessed by medical professionals.

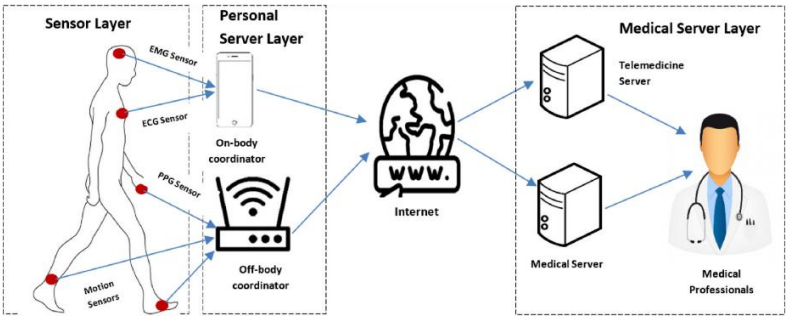


Figure 2-5 High-Level IoHT Architecture (from [56])

2.3 Ontology

The term ontology was introduced by philosophers in the 17th century as the 'philosophical study of nature and relations of being'. This term first appeared in a Computer Science-related conference in 1967 [57] and has since been used in Computer Science and Information Science. In these sciences and according to [58], an ontology is an explicit specification of a conceptualization, which represents objects with concepts and other entities that are presumed to exist in some domain, and the relationships between them.

According to [59], an ontology specifies the concepts, relationships and other distinctions relevant to modelling a domain, and this specification takes the form of definitions of a representational vocabulary (classes, relations, and so on), which provide meanings for the vocabulary and formal restrictions for its coherent use. Ontologies allow inferences based on the relationships between entities in a certain domain, which makes the use of ontologies a powerful feature for the Semantic Web.

In Computer Science, ontologies have been used mainly in the following areas [60]: Database and Information Systems; Artificial intelligence; and Software Engineering. In the context of Artificial Intelligence, an ontology defines a set of representational terms, associating the names of entities in the universe of discourse (*e.g.*, classes, relations, functions) with natural language texts, which describe the meanings of these names, and with formal axioms that restrict the interpretation and use of these terms.

There are several classifications for ontologies in the literature. As stated in [61] and illustrated in *Figure 2-6*, an ontology can be classified according to its level of generality and dependence into: (a) Top-Level Ontology, also called high-level or foundation ontology, describes concepts quite general (*e.g.*, Object, Property) that are independent of a particular domain; (b) Domain Ontology describes the vocabulary related to a generic domain (*e.g.*, Medicine, Course), specializing terms introduced in the Top-Level Ontology; (c) Task Ontology describes the vocabulary related to a generic task (*e.g.*, Diagnosis, Class), also specializing terms introduced in the Top-Level Ontology; and (d) Application Ontology describes concepts that depend on a specific domain and a particular task (*e.g.*, the roles played by entities from the Course domain during the execution of the Class task).

A top-level ontology facilitates the definitions of domain ontologies, and aims to support semantic interoperability between domain ontologies by providing a common basis for formulating these definitions. The reuse and maintenance of systems supported by domain ontologies requires a combination of these ontologies and, if they are derived from the same top-level ontology, this combination can be facilitated using appropriate tools. However, most available domain ontologies are not derived from the same top-level ontology. Furthermore, different domain perceptions, different usage intentions, and different languages give rise to specific application ontologies in the same domain that are often incompatible. To deal with these problems, application ontologies can be constructed from a domain reference ontology.

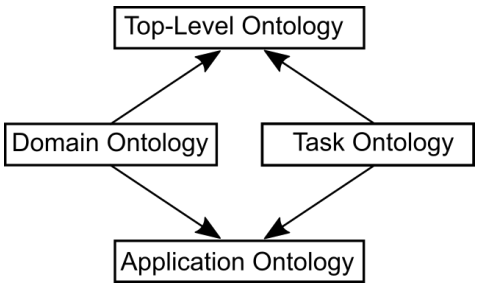


Figure 2-6 Ontology Classification (from [61])

2.3.1 IoT Ontologies

Interoperability in IoT systems has been extensively addressed at different levels, such as device, middleware and service, while semantic interoperability has received less attention. This fact can be observed when searching for semantic methodologies and tools for IoT and their use in practice, which requires the availability of ontologies in this area, preferably described in *Resource Description Framework Schema (RDFS)* [62] or *Web Ontology Language (OWL)* [63].

Most existing ontologies for IoT were developed in individual research projects, generally in the prototyping stage and incomplete, and were often abandoned. A exception is the *Semantic Sensor Network (SSN)* [64], which is an ontology for describing sensors and actuators, involved procedures, characteristics of interest, used samples, and observed properties. The SSN architecture is modular, and includes a lightweight core ontology, called Sensor, Observation, Sample, and Actuator (SOSA) [65], containing its elementary classes and properties. SSN and SOSA are capable of supporting a wide range of applications and use cases, such as satellite imagery, large-scale scientific monitoring, industrial and domestic infrastructure, social sensing, and observation-driven ontology engineering.

Another exception is the OpenIoT project [66], which developed an open source IoT platform that enables semantic interoperability of IoT services in the cloud. This platform is based on SSN and contains a sensor middleware that facilitates the collection of data from virtually any sensor, including mobile ones, ensuring appropriate semantic annotation. By combining Cloud Computing capabilities with sensory capabilities, OpenIoT enables on-demand cloud-based access to IoT resources. This platform offers several visual tools that make it possible to semi-automatically develop and deploy interoperable IoT applications, mainly in areas where semantic interoperability is crucial.

It is also worth highlighting the *Smart Appliances REFerence (SAREF)* ontology [67], which describes at a high level of abastraction the domain of smart home appliances. The basic concept in SAREF is the device, which represents a tangible object designed to perform a specific task, which requires the execution of one or more functions. For example, the device "washing machine" is designed to perform the task 'washing', and for this purpose one of its functions is "start/stop". SAREF offers a set of basic functions, which can be combined to define more complex functions of a single device. The model used in SAREF is generic enough to be used in other domains than smart home appliances, as it includes concepts such as device, sensor, actuator, service, state and function.

Although SAREF is most naturally applied to homes, offices, and limited public spaces, this ontology was extended for the eHealth/Ageing-well (EHAW) domain. This extension, named SAREF4EHAW [68], has been specified and formalised by investigating EHAW domain related resources, such as: potential stakeholders, standardization initiatives, alliances/associations, European projects, European Commission directives, existing ontologies, and data repositories. The SAREF4EHAW modular ontology is expected to:
(a) Allow the implementation of typical EHAW related use cases already identified, i.e., 'monitoring and support of healthy lifestyles for citizens', and 'Early Warning System (EWS) and Cardiovascular Accidents detection'; and
(b) Fulfil the EHAW related requirements, mainly the ontological ones that were mostly taken as input for the ontology specification.

2.3.2 e/mHealth Ontologies

Currently, the main data sources in e/mHealth are: WBAN-based systems, non-wearable sensors instantiated on medical devices, and the *Electronic Health Record (EHR)* [69]. EHR is a complete electronic record of all events and data related to a person's health status, which can be modelled in different ways depending on the application context. The creation of a reference model seems to be non-trivial or even impossible, especially given the different legal regulations that exist around the world. However, it is expected that the provision of semantic interoperability between different EHR models can deal with the most important problems of standardization, data sharing and reuse. In this sense, the development of high-level ontologies in e/mHealth will facilitate more generic interoperability with other application domains.

Although there are a reasonable number of ontologies for the Biomedical domain, the use of semantic technologies in these ontologies has not yet been widely adopted in e/mHealth. Key developments include:
(a) *Open Biological and Biomedical Ontologies (OBO)* [70] aim to provide a shared taxonomy across multiple biological and medical domains. The scientific ontology library was developed after specifying a set of best practices in ontology development, as an effort to promote interoperability. Among the numerous groups of ontologies in OBO, the Human Disease Ontology, the Clinical Measurement Ontology and the Symptom Ontology stand out;
(b) *International Classification of Diseases (ICDx)* [71,72,73] is a diagnostic classification developed by the WHO. Although the ICDx classifications only provide terminologies and not complete ontologies, there is a

mapping between the different classifications. The developed mapping mechanisms, between available classifications and terminologies, can become foundations to provide interoperability among IoT solutions in e/mHealth;

(c) *Systematized Nomenclature of Medicine Clinical Terms (SNOMED CT)* [74] is a taxonomy of medical terms used in clinical documentation and reports, which provides the basic general terminology for the EHR. SNOMED CT covers, for example, symptoms, diagnoses, procedures and structures of the body. It maps concepts defined in other international standards and classifications, such as ICD9 [71] and ICD10 [72]. SNOMED CT was used by the WHO in the development of ICD-11 [75], which is the new version of the international classification of diseases;

(d) *Health Level Seven (HL7)* [76] is an international organization for developing standards in health information technology. Initially, HL7 created the HL7 version 2 (HL7 v2) standards, which were later replaced by HL7 v3, a more formal set of standards based on a methodology for exchanging health-related data through point-to-point messages. With the development of semantic technologies, HL7 v3 began to be criticized for its weak interoperability and internal inconsistencies, and in 2007 HL7 developed the *HL7 Service-Aware Interoperability Framework (SAIF)*, which provides consistency between all HL7 artifacts, a base structure for further standardization, and a general high-level ontology. It provides a family of standards that explicitly describe governance semantics, behaviour, information, and compliance semantics necessary to achieve semantic interoperability. In 2014, HL7 *Fast Healthcare Interoperability Resources (FHIR)* was proposed, as a set of standards that describe data formats and elements for exchanging medical data. It provides support to represent and exchange of basic data elements (*e.g.*, patients, admissions, medications, diagnostic information) that can be referenced by their designated URIs, and its focus is to provide a set of APIs to enable the creation of interoperable mHealth applications. FHIR was built on top of HL7 v2, already implemented in many systems, provides a standardized ontology, and is already used in some e/mHealth projects, which makes it a good candidate for IoT applications; and

(e) *OpenEHR* [77] is a community that works on interoperability and computability in eHealth, with the main focus on the EHR. OpenEHR has developed a set of specifications, called archetypes, that define the reference model that can be used to implement specific clinical models, and allow the use of external healthcare terminologies (*e.g.*, SNOMED CT, ICDx). To apply semantic interoperability in e/mHealth, it is necessary to investigate medical data actually processed in healthcare systems. OpenEHR is a rich source of information on EHR modelling that should be considered a crucial part of e/mHealth.

2.4 Microservices

According to [21], 'microservices are an approach to distributed systems that promote the use of finely grained services that can be changed, deployed, and released independently'. A Monolithic Architecture (MA) and Microservices Architecture (MSA) are two different approaches to design and implement software systems. Although both architectures can be composed of several modules, an MA application has a single executable, while an MSA application is distributed so that its parts are microservices that can be executed independently [78]. The choice between MA and MSA depends on several factors as both architectures have their own strengths and weaknesses, and choice should be made based on quality requirements. Therefore, there is no definitive answer as to which is the best.

Companies can make the decision to keep their MA applications by only updating the Software Development and IT Operations (DevOps) tools and/or the programming language version, or migrate to MSA. This migration can be a challenging process, and many aspects can influence it, such as: the MSA complexity; the data structure complexity; the company organization; the experience of the work team in developing microservices; and the ability to properly split the MA to create the MSA. In [79], a comprehensive survey on MSA is presented, and the advantages and disadvantages of using MA and MSA are discussed.

Generally, it is recommended to keep MA if the application is small, with the following advantages: ease of development, deployment and debugging; and flexibility to work alone or on small projects. Otherwise, if the application is large or tends to grow a lot, MA has the following disadvantages: source code developed with a single language/technology; difficult to understand the business logic for making changes; greater complexity when adding new features; a single error or bug can crash the entire application; it is more difficult to perform functionality tests; necessary to retest all the application's functionalities in case of changes; and maintenance is difficult and expensive for the company, mainly when scaling infrastructure.

In the case of a large and complex application, the use of MSA can lead to improve understandability, availability, scalability, maintainability and resilience, since MSA has the following advantages: work teams are in charge of small and independent services; independent deployment, updating, replacement and scaling through the use of containers; ease of deployment using DevOps tools; ease of testing business functionalities; fault tolerance and better fault isolation; application security; shorter and faster development cycles; ease of reuse, replacement and innovation of services; and saves cost to company.

When developing an MSA application, a major problem is often the lack of MSA experience among the company's work team, which may stress the following disadvantages of MSA: time to learn MSA; difficulty in dealing with the use of different technologies; difficulty in splitting an MA application and developing microservices; difficulty in dealing with the data duplication due to MSA; difficulty in orchestrating an incremental number of microservices; difficulty in dealing with the possibly chatty interactions among microservices that cause communication delays; difficulty in dealing with the potential cyber-attacks to which microservices may be exposed; and difficulty in performing a full test on the MSA application.

At a high abstraction level, there are similarities between Service Oriented Architecture (SOA) and MSA, as both are architectural styles that structure a system as a set of services. MSA complies with SOA principles, but adds more principles and technologies. As this abstraction level is refined, significant differences between these architectures begin to appear [80]:

(a) *different technology stacks*. SOA applications typically use heavyweight technologies (*e.g.*, SOAP) and an Enterprise Service Bus (ESB), which is a smart pipe that contains business and message-processing logic to integrate the services. MSA applications typically use lightweight and open-source technologies, and the services usually communicate via message brokers or lightweight protocols (*e.g.*, REST);

(b) *different data treatments*. SOA applications typically have a global data model and share databases, whereas in MSA applications each service usually has its own database and domain model; and

(c) *different service sizes*. SOA applications usually consists of a few large services, whereas MSA applications typically consist of dozens or hundreds of smaller services.

Although decomposing a system into a set of services aiming to create an MSA depends on the designer's experience, in [80] two main strategies are presented that can help: Decomposition by Business Capability and Decomposition by Subdomain.

2.4.1 Decomposition by Business Capability

One strategy for creating an MSA is to decompose the system by *business capability*, a concept from business architecture modelling whose meaning is 'something that a business does in order to generate value' [80]. A business capability usually corresponds to a business object, for example: *Order Management* is responsible for *orders*; and *Customer Management* is responsible for customers.

According to [81], the decomposition by business capability as the following benefits: stable architecture since the business capabilities are relatively stable; work teams are cross-functional, autonomous, and organized around delivering business value rather than technical features; and services are cohesive and loosely coupled.

The business capabilities of a company are identified by analysing the company's purpose, structure, business processes and specialization areas. Starting points for identifying business capabilities are: *company structure*, different groups within a company might correspond to business capabilities or business capability groups; and *high-level domain model*, business capabilities often correspond to domain objects.

A service can be assigned to each business capability, except that this service is business-oriented rather than technical-oriented. The service specification consists of several components, including inputs, outputs, and service-level agreements.

Business capabilities are often organized into a hierarchy, and the set of capabilities for a given business depends on the type of business. For example, the top-level business capabilities of an online store include: *Product Order Management*, *Inventory Management*, *Order Management*, and *Delivery Management*. The corresponding MSA would have services corresponding to each of these capabilities, as shown in *Figure 2-7*.

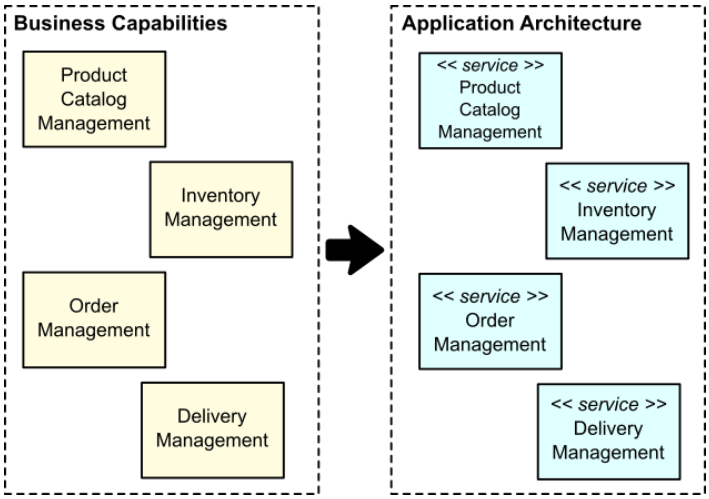


Figure 2-7 Example of Decomposition by Business Capability (adapted from [81])

2.4.2 Decomposition by Subdomain

According to [34], DDD is an approach for building complex software applications that is centred on the development of an object-oriented domain model. DDD refers to the application's problem space as domain,

and provides a vision of the system to be developed, generating a business domain model from where the knowledge of the business functionalities can be extracted. This domain model defines a common vocabulary called *Ubiquitous Language*, which is used by the work team, i.e., domain experts, architects and developers. DDD has two concepts that are applicable to MSA: *subdomains* and *bounded contexts*.

Subdomains are part of the domain and have their own domain models, and are identified using the same approach to identify business capabilities, i.e., analyse the business and identify the different areas of expertise, and the end result will likely be subdomains similar to business capabilities. Subdomains can be classified as follows [82]: *core*, key differentiator for the business and the most valuable part of the application; *supporting*, related to what the business does but not differentiator, and can be implemented in-house or outsourced; and *generic*, not specific to the business, and are ideally implemented using off the shelf software.

In DDD, the scope of a subdomain is called a bounded context. It is an explicit boundary within a business domain that provides functionality to the broader system while hiding its complexity, which includes the code artifacts that implement the model. We can create an MSA by applying DDD and assigning a service or a set of services to each subdomain within its bounded context.

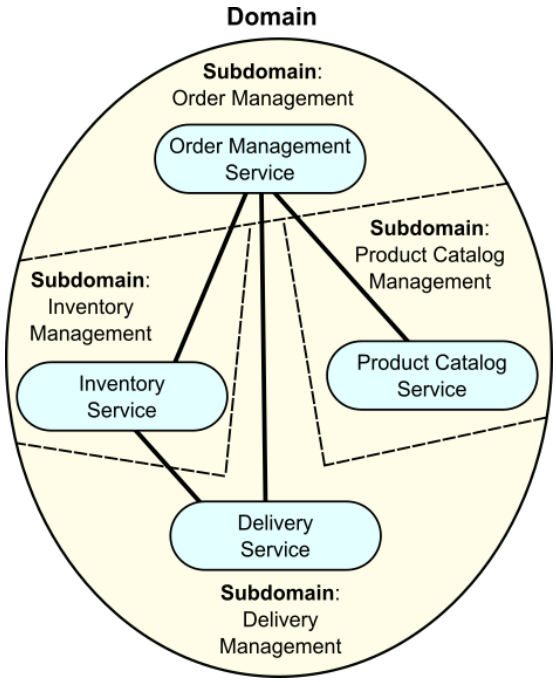


Figure 2-8 Example of Decomposition by Subdomains (adapted from [82])

According to [82], the decomposition by subdomain has exactly the same benefits as the decomposition by business capability, and has a similar main issue that is '*How to identify the subdomains?*', which must be addressed in the same way as in the decomposition by business capability. *Figure 2-8* shows the MSA for the online store example with its subdomains within their bounded contexts and the corresponding services.

Chapter 3

System Based on Internet of Things for Monitoring Patients with Hypertension

This chapter presents the *System Based on Internet of Things for Monitoring Patients with Hypertension (SBIoT-MPH)*, an RPM IoT system developed in the first Design Cycle of this research, which employs IoT, Ubiquitous Computing and Cloud Computing technologies. This system is the most basic version that was used as starting point and for benchmarking of this research. It allows patient's vital signs, such as blood pressure, heart rate and body temperature, to be captured through sensors built in a wearable device similar to a wristwatch. These signals are transmitted to the patient's mobile device for processing, and the generated clinical data are sent to the cloud to be properly presented and analysed by the health professionals responsible for the patient.

This chapter is organized as follows: Section 3.1 describes the SBIoT-MPH architecture, which was structured according to the IoT three-layer architecture, i.e., Sensor, Fog and Cloud; Section 3.2 deals with the Sensor Layer, describing the Sensor Platform developed for this layer; Section 3.3 deals with the Fog Layer, describing the mobile application developed for this layer; Section 3.4 deals with the Cloud Layer, describing the applications developed for this layer; Section 3.5 describes the performance evaluation of the communication technique used in SBIoT-MPH; and Section 3.6 describes the accuracy evaluation of blood pressure measurements performed by the Sensor Platform.

3.1 SBIoT-MPH Architecture

Several IoT architecture models have been proposed in the literature, but apparently these models have not converged to a common reference model. According to [19], some popular IoT multi-layered architectures are three-layer, five-layer, middleware-based layers, and service-based layers. In the e/m-Health domain, Cloud-Fog based architectures have been applied in the development of IoT applications [83], and in [84] a three-layer architecture (Sensor, Fog, Cloud) is proposed for monitoring patients and elderly. Since SBIoT-MPH also targets patient monitoring, this three-layer architecture was chosen to structure the components of this system, as shown in *Figure 3-1*.

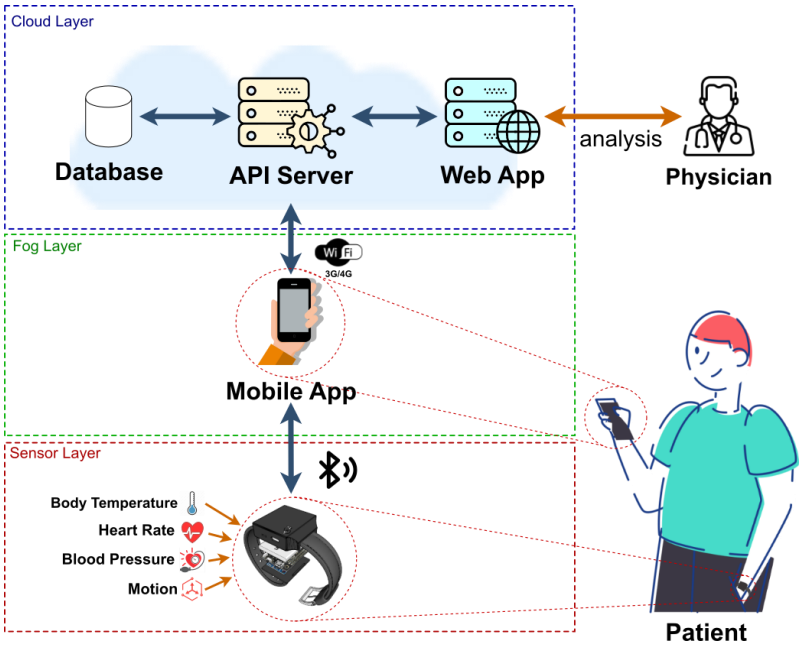


Figure 3-1 SBIoT-MPH Architecture

The Sensor Layer consists of a Sensor Platform built in a wearable IoT device similar to a wristwatch. The sensors of this platform capture the patient's vital signs (raw data) blood pressure, heart rate and body temperature. These raw data are filtered, aggregated and converted to digital format on the platform itself, and the resulting clinical data are transmitted to the Fog Layer in accordance with the Bluetooth Low Energy (BLE) standard, which is recommended for use in devices with power constraints, and that do not require a high data transfer rate and a high frequency, such as IoT sensors [85].

The Fog Layer consists of the Mobile App application that runs in the patient's mobile device (*e.g.*, smartphone, tablet) and where the clinical data are pre-processed, analysed, and classified according to health risk levels to be properly presented to the patient and for the eventual generation of alert signals. We chose to assign this functionality to the Fog Layer because in critical applications if many IoT devices would have to send raw data directly to be processed in the cloud, this communication path could potentially become a bottleneck. In this way, we assign delay-sensitive data processing to the Fog layer, which is close to the edge network and IoT devices, instead of processing these data in the cloud [86]. After that, the pre-processed clinical data are transmitted to the Cloud Layer via Wi-Fi or 3G/4G networks.

The Cloud Layer contains the API Server and Web App applications, which are responsible for processing, managing, and archiving clinical data. API Server provides an Application Programming Interface (API) for other applications to query and store data, while Web App provides a User Interface (UI) for health professionals to view and analyse patient clinical data and for system management.

3.2 Sensor Layer

The Sensor Platform consists of a Wireless Body Sensor Network (WBSN) that has been mounted in a bracelet to be worn by the patient on the wrist. The Sensor Platform captures some of the patient's vital signs (raw data), namely blood pressure, heart rate and body temperature. The clinical data obtained from these signals are transmitted via BLE to an application that runs on the user's mobile device (*e.g.*, smartphone, tablet). *Figure 3-2* shows the main modules of the hardware architecture of this platform.

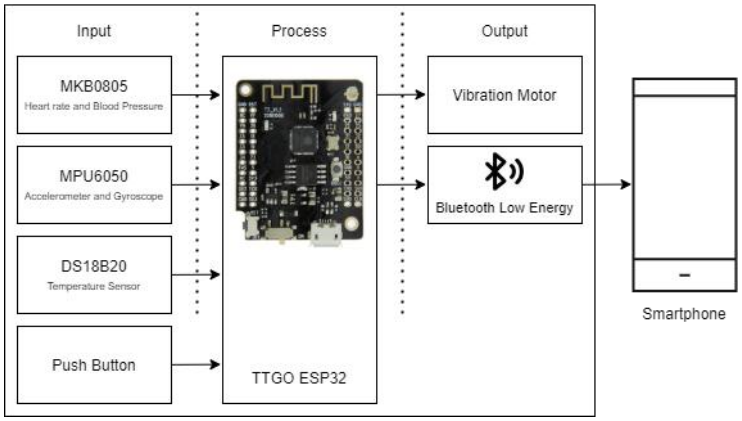


Figure 3-2 Hardware Architecture of the Sensor Platform

The MKB0805 module is a heart rate and blood pressure sensor that employs the Photoplethysmography (PPG) method to detect changes in blood volume in the microvascular tissue bed. This non-invasive and low-cost method consists of applying a light source to the surface of the skin and measuring the variations in light intensity caused by the absorption and reflection of this light by the skin tissue through a photodetector [87]. Since these variations depend on the amount of blood present in the optical path, this sensor reads the signals and calculates the SBP, DBP and heart rate, making these data available for reading by the TTGO T7 V1.3 MINI 32 ESP32 WiFi Bluetooth Module Development Board, shortly TTGO ESP32, which is the platform main module.

The MPU6050 module is a sensor that measures the acceleration of an object according to three coordinate axes (X, Y, Z), which is used to detect whether the patient is at rest for the reading of signals by the MKB0805 module. The DS18B20 module is a digital temperature sensor that measures temperatures between -55°C and 150°C with an accuracy of ±0.5°C.

The signals and data obtained by the sensors are processed in the TTGO SP32, which is often used in the development of IoT systems. This board is equipped with a low cost ESP32 microcontroller from Espressif Systems, which has low energy consumption and features Wi-FI 802.11b and Bluetooth v4.2 connectivity. The platform also has a Vibration Motor, which consists of a small motor capable of generating vibrations to alert the patient when notifications have to be generated.

Currently, the Sensor Platform captures only blood pressure, heart rate, body temperature and acceleration, but its modular architecture allows these sensors to be changed and/or new sensors to be added, for example, to capture other vital signs, so that other clinical data can be monitored.

3.2.1 Sensor Platform Implementation

Figure 3-3 shows the hardware scheme of the Sensor Platform with its components (TTGO SP32, MKB0805, MPU6050, DS18B20, Vibration Motor, Button, Led and Battery). Three types of buses are used for communication between the TTGO SP32 module and the sensor modules: a serial bus for the communication with the MKB0805 module; I2C for communication with the MPU6050 module; and OneWire for communication with the DS18B20 module.

Figure 3-4 shows two photos of the Sensor Platform prototype: (a) the prototype components fixed to a base structure to be inserted into the bracelet; and (b) the bracelet already fully assembled. The plastic frame

was manufactured with a 3D printer from 3D models designed using the FreeCad software.

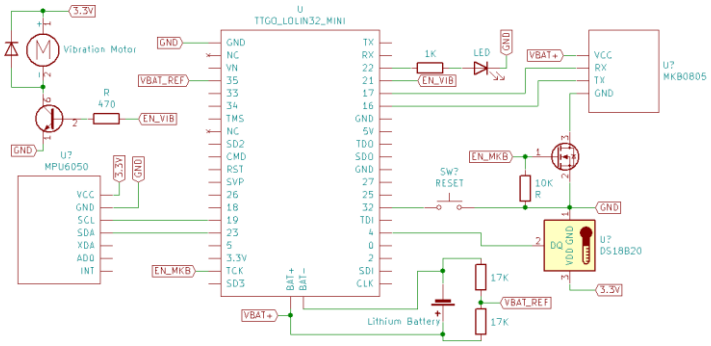


Figure 3-3 Hardware Scheme of the Sensor Platform

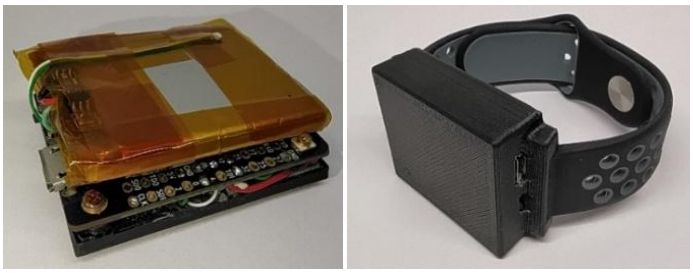


Figure 3-4 Prototype of the Sensor Platform

(a) Prototype Components (b) Assembled Prototype

The software embedded in the Sensor Platform has been developed using the ESP-IDF framework from the Espressif Systems, and the FreeRTOS Operating System kernel, which allows the execution of several tasks with different priority levels, thus enabling an efficient use of the TTGO ESP32 processors, and appropriate management of the peripherals (sensors). This software performs the following tasks:

(a) *Main*, which is executed when the platform is activated and is responsible for configuring the peripherals, initialising the sensors, describing the characteristics of each sensor, creating the services, registering the created services with BLE's GATT Server, and dynamically creating and allocating the other tasks;

(b) *Data Acquisition*, which is responsible for acquiring data from the sensor modules, processing these data, and storing the processed data in a non-volatile memory, until they are sent to the Mobile App application on the patient's mobile device; and

(c) *BLE*, which is responsible for maintaining the BLE communication with the patient's mobile device.

3.2.2 Blood Pressure and Heart Rate Reading

Since the accuracy of the measurement of SBP and DBP is affected by the patient's movement, the Sensor Platform reads these data only when the patient is at rest. To monitor the patient's movements, the acceleration of the patient in each of the three coordinate axes is obtained from the MPU6050 module at a sampling rate of 60Hz and is filtered, as illustrated in *Figure 3-5*.

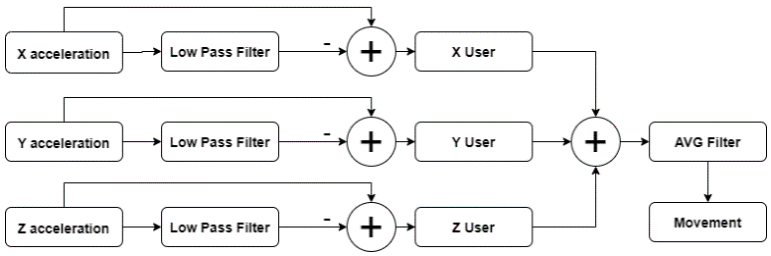


Figure 3-5 Digital Filter Diagram for Detecting the Patient's Movement

In addition to the patient's acceleration, the signal obtained by this module also captures the gravity force. Therefore, the filtering process illustrated in *Figure 4-5* is necessary to obtain an output signal that represents only the patient's movement. For this purpose, initially an Infinite Impulse Response (IIR) type low pass filter is applied to each signal, which attenuates all frequencies greater than 0.2 Hz, which correspond to the signal contribution of the gravity force that affects each coordinate axis. Then, the patient's acceleration on each axis is obtained by subtracting the gravity force from the total acceleration of the corresponding axis, and a single signal is obtained through the vector sum of these user accelerations. Finally, a moving average filter is applied to this single signal for attenuating noise, resulting in an output signal that represents the patient's movement.

The SBP, DBP and heart rate measures are obtained by the platform at 30-minute intervals in case the patient is stationary. To determine this, the patient's movement is analysed based on a predefined threshold value, and if this value is not exceeded in a 2-minute interval, the MKB0805 sensor is enabled and the average values of a total of 20 samples of each vital sign are sent to the patient's mobile device. If patient movement is detected during the sample collection interval, these samples are discarded and new ones are collected as soon as the platform detects that the patient has stopped moving. If the data collection process is not completed within a pre-set

time of up to 5 minutes, then the patient is warned to stop moving with a vibration generated with the vibration motor, and the patient is also warned later when the measurement process is completed. The values of all these time parameters can be set according to previously established recommendations.

3.2.3 Security

Data security was implemented in the Sensor Platform in two stages:
(a) the platform and the mobile device perform a pairing process, and establish an encrypted connection according to the methods and protocols defined in the security layer of the Bluetooth Core Specification; and
(b) the Mobile App performs a registration and authentication process in order to guarantee that the sensors that wish communicate with this application are legitimate data sources. *Figure 3-6* shows the registration and authentication process in the Sequence diagram of Unified Modelling Language (UML).

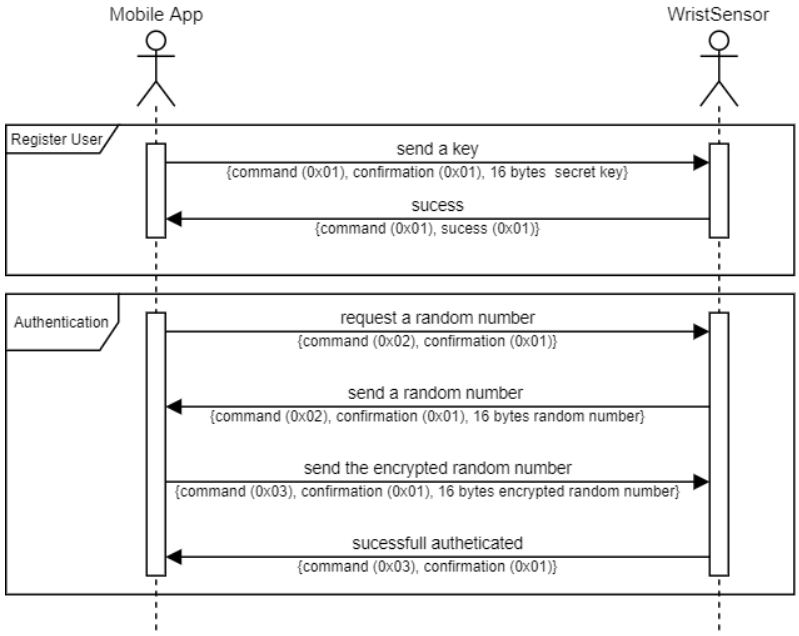


Figure 3-6 UML Sequence Diagram of the Mobile App Registration and Authentication Process

The registration and authentication process is performed only the first time the platform and mobile device are paired. To do so, the Mobile App sends a registration command and a 16-byte key to the platform, which in turn stores this key and returns a success message. Once the registration is complete, the Mobile App starts the authentication process by requesting

the platform a 16-byte string, which is randomly generated and sent to the Mobile App for encrypting this string using the registered key. Then, the Mobile App sends the encrypted string to the platform that in turn compares this string with the string it generated and encrypted using the same key. If the strings are identical (i.e., authenticity is confirmed), the platform returns a success message, authorising the Mobile App to read the platform's data.

3.3 Fog Layer

Mobile App is an application for Android devices that runs on the patient's mobile device, and acts mainly as a gateway, receiving data from the Sensor Platform and forwarding it to the cloud. *Figure 3-7* shows the Mobile App Use Case diagram, in which two actors interact with the Mobile App, namely *Patient* and *Background Service*.

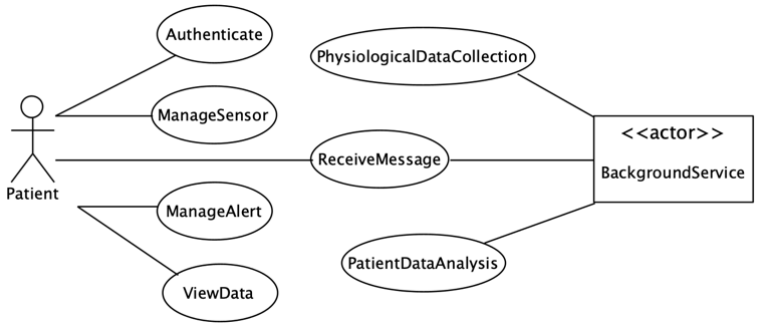


Figure 3-7 UML Use Case Diagram of Mobile App

The *Patient* actor represents the patient with hypertension, who triggers the following use cases:
(a) *Authenticate*: the patient provides the username and password previously registered in the Web App, and once authorised the patient can view their clinical data, messages, and risk alerts;
(b) *Manage Sensor*: the patient can check the connection status and battery level of the sensors; and
(c) *View Data*: the patient consults the data records of the last 24 hours, which are stored and made available in a database on the patient's mobile device.

The Background service actor represents a running process of the application that interacts with the Sensor Platform and triggers the following use cases:

(a) *Physiological Data Collection*: after the Mobile App and the Sensor Platform are connected, the patient's clinical data are collected, processed and sent to the cloud, where they are stored and made available to health professionals;
(b) *Patient Data Analysis*: the data received are analysed and classified according to four risk levels to the patient's health: NORISK; LOW, indicating hypertension can be controlled via changes in the patient's lifestyle, such as physical activities and healthier eating habits; MODERATE, indicating hypertension needs to be controlled by medication, in addition to lifestyle changes; and HIGH, indicating a crisis situation that means the patient has to consult a health professional immediately;
(c) *Manage Alert*: alerts are generated according to the risk levels and sent to the cloud for analysis by a health professional, and the patient automatically receives notifications regarding these alerts on the patient's mobile device and on the Sensor Platform. For MODERATE and HIGH risks, the health professional and an emergency contact immediately receive an SMS message with information related to the patient's health status; and
(d) *Receive Message*: messages recorded by the health professional on the Web App, such as recommendations for lifestyle changes, treatment adjustments involving changes in drug dosages, or the prescription of new drugs, are received and stored on the patient's mobile device, who in turn receives notifications related to these messages.

For each use case, we drew Sequence diagrams to define the internal operation to the Mobile App. *Figure 3-8* shows a Sequence diagram for the *Patient Data Analysis* use case, focusing on the analysis of the Blood Pressure data.

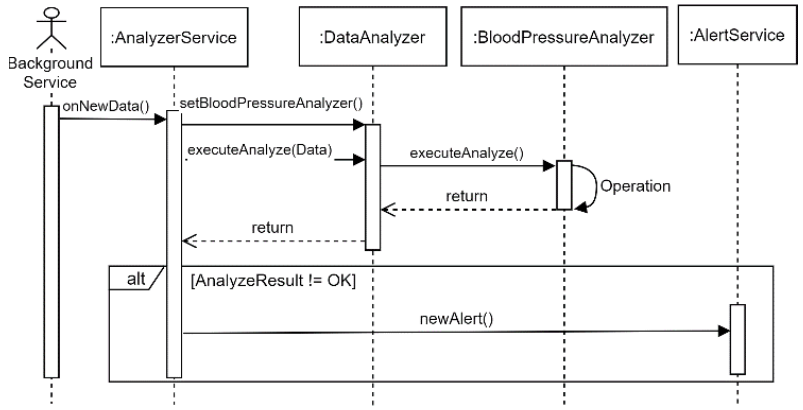


Figure 3-8 UML Sequence Diagram of the Patient Data Analysis Use Case

The following interactions occur when the Patient Data Analysis use case is performed:
(a) when receiving data, *Background Service* sends the OnNewData message to AnalyzerService;
(b) AnalyzerService configures a DataAnalyzer instance for handling the Blood Pressure data, and the received data are forwarded to this instance, which in turn performs data analysis through the routines defined in BloodPressureAnalyzer; and
(c) the analysis result is sent to Analyzer Service, which generates an alert to be handled by ManageAlert once any risk to the patient is detected.

The Mobile App software architecture was defined in terms of Class diagrams derived from the use cases and their corresponding Sequence diagrams, in accordance with the best practices of Object-Oriented software development. For example, Patient Data Analysis was implemented by employing the Strategy design pattern, which allows the definition of an algorithm family, and the selection of the actual algorithm for execution to take place at runtime. This pattern allows the Mobile App to be extended quite straightforwardly by adding new data analysers, without having to change the existing ones.

Figure 3-9 shows four main classes that realise the Patient Data Analysis use case according to the Strategy pattern. The BloodPressureAnalyzer and TemperatureAnalyzer classes correspond to the strategies of the Strategy pattern, and implement the analysis of the patient's clinical data, for Blood Pressure and Body Temperature, respectively. The DataAnalyzer class contains a reference to one of these analysis strategies, which is configured at runtime by the clients of this class. These strategies implement the IAnalizeImplementation interface, which exposes the unique executeAnalyze method for executing the selected algorithm.

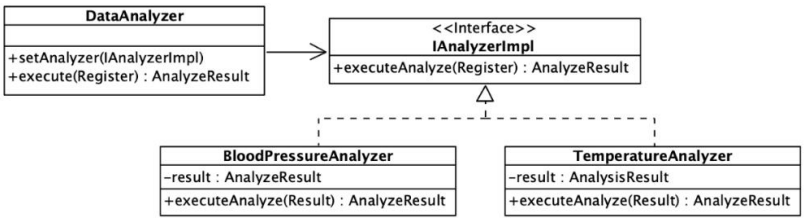


Figure 3-9 Excerpt of the UML Class Diagram for the Patient Data Analysis Use Case

Figure 3-10 shows an excerpt of the source code of executeAnalyze method of the BloodPressureAnalyzer class, in which the application first checks if the data record is of BLOOD_PRESSURE type and, if so, then the values of DIASTOLIC and SYSTOLIC pressures are captured. The RiskStatus is

classified based on the recommendations of the American Heart Association [44] and the blood pressure categories presented in *Table 2-1*.

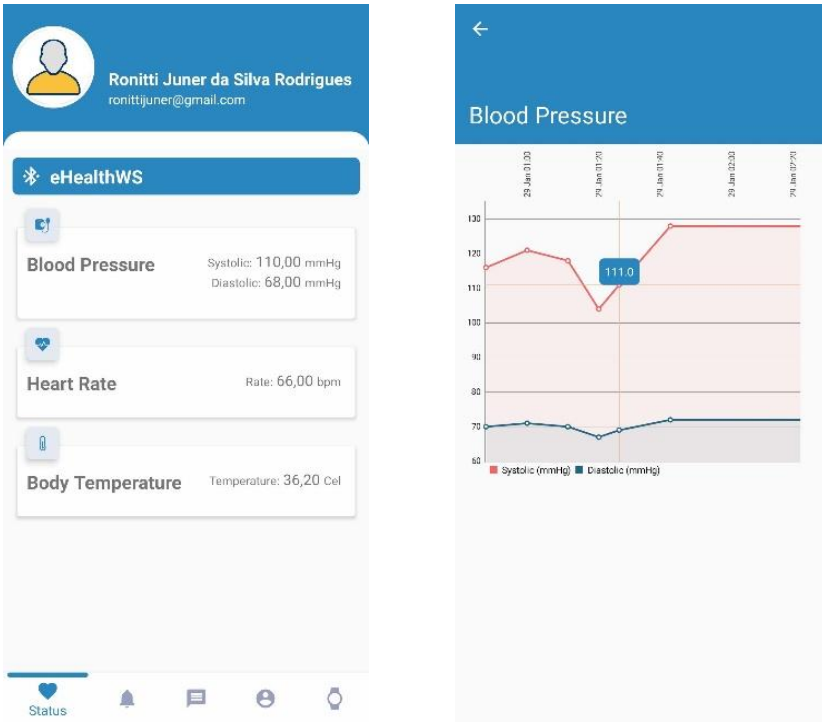
Figure 3-10 Excerpt of the Source Code of executeAnalyze Method

```
public AnalyzeResult executeAnalyze(Register register) {
  if (register.getType() == RegisterType.BLOOD_PRESSURE) {
      Data dia =
              register.getDataByType(DataType.DIASTOLIC);
      Data sys =
              register.getDataByType(DataType.SYSTOLIC);
      if (sys.getValue() < 120 && dia.getValue() < 80) {
          analyzeResult.setStatus(RiskStatus.NORISK);
      }
      else if ((sys.getValue() >= 120 && sys.getValue() <= 139) ||
              (dia.getValue() >= 80 && dia.getValue() <= 89)) {
          analyzeResult.setStatus(RiskStatus.LOW);
      }
      else if ((sys.getValue() >= 140 && sys.getValue() < 180) ||
              (dia.getValue() >= 90 && dia.getValue() < 120)) {
          analyzeResult.setStatus(RiskStatus.MODERATE);
      } else {
          analyzeResult.setStatus(RiskStatus.HIGH);
      }
  }
  return analyzeResult;
}
```

Mobile App was implemented in Java version 17.0.5 LTS using the API level 21 for Android 5.0 applications, which enables Mobile App for being compatible with around 94,1% of Android-based devices. For protecting the data stored on the patient's mobile device against eavesdropping, the SQLite library with the SQLChipher extension was used, which provides a database with 256-bit AES encryption. The Apollo GraphQL library 2.4 was used, which allows the generation of Java models for GraphQL queries [88].

Figure 3-11 shows two screenshots of the Mobile App User Interface (UI) displayed on the patient's mobile device:
(a) the interactive Data Collection screen, which allows the patient to view the sensors and the updated data collected by each sensor, and also gives access the screen for viewing the data history related to each sensor; and
(b) the Data History screen, which allows the patient to view historical data (*e.g.*, Blood Pressure historical data in *Figure 4-11*), which are stored on the patient's mobile device.

Figure 3-11 Screenshots of the Mobile App UI



(a) Data Collection Screen (b) Historical Data Screen

3.4 Cloud Layer

The Web App and API Server applications were deployed in the Cloud Layer so that they can request additional computing resources on demand whenever necessary and are accessible through the Internet.

3.4.1 Web APP

Web App can be accessed via a Web browser, both by health professionals to view and analyse their patients' clinical data, and by an administrator to manage these professionals and patients. *Figure 3-12* shows the Web App Use Case diagram where two actors, namely Administrator and Health Professional, interact with the Web App.

Figure 3-12 UML Use Case Diagram of Web App

The Administrator actor triggers the following use cases:
(a)*Manage Patients*: performs the registration and maintenance of patient data, such as their personal data and the types of diseases to be monitored;
(b) *Assign Patient to Health Professional*: assigns a health professional to the patient, who will be then responsible for monitoring the patient's clinical data; and
(c) *Manage Health Professionals*: performs the registration and maintenance of health professionals' data.

The Health Professional actor triggers the following use cases:
(a) *Analyze Health Data*: allows the health professional to access their patients' clinical data, and then analyse these data and make decisions regarding those patients' treatments;
(b) *Send Message*: sends messages to the patient's mobile device via the Mobile App, with recommendations regarding that patient's treatment or lifestyle. Such messages can be categorized by the health professional into 3 priority levels, which determines the order in which they are displayed to the patient; and
(c) *View Alerts*: visualizes alerts automatically generated in the Mobile App that were forwarded to the API Server, helping health professionals in the analysis of their patients' clinical data.

Figure 3-13 shows a screenshot of the Web App UI, which allows the health professional to apply filters, such as the desired clinical data type and the period for analysis. To help the health professional with this analysis, reference lines related to pre-established normal values are shown on the graph for each clinical data type.

Web App was implemented in JavaScript, and React and Material UI libraries were employed since they facilitate the implementation of user interfaces. Integration with the API Server was carried out via the Apollo Client library, as it supports the execution of queries to GraphQL APIs, managing the cache and automatically updating the UI data.

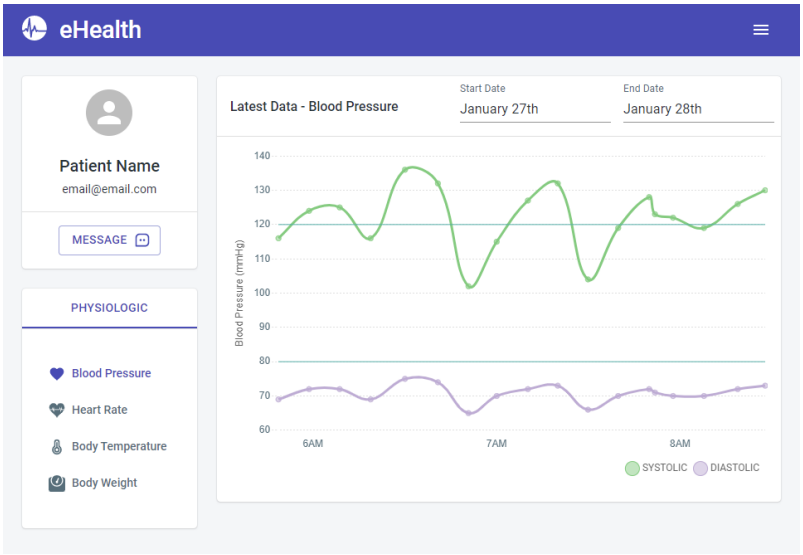


Figure 3-13 Screenshot of the Web App UI

3.4.2 API Server

API Server operates as a server, providing a secure API so that Mobile App and Web App can store and retrieve system information. This API employs the GraphQL query and data manipulation language, which were originally developed by Facebook as an alternative to REST.

GraphQL has been considered more efficient and able of mitigating the inflexibility and complexity associated with REST, such as: complex HTTP requests, when multiple requests to different endpoints are necessary to query different resources; data overfetching, when more data is returned than required by the application; data underfetching, when an endpoint does not provide all data required by the application; and versioning, when there is a need to create new API versions whenever new functions are required, in which case documentation for a large number of APIs becomes expensive to maintain [89].

Aiming to overcome these limitations, GraphQL allows APIs to be designed at a high level of abstraction from strongly typed schemas. It provides a single endpoint, through which client applications can obtain all and only the required data by means of a single request. This is a single API version to be maintained by constantly evolving, adding, or discontinuing fields of this version, without disturbing interoperability with available client applications. *Figure 3-14* shows an excerpt of the source code that specifies the API Server Schema and describes the Patient, Data and Register resources.

Figure 3-14 Excerpt of the Source Code of the API Server Schema

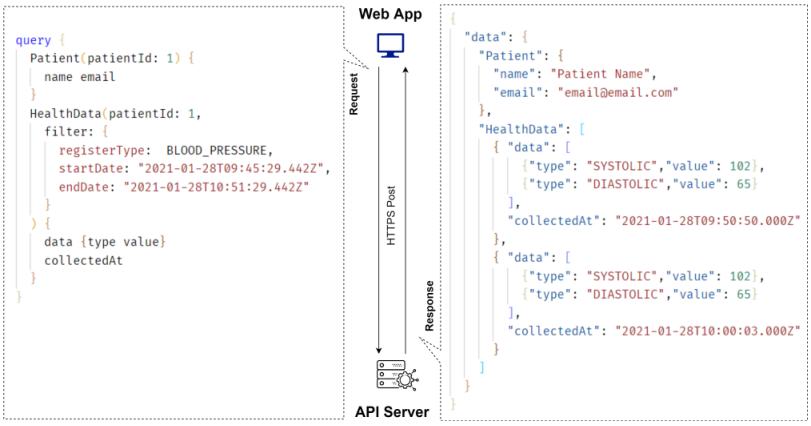
```
type Patient {
  id: Int
  name: String
  birth_date: Date
  gender: EnumGender
  susNumber: String
  phoneNumber: String
  email: String
  cpf: String
  addresses: [Address]!
}

type Data {
  type: DataType
  value: Float
}

type Register {
  type: RegisterType!
  data: [Data]!
  createdAt: Date
  collectedAt: Date
}
```

Figure 3-15 shows an excerpt of the source code of a GraphQL request, performed by Web App to API Server for obtaining the patient data to be displayed on the Data View screen, followed by the API Server response. In this request, the patientId argument is provided with value "1" for retrieving both the Patient resource (name and email), and the HealthData resource (clinical data and collecting date). In HealthData, an additional argument named filter is provided, aiming to obtain data of type BLOOD_PRESSURE in the interval between startDate and endDate. Upon receiving this request, API Server performs the query processing and returns only the requested data in JSON format.

Figure 3-15 Excerpt of the Source Code of a GraphQL Request and its Response



API Server was implemented in JavaScript, and Node.js was used as a server-side execution environment due to its better performance when compared with other similar techniques [90,91]. The Apollo Server open-source library version 2.16 was employed since it allows GraphQL APIs to be implemented that are compatible with any GraphQL client. This API provides an authentication and authorisation mechanism that uses JSON Web Token (JWT), guaranteeing secure data exchange via HTTPS.

3.5 Performance Evaluation

SBIoT-MPH has been designed to be scalable and support a large number of users, which can generate potential communication bottlenecks between Mobile App and API Server and between Web App and API Server. So, an experiment was performed similar to the one reported in [89] to compare REST and GraphQL in order to determine which of these technologies should be used in this system.

The experiment was performed to evaluate the system Response Time and Average Throughput, when these technologies are used in the API responsible for communicating with the Web App and Mobile App applications. Since API Server internal architecture is organised in layers that separate business rules from the API services, it was possible to perform this evaluation by reducing the impact caused by changes in the API implementation. Apache JMeter [92] was used in this experiment, which is a tool to perform functional and load performance tests in Web applications.

The tests were configured in two simulation scenarios that are illustrated in *Figure 3-16*. These scenarios simulate users sending concurrent requests to API Server. Scenario 1 simulates mobile devices that send health data (blood pressure and heart rate with their timestamps) to API Server via their Mobile App, while Scenario 2 simulates the queries by health professionals to API Server via Web App to obtain personal and health data, considering 20 health records per patient. In Scenario 2, to obtain the personal and health data of a patient using REST two requests are necessary, one for the personal data and another for the health data, while using GraphQL only one request is needed. Each simulation was executed for one hour, with the number of users varying from 1 to 500.

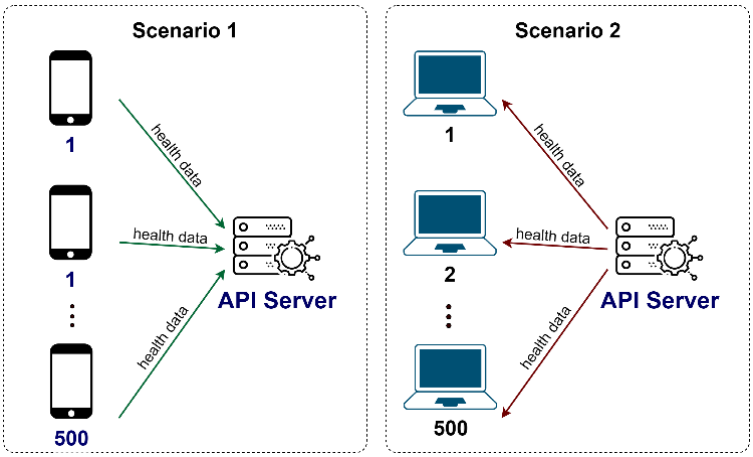


Figure 3-16 Simulation Scenarios

To avoid disturbances that could be caused by network instability, the simulations were carried out in a controlled environment composed of two computers, one acting as a Server and the other as a Client, interconnected by a 100 MBPS IEEE 802.3 Local Network, as illustrated in *Figure 3-17*. The Server machine has an Intel Core I5 3337U 1.8GHz processor, an 8GB RAM memory and a 32GB SSD, and the API Server and the PostgreSQL database were packaged in Docker containers running in the Ubuntu Server 18.04 LTS Operating System. The Client machine has an Intel Core I7 3337U 1.8GHz processor, an 8GB RAM memory and a 250GB SSD, and its Operating System is Windows 10 Single Language.

Figure 3-17 Simulation Environment

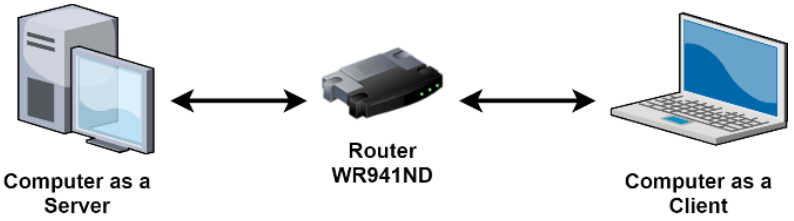
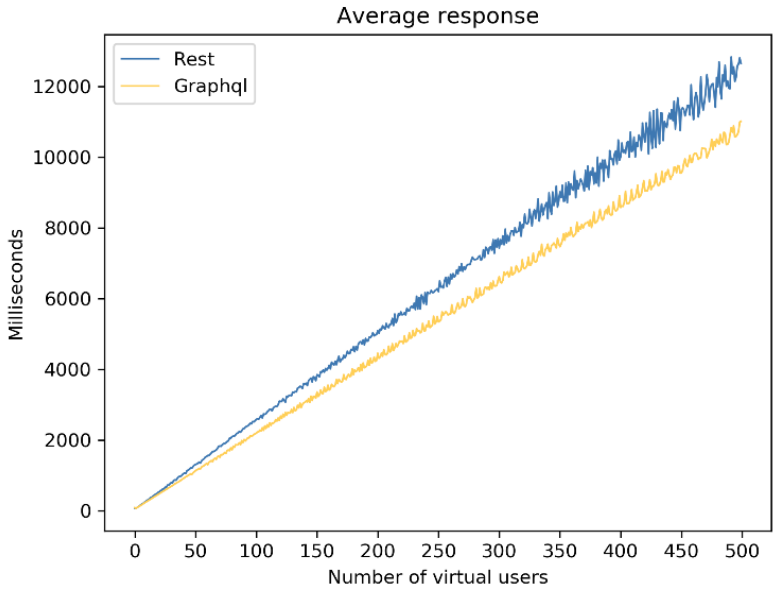


Figure 3-18 and *Figure 3-19* show the results of the simulations of Scenarios 1 and 2, respectively, showing the relationship between the average response time of the API Server and the number of users performing requests through Mobile App and Web App, respectively. These results indicate that in both scenarios the average response time was shorter when using GraphQL than using REST, independently of the number of users.

Figure 3-18 Average Response Time of the API Server for Sending Health Data



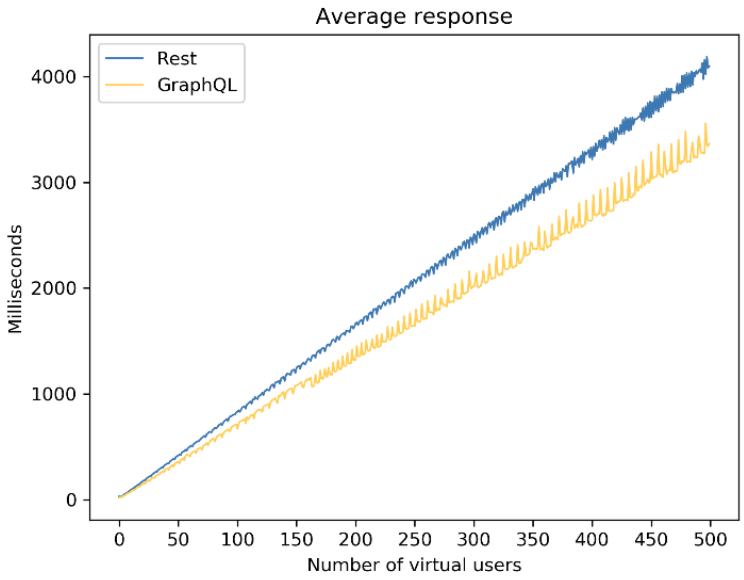


Figure 3-19 Average Response Time of the API Server for Querying Personal and Health Data

Table 3-1 and *Table 3-2* show the simulation results of Scenarios 1 and 2, showing the minimum and maximum response times of API Server, and the average data rates in Kbytes/s received and sent by the API Server.

Table 3-1 Simulation Results for Sending Health Data

End Point	Response Times (ms)		API Server (KB/s)	
	Min	Max	Received	Sent
REST	16	4332	70,33	43,35
GraphQL	13	3901	92,37	52,68

Table 3-2 Simulation Results for Querying Personal and Health Data

End Point	Response Times (ms)		API Server (KB/s)	
	Min	Max	Received	Sent
REST Health Data	28	9863	18,71	192,04
REST Personal Data	6	5278	14,83	30,13
REST Total	34	15141	33,54	222,17
GraphQL	26	11210	33,55	173,48

In both scenarios, the response times were shorter for GraphQL than for REST. In Scenario 1, the average data rates were lower for REST than for GraphQL, since the latter requires a larger header for each request. However, in Scenario 2, the average data rates received by the API Server were similar using GraphQL and using REST, while the average data rate

sent by the API Server was lower when using GraphQL than using REST, since the former allows personal and health data to be sent in a single response for each query.

Although this experiment showed a better performance of GraphQL than REST, limitations of this validation were that we neglected the GraphQL engine performance overhead on both sides (requester and responder), and that all components were in the same Local Area Network (LAN). Furthermore, this validation was done in the context of the adopted SBIoT-MPH architecture.

In [93], a similar comparative study is carried out on the performance of these technologies, but in the context of two MSA systems, with a REST gateway and a GraphQL gateway being implemented. The experiment was also conducted using JMeter, the analysed parameters were also response time and throughput, and the results showed a better performance of REST than GraphQL. However, the authors of [93] state that more effort was required to implement the REST gateway than the GraphQL gateway, which is in line with the results obtained in [89]. Given these potential benefits, we used GraphQL for the SBIoT-MPH internal communication.

3.6 Accuracy Evaluation

To evaluate the accuracy of the SBP and DBP measured by the Sensor Platform (our system under test), we compared these measurements with the ones obtained with a digital blood pressure monitor (our reference device).

We set up the Sensor Platform to measure the pair (SBP/DBP) every minute, and to send to the patient's mobile device the mean values of the last 10 SBP/DBP samples. To measure SBP/DBP with the digital monitor, we adopted the following recommended procedure [94]: measure SBP/DBP twice with an interval of 1-2 minutes between these measures; take the average value of the last two readings; and if these readings differ by more than 10 mmHg, one additional measurement should be performed and considered when calculating the average.

Two patients participated in this evaluation: (1) a 68-year-old male, 1,68 m tall and weighing 90 kg, with a long history of Hypertension controlled by medication; (2) a 37-year-old male, 1,71 m tall and weighing 75 kg, with no history of Hypertension.

As shown in *Figure 3-20*, for Patient 1 we obtained a total of 10 SBP/DBP measurements of the digital monitor and of the Sensor Platform. The mean of SBP and DBP measured by the digital monitor was 103,1

mmHg (range = 96 - 113 mmHg) and 60,5 mm Hg (range = 56 - 74 mmHg), respectively. The mean of SBP and DBP measured by the Sensor Platform was 99,9 mmHg (range = 88 - 108 mmHg) and 60,0 (range = 55 - 74 mmHg), respectively. The mean difference between the set of SBP measurements taken by the Sensor Platform and the set of SBP measurements taken by the digital monitor was 3,2 mmHg with a Standard Deviation (SD) of the first set of 5,2 mmHg. The mean difference between the set of DBP measurements taken by the Sensor Platform and the set of DBP measurements taken by the digital monitor was 0,5 mmHg with a SD of the first set of 5,1 mmHg. These results show that for Patient 1 the Sensor Platform passes on the pass/fail criterion 1 of the ANSI/AAMI/ISO 81060–2:2013 [95], which is the mean difference of test device versus reference device BP measurements ≤ 5 mmHg with SD ≤ 8 mmHg for SBP and DBP.

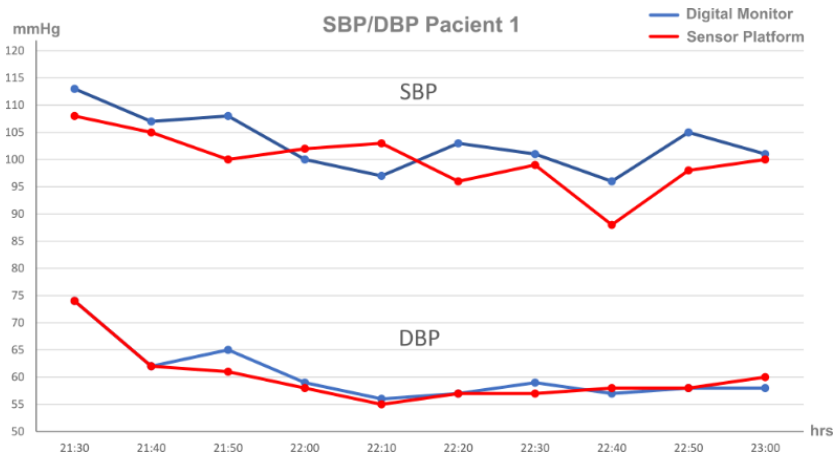
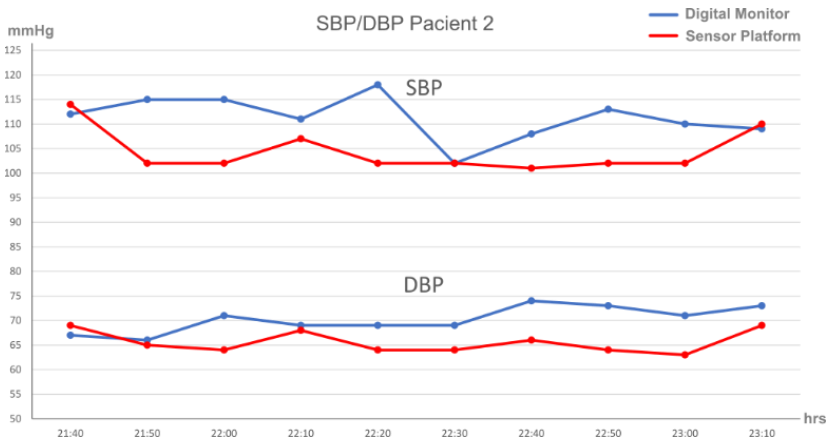


Figure 3-20 (SBP/DBP) Measurements for Patient 1 (taken on April 14, 2023)

As shown in *Figure 3-21*, for Patient 2 we obtained a total of 10 SBP/DBP measurements of the digital monitor and of the Sensor Platform. The mean of SBP and DBP measured by the digital monitor was 111,3 mmHg (range = 102 - 118 mmHg) and 70,2 mm Hg (range = 66 - 74 mmHg), respectively. The mean of SBP and DBP measured by the Sensor Platform was 104,4 mmHg (range = 101 - 114 mmHg) and 65,6 (range = 63 - 69 mm Hg), respectively. The mean difference between the set of SBP measurements taken by the Sensor Platform and the set of SBP measurements taken by the digital monitor was 6,9 mmHg with a SD of the first set of 4,2 mmHg. The mean difference between the set of DBP measurements taken by the Sensor Platform and the set of DBP measurements taken by the digital monitor was 4,6 mmHg with a SD of

the first set of 2,2 mmHg. These results show that for Patient 2 although the Sensor Platform passes criterion 1 for DBP, it fails for SBP.

Figure 3-21 (SBP/DBP) Measurements for Patient 2 (taken on April 17, 2023)



In *Figure 3-20* and *Figure 3-21* each value plotted referring to the Sensor Platform is an average of 10 values collected automatically and continuously by this platform over 10 minutes, while each value plotted referring to the digital monitor is an average of 2 values collected manually using this monitor over the 10 minutes. In addition, each measurement with the digital monitor requires user intervention, which can introduce noise in the readings, while the measurements with the Sensor Platform are taken continuously and automatically, possibly with less noise.

Chapter 4

Ontology-Driven IoT System for Monitoring Hypertension

This chapter presents the *Ontology-Driven IoT System for Monitoring Hypertension (ODIoT-SMH)*, an RPM IoT system developed in the second Design Cycle of this research, which is also structured according to the IoT three-layer architecture, reuses most of the SBIoT-MPH components and, in addition, it deals with semantic interoperability. For this purpose, a semantic model was conceived to be fully deployed in the ODIoT-SMH Cloud Layer, which was implemented through a semantic module introduced in this layer, communicating with the ODIoT-SMH applications of that layer, a triplestore and ontologies that were also introduced in the Cloud Layer. Regarding the development of the ODIoT-SMH ontology, we apply some IoT-specific best practices to reuse and extend existing ontologies for this system.

This chapter is organized as follows: Section 4.1 introduces the ODIoT-SMH semantic model; Section 4.2 describes the ODIoT-SMH semantic module and its main components; Section 4.3 deals with ontology engineering and the ODIoT-SMH ontology; Section 4.4 presents the scalability analysis of ODIoT-SMH; and Section 4.5 presents a comparative performance analysis between ODIoT-SMH and SBIoT-MPH.

4.1 ODIoT-SMH Semantic Model

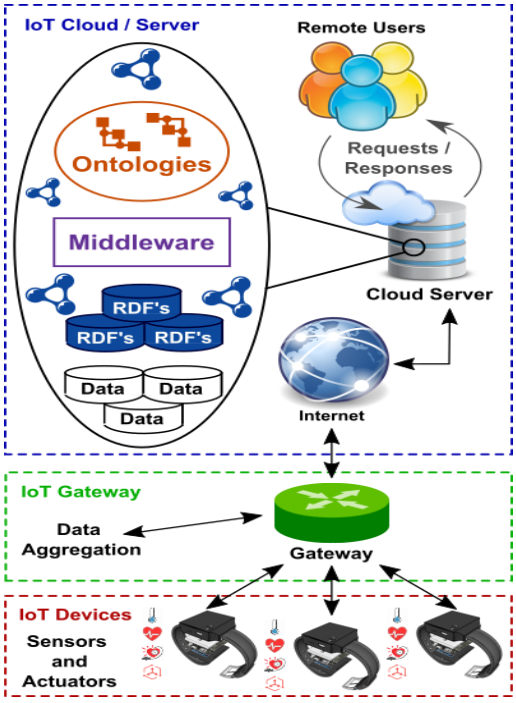
SBIoT-MPH was the starting point for the design of ODIoT-SMH. The Sensor Layer is the same for both systems, while the Fog Layer is similar, the main difference being the functionality that analyses and classifies the clinical data of blood pressure according to health risk levels for the

eventual generation of alert signals, which in the SBIoT-MPH belongs to the Mobile App, while in the ODIoT-SMH it was transferred to the Cloud Layer.

The Cloud Layer of ODIoT-SMH contains the same API Server and Web App applications of SBIoT-MPH, and the main differences between these two systems are the new components introduced in the Cloud Layer of ODIoT-SMH and the ontology developed for this system, to deal with semantic interoperability.

In an IoT-based platform, components may not be able to properly exchange and understand the raw data generated by IoT devices (*e.g.*, sensors, actuators) from different manufacturers due to the lack of common semantics. To solve this problem, the first step can be to send the data to a gateway for pre-processing and aggregation, aiming to increase their quality via an algorithm such as the one presented in [96]. These aggregated data can be then stored in the cloud so that semantic operations can be performed on them. In this regard, *Figure 4-1* depicts an IoT semantic model that we developed based on a model that has been proposed in [97].

Figure 4-1 IoT Semantic Model (Adapted from [47])



In this model, all processing to achieve semantic interoperability is performed in the cloud, which must comprise ontologies to provide

semantic annotations to the aggregated data, adapters to convert aggregated data into semantic data, a conventional database, and a triplestore [33]. The aggregated data are converted to the RDF triples format with the help of ontologies. Some work has already been performed in this direction [98], but further investigation is still necessary.

This IoT semantic model can improve ODIoT-SMH by addressing the following requirements of each layer:
(a) *Sensor Layer*, where the sensors have different data formats based on different data types with different semantics, and it is necessary to predefine data according to BluetoothGatt API;
(b) *Fog Layer*, where the interpretation of the data received from the Sensor Layer, with respect to the hypertension risk levels, is restricted to the Patient Data Analysis functionality; and
(c) *Cloud Layer*, where data exchange is not meaningful, and it is not possible to personalise the Patient Data Analysis functionality.

Therefore, we adapted the IoT semantic model shown in *Figure 5-1* to deal with these problems. We worked mainly on the Cloud Layer, in order to add semantic annotations to the aggregated data received from the Fog Layer, as illustrated in *Figure 4-2*.

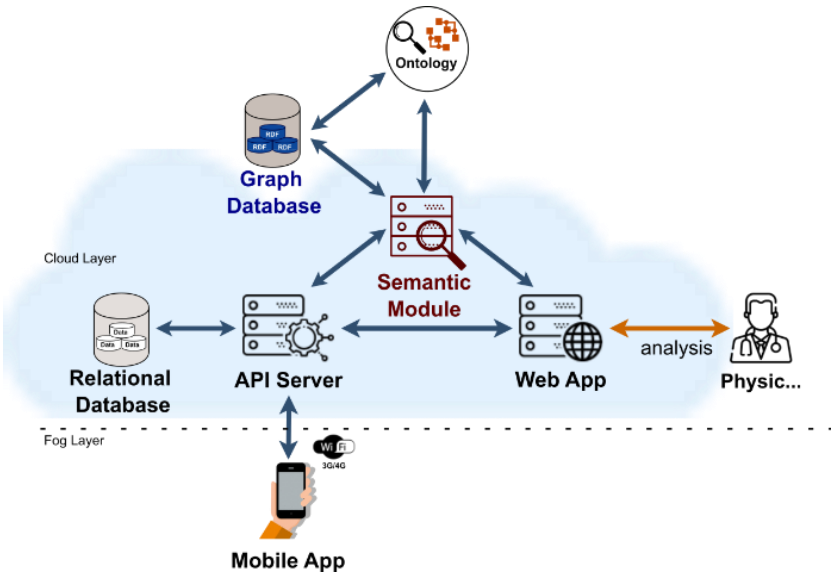


Figure 4-2 ODIoT-SMH Semantic Model

To add semantic annotations to the aggregated data it is necessary to extract concepts and properties from ontologies related to health domain, *e.g.*, SNOMED CT ontology [74], and from ontologies related to the IoT domain, *e.g.*, SAREF4EHAW [67].

When designing the ODIoT-SMH Semantic Model, we sought to keep as much as possible the original structure of SBIoT-MPH. To this end, a semantic module[1] was introduced in the Cloud Layer, playing the role of a middleware, and communicating with the ODIoT-SMH applications, the triplestore and the ontologies, all of them also in the Cloud Layer as shown in *Figure 4-2*.

4.2 ODIoT-SMH Semantic Module

ODIoT-SMH Semantic Module was designed following Service Oriented Architecture (SOA) principles, and it has seven main components as shown in *Figure 4-3*:

(a) *API Listener* transforms the aggregated data into semantic data for being consumed by the others components;

(b) *Inference Engine* enables the automatic classification of patients' data according to pre-defined rules;

(c) *Alert Handler* notifies the user based on context and results from the Inference Engine;

(d) *Web Manager* and *Web Mapper* handle all requests and data coming from the Web App;

(e) *Data Handler* manages the triplestore; and

(f) *MQTT-Broker* is a broker used by clients to exchange data.

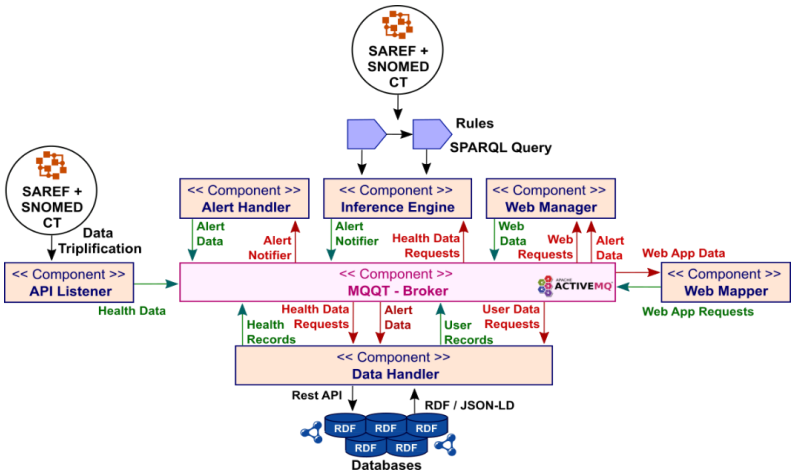


Figure 4-3 ODIoT-SMH Semantic Module Architecture

[1] We consider a module as an architectural component of the system, including the semantic module, which is a different concept than 'semantic model', which refers to an operational ontology, *e.g.*, formalized in RDF/OWL.

The MQTT messages are filtered by the MQTT-Broker topics and forwarded by the broker to interested clients. The broker acts as a server and intermediates the message exchange between clients, by receiving the published data and forwarding them to the clients signed up to receive them. A client is any device connected to the broker that can publish/subscribe topics.

4.2.1 API Listener

This component is responsible for capturing aggregated data from API Server, transforming them into RDF format and publishing the result of this transformation. *Figure 4-4* shows the main activities of API Listener.

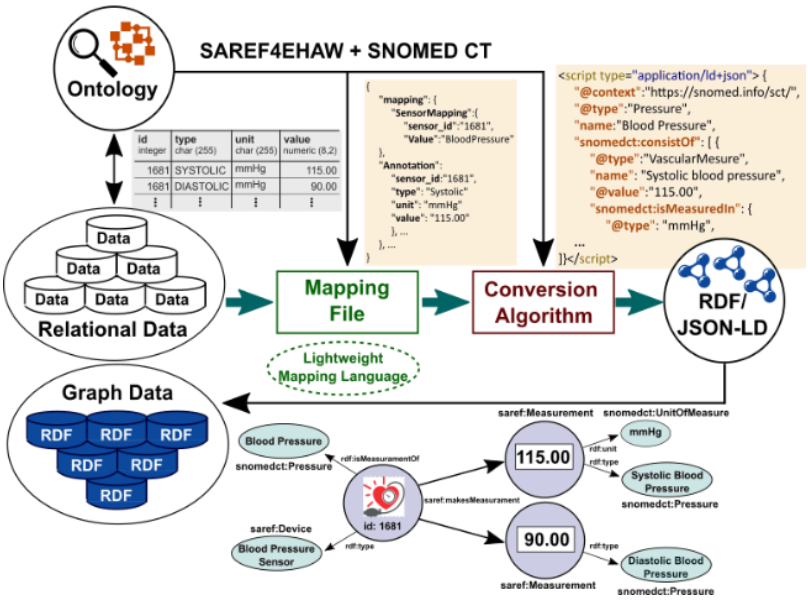


Figure 4-4 API Listener Main Activities

Initially, a GraphQL API has been employed to send to the API Listener all data in JSON format received by the API Server from the Mobile App. API Listener uses the GraphQL subscription operation, which acts like a publisher/subscriber protocol. Whenever API Server receives data from Mobile App, API Listener is notified and receives a copy of those data.

The next step has been to semantically enrich the aggregated data with RDF annotations, with additional knowledge expressed in ontologies. The ODIoT-SMH Semantic Module uses the SAREF4EHAW ontology [67] to represent devices, sensors, data, and actors. To incorporate concepts and properties related to Hypertension, the *Chronic disease* class of

SAREF4EHAW was extended with a subset of the NCDs-related SNOMED CT ontology [74].

A conversion algorithm transforms the aggregated data into a set of RDF triplets with the help of the ontology and a mapping file that employs a lightweight JSON-based mapping language. The relationship between the schema and the ontology is stored in this mapping file, which describes how to extract any information from the ontology [97]. An MQTT client function connected to the broker publishes the semantically structured data produced by the conversion algorithm as a JSON-RD format to the *FogHeathData* topic.

4.2.2 Inference Engine

This component is responsible for real-time processing of the patient data published by API Listener, and for detecting possible deviations from normal values based on a pre-defined classification of health data, such as the risk levels of *Patient Data Analysis*. Furthermore, it is responsible for triggering the Alert Handler component by publishing in the broker topics related to each detected deviation type.

Health data are classified based on acquired knowledge, being used in real health data to assess and predict situations. The Inference Engine deals with two types of knowledge: objective and subjective. Objective knowledge is present in general medical standards, well known by health professionals, easily found in the literature, and widely disseminated by large health organizations such as the WHO. Subjective knowledge is related to the patient's profile and context, such as medical history, genetic diseases, and personal lifestyle. Objective knowledge can be described in the ontology and, therefore, accessed by the Inference Engine. Subjective knowledge can be manually described by the user, usually the health professional responsible for the patient, and stored in the subjective rules database.

When a *FogHealthData* topic is received, the Inference Engine identifies the data type and the patient associated with these data, and the following steps are applied involving subjective rules:
(a) The Inference Engine verifies if there is any rule in the subjective rules database for this patient regarding this data type;
(b) For each rule found, it checks whether the data fit the rule; and
(c) If a deviation from normal values is detected, a specific topic describing this deviation is published, which triggers an alert.

Afterwards, the same steps are applied involving objective rules. The only difference is in the first step, where the Inference Engine verifies in the ontology for these rules regarding this data type.

Objective rules can be skipped if a subjective rule is found that explicitly states that. Some health data have variations by default, which invalidates standard analysis. This may occur due to the patient's profile and context, which must be informed by the health professional when storing a rule in the subjective rule database. *Figure 4-5* illustrates these steps.

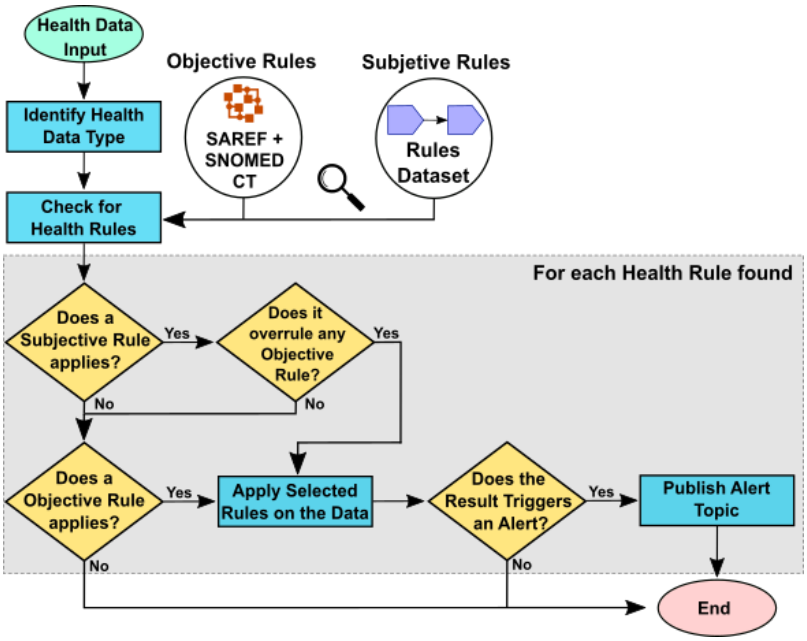


Figure 4-5 Steps of the Inference Engine

Rules are described as Semantic Web Rule Language (SWRL) [99] expressions of the form `IF health-data-preconditions THEN variants-effects`. SWRL expressions can be evaluated by description logic reasoners, such as Pellet, for the evaluation of health data, inferring at runtime new knowledge based on the ontology and rules database. For example, the SWRL rule below defines the Patient Data Analysis normal risk level:

```
Blood_Pressure(?systolic_mmhg ?diastolic_mmhg),
Patient(Pedro),hasMeasurement(Pedro,?systolic_mmhg,
lessThen(?systolic_mmhg, 120) ^
hasMeasurement(Pedro,?diastolic_mmhg),
lessThen(?diastolic_mmhg,80)->
hasHypertensionStage (Pedro, normal).
```

Each classified deviation from normal heath data values triggers a topic-related alert message, which is published and consumed by Alert Handler.

4.2.3 Alert Handler

This component is responsible for processing real-time alerts related to topics published by the Inference Engine, and for contacting the patient and their health professional. The main activities of the Alert Handler are shown in *Figure 4-6*.

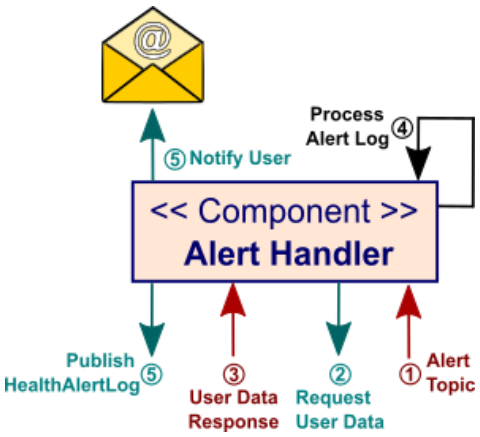


Figure 4-6 Alert Handler Main Activities

Three alerts related to the topics blood pressure, body temperature, and heart rate were respectively implemented in ODIoT-SMH. Alerts of these topics provide the following information: the user identifiers (patient and their health professional); the sort of data and their values, and the timestamp of the data and of their sort; and a set of actions to be executed. The actions include sending an email to notify the patient, their health professional, or both.

The user identifier is used so that a request to the Data Handler can be published for retrieving all required information to execute indicated actions (*e.g.*, patient email address). Data are used to create a historic log of all patient alerts, which can be useful for improving pervasive healthcare by helping health professionals to define more accurate subjective rules for the patient. Historic data can be also explored by data mining and knowledge discovery algorithms. An alert log is published as HealthAlertLog, and stored in the triplestore by the Data Handler.

4.2.4 Web Manager and Web Mapper

These components interact with Web App by capturing and redirecting all requests and responses to Data Handler. Furthermore, they provide additional information not available in the relational database to Web App (*e.g.*, alert log). The main activities of both components are presented in *Figure 4-7*.

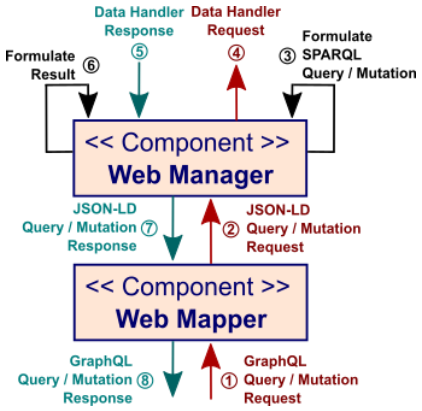


Figure 4-7 Web Manager and Web Mapper Main Activities

Web Mapper uses an interface similar to the API Server for receiving a Web App request described in the Apollo Client definition language for GraphQL. Web Mapper converts the received request to a JSON-LD request and publishes it under the HTMLRequests topic. After receiving the converted request, Web Manager formulates the corresponding SPARQL queries [100], and publishes them for being processed by the Data Handler, which in turn returns a response with the processing result. After receiving the response from the Data Handler, the Web Manager structures the received data for being published in the HTMLData topic. After receiving a response with these data, Web Mapper converts it into a GraphQL response for sending to the Web App.

4.2.5 Data Handler

This component processes any direct request to the triplestore that is made by other components. It is an endpoint channel that isolates the persisted data, and simplifies mutations and queries management, in accordance with the SOA architectural principles.

SPARQL was adopted for requests and responses because it is a structural query language designed for querying RDF data. Since SPARQL queries can be represented as query graphs, they can be answered by performing graph pattern matching over RDF graphs.

4.3 ODIoT-SMH Ontology Engineering

Semantic-driven IoT solutions rely on IoT ontologies to represent the objects and properties inside the IoT application domain. Over the past decade, numerous IoT ontologies were developed, most of them described in OWL [62], to improve the semantic interoperability of IoT artifacts,

aiming to find a common understanding of capabilities among platforms, devices, gateways, applications and networks involved in IoT solutions [19].

A set of guidelines for IoT ontology engineering covering best practices and methodologies from the Semantic Web community was proposed by the European Research Cluster on the Internet of Things group, enhanced by the FIESTA-IoT project experience reported in [101]. *Table 4-1* summarises the best practices we followed while developing ODIoT-SMH ontology based on the guidelines for IoT ontology engineering. This table organises the best practices into categories, each containing questions related to the same topic. Semantic technologies correspond to tools and methodologies recommended in [101] to address each question.

Table 4-1 Best Practices and Recommendations for IoT Ontology Engineering

Category	Questions	Semantic Technologies Recommended
Ontology & Dataset Publication (formats, serialisation)	- Did you validate your ontology's syntax?	Interlink datasets through unified RDF serialisation (*e.g.*, RDF/XML, JSON-LD, Turtle).
	- Does your ontology follow a unified terminology to describe sensor data?	Unique terminology for IoT data (*e.g.*, M3), linking synonyms terms with the same meaning (*e.g.*, rain and precipitation).
	- Did you correctly use the SSN ontology?	Always apply syntax checks of semantic documents (*e.g.*, RDF, OWL) to facilitate interoperability, using RDF and OWL Validators (*e.g.*, TripleChecker), which detects properties used but not declared or a wrong format date.
Ontology Quality	- Did you provide documentation attached to your ontology?	Easiness to lookup terms is a feature making the ontology dereferenceable. This is essential to automate tasks when analysing ontologies.
	- Is your ontology dereferenceable?	Many ontologies have open issues and common pitfalls, which may be addressed by creating new elements that fix the problem (*e.g.*, merging different concepts in the same class).
	- Did you follow tutorials and methodologies to design ontologies?	Consider the importance of ontology reusability. Thus, ontology engineering methodologies and tutorials should be adopted.

Ontology & Dataset Reuse	- Did you try to reuse the existing ontologies or datasets catalogues or from semantic search engines?	Share your ontology on Web ontology catalogues or semantic search engines to improve reuse and refinement. Linked Open Data provides tools to share and reuse datasets.
	- Are your ontology and dataset referenced on such tools?	Enrich ontology documents by adding precise element definitions and describing them in the ontology, to allow automatic generation of documentation websites from the ontology.
	- Is your ontology shared online?	When the ontology is reliable, we should directly reuse the concepts or properties for existing ontologies. Otherwise, redefining the same concept or property can be done and add *owl:equivalentClass*, *owl:equivalentProperty*.
		As SSN is the de facto IoT ontology, it is necessary to align the ontology with SSN through extensions, which enables the validation with the W3C SSN validator.

The main purpose and information requirements of ODIoT-SMH ontology regarding the NCDs and the IoT domain are described as Competency Questions (CQs) as shown in *Table 4-2*. While SNOMED-CT provides an accurate and deeper ontological analysis of Health domain terminology including NCDs, SAREF4HAW can be used as an IoT reference model for electronic(e)/mobile(m)-Health applications. Therefore, a set of CQs was defined according to the main elements of interest of these ontologies. The scope of these questions guided the selection and modularisation of the existing ontologies to be aligned, merged, extended and reused. Each step followed best practices based on the guidelines for IoT ontology engineering in *Table 4-1*.

Table 4-2 Competence Questions to be Responded by ODIoT-SMH Ontology

Id	Semantic Technologies Recommended
CQ01	What are the general elements that describe all users in ODIoT-SMH?
CQ02	What are the NCDs, and how are they categorised?
CQ03	What are the elements participating in a blood pressure recording session?
CQ04	What is blood pressure, and what are the types of blood pressure

CQ05	What type of properties can be measured from blood pressure, and what type of measurement a blood pressure sensor can measure?
CQ06	What is systolic/diastolic pressure, and how is it related to blood pressure?
CQ07	What is hypertension and how is it classified?
CQ08	What is a time series of measurements?
CQ09	How to integrate measurements from multiple sensors (*e.g.*, blood pressure, body temperature, heartbeat, accelerometer and battery monitor) of a sensor platform/BAN?

4.3.1 Ontology Modularity, Alignment, Merge and Extension

Modularity is essential for multidisciplinary knowledge environments like ODIoT-SMH. Modularity is also recommended in formal and applied ontology to reduce its design complexity and to facilitate ontology verification, reasoning, maintenance, extensibility and integration [102]. As a consequence, the ODIoT-SMH ontology was developed as the composition of three different modules (sub-ontologies): an e/m-Health application ontology, a user profile ontology, and a medicine ontology.

The e/m-Health application ontology specifies the core concepts in the e/m-Health smart application domain, such as health agents and health device models. The device models define sensors, actuators and their observed properties, such as the unit of measure and the time of each observation. Devices can also be combined and described as a Body Area Network (BAN).

The user profile ontology represents the user's attributes, extending the health agent's concepts, like personal account, personal information and contact address.

The medicine ontology states the different medical and clinical terminologies. For the proposed system, the medicine ontology was restrained to the concepts related to NCDs. The main reason was that most medicine ontologies found in the literature were too extensive, covering all medical subdomains, or too specific, lacking properties related to NCDs [19].

The e/m-Health application ontology was aligned, merged and extended with a user profile ontology and a medicine ontology. The first step is to align the ontologies by matching overlapping and related concepts from each ontology. The alignment is the output of this matching process.

Ontology alignment techniques can be classified into two categories: element-level and structure-level [103]. The first considers ontology entities or their instances in isolation from their relations with other entities or instances. The second considers the ontology entities or their instances to compare their relations with other entities and instances, focusing on the characteristics of the relationship between entities.

Among the element-level techniques, we applied the Levenshtein distance method, a string-based approach that matches two or more entities with similar names [104]. This method calculates the minimum number of insertions, deletions, and substitutions of characters required to transform one entity name into the other entity name. The smaller the distance is, the more related the entities are. A script in Python using the Owlready2 library [105] was developed to calculate Levenshtein distance and to match entities from different ontologies based on a minimum threshold value.

However, this method cannot provide a complete association between all equivalent entities from all ontologies, as sometimes an entity may have different names from one ontology to another. Therefore, we manually checked the matchings to validate them. The matching is followed by a merging operation based on axioms expressing the equivalence or the membership. For example, the entity User from the e/m-Health application ontology is considered as equivalent to the entity Person from the user profile ontology.

OWL provides helpful properties to support 1:1 mappings of different semantic models for ontology alignments. For example, for the instance level, the main predicate is *owl:sameAs*, which allows linking one instance to another instance that represents the same object. The *owl:equivalentClass* and *owl:equivalentProperty* are the properties used to map equivalent classes and properties on the terminology level.

The ontology must be extended after being aligned and merged to achieve specific requirements. For example, to express that a sensor observes a result and a particular patient wears this sensor, two objective properties should be defined to describe the relations between an observation and a sensor and between a sensor and a patient. The merged ontology must also be able to answer all CQs, therefore reinforcing its capability to cover ODIoT-SMH semantic requirements.

4.3.2 ODIoT-SMH Ontology

This section presents the three reused ontologies, their main concepts and properties, and the extensions (personalized data properties, objective properties and inference rules) performed to build the ODIoT-SMH

ontology and to answer the CQs. To refer to these reused ontologies, classes from these ontologies are indicated with the corresponding ontology prefix. If the class name is not prefixed, we refer to a class created specifically for the ODIoT-SMH ontology.

FOAF: User Profile Ontology

The Friend of a Friend (FOAF) ontology includes all possible aspects of a user's general profile. The *foaf:Person* class represents one user's profile and represents its different data properties, such as name, gender, age, account, contact number and contact address. It is also a subclass of the *foaf:Agent* class, which represents anything that does something in FOAF. Other types of agents include *foaf:Organization* and *foaf:Group*.

In addition, medical concepts and properties needed to be defined as a physical dimension status to fit the features of ODIoT-SMH environment defined in the SNOMED-CT medicine ontology. The *sct:PhysicalDimensionStatus* class defines additional properties to be assigned to the *foaf:Person* class, such as weight, height, and blood type, as subclasses. Another medical concept added to the *foaf:Person* class is represented with the *MedicalAlertHistory* class. This class represents a record of NCD alerts, including the disease type, alert type, and a timestamp that identifies the chronological occurrence of a medical alert.

The FOAF ontology was reused mainly to represent all concepts and properties related to the general profile of ODIoT-SMH environment users. We also extend it to incorporate new concepts related to the ODIoT-SMH Health domain, which includes physical dimensions status and medical history alert. As a result, the extended set of general profiles meets the requirements of CQ01.

SAREF4HAW: e/m-Health Application Ontology

Among the numerous IoT ontologies proposed in the literature to represent sensor data observations, the W3C Semantic Sensor Network (SSN/SOSA) and the ETSI Smart Appliance Reference Ontology (SAREF) are the most prominent approaches. Both ontologies were rigorously developed by ontology and domain experts over many years, being applied in several IoT use cases and supported by standardization initiatives [102].

Although SSN/SOSA and SAREF are aligned, covering similar concepts from the IoT domain, SAREF is more focused on applying a vocabulary oriented to the IoT industry solutions. The objective was to create a semantic layer connecting heterogeneous platforms, supporting the data exchange with different protocols from different manufacturers [102].

The SAREF ontology provides building blocks that enable the reutilization of other parts of the ontology according to specific requirements. Furthermore, SAREF has standardized extensions for particular IoT domain applications that follow the ETSI standardization procedure, which are guidelines to avoid semantic interoperability issues. SAREF for e-Health/Ageing-well (SAREF4EHAW) is an extension of SAREF developed and standardized by ETSI for the e-Health/Ageing-well domain [106].

The *s4ehaw:HealthActor* class represents e/m-Health system actors, such as caregivers, patients, users, and helpers. *s4ehaw:Patient* is an subclass of *s4ehaw:HealthActor* (*rdfs:subClassOf* relation) and may have one or more instances of the *NonComunicableDisease* class such as *Hypertension*. A health actor also uses a BAN (*s4ehaw:usesBan* relation) for health monitoring purposes.

Regarding a measurement observed by a *s4ehaw:HealthSensor*, SAREF4EHAW represents it through the *saref:makesMeasurement* object property of a *s4ehaw:HealthDevice* to *saref:Measurement(s)*, representing the relation between a device and the measurements performed by this device. A *saref:Measurement* element represents a measurement value (*saref:hasValue*), the unit of measure of this value (*saref:UnitOfMeasure*) and the property under observation (*saref:Property*). For example, the *BloodPressureMeasurement* class is a sub-class of *saref:Measurement* representing all the observations of the pressure of the feature *Blood* of interest, which is a subclass of *saref:Property*, and has 110/78 (*saref:hasValue*) millimetres of mercury (*saref:UnitOfMeasure*). *BloodPressureMeasurement* has two values because according to the medicine ontology blood pressure has two observations, namely *SystolicBloodPressureMeasurement* and *DyastolicBloodPressureMeasurement*, which both are sub-classes of *BloodPressureMeasurement*.

The features of interest in ODIoT-SMH ontology are *Blood*, *Oxygen*, *Temperature*, *Glucose*, and *Heart*. These are the features that can be observed to detect possible NCDs. Each feature of interest is measurable using a measures property represented by the *saref:measuresProperty* class. For instance, the *Pressure* class is a sub-class of *saref:measuresProperty* of the *Blood* class. In the ODIoT-SMH ontology, the following set of measure properties is defined: *Pressure*, *Level*, *Rate*, and *saref:hasValue*. Therefore, *s4ehaw:HealthSensor* is specified in the following sub-classes decomposed per measures properties: *BloodPressureSensor*, *GlucoseSensor*, *HeartRateSensor*, *OxygenSensor* and *TemperatureSensor*. Each *s4ehaw:HealthSensor* value (*saref:hasValue*) is associated with a *Range* expressing that represents the

status of the corresponding observation: *Normal*, *Abnormal*, *Disorder* or *Wrong*.

The core concepts in ODIoT-SMH ontology come from SAREF4HAW. Therefore, SAREF4HAW is the most reused ontology and has the majority of concepts and properties that meet CQs requirements. As a result, ODIoT-SMH ontology was developed by reusing SAREF4HAW and extended to cover the remaining CQs. *Table 4-3* summarises the main class and property definitions from SAREF4HAW that are used in this project, along with their super classes, restrictions, extensions and the CQs they answer.

Table 4-3 Main Elements of SAREF4HAW Used in this Project and Related CQs

Element	Semantic description	Related CQs
s4ehaw:HealthActor	*rdfs:subClassOf s4ehaw:User* *rdfs:sameAs foaf:Person* Restrictions: *s4ehaw:usesBan some s4ehaw:Ban* *s4ehaw:hasActivity some s4ehaw:Activity* *hasNonCommunicableDisease some NonCommunicableDisease*	CQ01
s4ehaw:Ban	Restrictions: *s4ehaw:hasBanCommunicationType min 1 (s4ehaw:BanCommunicationType)* *s4ehaw:hasApplicationDomain min 1 (s4ehaw:ApplicationDomain)* *s4ehaw:contains min 1 s4ehaw:HealthDevice* *s4ehaw:hasContact min 1 s4ehaw:Contact*	CQ09
s4ehaw:HealthDevice *(rdfs:subClassOf)* *s4ehaw:HealthSensor* *s4ehaw:HealthActuator*	Restrictions: *saref:controlsProperty only saref:Property* *saref:hasFunction min 1 saref:Function* *s4ehaw:isAttachedTo some s4ehaw:s4ehaw:HealthActor* *s4ehaw:hasCurrentMode only s4ehaw:Mode* *s4ehaw:hasDeviceCharacteristics max 1 s4ehaw:DeviceCharacteristics* *s4ehaw:hasInterface min1 s4ehaw:Interface* *saref:offersOnly saref:Service* *saref:consistsOf only saref:Device* *saref:measuresProperty only saref:Property* *saref:makesMeasurement only saref:Measurement* *saref:accomplishes min 1 saref:Task*	CQ03 CQ04 CQ05 CQ06 CQ08

SNOMED-CT: Medicine Ontology

Systematized Nomenclature of Medicine - Clinical Terms (SNOMED-CT) is an international standardised clinical terminology used in the Healthcare and Medical domain that provides medical terms, concepts, and relationships designed to support the sharing and analysis of health-related information. This ontology undergoes regular updates and maintenance managed by the International Health Terminology Standards Organisation[2] (IHTSO), which is an international non-profit organisation. IHTSO also provides a standardisation procedure to ensure consistency and semantic interoperability issues. As a result, SNOMED-CT has been one of the most used and extended standard terminologies in health information systems in the past years [107].

All the possible human actors in a healthcare or clinical system are defined under the *sct:SocialContext* class (*rdfs:subClassOf* relation). This class is divided into 10 sub-contexts: Community, Family, Group, Institution, Life style, Occupation, Person, Population, Religion/Philosophy, and Social Status. Healthcare professionals are represented as sub-classes of *sct:HealthcareProfessionals*, for example, *sct:Cardiologist* (*rdfs:subClassOf sct:Physician*) and *sct:Neurosurgeon* (*rdfs:subClassOf sct:Surgeon*). Non-healthcare professionals are represented as sub-classes of *sct:Person*, such as *sct:Patient* and *sct:Caregiver*. This classification contains more classes than necessary for the ODIoT-SMH ontology requirements, overfitting in this way CQ01. Therefore, we used it more as a guideline when extending the *s4ehaw:HealthActor* class. Most SNOMED-CT reused concepts came from the clinical observations subdomain (*sct:ClinicalFinding* class and its sub-classes).

The *sct:CardiovascularFindings* class represents all clinical observations of the human body's cardiovascular system. *sct:CardiovascularMeasurement* is a sub-class of *sct:CardiovascularFindings* and has defined sub-classes to each blood measurement, such as *sct:SystolicArterialPressure* and *sct:DiastolicArterialPressure*. In general, each phenomenon in *sct:CardiovascularFindings* can be interpreted as *Normal, Abnormal*, *Increasing*, *Decreasing* and *Disorder*, depending on its behaviour through a time-lapse. *sct:HypertensiveDisorder* is characterised by *sct:SystolicHypertension* and *sct:DiastolicHypertension* (*sct:dueTo* property), but not necessarily both. For example, if systolic blood pressure is elevated (*Disorder* behaviour), but diastolic blood pressure stays within an acceptable range (*Normal* behaviour), the condition is *sct:SystolicEssentialHypertension*.

---

[2] International Health Terminology Standards Organisation: https://www.snomed.org/

Most NCDs are represented as sub-classes of the *sct:Disease* class due to the diversity of their clinical characteristics. *Hypertension*, however, is not a really cardiovascular disease but rather a condition, so it is under *sct:ElevatedBloodPressure*, which is a sub-class of *sct:CardiovascularFindings*. Therefore, the *NoncommunicableDiseases* class was defined to model all NCDs-related classes. Each NCD can diagnosed through the *sct:dueTo* object property of a *sct:ClinicalFinding*, and their values can be interpreted through the *sct:hasInterpretation* object property. For example, the class *HypertensionType1* is a sub-class of *sct:Hypertension* representing all the cardiovascular findings that are observed as *sct:ElevatedBloodPressure* (*sct:dueTo)* in which both *sct:SystolicArterialPressure* and *sct:DiastolicArterialPressure* are interpreted (*sct:hasInterpretation*) as *Increasing*. In contrast, *sct:HypertensionCrysis* is a sub-class of *sct:Hypertension*, representing all the cardiovascular findings that are observed as *sct:HypertensiveDisorder* (*sct:dueTo)* in which at least one *sct:CardiovascularMeasurament* is interpreted as *Disorder* behaviour (*sct:DiastolicHypertension*, for example).

Since the SNOMED-CT ontology has regular updates, maintenance, and standardisation procedures in compliance with the best practices and methodologies for ontology engineering, identifying the main concepts and relations that meet the requirements and CQs of the ODIoT-SMH ontology was straightforward. However, the merging process was laborious since only a relatively small part of the ontology was reused. *Table 4-4* summarises the main class and property definitions from SNOMED-CT that were used in this project, along with their super-classes, restrictions, extensions and the CQs they answer.

Table 4-4 Main Elements of SNOMED-CT Used in this Project and Related CQs

Element	Semantic description	Related CQs
sct:CardiovascularMeasurament	*rdfs:subClassOf* *sct:CardiovascularFindings* Restrictions: *sct:hasInterpretation min 1*	CQ06
sct:Asthma	*rdfs:subClassOf* *sct:RespiratoryFinding*	CQ02
sct:ElevatedBloodPressure *sct:SystolicArterialPressure* *sct:DiastolicArterialPressure*	*rdfs:subClassOf* *sct:CardiovascularMeasurament* Restrictions: *sct:hasInterpretation min 1*	CQ06 CQ07

sct:Hypertension	*rdfs:subClassOf* *sct:CardiovascularMeasurament* Restrictions: *sct:dueTo max 1* *sct:SystolicArterialPressure* *sct:dueTo max 1* *sct:DiastolicArterialPressure* *sct:hasInterpretation min 1*	CQ02 CQ07
sct:HypertensiveDisorder	*rdfs:subClassOf* *sct:CardiovascularMeasurament* Restrictions: *sct:dueTo max 1* *sct:SystolicHypertension* *sct:dueTo max 1* *sct:DiastolicHypertension*	CQ02 CQ07
sct:Disease	*sct:CardiovascularDisease* *sct:ChronicObstructivePulmonaryDisease* *sct:Diabetes* *sct:Cancer*	CQ02

4.3.3 Dynamic Analysis and Protégé Implementation

One of the main contributions of our work is the dynamic analysis of the sensors' measurements of a particular patient using objective and subjective knowledge. Objective knowledge is present in general medical standards, is well known by health professionals, can be easily found in the literature, and is widely disseminated by large health organisations such as the WHO. Subjective knowledge is related to the patient's profile and context, such as medical history, genetic diseases, and personal lifestyle. Subjective knowledge is defined and updated by a system user, usually the health professional responsible for the patient, and stored in the subjective rules database. That allows a personalised inference analysis of the sensor's observations, assisting in data diagnosis so that, if a Disorder is detected, an alert protocol is triggered [32].

The ODIoT-SMH reasoner is implemented based on the Pellet[3] reasoner, which is an open-source Java reasoner that uses Jena and an OWL-API interface to perform ontology reasoning. It supports expressive description logic, checks the consistency of ontologies, computes the

[3] Pellet Github: https://github.com/stardog-union/pellet

classification hierarchy, explains inferences, and answers SPARQL queries. Pellet allows reasoning on a data set using a proposed set of rules specified in SWRL [99].

ODIoT-SMH ontology was developed using Protégé version 5.5.0 [108], a free open-source ontology editor and a knowledge management system. It allows for the definition of both the ontology and the set of SWRL rules. *Figure 4-8* depicts an excerpt of ODIoT-SMH ontology in OntoGraf/Protégé, showing the topology and main concepts reused and extended. *Figure 4-9* shows the validation of the proposed ontology with OntoDebug/Protégé[4], which is a plugin that supports automatic discovery and identification of axioms responsible for inconsistency or incoherence in ontologies by applying interactive ontology debugging.

Figure 4-8 Excerpt of the ODIoT-SMH Ontology

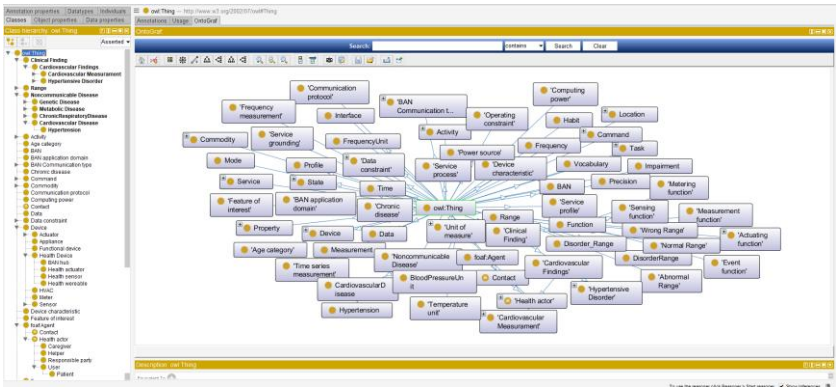
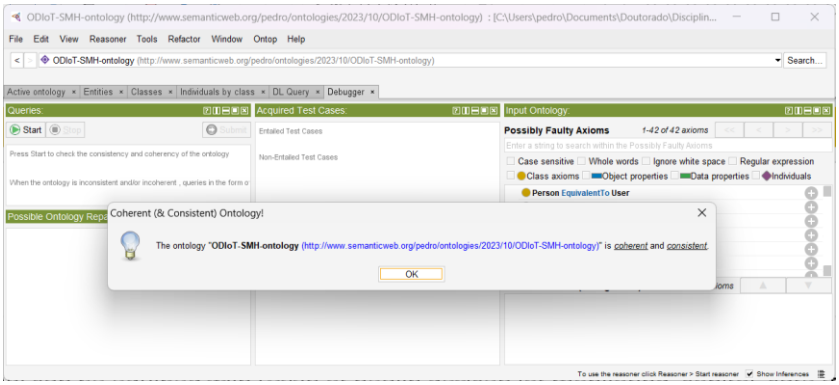


Figure 4-9 Validation of the ODIoT-SMH Ontology



[4] OntoDebug: https://protegewiki.stanford.edu/wiki/OntoDebug

4.4 ODIoT-SMH Scalability Analysis

Due to the high costs and long time needed to develop applications in Edge/Fog computing environments, the efficient management of these applications and the resources of these environments becomes essential. Since simulation is an effective way to assess this efficiency, several tools have been proposed in the literature to simulate the physical behaviour of these environments. However, many of these tools have limitations in terms of usability, efficiency, compatibility and portability [109]. They often fail to support advanced service management features because of their monolithic architecture, the use of synthetic data, and limited scope for periodic update.

To deal with these problems, the basic components of iFogSim [110], an Edge/Fog computing simulator successor to CloudSim [111], were extended with modular simulation models for service migration, dynamic formation of distributed clusters and orchestration of microservices, all based on real datasets. This new simulator, called iFogSim2 [112], was used to validate and evaluate the scalability of ODIoT-SMH. To this end, the following activities were carried out in iFogSim2: system modelling; simulation environment definition; and simulation results. The next subsections describe these activities.

4.4.1 System Modelling

The Java classes in iFogSim2 are annotated so that users without a prior knowledge of CloudSim can define the infrastructure, service placement, and resource allocation policies for Fog and Cloud computing. The main Java classes of iFogSim2 are depicted in *Figure 4-10* and *Figure 4-11*.

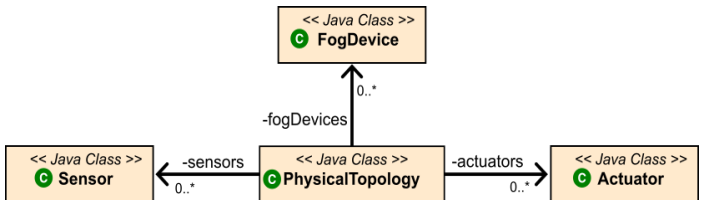


Figure 4-10 Physical Topology Java Classes of iFogSim2 (adapted from [110])

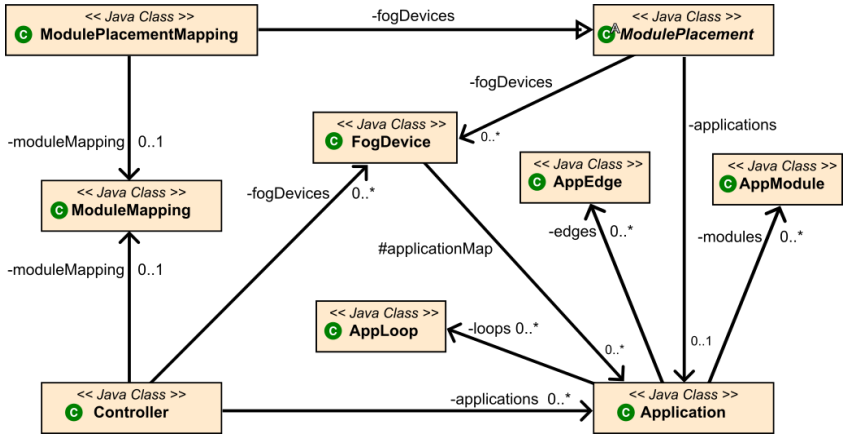


Figure 4-11 Fundamental Java Classes of iFogSim2 (adapted from [110])

Table 4-5 provides a brief description of the main iFogSim2 Java classes. More details regarding these classes and their interactions can be found in [110,112].

Table 4-5 Description of the Main Java Classes of iFogSim2

Java Class	Description
FogDevice	Specifies the hardware characteristics of a Fog device and its connections to sensors and actuators and to other Fog devices. The main attributes of a Fog device include accessible memory, processor, storage size, and uplink/downlink bandwidths. The methods in this class define how the resources of a Fog device are scheduled in the running application modules, and how these modules are deployed and decommissioned.
Sensor	Defines the characteristics of a sensor, ranging from its connectivity to its output, including a reference to the gateway device in the Fog to which it is connected with the latency of this connection. Defines the sensor output characteristics and the distribution of tuples inter-transmission, which identifies the tuple arrival rate at the gateway.
Actuator	Defines the effects of an actuator and its network connection properties, including a reference to the gateway device in the Fog to which it is connected with the latency of this connection. Defines a method to perform an action on arrival of a tuple from an application module, which can be overridden to implement custom effects of actuation.
Tuple	It is the fundamental unit of communication between Fog entities, and it is characterized by its type and the source and destination application modules. A tuple specifies the processing requirements in millions of instructions (MI), and the data length encapsulated in the tuple.

Application	The application design follows the Distributed Dataflow Model (DDF), in which an application is modelled as a Directed Acyclic Graph (DAG), with its vertices representing modules that process incoming data, and its edges denoting data dependencies between modules. These entities are specified in the three classes described below.
AppModule	Represents Fog application processing elements and specifies the DAG vertices. For each incoming tuple, an AppModule instance processes it and generates output tuples that are sent to next modules in the DAG. The number of output tuples per input tuple is decided using a selectivity model, which can be based on a fractional selectivity or a bursty model.
AppEdge	Denotes the data dependency between a pair of application modules and represents a directed edge in the DAG. Each edge is characterized by the type of tuple it carries, which is captured by the *tupleType* attribute of *AppEdge* class along with the processing requirements and length of data encapsulated in this tuple type. iFogSim2 supports two types of application edges: *periodic,* where tuples are emitted at regular intervals; and *event-based*, where a tuple an edge $e = (u, v)$ is sent when the source module u receives a tuple, and the selectivity model of u allows the emission of tuples to module v carried by e.
AppLoop	Specifies the process-control loops of interest to the user, who can specify the control loops to measure end-to-end latency. An *AppLoop* instance is fundamentally a list of modules starting from the origin of the loop to the module where the loop ends.
Controller	Contains the object references of all core classes, such as *FogDevice*, *Sensor*, *Actuator* and *AppModule*, and through an *Application* object, the *Controller* class can also access the *Tuple* class. It launches the *AppModule* instances on their assigned *Fogdevice* instances, following the placement information provided by the *ModuleMapping* object and periodically manages the resources of *Fogdevice* instances.
ModuleMapping	Maps the node name to the module name.
ModulePlacement Mapping	Provides the placement mapping information.
ModulePlacement	Contains the abstract placement policy that needs to be extended for integrating new policies.

The simulator iFogSim2 is packaged with two application module placement strategies:

(a) *Cloud-only placement*, which uses traditional cloud-based deployment, where all modules of an application are executed in data centres. The sense-process-actuate loop in such applications is implemented through sensors that transmit the sensed data to the cloud for processing, and through actuators that are informed if action is required.

(b) *Edge-ward placement*, which tends to deploy application modules on Fog devices that are close to the network's edge. If these devices have not enough computation power to host all application's modules, this strategy iterates Fog devices towards the cloud and tries to place the remaining modules on alternative devices.

To align with the distributed data flow approach adopted by iFogSim2, we model the applications for SBIoT-MPH and ODIoT-SMH through the DAGs shown in *Figure 4-12* and *Figure 4-13* respectively. The objective is to explore the path of clinical data from the Sensor Platform to its visualisation by the user (patients and health professionals), so that we can analyse the performance of alert notifications. For example, if the results of the clinical data analysis point to Hypertension Stage 1, an alert notification should be issued to the patient to take medication. The mobile display and the Web App service were modelled as actuators since they are the visual interfaces for users to access clinical data and alert notifications. Smartphones running the Mobile App can access the API Server through an Internet Service Provider (ISP) proxy. Depending on the patient's location, a different ISP proxy will be used to access the Internet. As we are simulating only one API Server instance in all scenarios, the ISP proxies involved will have different latency values for transferring clinical data to the cloud, which will be calculated based on their locations.

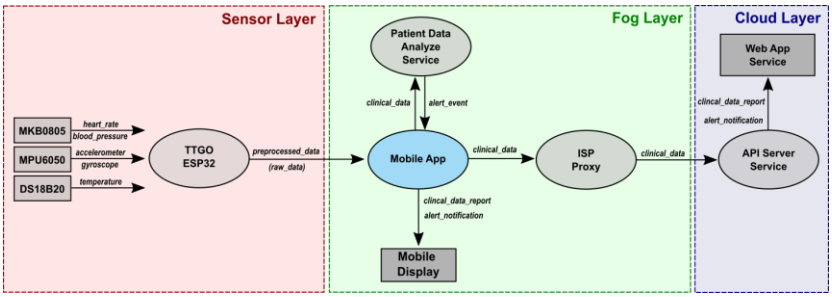


Figure 4-12 SBIoT-MPH application model for the iFogSim2 simulations

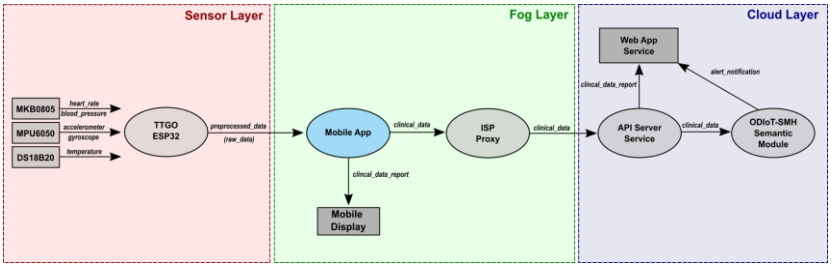


Figure 4-13 ODIoT-MPH application model for the iFogSim2 simulations

4.4.2 Simulation Environment

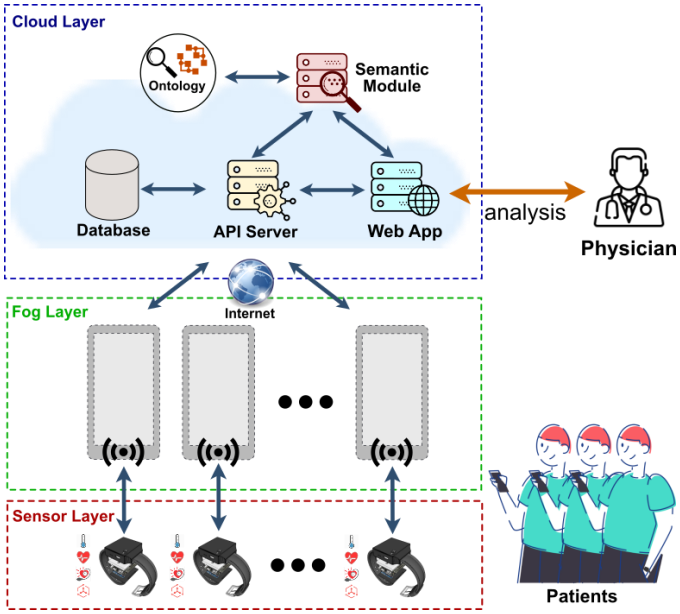
To investigate ODIoT-SMH scalability and compare its performance to that of SBIoT-MPH, we developed three usage scenarios for each model, based on the following relation between the number of patients and the number of health professionals:

(a) *1patient- to-1 health professional (1-to-1)*;

(b) *M patients-to-1 health professional (M-to-1)*, which is illustrated in *Figure 4-14*, and that we set the number of patients to 4, 8 and 12; and

(c) *M patients-to-N health professionals (M-to-N)*, which is illustrated in *Figure 4-15*, and that we set the number of patients to 8, 16 and 24, and the number of health professionals to 2, 4 and 6, with each health professional always responsible for 4 patients.

Figure 4-14 Usage Scenario for the Relation M-to-1



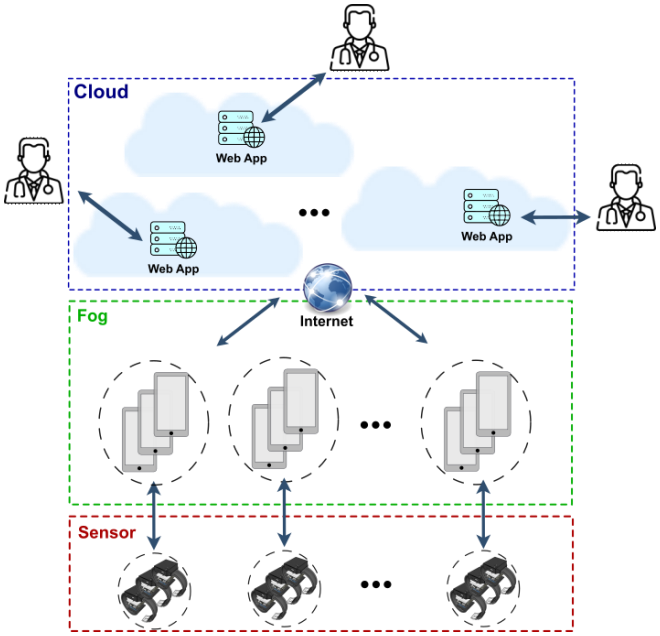


Figure 4-15 Usage Scenario for the Relation M-to-N

We selected these three usage scenarios to analyse the impact of the semantic module on ODIoT-SMH performance as the number of users increases, especially when the clinical data analysis results trigger alert notifications. To this end, we evaluated the efficiency of ODIoT-SMH and SBIoT-MPH in terms of latency, network usage and energy consumption, for each usage scenario. Each of these criteria is calculated using built-in features available in iFogSim2 [86]: latency is computed using the processing latency of the Fog nodes and the cloud network latency, both consisting of data transmission and propagation delay; network usage is computed based on the amount of data propagated across the network during the simulation time; and energy consumption is determined considering its use in busy and idle nodes.

Table 4-6 shows the default configurations for the entities Cloud VM, ISP Proxy, Smartphone, and end devices (Sensor Platform) in terms of speed in Million Instructions Per Second (MIPS), RAM in Gigabytes (GB), uplink and downlink bandwidths in Megabytes Per Second (MBPS), and busy and idle powers. The configuration values for the Patient Data Analysis Service, ODIoT-MPH semantic module and the Sensor Platform (TTGO ESP32) were taken when testing the SBIoT-MPH and ODIoT-MPH systems and are shown in *Table 4-7*.

The simulations of the usage scenarios were carried out on a machine with the following configuration: CPU - Intel Core I5 3337U 2-Core; memory - 16GB RAM; storage - 1024GB SDD; and OS - Windows

11(build 22631). Each simulation was performed 10 times to minimize the impact of outlier records, and the results of these simulations are discussed in the next section.

Table 4-6 Default Configurations of Entities in iFogSim2

Configuration	Cloud VM	ISP Proxy	Smartphone	Sensor Platform
Speed (MIPS)	2500–3000	2500–3000	500	100
RAM (GB)	16	16	1	1
Uplink (MBPS)	100	10	100	100
Downlink (MBPS)	100	20	200	200
Busy power (MJ)	107,3	107,3	87,5	47,5
Idle power (MJ)	83,4	83,4	82,4	12,4

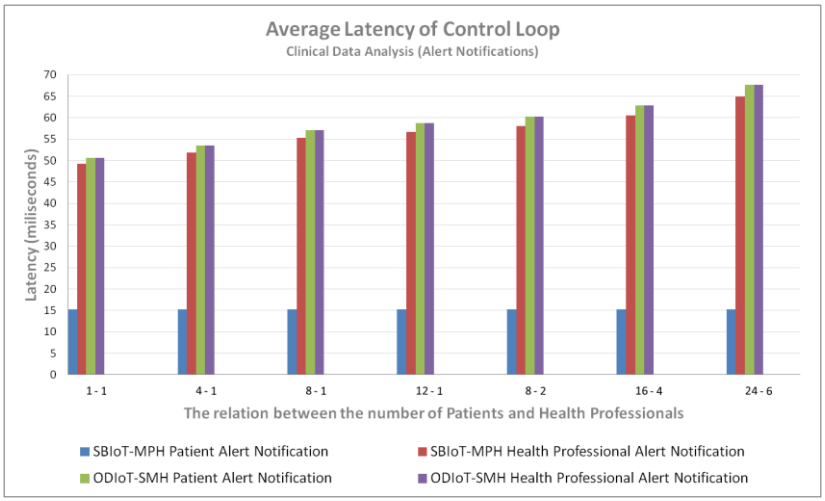
Table 4-7 Configurations of both systems components in iFogSim2

Module	RAM (GB)	Input (MB)	Output (MB)	CPU Length (MI)
Mobile App	0,5	0,5	0,5	1000
Patient Data Analysis	0,1	0,5	0,5	1000
API Server	4	2,5	1,5	4000
ODIoT-MPH Semantic Module	2	2,5	1,5	4000
WebApp	0,1	1	1	1000

4.4.3 Simulation Results

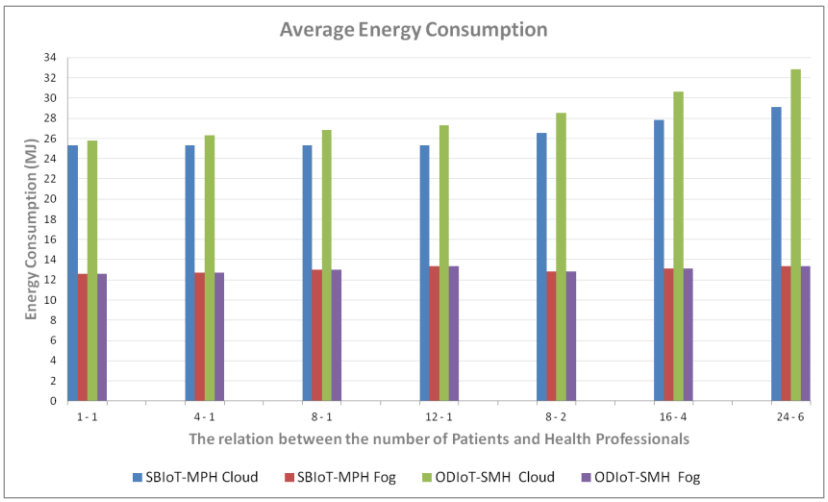
The average latencies for the control loop of interest, which is the path of clinical data from the Sensor Platform to its viewing by the user as an alert notification, are shown in *Figure 4-16*. Patients using SBIoT-MPH see alert notifications 60% faster than patients using ODIoT-SPH. The main reason is that the SBIoT-MPH clinical data analysis is processed in the Fog Layer using the Patient Data Analysis service. On the other hand, ODIoT-SPH processes the clinical data in the Cloud Layer using the Semantic Model. Delays caused by network latency when transferring data between Fog devices until reaching the Cloud Layer result in a significant decrease in data processing performance, when comparing cloud-centric strategies with Edge-ward strategies [110, 112]. Another reason with less impact is the component structures, since the Patient Data Analysis service of SBIoT-MPH uses in-process-based methods and function calls to communicate with the Mobile App, while the ODIoT-SPH uses a set of components that communicate through an MQTT broker. In this case, the more complex the communication implementation, the grater the latency of clinical data processing.

Figure 4-16 Average Latency of Control Loop



The average energy consumptions in the Fog and Cloud layers are shown in *Figure 4-17*. The increase in the number of users practically did not affect the energy consumption in the Fog Layer of both systems, since there was an increase of less than 1% in energy consumption between the 1-to-1 and M-to-N relations. This is due to the fact that the ISP proxy handles multiple transfers of clinical data to the Cloud Layer as the number of patients increases. The Cloud Layer consumes more energy as the number of users increases for both systems. ODIoT-SMH has a 20% higher average energy consumption increase rate than SBIoT-MPH because all ODIoT-SMH users access alert notifications through Cloud Lyer. On the other hand, SBIoT-MPH increases energy consumption in its Cloud Layer only when the number of health professionals increases.

Figure 4-17 Average Energy Consumption



The average amount of data transferred on the network is essential for evaluating different systems. High data transmissions may lead to network congestion, service interruption, and increased average application control loop delay, especially in high-density networks. *Figure 4-18* shows the average network usage for both systems. At the Fog Layer, ODIoT-SMH has much higher network usage than SBIoT-MPH. The main reason is that ODIoT-SHM only has the Web App service in the Cloud Layer as an actuator to access alert notifications. Therefore, if a patient wants to access the Web App from their mobile device, the alert notification data must travel from the Cloud Layer to the Fog Layer to reach that device. This is typical behaviour in a cloud-based deployment strategy, where most of the data processing capability is placed in the Cloud Layer [112].

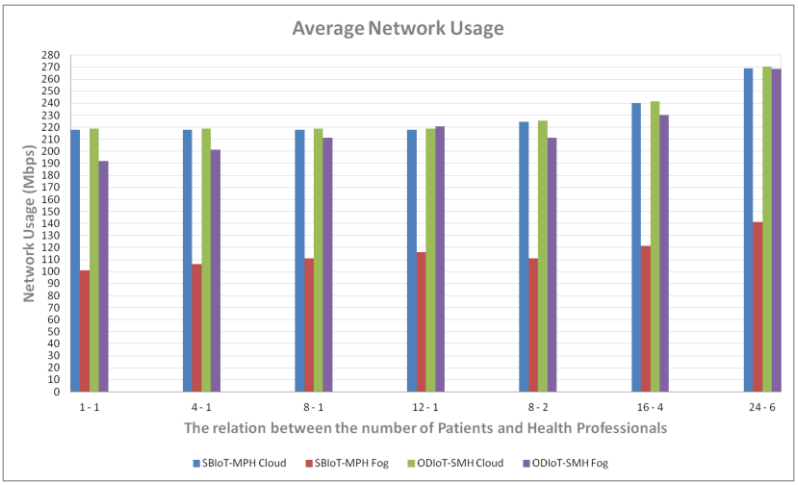


Figure 4-18 Average Network Usage

4.4.4 Discussion

The ODIoT-SMH semantic model was developed to improve interoperability in SBIoT-MPH, thus covering different aspects in each of its layers (see section 4.1 of this thesis). Furthermore, ODIoT-SMH allows for a broader interpretation of the clinical data provided by the Sensor Layer, which allows different customizations of the Patient Data Analysis functionality according to the patient's health profile. However, when comparing the performance of both systems using iFogSim2, the introduction of the semantic model in the Cloud Layer of ODIoT-SMH had a visible cost in analysing the system's scalability in terms of latency, network usage and energy consumption. SBIoT-MPH had better results than ODIoT-SMH in all these criteria. Latency becomes the most worrying of these criteria when related to the speed with which the user accesses alert notifications issued as a result of clinical data analysis.

However, we still believe that adding the semantic model was fundamental in the ODIoT-SMH development. It does not matter if the user can have faster access to the clinical data analysis if the result does not fit the patient's health profile. By removing the restriction from the rules defined in the Patient Data Analysis functionality, we ensure that any alert notification is meaningful for any heath profile.

One strategy that can narrow the performance gap between ODIoT-SMH and SBIoT-MPH is to allocate the Semantic Module services of ODIoT-SMH partially in the Cloud Layer and partially in the Fog Lyer. It is the same principle as the edge-ward placement, which deploys application modules on Fog devices close to the network's edge. In ODIoT-SMH, our Fog device is a mobile device, which has enough resources to process some semantic services such as the customized Patient Data Analysis. As a result, like in previous studies found in the literature [110,111,112], the values obtained for the latency, network usage and energy consumption criteria, tend to decrease when functionalities are pushed from the Cloud Layer to the Fog Layer.

Chapter 5

Microservice-Based IoT System for Monitoring Hypertension

This chapter presents the *Microservice-Based IoT System for Monitoring Hypertension (MBIoT-SMH)*, an RPM IoT system developed in the third Design Cycle of this research, which is also structured according to the IoT three-layer architecture, reuses the components of the Sensor Layer and Fog Layer of SBIoT-MPH, and whose Cloud Layer was redesigned using microservices to improve system scalability and maintainability, and to increase sensor data throughput. For this purpose, DDD was first applied to create domain models and identify subdomains and their bounded contexts. Next, the MSA for the MBIoT-SMH Cloud Layer was designed following MSA's principles.

This chapter is organized as follows: Section 6.1 introduces MSA and DDD; Section 6.2 depicts the domain model and bounded contexts obtained from SBIoT-MPH; Section 6.3 presents the MBIoT-SMH architecture, describing the microservices of its Cloud Layer; Section 6.4 deals with MBIoT-SMH implementation; and Section 6.5 presents an experiment carried out with SBIoT-MPH and MBIoT-SMH to analyse and compare the performance of both systems.

5.1 MSA and DDD

Microservices Architecture (MSA) has emerged as a specialisation of Service Oriented Architecture (SOA) to address challenges, such as scalability and maintainability in Monolithic Architecture (MA). While SOA and MSA revolve around the concept of loosely coupled services that encapsulate business functions, they largely diverge in governance patterns. For example, in traditional SOA, services are more commonly

orchestrated by an Enterprise Service Bus (ESB) that centralises governance functions like routing, authentication and integration of services, whereas, MSA promotes the decentralisation of the architecture, which allows the services to evolve in isolation. Furthermore, traditional SOA favours the orchestration of business transactions, whereas MSA encourages choreography.

According to [27], there are similarities between the requirements for IoT-based and MSA-based systems, and as the former are composed of a large number of IoT devices that generate large volumes of data, a traditional monolithic system might not be able to handle the required load. In [20] the authors claim that efficient architectures employing IoT and microservices are invaluable components of the eHealth domain, and propose an MSA-based reference architecture for RPM, which performs real-time data analysis on the patient's smartphone and then forwards that data to the cloud.

A popular approach to microservice decomposition is to apply DDD [27], a pattern to manage software complexity that revolves around modelling the software in line with the business domain, and that makes a distinction between the problem space and the solution space. The problem space concentrates on what business problems are going to be solved (it is more abstract), whereas the solution space concentrates on how the problems in the problem space are going to be solved (it is more concrete).

The problem space describes the business capabilities, words, requirements and use cases. In addition, it entails the domain, which is a set of patterns aimed at solving business needs, and the subdomains, which encapsulate business functions into manageable pieces. Consequently, when translating this into a software architecture, DDD requires engineers and domain experts to create domain models for the subdomains, and separate them by their scopes, called bounded contexts, which can be identified by looking at where the same domain model has a different meaning [34].

This modular thinking of the system is what makes DDD extremely useful and popular for microservices identification. When creating an MSA by applying DDD, each subdomain with its bounded context will be an MSA's service or possibly a set of services [80].

The DDD's principle of each domain model being owned and developed by a single team is aligned with the MSA's principle of autonomous teams owning and developing MSA services [80]. By splitting the domain into subdomains with their bounded contexts, a cohesive design can be achieved, and if new requirements arise that require changes

in a bounded context of a subdomain, they do not extend to the bounded contexts of others subdomains, which is also aligned with the MSA's principle of independent deployability [21].

5.2 MBIoT-SMH Domain Model

We apply the DDD decomposition methodology proposed in [34] to create a domain model composed of subdomains and their bounded contexts, each subdomain being able to have its own domain model, as they are good candidates for services in an MSA. The domain model has been defined by identifying the domain concepts and their relationships, and has been confirmed by analysing the SBIoT-MPH source code. Bounded contexts have been identified by grouping domain concepts together that are strongly related. *Figure 5-1* shows our domain model, composed of subdomains and their bounded contexts (in yellow), and the domain concepts (in green).

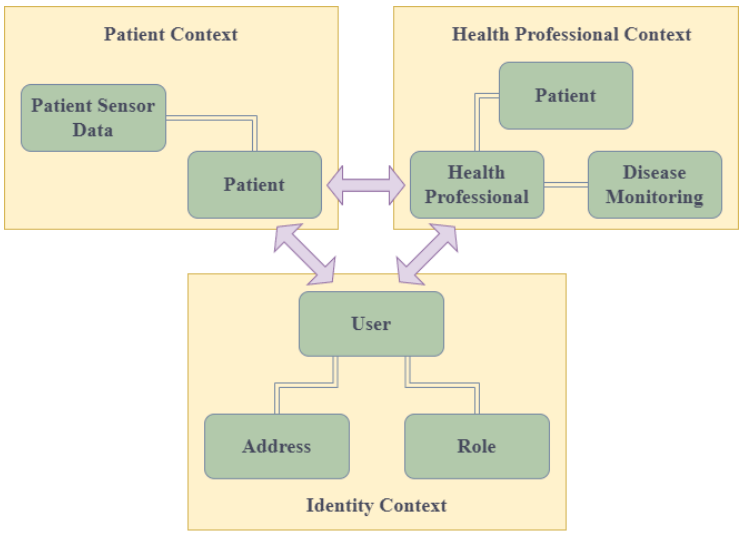


Figure 5-1 Domain Model with Subdomains and their Bounded Contexts and Domain Concepts

The Patient Context encapsulates the patient's information, medical records and sensor data. The Health Professional context manages health professional information, disease monitoring and the patients assigned to this health professional. The Patient concept in this context only holds the information that is relevant to the health professional. Lastly, the Identity Context is where general user management takes place. Consequently, it is not a core subdomain, but rather a supporting one since it is necessary to manage user authentication and roles.

By splitting the domain into subdomains with their bounded contexts, we obtained a cohesive design. For instance, when working with a patient's sensor data, it is often required to correlate that information and perhaps perform some modifications in the context of a patient because it is the only place where we have a complete view of the patient's data. Therefore, if new requirements arise, changes to the sensor data processing do not span other bounded contexts.

5.3 MBIoT-SMH Architecture

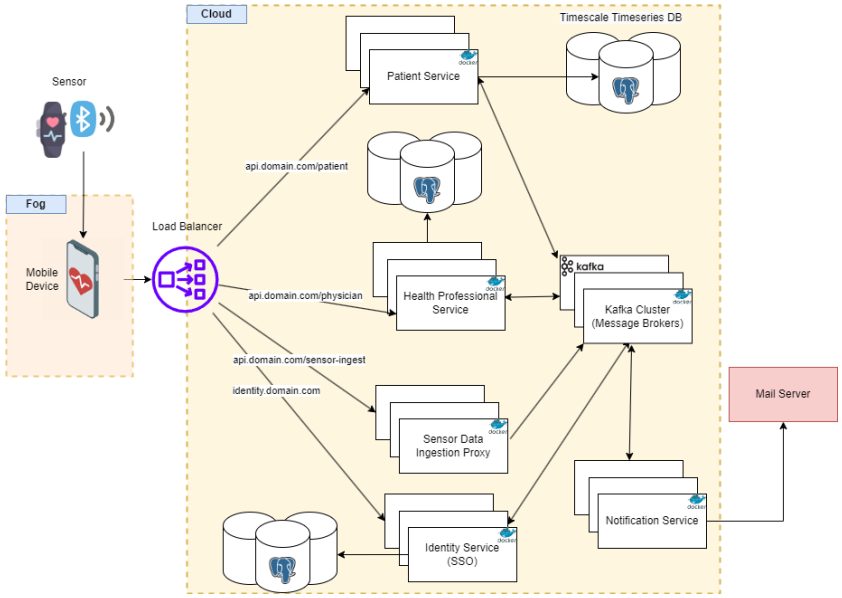
The MBIoT-SMH Cloud Layer consists of the Patient, Health Professional, Identity and Notification services, and a Sensor Data Ingestion (SDI) Proxy component, which works as gateway for the Fog Layer to forward clinical data to the Cloud Layer. Each of these services has its own database, and maintains a connection to a publish/subscribe message broker (Kafka Cluster) and runs an isolated process space (Docker container).

Figure 5-2 shows that MBIoT-SMH is also structured according to the IoT three-layer architecture. The Sensor and Fog layers remain the same of for SBIoT-MPH, but the differences start with the interactions between the Fog and Cloud layers. In SBIoT-MPH the mobile device forwards the data to the API Server using the GraphQL protocol, while in MBIoT-SMH the mobile device uses a REST API and transfers the data via HTTP JSON to the SDI Proxy. All calls to the services are routed by the Load Balancer, which can select the least congested service instance (microservice) through service discovery. Another advantage of having a Load Balancer is that internal topology can change, with new service instances being added or removed, making them available to the clients instantly without waiting for DNS changes to propagate. In this architecture, each service has its own database that cannot be accessed by other services.

MBIoT-SMH uses an asynchronous communication pattern between services through a message broker, making these services loosely coupled and more resilient to cascading failures in case of an unhealthy service instance [114]. Moreover, the developed services are unaware of each other's existence and only publish the events that interested parties (other services) can consume and react to, thereby, making this an Event-Driven-Architecture (EDA). Events in this case are statements about an action that has happened. To enable this communication pattern, we deployed a Kafka Cluster consisting of three brokers in the KRAFT mode, which uses the RAFT consensus algorithm that performs leader elections in the cluster without centralised cluster management software.

Figure 5-2 shows that each service and infrastructure-related software runs in a Docker container to support the requirement of isolation outlined in [21,27]. In addition, containerisation creates a replicable environment, since containers abstract the underlying Operating System (OS). That ensures that if an image is built for a desired computer architecture, it can be deployed to any cloud instance with no compatibility problems.

Figure 5-2 MBIoT-SMH Cloud Layer Architecture



5.3.1 Identity Service

Authentication and authorisation are crosscutting concerns that must be tackled with care to ensure data Confidentiality, Integrity and Availability (CIA). Authentication is rather a trivial task when implemented using 3rd party tools in a monolithic application, since all the logic and interactions reside in one application. However, in MSA if each service authenticates its users, this would lead not only to code duplication but also to poor User Experience (UX) because the user would to present the credentials to each service [21]. Therefore, [21,115] stress the importance of the Single-Sign-On (SSO) in an MSA. SSO features a seamless authentication experience where the user presents the credentials only once and can interact with a variety of services afterwards. The core component of SSO is an identity provider, which can be either managed like Okta, Auth0 or AWS Cognito [21], or can be selfhosted.

Furthermore, to address the challenge of propagating the user's authentication state, we have applied JSON Web Tokens (JWT), which are issued by the service upon successful authentication. Issued JWTs are cryptographically signed using RSA private key and SHA512 hashing algorithm. JWTs were selected because they provide a lightweight approach to transmit an authentication state that makes it unnecessary to call the Identity Service to validate whether the user is authenticated and/or authorised [21].

Moreover, the Identity Service was extended to support signing key rotation and utilises PKCS12 key store type to store private RSA 2048-bit signing keys. In contrast, public keys are collected into a JSON Web Key Set (JWKS) and are exposed via RESTful API so that other services can load the keys at start-up and validate each incoming request, without the need to call the Identity Service.

The Identity Service also provides user management capabilities, such as patient and health professional registration, and as shown in *Figure 5-2*, it stores the user data in a PostgreSQL database. Each update to user information generates events that are forwarded by the Kafka message broker so that other services that duplicate user information can react to those events and update the local user view.

Lastly, health professional and patient registration can be achieved by emitting the respective events from the Health Professional and Patient services, to which the Identity Service reacts by creating the user record, generating the activation token and sending the User Created event, which is consumed by the Notification Service to generate a welcome email.

5.3.2 Patient Service

The Patient Service encapsulates patient-related functions and has its own TimescaleDB, an open-source time series database, to store both patient information and health sensor readings. TimescaleDB is a suitable data store for IoT sensor data because it provides high write throughput and low read latencies [114,115,116]. Furthermore, TimescaleDB extends an ordinary PostgreSQL instance with time-series capabilities, where the data is partitioned based on its timestamp, and does not require an additional query language since the data can be queried using ordinary SQL syntax. TimescaleDB also allows for both Online Transaction Processing (OLTP) and Online Analytics Processing (OLAP) workloads, so that only one database can be used for both patient data and sensor readings.

The Patient Service also exposes a RESTful API for administrators to create, update and delete patients, leading to the emission of respective domain events. Lastly, it provides the interface for patients and health

professionals to query the sensor readings for a given time range, sensor type and patient.

5.3.3 Sensor Data Ingestion (SDI) Proxy

The SDI Proxy has two primary roles:
(1) Analogous to the Technology Integration Adapter in [115], the SDI Proxy exposes the modalities of IoT sensors and provides an interface for sensor integration. In the context of the SBIoT-MPH, the only integration that was implemented is a RESTful API that accepts clinical data from a mobile device, but it has limitless capabilities to be extended and make it work with various IoT sensors and communication protocols without changing other system components [115]; and
(2) The SDI Proxy acts as Inbound Pipeline [115] and as a Sensor Gateway [114], working as a buffer between the Fog layer and the Patient Service. Its benefits are twofold: (a) it potentially increases the throughput and reduces the write latency [114]; and (b) it allows the development of a loosely coupled event-based system [115].

When the Fog layer preprocesses the sensor data, it sends the resulting clinical data to the SDI Proxy, which performs an integrity check of the data and publishes it as an event to the Kafka cluster for the other services to consume. In case of high load spikes, the number of instances of SDI Proxy can be increased, whereas the other services can remain unchanged and process the data at their own pace without affecting the load and write times.

Just like Web servers, the SDI Proxy has to efficiently process a possibly high traffic load of HTTP requests. Web servers are often implemented as a thread-per-request pattern that works well if the operations do not require frequent IO interactions. However, in the case of the SDI proxy, all interactions except for data integrity checks are IO related, such as serialising/de-serialising HTTP requests and pushing the data to the Kafka broker. Consequently, if a thread-per-request pattern were applied to the SDI Proxy, most of the write requests would have to be blocked waiting for the OS to complete the data transfer, which would preclude those threads from servicing the requests of other users. Therefore, the SDI Proxy was built based on the Reactive Programming Paradigm (RPP), which favours nonblocking IO operations, and supports high throughput under high load.

5.3.4 Health Professional Service

The Health Professional Service provides a RESTful API for administrators to create, update and delete health professionals, and analogous to Patient

Service, it emits events that are consumed by the Identity Service for user account management.

The Health Professional Service exposes the endpoints to query all health professionals in the system as well as health professionals by patient and all patients for a given health professional. Additionally, this service exposes an API to assign a health professional to a patient, which creates a many-to-many connection. This service also allows a health professional to send a message to an assigned patient. The health professional can also select a priority message, and upon validation of patient assignment, this message is emitted as an event to the Notification Service, and can be then queried by the patient.

5.3.5 Notification Service

The Notification Service is connected to an external mail server outside the private network through SMTP protocol. This service consumes the notification events from Kafka brokers such as User Activation Required and Message events, to which it reacts and sends the notifications.

Furthermore, the Notification Service follows a Denormalization strategy [117], used in a previously-normalised database to increase performance, and stores partial records of the users with their respective emails. Lastly, this service provides a RESTful API to query all messages by health professionals and patients.

5.4 MBIoT-SMH Implementation

MBioT-SMH was developed in Java version 17 using the Spring framework [118], mainly Spring Boot version 3 and Spring Cloud version 2023. Spring framework is a mature library for building server-side applications, with extensive tooling and an active developer community. Furthermore, Spring Cloud is a sub-project of the Spring ecosystem that provides the tools to build cloud-native applications. Spring Cloud provides out-of-box implementations for security, API gateway, service discovery, caching, message broker integration and others, which makes it a great tool for building microservice-based applications.

Since MBioT-SMH is event-driven, we built our own Identity Service using Spring Boot, Spring Security and Spring Authorization Server. In particular, Spring Authorization Server provides a robust implementation of OAuth 2.0 and OpenID Connect (OIDC) 1.0 standards, which are commonly used to achieve SSO. OIDC 1.0 protocol is used for authentication, whereas OAuth 2.0 is used for authorisation.

Like other services, the Notification Service was built using Spring Boot, and uses the PostgreSQL database to store user emails and messages from health professionals.

5.5 MBIoT-SMH and SBIoT-MPH Performance Analysis

MSA emerged from the enterprise software domain, where sizeable MA applications have shown to be hard to scale and maintain. Thus, the idea was to split an MA into smaller modular pieces to obtain a distributed application highly scalable and fast-changing. To compare the efficiency of MA and MSA in the cloud, we developed a set of simulations to evaluate the performances of SBIoT-MPH and MBIoT-SMH.

One of the main challenges in migrating from MA to MSA is defining an appropriate communication technology. SBIoT-MPH components communicate through GraphQL, a graph-based query language, while MBIoT-SMH components communicate through REST, an abstract model for Web architecture used to guide the definition of HTTP and URIs. To minimise the interference caused by heterogeneous environments and focus on the aspects of each communication technology, most studies that analyse the performance of these technologies use similar architectures to run their simulations [119,120,121]. Although SBIoT-MPH and MBIoT-SMH have very different architectures, we want to verify how GraphQL and REST influence the performance of these systems.

5.5.1 Simulation Design

We designed the two simulation scenarios illustrated in *Figure 5-3* for the performance analysis: Scenario 1 for the MA of SBIoT-MPH Cloud Layer; and Scenario 2 for the MSA of the MBIoT-SMH Cloud Layer. As the API Server of SBIoT-MPH is responsible for communications involving Mobile App and Web App, the load tests that simulate health data input and user data requests are performed through the API Server. For MBIoT-SMH, load tests must involve three different microservices: the SDI Proxy for health data input, the Patient Service for patient data requests, and the Health Professional Service for health professional data requests (see sections 5.3.2, 5.3.3 and 5.3.4).

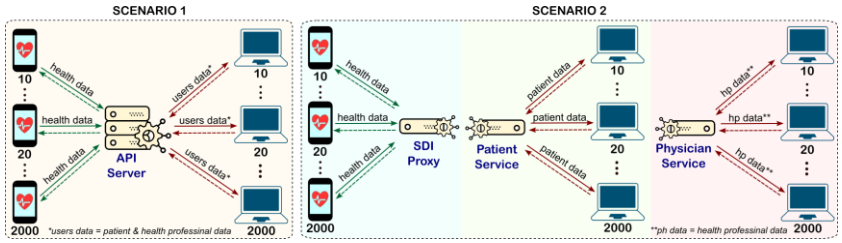


Figure 5-3 Simulation Scenarios

The total processing time for the complete message flow in these simulation scenarios is defined as

$$t = \left(\sum_{i=1}^{n=y} C_i \rightarrow C_{i+1}\right) + \left(\sum_{i=0}^{n=y-1} C_{y-i} \rightarrow C_{y-i-1}\right)$$

where t is the message processing time, C_i is the component with index i, y is the number of components involved in the communication, and $\rightarrow$ is a request/response operation. Different types of messages were sent to evaluate and compare the impact of message complexity on latency and throughput for each architecture. The total time between sending a request and receiving its response was measured and used to calculate latency and throughput.

5.5.2 Evaluation Criteria

In order to guide our performance analysis, we used the following five performance criteria to compare the architectures and the communication technologies employed in the Cloud layer of SBIoT -MPH and MBIoT-SMH:

(1) *Performance*, latency and throughput were used to evaluate and analyse architecture and communication performance as follow:

(a) *Latency* was measured in milliseconds by processing messages of different complexities during requests from a single client; and

(b) *Throughput* was measured by the number of successful requests per second (RPS), with a request being considered successful if the response is received within 1 second. Throughput was measured by processing the same messages used to measure latency, but with an increased request load. During the experiment, the request load started with 10 clients and increased by 10 clients every 30 seconds until it reached 2000 clients.

(2) *Message complexity*, each performance assessment was measured using three types of messages that both systems use in a 'real world' context, i.e., health data, patient data, and health professional data. The complexity of each message is based on the number of properties it sends or requests, as follows:

(a) *Health data* corresponds to the blood pressure sent by the sensor platform, it is composed of 2 data and 1 timestamp, with each data having a type and a value, and it sends a total of 5 properties;
(b) *Patient data* returns all blood pressure data within a time interval, with each patient having 20 blood pressure data registered, and each request returning a total of 50 properties; and
(c) *Health professional data* returns the names and emails of all patients associated with a given health professional ID, with each health professional having 10 registered patients, and each request returning the total of 20 properties.

(3) *Message rate*, the average rates of messages sent and received are measured in kilobytes per second, and this metric can help to identify the potential impacts of communication technologies on network load.
(4) *CPU load*, this metric is stored every 30 seconds, is evaluated at the same time as throughput, and since both architectures run in Docker containers it is evaluated within them, which minimizes interference from other loads, services and processes not related to the architectures.
(5) *Memory utilization*, this metric is also stored every 30 seconds and is computed under the same conditions as the CPU load, i.e., it is restricted to containers during throughput and CPU load assessments.

Figure 5-4 Messages Structures with Different Complexities

Rest and GraphQL Health Data Input::

```
{
    "type": "BLOOD_PRESSURE",
    "data":[{
        "type": "DIASTOLIC",
        "value": 102},{
        "type": "DIASTOLIC",
        "value": 80}
    ],
    "time": "2023-05-14T16:03:00Z"
}
```

REST Patient Data Request::

```
GET {{patient_host}}/v1/patients/{{user_id}}/
sensor-readings?past=320m&from=2023-05-17T08:58:00Z&size=10&page=0&type=BLOOD_PRESSURE
```

GraphQL Patient Data Request::

```
query{
    HealthData(physicianId: 1,
    filter: {
        registerType: BLOOD_PRESSURE,
        startDate: "2023-05-17T08:58:00Z",
        endDate: "2023-05-14T14:58:00Z"
    }
    ){
        data {type value}
        CollectedAt
    }}
```

REST Health Professional Data Request

```
GET {{physician_host}}/v1/physicians/{{physician_id}}/patients?page=0&size=25
```

GraphQL Health Professional Data Request

```
query{
    Patients(physicianId: 1){
        data {name email}
    }}
```

Figure 5-4 exemplifies each type of message and its corresponding response, which is issued by SBIoT-MPH with GraphQL and MBIoT-SMH with REST.

5.5.3 Simulation Tools

To minimize interference, which could be caused by Internet instabilities, the simulations were carried out in a controlled environment composed of two computers, one acting as a Server and the other as a Client, interconnected by RTF8115VW router with a 300Mbps IEEE 802.3 local network. It is the same configuration used for the SBIoT-MPH performance evaluation (see *Figure 3-17* in Section 3.5).

Because each architecture and its software infrastructure run in Docker containers, we deployed these containers on a physical machine that ran as the server with the following configuration: CPU - AMD Ryzen 5 5600X 6-Core; memory - 16GB RAM; storage - 1024GB SDD; and OS - Windows 11 (build 22631). Another physical machine ran as the client with the following configuration: CPU - Intel Core I5 3337U 2-Core; memory - 16GB RAM; storage - 1024GB SDD; and OS - Windows 11(build 22631). We deployed the Apache JMeter [92] load testing tool in the client machine for simulating concurrent users and generating large volumes of synthetic data. For the CPU load and memory utilization metrics, we used PerfMon Metrics Collector [122], a JMeter plugin for server health monitoring, and Docker stats [123], a Docker command for container runtime metrics.

Furthermore, in MBIoT-SMH each service was deployed as a single instance for better comparison with SBIoT-MPH, since this monolithic system uses only one instance of the API service. Also, there were no changes to the source codes of the architectures of these systems, both for tracking component communications and for capturing metrics for simulations.

5.5.4 Simulation Results

We performed two simulations to cover our set of criteria: simulation for the latency criterion, which was measured by processing messages of different complexities requested by 1 client; and simulation for the criteria throughput, message rate, CPU load and memory utilization, which were measured by processing messages of different complexities while increasing the number of clients by 10 every 30 seconds until reaching 2000 clients. Each simulation was performed 20 times to minimize the impact of outlier records.

The results of the SBIoT-MPH and MBIoT-SMH simulations for the latency criterion are shown in *Table 5-1*, where its columns contain the mean, median, minimum (Min), and maximum (Max) latency results for the three types of messages. Regarding the complexity of messages, these results show that the latency of SBIoT-MPH was always better than that of MBIoT-SMH, as it was 3,65% lower for health data, 1,73% lower for patient data, and 1,06% lower for health professional data.

Table 5-1 Simulation Results for the Latency Criterion

	Message Complexity	Mean (ms)	Median (ms)	Min (ms)	Max (ms)
SBIoT-MPH (GraphQL API)	**Heath Data**	12,598	12,362	9,182	17,604
	Patient Data	17,748	17,323	15,089	22,734
	Health Professional Data	12,808	12,568	9,335	17,897
MBIoT-SMH (REST API)	**Health Data**	13,076	12,833	9,482	18,454
	Patient Data	18,060	17,629	15,240	23,414
	Health Professional Data	12,946	12,704	9,318	18,173

The results of the SBIoT-MPH and MBIoT-SMH simulations for the throughput criterion are shown in *Figure 5-5* and *Figure 5-6* respectively. For SBIoT-MPH, the best average result, 141,33 RPS, was achieved by processing health data, and the worst average result, 122,42 RPS, was achieved by processing patient data. For MBIoT-SMH, the best average result, 136,86 RPS, was also achieved by processing health data, and the worst average result, 119,63 RPS, was also achieved by processing patient data. Up to 410 concurrent users, SBIoT-MPH performed slightly better than MBIoT-SMH as MA processed on average 1,89% more health data, 1,79% more patient data, and 1,83% more health professional data than the MSA. Above 410 concurrent users, MBIoT-SMH performed better than SBIoT-MPH as MSA processed on average 4,69% more health data, 3,88% more patient data, and 4,33% more health professional data than the MA.

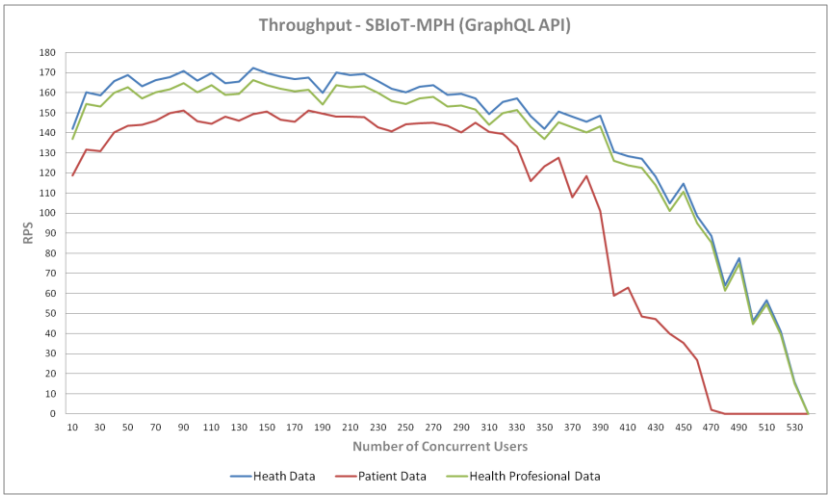


Figure 5-5 SBIoT-MPH Simulation Results for the Throughput Criterion

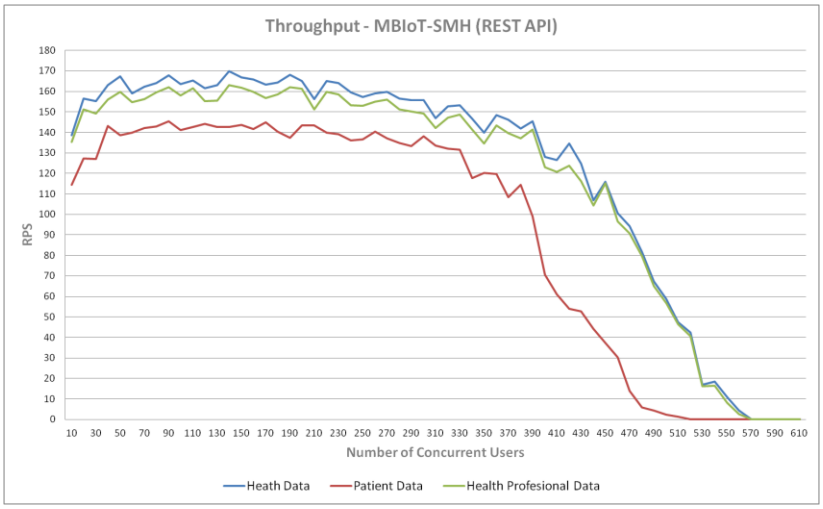


Figure 5-6 MBIoT-SMH Simulation Results for the Throughput Criterion

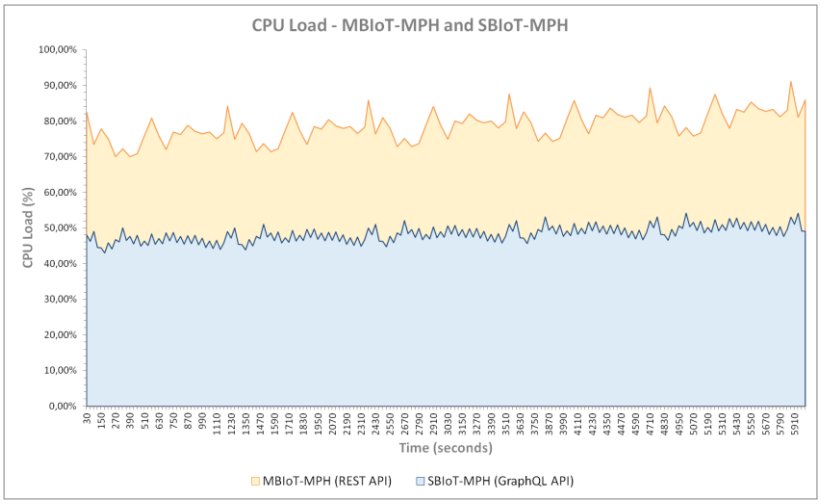
The average rates of messages sent and received and their payloads are shown in *Table 5-2*. For all types of messages, SBIoT-MPH showed slightly higher average rates than MBIoT-SMH. The two main reasons for this are that GraphQL has a higher data overhead and a larger payload in data requests than REST. This behaviour is expected because in REST the client makes an HTTP request and the data is sent as an HTTP response, while in GraphQL the client requests data with queries. It is also worth mentioning that in REST the structure of the request object is defined on the server, while in GraphQL it is defined on the client.

Table 5-2 Simulation Results for the Message Rate Criterion

	Message Complexity	Received (KB/s)	Sent (KB/s)	Payload (Bytes)
SBIoT-MPH (GraphQL API)	**Heath Data**	54,26	95,14	90
	Patient Data	178,68	31,45	170
	Health Professional Data	171,12	21,50	120
MBIoT-SMH (REST API)	**Health Data**	44,65	72,44	80
	Patient Data	159,80	19,27	150
	Health Professional Data	153,85	15,27	111

The results of the SBIoT-MPH and MBIoT-SMH simulations for the CPU load criterion are shown in *Figure* 5-7. At the peak usage, the MSA used 88,45% CPU and remained slightly fluctuating in heavy usage, with an average of 78,74% CPU within the range of 70 - 88,45 % CPU. Meanwhile, the MA is consistently steady with an average of 48,29% within the range of 44 - 53 % CPU.

Figure 5-7 Simulation Results for the CPU Load Criterion



The results of the SBIoT-MPH and MBIoT-SMH simulations for the memory utilization criterion are shown in Figure 5-8. The MSA had an average memory utilization rate of 76,99 MB, and tended to fluctuate

periodically with alternating peaks and valleys within the range of 62,50 - 91,43 MB and Standard Deviation (SD) of 5,89 MB. Meanwhile, the MA proved to be very stable and low in memory consumption, with an average memory utilization rate of 53,47 MB within the range of 48, 6 - 60.95 MB and SD of 1,96 MB.

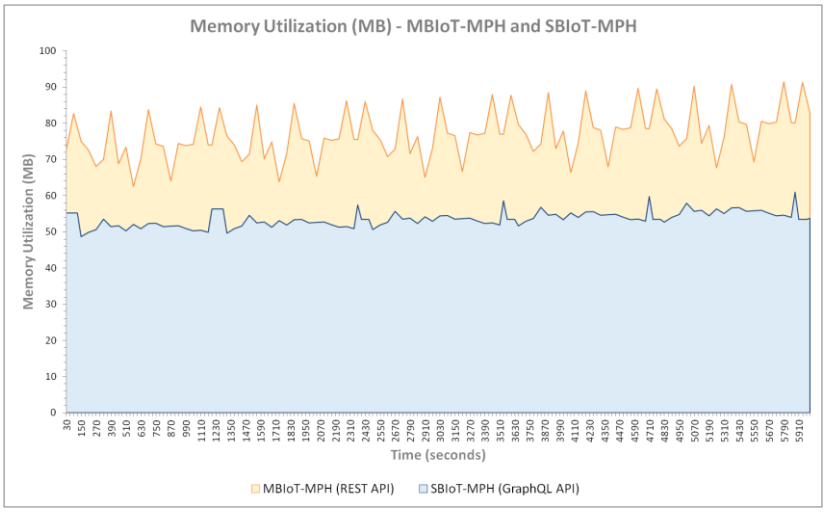


Figure 5-8 Simulation Results for the Memory Utilization Criterion

5.5.5 Discussion

From the architecture design perspective, the results obtained with the simulations for the latency and throughput criteria shows a similar performance between the MA of SBIoT-MPH using GraphQL, and the MSA of MBIoT-SMH using REST. The SBIoT-MPH throughput was slightly better than that of MBIoT-SMH until the simulation hit 410 concurrent users, indicating that the MA can perform better than MSA under a small number of users. Above 410 concurrent users, the MBIoT-SMH throughput started to be slightly and increasingly better than that of SBIoT-MPH, indicating that MSA can scale better than MA. The main reason for this behaviour is that MBIoT-SMH relies on Kafka cluster for asynchronous inter-service communication, while the SBIoT-MPH uses in-process-based methods and function calls. Although asynchronous communication allows loose runtime coupling and improves availability, it requires a more complex implementation and more resources, which can be seen when analysing the CPU load and memory utilization criteria. On runtime, in the MSA the Kafka cluster had higher rates and SD than the microservices, and it was the only component that followed the MBIoT-SMH CPU load and memory utilization pattern.

The Kafka cluster can buffer the data for the microservices, which minimizes possible bottlenecks in data processing. The difference in performance becomes more evident when both architectures are submitted to a stress performance test. In one of the throughput test cycles we performed, we did not limit the number of concurrent users for the health data. Up to 2,500 concurrent users, MBIoT-SMH achieved almost 10% higher throughput than SBIoT-MPH, and from 2500 to 5000 concurrent users this difference raised to 28%. Above 5000 concurrent users the SBIoT-MPH collapsed, as the average error rate (unsuccessful health data input) was over 73%, making the MA unsuitable. One main reason for the SBIoT-MPH failing under this stress performance test was the database write operations, where many user requests were idle for a long time waiting for these operations to complete.

From the communication technology perspective, the results obtained with the simulations for the message rate criteria shows that REST has always smaller messages than GraphQL, and if this is the most important criterion when choosing communication technology, then REST is recommended. Furthermore, GraphQL has the resolvers, an extra layer from the service side, which maps the schema with the query, adding an overhead to data processing. Although we were unable to measure the impact of the resolver layer in our simulations, previous studies have shown that it impacts performance compared with other communication technologies [124].

On the other hand, because GraphQL supports remote queries, a single GraphQL request/response could potentially transfer as much information as multiple REST requests/responses. This situation happens in the SBIoT-MPH performance evaluation (see section 3.5), where for its MA and single database one GraphQL query is enough to handle a patient's personal data and their healthcare records while REST needs two requests. However, as in an MSA each microservice usually have its own database, which tends to be a fraction of an MA database, the possibility of reducing the number of requests/responses in an MSA by changing Rest to GraphQL seems unlikely, as information is spread across several microservices.

In conclusion, MSA is not the most suited for all contexts, as MA seems to be a better choice for simple and small systems that do not need to support a large number of concurrent users. Previous studies comparing the performance of MA and MSA obtained results that corroborate our statement [124,125,126].

The same applies to REST, it is not the most suited communication technology for all contexts, as GraphQL seems to be a better choice for

MA where the benefits of data modelling with queries stand out. The ability for the client to select data fields or properties for specific requests can result in much less and more efficient use of computing resources compare to other communication technologies. Additionally, the request is made only once on a single endpoint, and the client can specify flexible response formats to avoid very large data transfers (overfetching) or sparse data responses (underfetching), as can happen with REST. Previous studies comparing the performance of GraphQL and REST in Web services obtained results that corroborate our statement [119,120,121].

Chapter 6

Approach for Designing Interoperable, Scalable, and Maintainable Remote Patient Monitoring IoT Systems

This chapter presents an approach, described through a process, to design interoperable, scalable and maintainable RPM IoT system, which we distilled from the design interactions of chapters 3 to 5, and is organized as follows: Section 6.1 gives an overview of this approach describing its main activities; Section 6.2 deals with how system requirements should be captured depending on the NCD to be monitored, focusing on the patients and health professionals' purposes for remote monitoring of this NCD, the objectives, scope and scenarios of the system; Section 6.3 deals with the design of the system ontology, focusing on ontology engineering; Section 6.4 deals with the design of the Sensor Layer components, focusing on the sensor platform; Section 6.5 deals with the design of the Fog Layer components, focusing on the mobile application; Section 6.6 deals with the design of the Cloud Layer components, focusing on the semantic model, ontology and MSA; and Section 6.7 deals with the building and validation of the entire system.

6.1 Overview

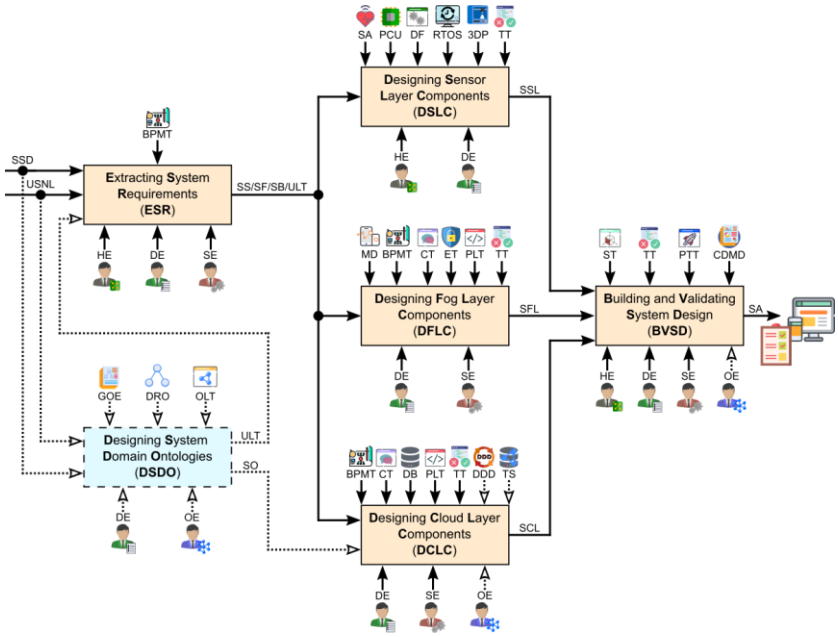
Based on the experience gained with the development of artifacts in the three Design Cycles of this research, we propose an approach, to be used mainly in the development of IoT systems for monitoring patients suffering

from NCDs that can be monitored remotely, such as Hypertension, Diabetes, Asthma and Obesity.

Since the three-layer architecture (Sensor, Fog, Cloud) has been used in the development of IoT systems for monitoring patients and elderly [84], and for that reason it was also used in the development of the artifacts of this research, the approach proposed here was structured according to this architecture.

Figure 6.1 provides an overview of our approach, described through a process represented by a diagrammatic notation similar to the Structured Analysis and Design Technique (SADT) [127], which was also used in [119]. In this diagram, a rectangle represents a main activity of this approach, while an arrow on a rectangle can have one of the following meanings: on the left-hand side represents an activity input; on the right-hand side represents an activity output; at the top represents a resource used in the activity; and at the bottom represents a role that should be played by some member of the Development Team (DT) that performs the activity.

Figure 6-1 Approach Overview



The main activities of our approach are: *Extracting System Requirements (ESR)*, *Designing System Domain Ontologies (DSDO); Designing Sensor Layer Components (DSLC)*, *Designing Fog Layer Components (DFLC)*, *Designing Cloud Layer Components (DCLC)*, and *Building and Validating System Design (BVSD)*.

In the ESR activity common and specific requirements related to NCDs, patients and health professionals are captured. In the DSDO activity an ontology is developed for the specific system domain, in this case the NCD to be monitored. This is a mandatory activity if semantic interoperability is to be handled, otherwise it is optional. Depending on the DT composition, ESR and DSDO may be carried out in parallel, possibly with some overlap. In the DSLC, DFLC and DCLC activities, components of the Sensor, Fog and Cloud layers are designed, respectively, for the specific system domain. Depending on the DT composition, these activities may be carried out in parallel, possibly with some overlap. In the BVSD activity the system layers to build a System Artifact (SA) are integrated, and the system is validated. Several validation strategies can be applied in this activity.

The inputs and outputs of these activities are: *System Domain Documents (SDD)*, *User Stories in Natural Language (USNL)*, *System Scenarios (SS)*, *System Features (SF)*, *System Behaviours (SB)*, *Ubiquitous Language Terminology (ULT)*, *System Ontology (SO)*, *System Sensor Layer (SSL)*, *System Fog Layer (SFL)*, *System Cloud Layer (SCL)*, and *System Artifact (SA)*.

The main resources employed by these activities are: *Business Process Modelling Techniques (BPMT)*, *Guidelines for Ontology Engineering (GOE)*, *Domain Reference Ontologies (DRO)*, *Ontology Languages and Tools (OLT)*, *Sensors and Actuators (SA)*, *Processing and Communication Units (PCU)*, *Development Framework (DF)*, *Real-Time Operating System (RTOS)*, *3-Dimensional Printer (3DP)*, *Clinical Data Measurement Devices (CDMD)*, *Mobile Devices (MD)*, *Encryption Techniques (ET)*, *Communication Technologies (CT)*, *Programming Languages and Tools (PLT)*, *Testing Tools (TT)*, *Performance Testing Tools (PTT)*, *Simulation Tools (ST)*, *Database (DB)*, *Triplestore (TS)*, and *Domain Driven Design (DDD)*.

The DT should be multidisciplinary and their members should perform the following roles in order to generate appropriate system artifacts:
(a) *Domain Expert (DE)* is responsible to contribute to the activities with their knowledge of the system domain;
(b) *Ontology Engineer (OE)* is responsible for defining and performing the ontology engineering process to obtain the system domain ontologies, and also for assisting in the formalization of system requirements and the design of the semantic model;
(c) *Software Engineer (SE)* is responsible for tracking the entire software life cycle, ensuring its proper designing and validation in accordance with the system's software requirements; and

(d) *Hardware Engineer (HE)* is responsible for tracking the entire hardware life cycle, ensuring its proper designing and validation in accordance to the system hardware requirements.

6.2 Extracting System Requirements (ESR)

The ESR activity starts by identifying the characteristics of the NCD to be monitored, the patients and health professionals' purposes for remote monitoring of this NCD, based mainly on System Domain Documents (SDD). The purposes and scope of the system, as well the System Scenarios (SS), are defined based mainly on User Stories in Natural Language (USNL). If the DSDO activity is not planned, a Ubiquitous Language Terminology (ULT), to be used in all activities, should be defined in the ESR activity.

These scenarios can be described by means of Business Process Modelling Techniques (BPMT) [129], such as UML diagrams, Business Process Model Notation (BPMN) [121] and flowcharts, to illustrate key system requirements. The ULT, defined in this activity or provided by the DSDO activity, should be used in this description.

Based on these scenarios, the main functionalities of the system are defined, and the System Features (SF) and System Behaviours (SB) for providing such functionalities are described using, for example, a Behaviour Driven Development (BDD)-like notation [131]. The ULT should also be used in these descriptions.

Table 6.1 summarizes the inputs, outputs, resources, roles and the main tasks of the ESR activity, and an example of this activity is provided in [128].

Table 6-1 Inputs, Outputs, Resources, Roles and Tasks of ESR Activity

Inputs	System Domain Documents (SDD) User Stories in Natural Language (USNL) Ubiquitous Language Terminology (ULT)
Outputs	System Scenarios (SS) System Features (SF) System Behaviours (SB) Ubiquitous Language Terminology (ULT)
Resources	Business Process Modelling Techniques (BPMT)
Roles	Software Engineer (SE) Hardware Engineer (HE) Domain Expert (DE)
Tasks	(a) Identify the NCD characteristics, and the purposes of patients and health professionals; (b) Define the system objectives, scope, and SS;

	(c) Define ULT if the DSDO is not planned; (d) Draw up SS; (e) Define hardware and software functionalities; and (f) Draw up SF and SB.

6.3 Designing System Domain Ontologies (DSDO)

The DSDO activity starts by searching for Guidelines for Ontology Engineering (GOE) in IoT covering best practices and methodologies to be followed during the development of the System Ontology (SO). Then, the SO characteristics (*e.g.*, purposes, requirements), regarding the NCD to be monitored, are defined based mainly on the same SDD and USNL used in the ESR activity.

This activity continues by searching for Domain Reference Ontologies (DRO) for the system, such as SAREF4EHAW [65], which can be used as an IoT reference ontology for e/m-Health systems, and SNOMED-CT [72], which can be used as a Health reference ontology for providing Health domain terminology including NCDs. In this way, the DROs provide the means for defining the ULT to be employed in all activities.

Based on the SO characteristics and based on the main elements of interest of the DROs, a set of Competency Questions (CQ) can be defined, which should be answered during the following steps of SO development: modularisation, alignment, merging, and extension of DROs. The scopes of these CQs can guide these development steps.

In the SO development, appropriate Ontology Languages and Tools (OLT), such as OWL [62] and Protégé [108], OntoUML [132] and OntoUML Lightweight Editor (OLED) [133], must be employed. Besides these resources, each one of the development steps should follow the best practices and methodologies of the chosen GOE.

The quality of SO can be evaluated according to the structural and functional dimensions [125]. Structural validation considers the ontology logical structure, focusing on its syntax and formal semantics. Functional validation focuses on the ontology usage, which includes evaluation by domain experts, user satisfaction, task assessment, and topic assessment (see section 4.3).

Table 6.2 summarizes the inputs, outputs, resources, roles and the main tasks of the DSDO activity, and its development is illustrated in section 4.3 of this thesis.

Table 6-2 Inputs, Outputs, Resources, Roles and Tasks of DSDO Activity

Inputs	System Domain Documents (ADD) User Stories in Natural Language (USNL)
Outputs	Ubiquitous Language Terminology (ULT) System Ontology (SO)
Resources	Guidelines for Ontology Engineering (GOE) Domain Reference Ontologies (DRO) Ontology Languages and Tools (OLT)
Roles	Ontology Engineer (OE) Domain Expert (DE)
Tasks	(a) Search for GOE in IoT; (b) Define SO characteristics; (c) Search for DRO for the system; (d) Define ULT; (e) Define Competency Questions (CQ); (f) Perform modularisation, alignment, merging, and extension of DROs to obtain SO; and (g) Perform SO validation.

6.4 Designing Sensor Layer Components (DSLC)

The DSLC activity starts by searching for IoT technologies to collect, process and transmit physiological data related to the NCD to be monitored. This search is mainly based on the SS, SF and SB, related to the Sensor Layer, which are provided by the ESR activity. Then, a sensor platform should be developed using the chosen technologies. The ULT provided by the DSDO activity or by the ESR activity should be employed in this development. Finally, the System Sensor Layer (SSL) must be tested using appropriate Testing Tools (TT).

Table 6.3 summarizes the inputs, outputs, resources, roles and the main tasks of the DSLC activity, and its development is illustrated in section 3.2 of this thesis.

Table 6-3 Inputs, Outputs, Resources, Roles and Tasks of DSLC Activity

Inputs	System Scenarios (SS), System Features (SF), System Behaviours (SB) Ubiquitous Language Terminology (ULT)
Outputs	System Sensor Layer (SSL)
Resources	Sensors and Actuators (SA) Processing and Communication Units (PCU) Development Framework (DF) Real-Time Operating System (RTOS) 3D Printer (3DP) Testing Tools (TT)
Roles	Hardware Engineer (OE) Domain Expert (DE)

Tasks	(a) Search for SA to capture/send the necessary signals for monitoring the chosen NCD; (b) Search for PCU to process/transmit the necessary clinical data for monitoring the chosen NCD; (c) Search for DF and RTOS for developing software to be embedded on the Sensor Platform (SP); (d) Implement SP with the chosen components; (e) Integrate SP into an IoT wearable device; and (f) Test the System Sensor Layer (SSL).

6.4.1 Sensor Platform (SP)

The Sensor Platform SP) contains a Wireless Body Sensor Network (WBSN), which must have the Sensors and Actuators (SA) to capture the patient's vital signs (raw data) relevant to the NCD to be monitored, and eventually act, for example issuing alert signals.

Given that this raw data should be filtered, aggregated and converted to digital format on the platform itself, and the resulting clinical data must be transmitted to the Fog Layer, SP must also contain Processing and Communication Units (PCU), supporting appropriate protocols, to carry out these tasks and also to deal with security issues.

As software must be developed to be embedded on SP, a Development Framework (DF) and a Real-Time Operating System (RTOS) can be used for this purpose.

Then, SP must be implemented and integrated into an IoT wearable device to be worn by the patient. For this purpose, a 3-Dimensional Printer (3DP) must be employed. Finally, this IoT wearable device, which constitutes the System Sensor Layer (SSL), must be tested using appropriate Testing Tools (TT).

6.5 Designing Fog Layer Components (DFLC)

The DFLC activity starts by searching for Mobile Devices (MD), such as smartphones and tablets, to act mainly as a gateway, receiving data from the Sensor Layer and sending data to the Cloud Layer. The chosen MD must host a Mobile Application (MAp) that will receive clinical data from the IoT wearable device, pre-process this clinical data, to be properly presented to the patient and for the eventual generation of alert signals, and send the pre-processed clinical data to the cloud. The development of MAp is mainly based on the SS, SF and SB, related to the Fog Layer, which are provided by the ESR activity. The ULT provided by the DSDO activity or by the ESR activity should be employed in this development. Finally,

the System Fog Layer (SFL) must be tested using appropriate Testing Tools (TT).

Table 6.4 summarizes the inputs, outputs, resources, roles and the main tasks of this activity, and its development is illustrated in section 3.3 of this thesis.

Table 6-4 Inputs, Outputs, Resources, Roles and Tasks of DFLC Activity

Inputs	System Scenarios (SS), System Features (SF), System Behaviours (SB) Ubiquitous Language Terminology (ULT)
Outputs	System Fog Layer (SFL)
Resources	Mobile Devices (MD) Business Process Modelling Techniques (BPMT) Communication Technologies (CT) Encryption Techniques (ET) Programming Languages and Tools (PLT) Testing Tools (TT)
Roles	Software Engineer (SE) Domain Expert (DE)
Tasks	(a) Search for MD to act as a gateway between the Sensor and Cloud layers and to host a Mobile Application (MAp); (b) Develop MAp using BPMT; (c) Implement MAp using appropriate CT, ET and PLT; (d) Integrate MAp into the MD; and (e) Test the System Fog Layer (SFL).

6.5.1 Mobile Application (MAp)

The development of the Mobile Application (MAp) should be supported by BPMT. If UML is chosen for that purpose, this development can involve the following steps: definition of MAp use cases using UML Use Case Diagram; for each use case, definition of MAp internal operations using UML Sequence Diagram; and based on the use cases and the corresponding sequence diagrams, definition of MAp software architecture using UML Class Diagram.

From these UML diagrams and using appropriate Programming Languages and Tools (PLT), an MAp implementation can be achieved. Regarding the security of the stored data in MD, Encryption Techniques (ET) can be used to this end. Another major concern should be the communication between MAp and the Cloud Layer applications, and Communication Technologies (CT), such as REST and GraphQL, should be considered for this purpose.

6.6 Designing Cloud Layer Components (DCLC)

The DCLC activity starts by choosing an appropriate architecture for the Cloud Layer. This choice depends on the complexity of the System Artifact (SA) that is being developed, and can be between a Monolithic Architecture (MA) and a Microservices Architecture (MSA). This choice also depends on how the company responsible for developing this activity is organized and the experience of its development teams (see section 2.4 of this thesis).

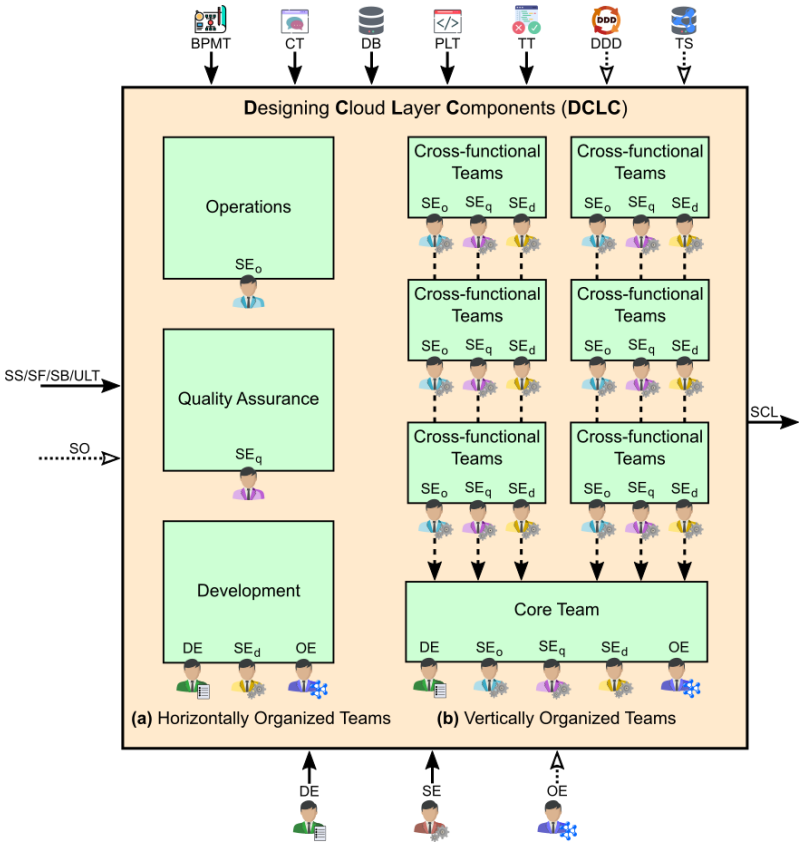


Figure 6-2 Development Team Organizations (adapted from [135])

As illustrated in *Figure 6.2*, if MA is chosen, the Development Team (DT) is generally organized horizontally into teams separated by functionality, such as development, quality assurance and operations teams. If MSA is chosen, the DT should be organized vertically into small cross-functional service teams, such as *Software Engineering for Development* (SE_d), *Software Engineering for Quality Assurance* (SE_q) and *Software Engineering for Operations* (SE_o), with each service team incorporating all defined

functionalities and being responsible for its own services. It should also include a core team, composed of representatives from service teams, the Domain Expert (DE) and eventually the Ontology Engineer (OE), so that this team can have an overview of service interactions across the entire Cloud Layer, and can make critical decisions regarding this layer [135].

This activity continues by deciding whether or not the chosen Cloud Layer architecture should handle Semantic Interoperability (SI). This decision depends on SA requirements, such as support for different devices, interoperation of its components, extensibility to cope with other diseases, and integration with other systems (see section 2.3 of this thesis).

The development of this activity is mainly based on the SS, SF and SB, related to the Cloud Layer, which are provided by the ESR activity. If SI is to be handled, the SO provided by the DSDO activity must be employed in this development. The ULT provided by the DSDO activity or by the ESR activity should be employed in this development. Finally, the System Cloud Layer (SCL) must be tested using appropriate Testing Tools (TT).

Table 6.5 summarizes the inputs, outputs, resources, roles and the main tasks of this activity, and its development is illustrated in sections 3.4, 4.1, 4.2, 5.2 and 5.3 of this thesis.

Table 6-5 Inputs, Outputs, Resources, Roles and Tasks of DCLC Activity

Inputs	System Scenarios (SS), System Features (SF), System Behaviours (SB) Ubiquitous Language Terminology (ULT) System Ontology (SO)
Outputs	System Cloud Layer (SCL)
Resources	Business Process Modelling Techniques (BPMT) Communication Technologies (CT) Database (DB) Programming Languages and Tools (PLT) Testing Tools (TT) Domain Drive Development (DDD) Triplestore (TS)
Roles	Software Engineer (SE) Domain Expert (DE) Ontology Engineer (OE)
Tasks	(a) Choose an architecture for the Cloud Layer; (b) Decide if the chosen architecture should handle Semantic Interoperability (SI); (c) If a Monolithic Architecture (MA) was chosen, (c_1) develop two applications using BPMT, the first to provide an API and the second an UI; (c_2) Implement then using appropriate CT and PLT; If SI must be handled, (c_3) Develop a Semantic Model (SM), which uses SO,

	DB and TS, to be deployed in the cloud; (c_4) Develop a Semantic Module (SMo) based on SM and using BPMT, which communicates with the applications, SO, DB and TS; (c_5) Implement SMo using appropriate CT and PLT; If an Microservices Architecture (MSA) was chosen, (c_1) Decompose the Cloud Layer into a set of services If DDD was chosen for this decomposition ($c_{1.1}$) Define the subdomains with their model concepts and bounded contexts; ($c_{1.2}$) Map each subdomain with its bounded context to a service or a set of services; ($c_{1.3}$) Define the supporting services; ($c_{1.4}$) Develop the services using BPMT; ($c_{1.5}$) Implement the services using appropriate CT and PLT; If SI must be handled, (c_2) Include in the Cloud Layer decomposition a subdomain with its bounded context and its model concepts related to the conceived SM; ($c_{2.1}$) Map this subdomain with its bounded context to a service or a set of services; ($c_{2.2}$) Develop these new services using BPMT; ($c_{2.3}$) Implement these new services using appropriate CT and PLT; (d) Test the System Cloud Layer (SCL).

6.6.1 Monolithic Architecture (MA)

The Cloud Layer design can follow the Service Oriented Architecture (SOA) principles, in which case its MA should contain at least two applications for processing, managing, and archiving clinical data. The first application must provide an Application Programming Interface (API) so that other applications in the Cloud and Fog layers can query and store data in a conventional Database (DB). The second application must provide a User Interface (UI) for health professionals to visualize and analyse patient clinical data and for system management. The development of these applications should be supported by BPMT, such as UML diagrams, and these applications should be implemented using appropriate PLT that take into account important issues, such as access control and data security. Also, appropriate CT, such as REST and GraphQL, should be considered for communication between these two applications and with MAp in the Fog Layer (see sections 3.4 and 3.5 of this thesis).

For SI to be addressed, a Semantic Model (SM), which uses SO, DB and a Triplestore (TS), should be conceived to be fully deployed in the

cloud. This deployment involves the development of a Semantic Module (SMo) for the Cloud Layer, which must play the role of a middleware, communicating with the applications, SO, DB and TS, all of them also in this layer (see section 4.1 of this thesis).

The main functionalities of the SMo include: transforming aggregated data into semantic data to be consumed by other components; enable automatic classification of patient data according to pre-defined rules; notify the patient and their health professional based on the context and obtained results; handle all requests and data coming from the Web; manage DB and TS; and allow customers to exchange data. The SMo development can also follow the SOA's principles and should supported by BPMT, and it should be implemented using appropriate PLT, such as Java SDK, Phyton and IntelliJ IDEA, and appropriate CT, such as Apache ActiveMQ message broker (see section 4.2 of this thesis).

6.6.2 Microservices Architecture (MSA)

The Cloud Layer design can follow the MSA's principles, in which case it is recommended to first decompose this layer into a set of services, and then obtain an MSA. This decomposition can be by business capability or by subdomain, and in the second case Domain Drive Development (DDD) should be employed (see sections 2.4.1 and 2.4.2 of this thesis).

If DDD is applied, a domain model composed of subdomains, each one being able to have its own domain model, must be defined. These subdomains are separated by their scopes, called bounded contexts. A domain model can be defined by identifying its domain concepts and their relationships, while a bounded context can be identified by grouping domain concepts that are closely related (see sections 5.1 and 5.2 of this thesis).

Then, each subdomain with its bounded context must be mapped to a service (or possibly a set of services), and one or more instances of this service (microservice) can be created. These services with other supporting services, such as proxies and brokers, constitute an MSA, and they must be isolated from each other using, for example, containers. The MSA development should be supported by BPMT, and the use of appropriate CT for communication between services and with Map in the Fog layer, such as REST API, HTTP JSON and Load Balancer, should also be considered. The MSA services should be implemented using appropriate PLT, such as Java and Spring framework (see sections 5.3 and 5.4 of this thesis).

For SI to be addressed, the Cloud Layer decomposition using DDD should include a subdomain with its bounded context and its domain

concepts related to the conceived SM. Then, this subdomain with its bounded context must be mapped to a service (or a set of services, possibly containing supporting services), which will use SO, DB and TS, and communicate with other MSA services, in order to deal with this type of interoperability. These new services should also be developed using BPMT and should also be implemented using appropriate CT and PLT.

6.7 Building and Validating System Design (BVSD)

The BVSD activity starts by integrating the Sensor, Fog and Cloud layers to build the System Artifact (SA), and then by testing SA using appropriate Testing Tools (TT). This activity continues by choosing the validation modalities that will be carried out in SA, among which the following stand out:

(a) To choose the most appropriate Communication Technologies (CTs) for SA, experiments should be carried, using appropriate Performance Testing Tools (PTT), to compare the use of different CTs in SA (see section 3.5 of this thesis);

(b) To assess the accuracy of the clinical data readings taken by SA, experiments should be carried out to compare these readings with the readings taken by appropriate Clinical Data Measurement Devices (CDMD) (see section 3.6 of this thesis);

(c) To verify whether SA satisfies the specified requirements, simulations of SA application scenarios should be carried out, using appropriate Simulation Tools (ST), where a patient is monitored remotely by their health professional (see section 4.4 of this thesis);

(d) To evaluate the SA scalability, simulations of SA application scenarios should be carried out, using appropriate STs, considering different groups of geographically distributed patients and their health professionals (see section 4.4.3 of this thesis); and

(e) To compare the performance of SAs that have been developed, for example in terms of throughput, service latency, energy consumption, network usage and total cost, experiments and/or simulations should be carried out, using appropriate PTTs and/or STs, of application scenarios of these SAs (see section 5.5 of this thesis).

Table 6.6 summarizes the inputs, outputs, resources, roles and the main tasks of this activity, and its development is illustrated in Chapters 3, 4 and 5 of this thesis.

Table 6-6 Inputs, Outputs, Resources, Roles and Tasks of BVSD Activity

Inputs	System Sensor Layer (SSL) System Fog Layer (SFL) System Cloud Layer (SCL)
Outputs	System Artifact (SA)
Resources	Simulation Tools (ST) Testing Tools (TT) Performance Testing Tools (PTT) Clinical Data Measurement Devices (CDMD)
Roles	Software Engineer (SE) Hardware Engineer (HE) Domain Expert (DE) Ontology Engineer (OE)
Tasks	(a) Integrate SSL, SFL and SCL to build SA; (b) Test SA; (c) Perform experiments to choose CTs for SA; (d) Perform experiments to assess the accuracy of SA clinical data readings; (e) Perform simulations of application scenarios to verify SA requirements; (f) Perform simulations of application scenarios to evaluate SA scalability; and (g) Perform experiments and/or simulations of application scenarios to compare SAs performances.

Chapter 7

Related Work

Many initiatives using IoT, Ubiquitous Computing and Cloud Computing technologies, some of them also using ontologies and/or microservices, for the development of systems in the Health domain have been proposed in the literature, which demonstrates the relevance of our research. This chapter discusses some of these initiatives comparing them to our work, and it is organized as follow: Section 7.1 presents some relevant works that propose RPM IoT systems for Hypertension and compares them with SBIoT-MPH; Section 7.2 presents some relevant works that propose interoperable IoT systems in Health and compares them with ODIoT-SMH; and Section 7.3 presents some relevant works that propose scalable and maintainable IoT systems in Health and compares them with MBIoT-SMH.

7.1 RPM IoT Systems for Hypertension

We carried out a literature review on RPM IoT systems for Hypertension, and the most relevant works are presented below and compared with SBIoT-MPH, the artifact that was developed in the first Design Cycle of this research.

A low-cost RPM IoT system for heart rate monitoring, which help tracking patients in real-time and can also be used in hospital and residential areas, is proposed in [136]. This system employs a photoplethysmography-based sensor and a piezoelectric sensor to continuously monitor the patient's pulse rate and resonant frequency. The collect data is sent over a wireless network to the cloud and can be used for further analysis by health professionals. *Figure 7-1* provides an overview of the block diagram of the proposed system.

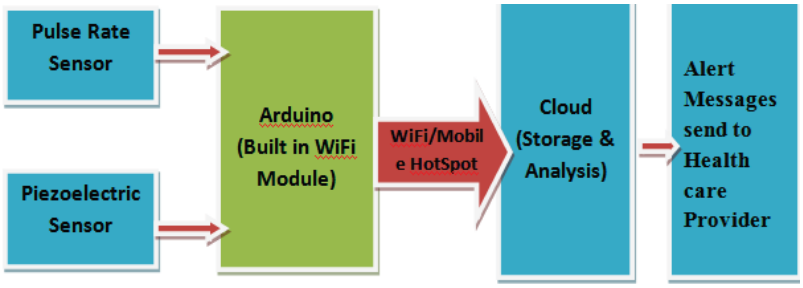


Figure 7-1 Block Diagram of the RPM IoT System for Heart Rate Monitoring (from [136])

An RPM IoT system for monitoring pre-hypertensive patients, which collect heart rate variability data, is proposed in [137]. The collected data are transmitted via the Message Queuing Telemetry Transport (MQTT) protocol to a server, where they are stored and can be accessed by health professionals through a Web application. The server's application analyses the received data, and when they reach a predefined risk range, alerts are sent via SMS to the health professionals. *Figure 7-2* provides an overview of the architecture of the proposed system.

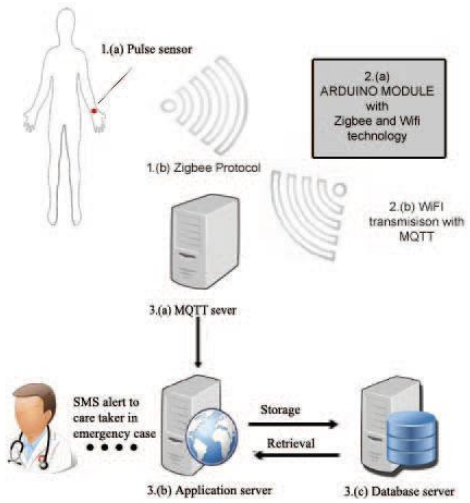


Figure 7-2 Architecture of the RPM IoT System for Monitoring Pre-Hypertensive Patients (from [137])

A Machine-To-Machine (M2M) RPM IoT system for monitoring hypertensive patients, which is able to provide medical recommendations aimed at maintaining blood pressure within normal levels, is proposed in [138]. Blood pressure data are read via a sphygmomanometer connected to a microcontroller, and then sent to a server, where they are analysed for providing medical recommendations on medication and medicine doses. Recommendations are forwarded to the responsible health professional for approval, and once approved they are sent and received by the patient via a

mobile application. A medicine box connected to a microcontroller is used to monitor whether patients are taking their medication correctly. *Figure 7-3* provides an overview of the logic flow of the proposed system.

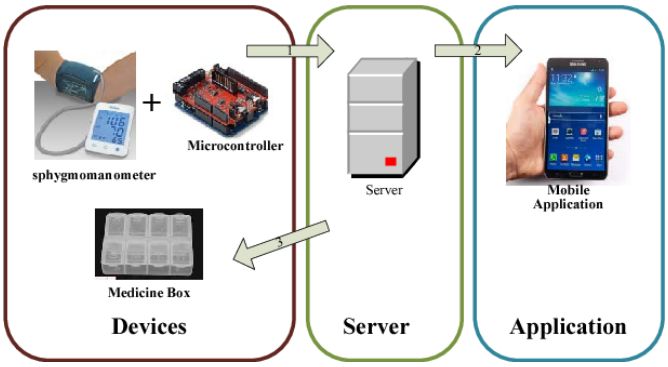


Figure 7-3 Logic Flow of the M2M RPM IoT System for Monitoring Hypertension (from [138])

An RPM IoT system for monitoring elderly hypertensive patients called oHealth, which obtains data collected by a blood pressure reading device connected to the patient's smartphone, and allows data related to diet, medication and exercise to be manually recorded, is proposed in [139]. All data records are sent to the cloud, where they are stored and analysed using the Multiclass Logistic Regression method. The analysed data can then be shared with the patient's physician and family members. *Figure 7-4* provides an overview of the oHealth architecture.

Figure 7-4 Architecture of the oHealth RPM IoT System (from [139])

An RPM IoT framework for monitoring blood pressure and other health parameters, which aims to identify the stage of Hypertension and to

predict hypertensive attacks using Artificial Neural Networks (ANN), is proposed in [140]. This framework is able to generate alerts related to blood pressure fluctuations that are sent to a mobile application on the patient's smartphone. The results of data analysis are forwarded to a subsystem located in the cloud, where they are stored and made available for further access by health professionals previously authorised by the patient. *Figure 7-5* provides an overview of the proposed framework.

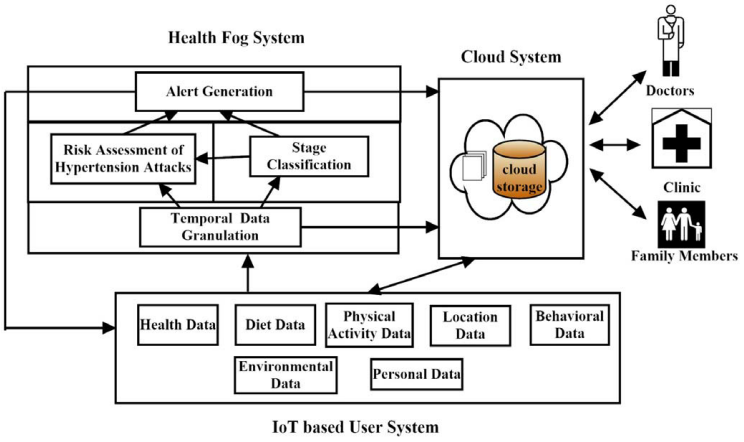


Figure 7-5 Overview of the RPM IoT Framework for Monitoring Hypertension (from [140])

All these initiatives aim at monitoring hypertensive patients. For this purpose, they employ different sensors or devices, most of them require the patients to perform measurements manually or cannot guarantee that the patient is resting during the clinical data readings. To obtain more accurate readings of a patient's blood pressure, the SBIoT-MPH's sensor platform continuously analyses the patient's movement and reads clinical data only when the patient is at rest. This platform was designed and implemented for not interfering with the patient's daily activities, and guarantees the confidentiality of the patient clinical data. In addition, the platform hardware was designed to support flexibility, so that other types of sensors could be incorporated to this platform, thus allowing the capture of other types of clinical data.

Regarding the SBIoT-MPH architecture, it was designed to enable the use of other sensors that can provide additional data for monitoring hypertension patients, or even to enable the monitoring of patients with other NCDs, such as Diabetes, Asthma and Obesity. This architecture also enables the processing of clinical data on the patient's mobile device, thus allowing to identify risk levels and to alert immediately the patient and their emergency contacts, by generating alert signals even if this device is

temporarily disconnected from Internet. The processing carried out on the mobile device also reduces the load for treatment and analysis in the cloud.

Furthermore, SBIoT-MPH was developed following the Design Science Methodology (DSM) as an artifact of the Design Science framework in a first Design Cycle. During the treatment validation phase of this cycle, in addition to carrying out the usual validation activities, we also carried out an experiment to compare REST and GraphQL in order to determine which these technologies would best fit into the SBIoT-MPH architecture.

7.2 Interoperable IoT Systems in Health

We carried out a literature review on interoperable IoT systems in Health, and the most relevant works are presented below and compared with ODIoT-SMH, the artifact that was developed in the second Design Cycle of this research.

An extended version of SSN ontology for dynamic IoT environments, called IoT-based large scale SOA (IoT-LSS), is proposed in [141]. According to the authors, IoT-LSS can be integrated with any other ontologies, and can support large-scale service composition, service clustering, and service discovery mechanism. As IoT-LSS entirely focuses on the SOA architecture, it supports interoperability, scalability, extendibility, flexibility, manageability, and heterogeneity among different entities. IoT-LSS is illustrated in [6] using a case study on Clinical Decision Support System (CDSS), where the elements of CDSS were mapped with IoT-LSS. Figure 7-6 illustrates the System class of IoT-LSS using OntoGraf plugin.

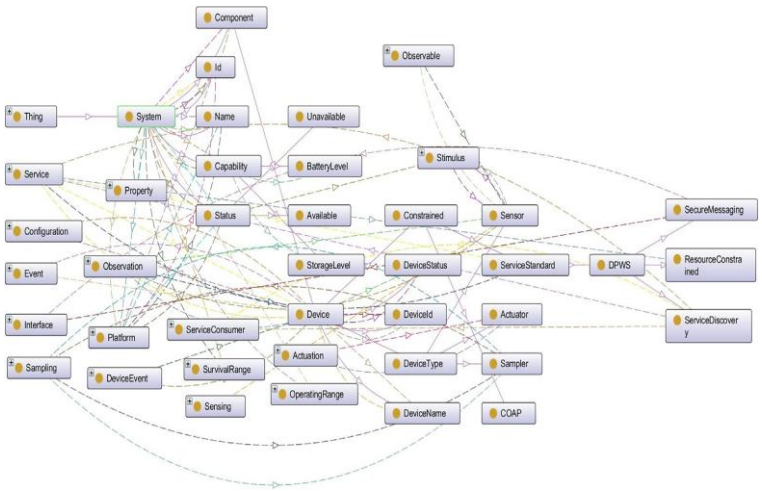


Figure 7-6 System Class of IoT-LSS Using OntoGraf Plugin (from [141])

Although this work is interesting from the point of view of developing ontologies aimed at IoT semantic interoperability, it does not incorporate a semantic reasoner to allow the automatic generation of large-scale services, and neither has the proposed ontology been used in a realistic system such as the ODIoT-SMH.

A framework for the Internet of Medical Things (IoMT), aiming to perform data interoperability where clinical data originates, is proposed in [142]. This framework, called Medical Data Interoperability through Collaboration (MeDIC), employs probing and translating agents at the network edge to allow IoMT devices to share their resources for data translations, thus minimizing cloud access. According to the authors, MeDIC is distributed, scalable, and extendable to other IoMT applications, and will be extended to support protocol and semantics compatibility. *Figure 7-7* provides an overview of the MeDIC framework.

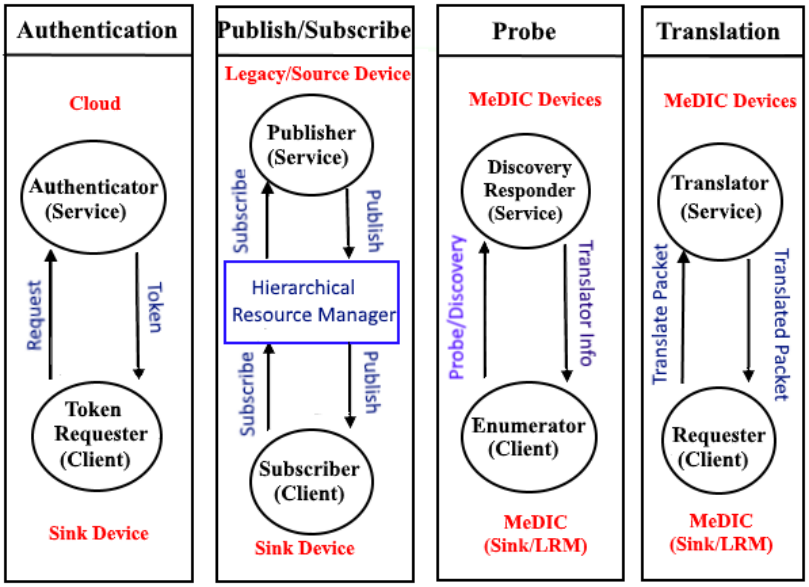


Figure 7-7 Overview of the MeDIC Framework (from [142])

The main contribution of this work is the proposition of a framework in which medical devices collaborate to enrich data and translate incompatible data formats, allowing in this way data interoperability at the network edge. For this purpose, it employs agents in this layer and not ontologies in the cloud as is the case with ODIoT-SMH.

An Ontology-based RPM IoT system called Do-Care for monitoring outdoor and indoor patients suffering from chronic diseases is proposed in [143]. The main contributions of this work are: a modular and extended ontology that integrates the ICNP medicine ontology, SSN/SOAS sensor

network ontology, and FOAF personal profile ontology; and a decision-making approach based on a dynamic set of personalized SWL rules, which considers objective and subjective knowledge to determine the patient's health condition. As future work, the authors intend to integrate the Decision Tree Learner algorithm into Do-Care for efficient disease prediction, and for providing additional support to physicians in preventive medication recommendations. *Figure 7-8* shows the Do-Care architecture.

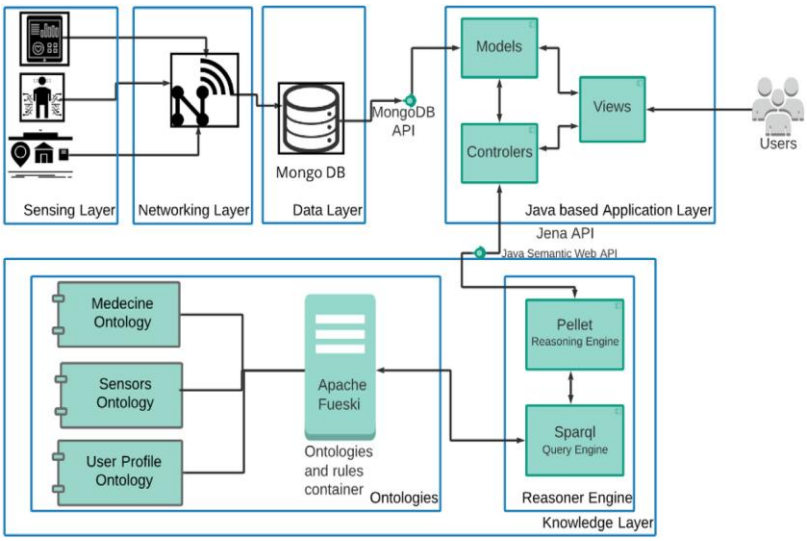


Figure 7-8 Architecture of the Do-Care System (from [143])

Although this work addresses an RPM IoT system for monitoring patients with chronic diseases, it focuses on the development of ontologies and not on the RPM IoT system itself as our work. Furthermore, Do-Care uses a particular five-layer architecture, where the primary entity is a Knowledge Layer that contains the ontologies and a reasoner engine, and not the three-layer architecture normally employed for the development of RPM IoT systems as is the case of ODIoT-SMH.

An eHealth platform called inCASA, aiming to integrate vital signs monitoring (Telehealth) with behavioural analysis based on home care sensors monitoring (Telecare), is proposed in [144]. This platform allows the deployment of services to follow-up the health evaluation of elderly, based on monitoring a set of parameters per disease, and to profile elderly's habits and diagnose deviations from their usual activities. inCASA uses a Service Oriented Architecture (SOA) middleware that collects data from heterogeneous Telecare and Telehealth gateways, and provides the upper layers with a unified, standards-compliant message, thus enabling an integrated view of this data and alerts in a backend Web portal, which is

accessed by health professionals and caregivers. *Figure 7-9* shows the inCASA architecture.

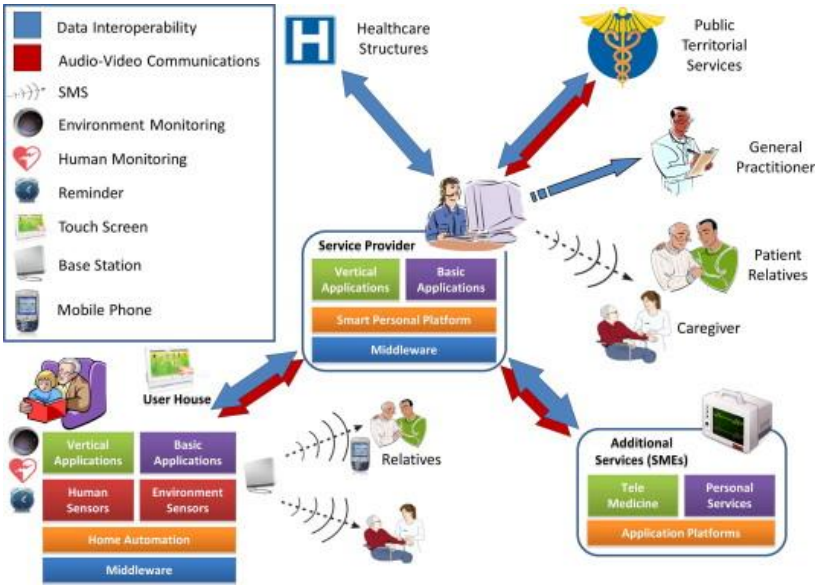


Figure 7-9 Architecture of the inCASA Platform (from [144])

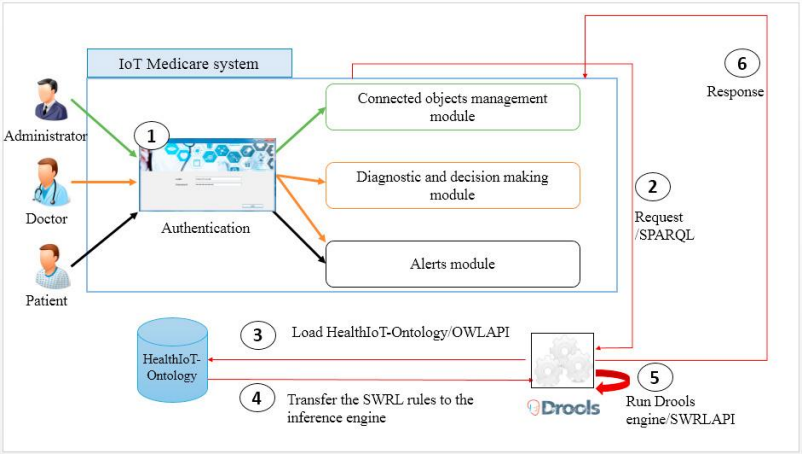
Although this work shows how a remote monitoring platform may achieve Telecare and Telehealth interoperability at a service level, it does not employ ontologies like ODIoT-SMH. It employs a SOA that relies on the LinkSmart Middleware, which receives measurements from proprietary Telehealth and Telecare gateways and transforms them into Health Level 7 (HL7) compliant data.

A HealthIoT ontology for the semantic representation of connected medical objects and their data, and an IoT Medicare system that uses this ontology are proposed in [145]. The development of the HealthIoT ontology was divided in four stages: data collection and pre-processing, knowledge extraction, semantic knowledge modelling, and semantic reasoning. The IoT Medicare system was organized into three main modules: connected object management, diagnosis and decision making, and alerts. This system supports health professionals in analysing the obtained vital signs and providing an appropriate service for their patients. *Figure 7-10* shows the IoT Medicare architecture.

Although this work has similarities with ours, the development of the HealthIoT ontology was not carried out following a set of guidelines for IoT ontology engineering, covering best practices and methodologies and driven by a set of Competency Questions (CQs), like the development of the ODIoT-SMH ontology. Regarding the IoT Medicare system, it appears

to be in the early stages of development, since its modules are briefly described in [136], as well as the validation carried out.

Figure 7-10 Architecture of the IoT Medicare System (from [145])



All these initiatives, including ODIoT-SMH, aim to develop interoperable IoT systems in Health. However, ODIoT-SMH was developed following DSM as an artifact of the Design Science framework in a second Design Cycle, and its development was triggered by a validated treatment of SBIoT-MPH, which allowed the reuse of components of this system.

7.3 Scalable and Maintainable IoT Systems in Health

We carried out a literature review on scalable and maintainable IoT systems in Health, and the most relevant works are presented below and compared with MBIoT-SMH, the artifact that was developed in the third Design Cycle of this research.

An MSA-based RPM IoT system for sleep monitoring is reported in [113]. All the participating microservices are unaware of the existence of other services and only consume/publish events, which are statements about the world depicting an action or notifying about data changes. A sensor gateway is introduced that buffers the sensor data and publishes it to the message broker in the form of events. A data persistence service consumes the events and stores the data in a time-series database to achieve high throughput and low read latencies. For validation, large data flows from Fog Layer to the Cloud Layer are simulated, and empirical analysis of metrics comparing monolithic and MSA-based systems are performed. *Figure 7-11* shows the architecture of this RPM IoT system.

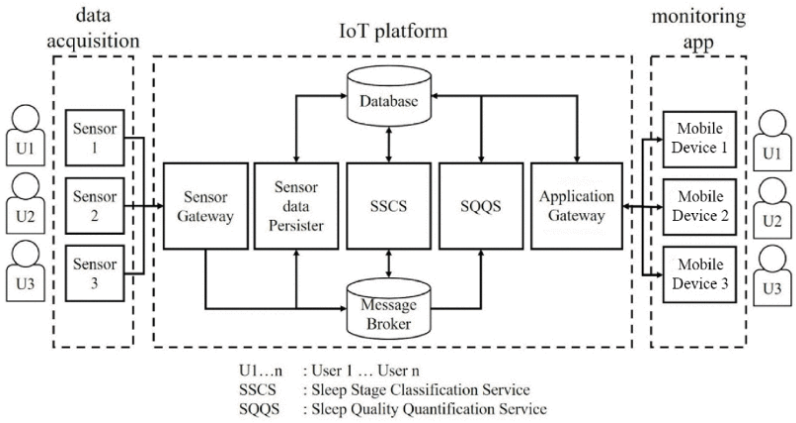


Figure 7-11 IoT Architecture of an RPM IoT System for Sleep Monitoring (from [113])

A layered architecture that enables the development of a near-real-time, multi-tenant RPM IoT system for monitoring chronic metabolic disorders is proposed in [114]. One of the main advantages of the system is that it uses an abstraction layer called Technology Integration Adapters (TIA), which allow the integration of heterogeneous IoT devices without changes that span the whole system. The developed artifact leverages an MSA and is designed to be deployed in the cloud. Furthermore, to address scalability and maintainability, the authors created an Inbound Pipeline microservice that receives the IoT sensor data and publishes it to a message broker. *Figure 7-12* shows the architecture of this RPM IoT system.

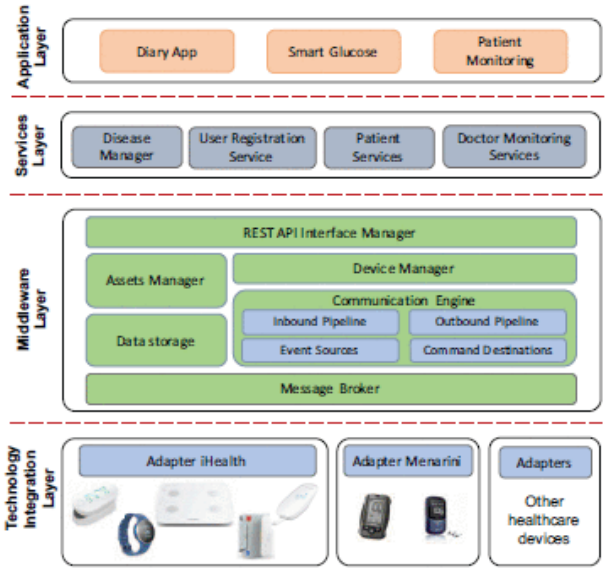


Figure 7-12 IoT Architecture of an RPM IoT System for Monitoring Chronic Metabolic Disorders (from [114])

An MSA-based RPM IoT system to send patient's data from Intensive Care Unit (ICU) to doctors and guardians is proposed in [115]. The system includes five microservices that leverage a variety of communication protocols like HTTP, MQTT and Web sockets. It has a general service built using the Django framework that handles user authentication and authorisation, enables chat capabilities with medical practitioners and provides patient data over HTTP protocol, whereas other services include MQTT broker, patient data, reporting and timeseries data ingestion services. *Figure 7-13* shows the MSA of this RPM IoT system.

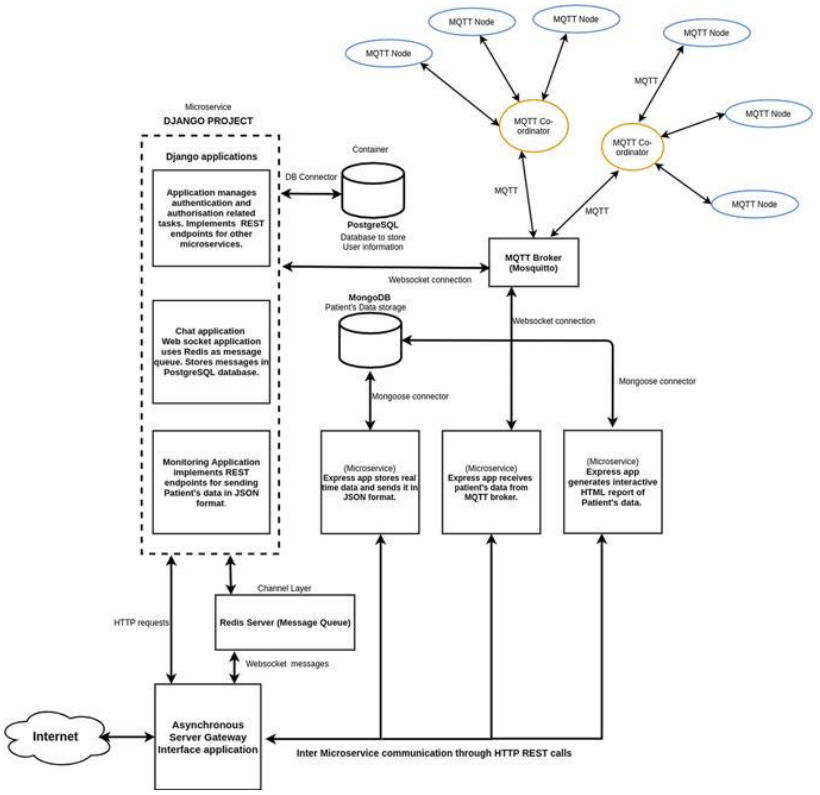


Figure 7-13 MSA of an RPM IoT System to Send Patient's Data from ICU to Doctors and Guardians (from [115])

A domain-specific architecture of a non-invasive RPM Internet of Health Things (IoHT) system for monitoring elderly is proposed in [146]. It employs a Raspberry PI 4 micro-computer to simulate IoHT devices generating pulse, body temperature, blood oxygen level, and blood pressure data as well as mimicking environmental, motion and wearable sensors. The data is summarised and pre-processed at the Fog Layer, enabling real-time decision-making and data buffering. Also, it is composed of different microservices, such as advanced analytics,

behavioural monitoring, triggers and alerts, and patient data services. Although the work reported in [137] does not provide any results and implementation, it provides a solid theoretical framework and argumentation. *Figure 7-14* shows the architecture of this RPM IoHT system.

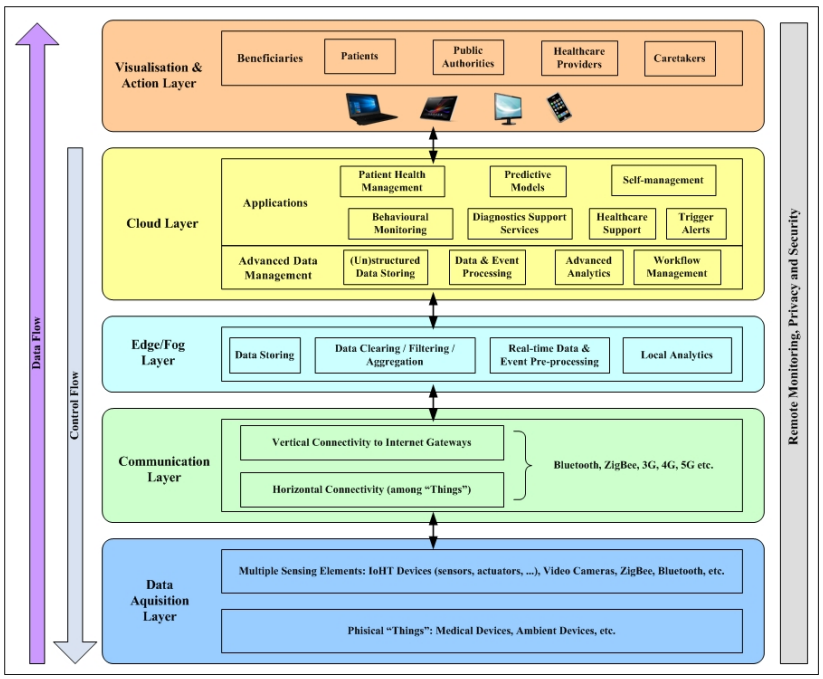


Figure 7-14 Architecture of an RPM IoHT System for Monitoring Elderly People (from [146])

An MSA-based system with a Machine Learning (ML) model to assess frailty in the elderly is proposed in [147]. This system collects sensory data from wearable devices while older adults perform Basic Activities of Daily Living (BADL), which involve a physical dimension (*e.g.*, toilet hygiene, functional mobility), in combination with Instrumental Activities of Daily Living (IADL), which involve a physical dimension but also a cognitive and a social dimension (*e.g.*, house cleaning, shopping). According to the authors, this system is accurate, flexible, non-intrusive, ecological, cost-effective and time-saving, and can help health professionals automatically detect frailty. *Figure 7-10* shows the MSA of this system.

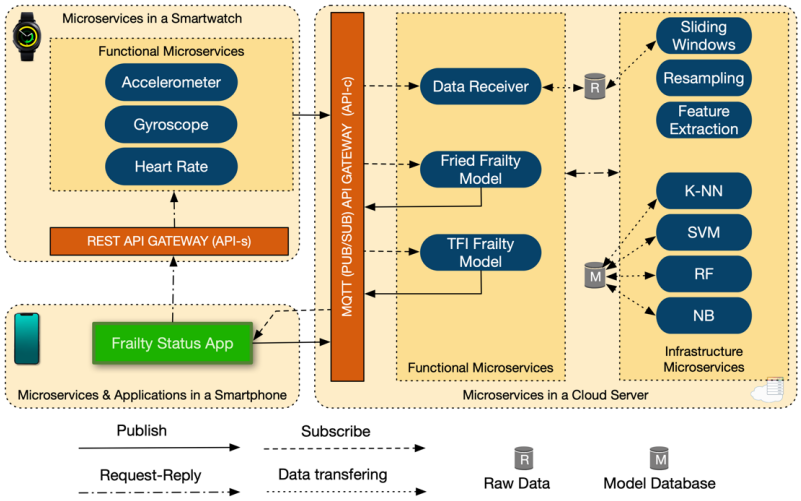


Figure 7-15 MSA-Based System to Assess Frailty in Elderly People (from [147])

All these initiatives, including MBIoT-SMH, aim to develop scalable and maintainable RPM systems in Health focusing on the use of microservices. However, MBIoT-SMH was developed following DSM as an artifact of the Design Science framework in a third Design Cycle, and its development, like that of ODIoT-SMH, was also triggered by a validated treatment of SBIoT-MPH, which allowed the reuse of components of the Sensor and Cloud layers of this system.

Furthermore, we redesign the Cloud Layer to obtain an MSA for this layer, first applying DDD to create a domain model composed of subdomains and their bounded contexts, and then designing the MSA based on this decomposition and following the MSA's principles. We also carried out an experiment similar to the one performed in the first Design Cycle to compare REST and GraphQL, this time to determine which these technologies would best fit into the MBIoT-SMH architecture.

Chapter 8

Conclusion

This chapter concludes the work presented in this thesis, identifies topics that we recommend for future research, and is organized as follow: Section 8.1 presents some general remarks of this research; Section 8.2 elaborates on the most important contributions of this thesis; and Section 8.3 discusses directions for future research.

8.1 General Remarks

According to the World Health Organization (WHO), Noncommunicable diseases (NCD) are one of the biggest challenges in Health. For a more effective treatment of NCD patients, WHO recommends a periodic monitoring of the health conditions of these patients. Some NCDs, such as Hypertension, Diabetes, Asthma and Obesity, allow for continuous Remote Patient Monitoring (RPM), enabling the patients and their health professionals to actively exert disease control. Internet of Things (IoT), Ubiquitous Computing, and Cloud Computing can be used in combination to build RPM systems.

One of the big challenges in IoT is dealing with the huge amount of data produced by their things, in addition to dealing with the data heterogeneity, the varied capabilities of the things, the diverse offered services, and the different IoT platforms. Therefore, ensuring semantic interoperability when multiple and different IoT devices and system components must interoperate is one of the problems that needs extensive investigation in this area. In addition to generating large volumes of data that require relatively complex processing, IoT systems have become increasingly complex and hard to maintain. Furthermore, they require high scalability, availability and data throughput in the context of critical applications such as in the Health domain.

In this context, this research investigated how to design interoperable, scalable and maintainable RPM IoT systems for NCDs that can be monitored remotely. This investigation has followed the Design Science Methodology (DSM) by employing the Design Science framework, where three RPM IoT systems for the NCD Hypertension were designed as artifacts of three Design Cycles. Based on the experience gained from this investigation, this research also proposed an approach to be used mainly in the development of IoT systems for monitoring patients suffering from NCDs that can be monitored remotely, such as Hypertension, Diabetes, Asthma and Obesity.

8.2 Research Contributions

Section 1.4 presented the main research questions to be addressed in this thesis. The current section discusses the research contributions of this thesis, reflecting on the results obtained and describing how these research questions were addressed.

As the research questions of this thesis were split according to DSM into knowledge questions (KQ) and technical research questions (TQ), and the KQs were addressed through comprehensive and systematic literature reviews, in this section we will focus on describing how TQs were addressed.

TQ_1 *How to populate the layers of the chosen architectural model with components of RPM IoT systems?*

As our investigation targeted RPM, and the three-layer architecture (Sensor, Fog and Cloud) was proposed for monitoring patients and elderly, we adopted this architecture to structure the artifacts that were developed in our research.

Chapter 3 describes the design of the SBIoT-MPH artifact, an RPM IoT system for the NCD Hypertension developed in the first Design Cycle of this research, showing how the Sensor, Fog and Cloud layers of this artifact were populated with components, and showing how these components were developed. The main components of these layers are: the Sensor Platform in the Sensor Layer; the Mobile App application in the Fog Layer; and the API Server and Web App applications in the Cloud Layer. A major concern in the SBIoT-MPH design was employing off-the-shelf (OTS) components to keep the system affordable.

The Sensor and Fog layers components of the ODIoT-SMH and MBIoT-SMH artifacts remained the same as the SBIoT-MPH, and the

developments of the Cloud Layer components of these artifacts are described in Chapters 4 and 5, respectively.

TQ_2: *How to collect, process, and analyse clinical data of RPM IoT systems according to the chosen architectural model?*

As the current landscape of IoT platforms is broad, heterogeneous, and IoT architectures are characterized by a lack of standardization, we developed a sensor platform to acquire physiological signals from patients, and to send the corresponding clinical data to be processed and analysed in the upper architecture layers.

Section 3.2 describes the design of the Sensor Platform, which was built on a wristwatch-like wearable IoT device. The sensors of this platform capture the patient's vital signs blood pressure, heart rate and body temperature, these raw data are filtered, aggregated and converted to digital format on the platform itself, and the resulting clinical data are transmitted to the Fog Layer in accordance with the Bluetooth Low Energy (BLE) standard. Since the accuracy of the blood pressure measurements is affected by the patient's movement, this platform has a movement sensor that allows to read these data only when the patient is at rest.

Section 3.3 describes the design of the Mobile App application, which runs on the patient's mobile device. This application receives clinical data from the Sensor Platform, pre-processes, analyses and classifies this data according to health risk levels, so that it is properly presented to the patient and for the eventual generation of alert signals. This latter functionality has been assigned to the Fog Layer, rather than the Cloud Layer, to keep processing of delay-sensitive data closer to the edge network and IoT devices. Then, the pre-processed clinical data are transmitted to the Cloud Layer via Wi-Fi or 3G/4G networks.

Section 3.4 describes the design of the API Server and Web App applications, which are responsible for processing, managing, and archiving clinical data. These applications were deployed in the Cloud Layer so that they can request additional computing resources on demand whenever necessary and are accessible through the Internet. API Server provides a secure Application Programming Interface (API) so that Mobile App and Web App can store and retrieve system information, and employs the GraphQL query and data manipulation language. Web App provides a User Interface (UI) and is accessed via a Web browser, both by health professionals to view and analyse their patients' clinical data, and by an administrator to manage these professionals and patients.

TQ_3: *How to design a semantic model for RPM IoT systems according to the chosen architectural model?*

As IoT is intrinsically heterogeneous, both in terms of hardware and software at different levels, and it is necessary to deal with this heterogeneity to achieve interoperability, and one possible way is semantic interoperability, we developed a semantic model, based on devices and diseases ontologies, to be fully deployed in the Cloud Layer.

Chapter 4 describes the design of the ODIoT-SMH artifact, an RPM IoT system for the NCD Hypertension developed in the second Design Cycle of this research. Since this artifact reuses most of the SBIoT-MPH components of the Sensor and Fog layers and also the API Server and Web App applications of the Cloud Layer, this chapter focuses on showing how ODIoT-SMH handles semantic interoperability and how its semantic model and ontology were developed.

Section 4.1 describes the design of the ODIoT-SMH Semantic Model, which comprises ontologies to provide semantic annotations to the aggregated data, adapters to convert aggregated data into semantic data, a conventional database, and a triplestore. The aggregated data are converted to the RDF triples format with the help of ontologies.

Section 4.2 describes the implementation of the ODIoT-SMH Semantic Model carried out through a semantic module introduced in the Cloud Layer, playing the role of a middleware and communicating with the ODIoT-SMH applications of this layer, the triplestore and the ODIoT-SMH Ontology, all of them also in the Cloud Layer. ODIoT-SMH Semantic Module was designed following Service Oriented Architecture (SOA) principles, and it has seven main components that are described in this section.

Section 4.3 describes the design of the ODIoT-SMH Ontology, which was developed following a set of guidelines for IoT ontology engineering covering best practices and methodologies in order to reuse and extend existing ontologies. The main purpose and information requirements of the ODIoT-SMH Ontology, regarding the NCDs and the IoT domain, were described as Competency Questions (CQs). We used SNOMED-CT as it provides an accurate and deeper ontological analysis of Health domain terminology including NCDs, and SAREF4HAW as it can be used as an IoT reference model for electronic(e)/mobile(m)-Health applications. The scope of these CQs guided the selection and modularisation of these ontologies that were aligned, merged, extended and reused. Each one of these steps followed best practices based on the guidelines for IoT ontology engineering.

TQ_4: *How to design an MSA for RPM IoT systems according to the chosen architectural model?*

As IoT systems have become increasingly complex and hard to maintain, and they require high scalability, availability and data throughput in the context of critical applications such as in the Health domain, we developed a Microservices Architecture (MSA) for the Cloud Layer.

Chapter 5 describes the design of the MBIoT-SMH artifact, an RPM IoT system for the NCD Hypertension developed in the third Design Cycle of this research. Since this artifact reuses the SBIoT-MPH components of the Sensor and Fog layers, this chapter focuses on showing how the Cloud Layer was redesigned using microservices to improve system scalability and maintainability, and to increase sensor data throughput.

Section 5.2 describes how Domain Driven Design (DDD) was applied to decompose the system by creating a domain model composed of subdomains and their bounded contexts, with each subdomain being able to have its own domain model and being a good candidate for a service or set of services in the MSA. By splitting the domain into subdomains with their bounded contexts we obtained a cohesive design. The domain model was defined by identifying the domain concepts and their relationships, and the bounded contexts were identified by grouping domain concepts that are strongly related.

Section 5.3 describes the design of the MSA for the MBIoT-SMH Cloud Layer, which followed the general principle 'code that changes together must stay together' and the Database-Per-Service pattern. The MBIoT-SMH Cloud Layer consists of the Patient, Health Professional, Identity and Notification services, and a Sensor Data Ingestion (SDI) Proxy component, which acts as a gateway for the Fog Layer to forward clinical data to the Cloud Layer. Each of these services has its own database, and maintains a connection to a publish/subscribe message broker (Kafka Cluster) and runs an isolated process space (Docker container). The Sensor and Fog layers remain the same of for SBIoT-MPH, but the differences start with the interactions between the Fog and Cloud layers. In SBIoT-MPH the mobile device forwards the data to the API Server using the GraphQL protocol, while in MBIoT-SMH the mobile device uses a REST API and transfers the data via HTTP JSON to the SDI Proxy. All calls to the services are routed by the Load Balancer, which can select the least congested service instance (microservice) through service discovery.

In addition to allowing us to answer these TQs, the experience gained from developing these artifacts in the three Design Cycles of this research, also allowed us to propose an approach for design interoperable, scalable and maintainable RPM IoT systems, to be used primarily in the development of IoT systems for monitoring patients suffering from NCDs that can be monitored remotely, such as Hypertension, Diabetes, Asthma and Obesity. This approach, depicted as a process and using a diagrammatic notation similar to the Structured Analysis and Design Technique (SADT), is described in Chapter 6 and is another research contribution of this thesis.

8.3 Directions for Future Research

As the Sensor Platform prototype was built employing OTS components to keep it affordable, we began to redesign this platform using Arduino and better precision sensors, in order to achieve better accuracy in clinical data measurements carried out by this platform, thus meeting the standards required in Health. As the mobile IoT device where the platform was embedded presented ergonometric problems, we also began to redesign this device based on existing commercial devices.

Aiming to enable semantic interoperability in ODIoT-SMH, a semantic model for the Cloud Layer was designed to incorporate reasoning through the use of disease and device ontologies. To this end, a semantic module was introduced in the Cloud Layer, playing the role of a middleware, and communicating with the ODIoT-SMH applications, the RDF triplestore database and the ontologies, all of them also in the Cloud Layer. A problem related to our databases is that ODIoT-SMH has duplicated the persisted data into two different formats: relational and RDF triplestore. While duplicated data promotes resilience, it also requires mechanisms to guarantee its integrity. For ODIoT-SMH, this mechanism may be more complex since the databases of this system use different technologies. We intend to investigate this problem by simulating different scenarios to check the viability to keep more than one database.

Although we followed the best practices and methodologies for IoT ontology engineering recommended by the Semantic Web community, we cannot guarantee 100% the consistency and coherence of the ODIoT-SMH Ontology. The OntoDebug/Protégé plugins helped us to identify some inconsistencies but it is limited by the OWL language expression. A solution for this problem may be to apply a systematic ontology development approach, such as the Systematic Approach for Building Ontologies (SABiO) [148], which consists of the following phases:

(a) identify the purpose and requirements of the ontology by defining Competency Questions (CQ) and modularisation;
(b) perform ontology capture and formalization;
(c) generates an operational artifact by designing and implementing the ontology; and
(d) test the artifact

In the ontology capture, a foundational ontology is selected to carry out knowledge acquisition, which in the ODIoT-SMH ontology comprises identifying NCDs and e/m-Health application components. Then, concepts are selected from an existing ontology domain to be reused and extended in the foundation ontology. In the formalization, a conceptual model is developed by identifying and organising relevant concepts and relations with a graphical model. In this phase, the domain ontology is grounded in a foundational ontology, by defining the conceptual model using concepts of the foundational ontology.

The first and second phases were already achieved in the ODIoT-SMH ontology development, and the next step is to formalise this ontology and ground it into a foundation ontology. For this purpose, we intend to use OntoUM [132], an ontologically well-founded language for Ontology-driven Conceptual Modelling, built as a UML extension based on the Unified Foundational Ontology (UFO). We intend to use also OntoUML Lightweight Editor (OLED) [133], an environment for the development, evaluation and implementation of domain ontologies in OntoUM, which includes model verification, model validation, and model transformation to languages such as OWL for supporting computationally efficient automated reasoning.

In the literature, several fog-based approaches have been reported that provide solutions for heterogeneity problems, but semantic approaches are usually implemented in the cloud. As our system allow pre-processing of delay-sensitive data in the Fog Layer that is close to the network edge, while the delay-tolerant data are processed in the Cloud Layer, as future work we intend to investigate and develop a new artifact in a fourth Design Cycle with a semantic model partially deployed in the Fog Layer and partially in the Cloud Layer. *Figure 8-1* provides an overview of the architecture of this artifact with its main functionalities distributed across its layers.

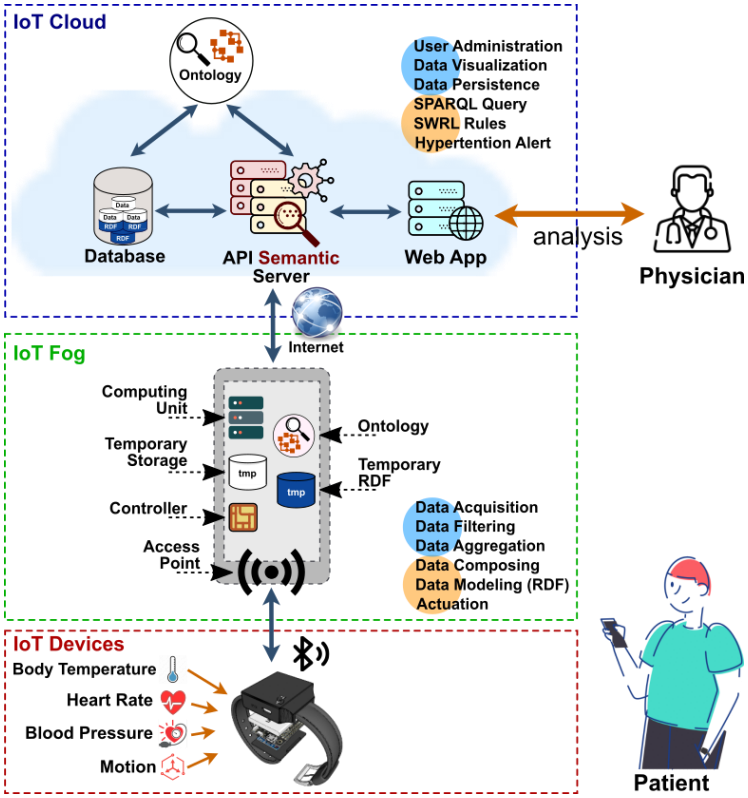


Figure 8-1 System Architecture with Semantic Model Deployed in the Fog and Cloud Layers

As future work, we intend to extend the Sensor Platform for allowing to collect other types of clinical data, and to adapt the system applications for monitoring patients with other types of NCDs, such as Diabetes, Asthma and Obesity. *Figure 8-2* illustrates a comprehensive usage scenario for this adapted/extended system, where different groups of patients, geographically distributed, with different types of NCDs are monitored by their respective health professionals, also geographically distributed.

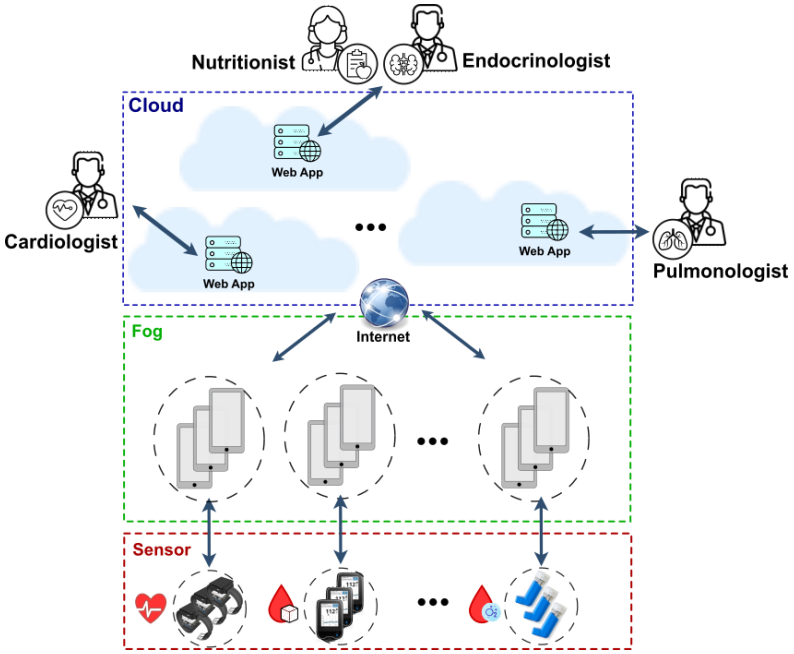


Figure 8-2 Usage Scenario for Monitoring Geographically Distributed Patients with

In addition to the monitoring of patients with NCDs, WHO also recommends the practice of physical activity for preventing NCDs. In [149], a systematic review on the use of IoT in physical activity recognition and monitoring is reported. Since the scenarios for monitoring physical activity presented in this study are similar to the ones for monitoring NCDs patients, as future work we also intend to investigate how to adapt/extend our system to be used for this purpose. For example, the public health system in Brazil, known as "Sistema Único de Saúde (SUS)", has a program linked to Family Health Units (FHUs), where Physical Activity Groups (PAGs) carry out regularly Physical Activity Sessions (PASs). Our system could be adapted/extended to monitor the PAGs participants during the PASs in the FHUs of SUS. For this purpose, the Sensor Platform needs to be reconfigured, for example to capture blood pressure and heart rate data of PAGs participants when they are practicing physical activity, and must also be extended to support other types of sensors (*e.g.*, gyroscope, spirometer) to capture relevant data for monitoring physical activity. It will also be necessary to extend the system applications for handling the data coming from these sensors and support this new monitoring process. Such programs generate a huge volume of data that can be analysed with a view to establishing public policies on physical activity practices and also evaluating, in the long term, how effective the practice of such activities is in preventing NCDs. *Figure 8-3*

provides an overview of the architecture of this adapted/extended system with its main functionalities distributed across its layers.

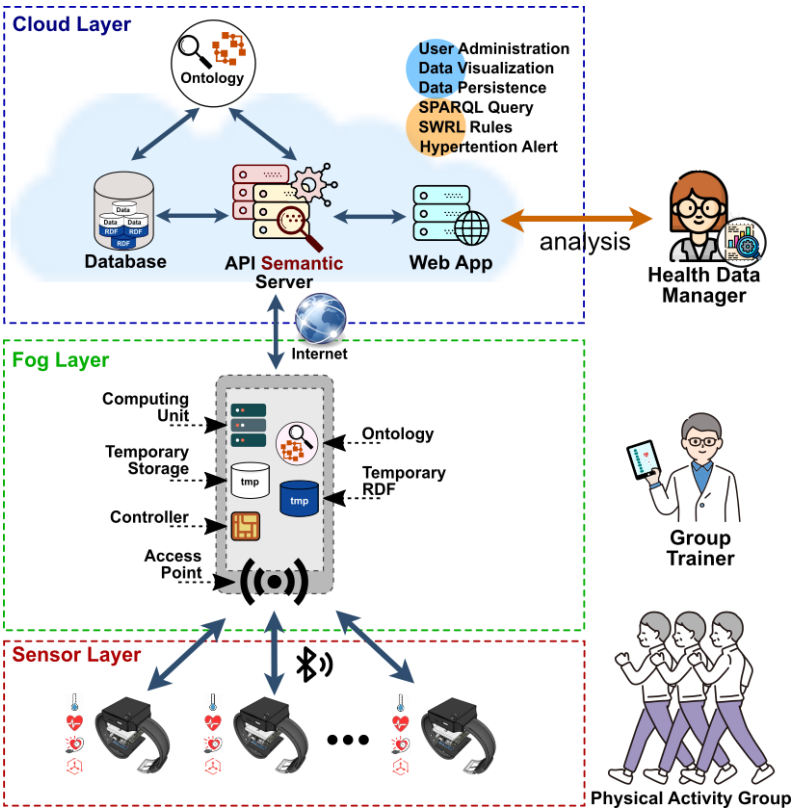


Figure 8-3 System Architecture for Monitoring Physical Activity

Companies, such as Netflix, Amazon, Spotify and Paypal, have used microservices in the development of their mobile applications to ease up the addition of new requirements, functionalities and storage to the database. Last but not least, to facilitate all necessary adaptations and extensions to make up our system more comprehensive, we intend to investigate the use of Microservices Architecture (MSA) also in its Fog Layer, thus facilitating the construction, testing and maintenance of the entire system.

5. References

[1] Arredondo, A.; Alives, R. *Costs and Epidemiological Changes of Chronic Diseases: Implications and Challenges for Health Systems*. PLOS ONE, Volume 10, Issue 3, 12 pp., March 2015.

[2] Varshney, U. *Pervasive Healthcare: Applications, Challenges and Wireless Solutions*. Communications of the Association for Information Systems, Volume 16, Article 3, pp. 57-72, July 2005.

[3] Varshney, U. *Pervasive Healthcare Computing: EMR/EHR, Wireless and Health Monitoring*. Springer, 285 pp., 2009.

[4] Bardram, J.E; Mihailidis, A.; Wan, D. *Pervasive Computing in Healthcare*. CRC Press, 336 pp., 2007.

[5] Boye, N. *Pervasive Healthcare: Problems and Potentials*. In: Human, Social, and Organizational Aspects of Health Information Systems, Kushniruk, A.W and Borycki, E.M. (eds.), Medical Information Science Reference, IGI Global, pp. 84-101, 2008.

[6] WHO. *Noncommunicable diseases*. World Health Organization, 2023. https://www.who.int/news-room/fact-sheets/detail/noncommunicable-diseases (accessed 20 October 2023).

[7] WHO. *Noncommunicable Diseases Progress Monitor 2020*. World Health Organization, 2020. https://www.who.int/publications/i/item/9789240000490 (accessed 20 October 2023).

[8] WHO. *Noncommunicable Diseases Country Profiles 2018*. World Health Organization, 2018. https://apps.who.int/iris/handle/10665/274512 (accessed 06 June 2023).

[9] Malta, D.C. et al. *Progress with the Strategic Action Plan for Tackling Chronic Non-Communicable Diseases in Brazil, 2011-2015*. Epidemiologia e Serviços de Saúde, Volume 25, Issue 2, pp. 373-390, 2016.

[10] WHO. *A global brief on hypertension: silent killer, global public health crisis: World Health Day 2013*. World Health Organization, 2013. https://apps.who.int/iris/handle/10665/79059 (accessed 06 June 2023).

[11] WHO. *Global NCD target: reduce high blood pressure*. World Health Organization, 2016. https://www.who.int/publications/i/item/global-ncd-target-reduce-high-blood-pressure (accessed 06 June 2023).

[12] Kjeldsen, S.E. *Hypertension and cardiovascular risk: general aspects*. Pharmacological Research, Elsevier, Volume 129, pp. 95-99, 2018.

[13] Sandi, G.; Nugraha, I.G.B.B.; Supangkat, S.H. *Mobile health monitoring and consultation to support hypertension treat*ment. In: International Conference on ICT for Smart Society, pp. 1-5, 2013.

[14] Husain, M.S. et al. *Pervasive Healthcare - A Compendium of Critical Factors for Success*. EAI/Springer Innovations in Communication and Computing, Springer, 379 pp., 2022.

[15] Kelly, J.T. et al. *The Internet of Things: Impact and Implications for Health Care Delivery*. Journal of Medical Internet Research, Volume 22, Issue 11, 11 pp., November 2020.

[16] Salih, K.O.M. et al. *A Comprehensive Survey on the Internet of Things with the Industrial Marketplace*. Sensors, MDPI Open Access Journals Volume 22, Issue 3, pp. 1-30, 2022.

[17] Gawanmeh, A.; Al-Karaki, J.N. *Disruptive Technologies for Disruptive Innovations: Challenges and Opportunities*. In: Proceedings of 18th International Conference on Information Technology: New Generations (ITNG 2021), Advances in Intelligent Systems and Computing, Springer, Volume 1346, Chapter 55, pp. 427-434, 2021.

[18] Noura, M.; Atiquzzaman, M.; Gaedke, M. *Interoperability in Internet of Things: Taxonomies and Open Challenges*. Mobile Networks and Applications, Springer, Volume 24, pp. 796-809, 2018.

[19] Lopes de Souza, P; Lopes de Souza, W; Ciferri, R.R. *Semantic Interoperability in the Internet of Things: A Systematic Literature Review*. In: ITNG 2022 Proceedings of 19th International Conference on Information Technology: New Generations. Advances in Intelligent Systems and Computing, Elsevier, Volume 1421, pp. 333-340, 2022.

[20] Petrakis, E.G.M.; Sotiriadis, S.; Soultanopoulos, T.; Renta, P.T.; Buyya, R.; Bessis, N., *Internet of Things as a Service (iTaaS): Challenges*

and solutions for management of sensor data on the cloud and the fog. Internet Things, Volume 3-4, pp. 156-174, 2018.

[21] Newman, S. *Building Microservices: Designing Fine-Grained Systems*. Second Edition, O'Reilly Media, Inc., 615 pp., 2021.

[22] Mehta, R.; Sahni, J.; Khanna, K. *Internet of Things: Vision, Applications and Challenges*. Procedia Computer Science, Elsevier, Volume 132, pp. 1263-1269, 2018.

[23] Bhuiyan, M.N. et al. *Internet of Things (IoT): A Review of Its Enabling Technologies in Healthcare Applications, Standards Protocols, Security, and Market Opportunities*. IEEE Internet of Things Journal, IEEE, Volume 8, Issue 13, pp. 10474-10498, July 2021.

[24] Qadri, Y. A. et al. *The Future of Healthcare Internet of Things: A Survey of Emerging Technologies*. IEEE Communications Surveys & Tutorials, IEEE, Volume 22, Issue 2, pp. 1121-1167, 2020.

[25] Chai, S. et al. *Wireless Sensor Networks*. Springer, 291 pp., 2020.

[26] Balas, V.E.; Pal, S. (eds.). *Healthcare Paradigms in the Internet of Things Ecosystem*. Academic Press, Elsevier, 397 pp., 2021.

[27] Butzin, B.; Golatowski, F.; Timmermann, D. *Microservices Approach for the Internet of Things*. In: 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), pp. 1-6, 2016.

[28] Kristoffersson, A.; Lindén, M. *Wearable Sensors for Monitoring and Preventing Noncommunicable Diseases: A Systematic Review*. Information, MDPI, Volume 11, Issue 11, 31 pp., November 2020.

[29] Al-Siddiq, W. *Why IoT might prove to be the missing link in chronic disease management*. HIT Leaders and News, 2023. https://us.hitleaders.news/why-iot-might-prove-to-be-the-missing-link-in-chronic-disease-management/ (accessed 08 June 2023).

[30] Wieringa, R.J. *Design Science Methodology for Information Systems and Software Engineering*. Springer, 327 pp., 2014.

[31] Rodrigues, R.J.S. *SBIdC-MPH: Sistema Baseado em Internet das Coisas para o Monitoramento de Pacientes com Hipertensão*. MSc dissertation in Portuguese, Graduate Program in Computer Science (PPG-CC), Federal University of São Carlos (UFSCar), 110 pp., 2022.

[32] Lopes de Souza, P. et al. *Ontology-Driven IoT System for Monitoring Hypertension*. In: Proceedings of the 25th International Conference on

Enterprise Information Systems (ICEIS 2023), SCITEPRESS - Science and Technology Publications Lda, Volume 1, pp. 757-767, 2023.

[33] W3C. *Resource Description Framework (RDF)*. W3C Recommendation, RDF Working Group, 2014. https://www.w3.org/RDF/ (accessed 05 October 2023).

[34] Evans, E. Domain-Driven Design: *Tackling Complexity in the Heart of Software*. Addison-Wesley, 382 pp., 2004.

[35] European Commission. *The General Data Protection Regulation (GDPR)*. Data protection in the EU. https://commission.europa.eu/law/law-topic/data-protection/data-protection-eu_en (accessed 20 June 2023).

[36] U.S. Department of Health and Human Services. *HIPAA & Your Health Rights*. https://www.hhs.gov/programs/hipaa/index.html (accessed 20 June 2023).

[37] Karimi, M.; Brazier, J. *Health, Health-Related Quality of Life, and Quality of Life: What is the Difference?*. Pharmacoeconomics, Springer, Volume 34, Issue 7, pp. 645-649, 2016.

[38] Wang, X et al. *Leveraging Mobile Cloud for Telemedicine: A Performance Study in Medical Monitoring*. In: Proceedings of 2013 39th Annual Northeast Bioengineering Conference, pp. 49-50, 2013.

[39] Huzooree, G.; Kumar Khedo, K.; Joonas, N. *Pervasive mobile healthcare systems for chronic disease monitoring*. Health Informatics Journal, Sage Journals, Volume 25, Issue 2, pp. 267-291, 2019.

[40] UN. *Transforming our World: the 2030 Agenda for Sustainable Development*. United Nations, 2015. https://sdgs.un.org/2030agenda (accessed 25 October 2023).

[41] WHO. *Global Action Plan for the Prevention and Control of Noncommunicable Diseases 2013-2020*. World Health Organization, 2013. https://www.who.int/publications/i/item/9789241506236 (accessed 25 October 2023).

[42] WHO. *Implementation Roadmap 2023-2030 for the Global Action Plan for the Prevention and Control of NCDs 2013-2030*. World Health Organization, 2023. https://www.who.int/teams/noncommunicable-diseases/governance/roadmap (accessed 25 October 2023).

[43] McKenney, J.M. *Patient Education and Compliance: How to Make It Cost-Effective*. Value in Health, Elsevier, Volume 1, Issue 4, pp. 212-215, 1998.

[44] American Heart Association. *Understanding Blood Pressure Readings*. https://www.heart.org/en/health-topics/high-blood-pressure/understanding-blood-pressure-readings (accessed 05 July 2023).

[45] Whelton, P.K. et al. *2017 ACC/AHA/AAPA/ABC/ACPM/AGS/APhA/ASH/ASPC/NMA/PCNA Guideline for the Prevention, Detection, Evaluation, and Management of High Blood Pressure in Adults: A Report of the American College of Cardiology/American Heart Association Task Force on Clinical Practice Guidelines*. Hypertension, Volume 71, Issue 6, pp. e13-e115, 2018.

[46] Weiser, M. *The Computer for the 21st Century*. Scientific American, Volume 265, Issue 3, pp. 94-105, 1991.

[47] Adelstein, F. et al. *Fundamentals of Mobile and Pervasive Computing*. McGraw-Hill, 422 pp., 2005.

[48] Lyytinen, K.; Yoo, Y. *Issues and Challenges in Ubiquitous Computing*. Communications of the ACM, Volume 45, Issue 12, pp. 62-65, 2002.

[49] Singh, A.; Payal, A.; Bharti, S. *A walkthrough of the emerging IoT paradigm: Visualizing inside functionalities, key features, and open issues*. Journal of Network and Computer Applications, Elsevier, Volume 143, pp. 111-151, 2019.

[50] Ashton, K. *That 'internet of things' thing*. RFID journal, Hauppauge, Volume 22, Issue 7, pp. 97-114, 2009.

[51] Atzori, L.; Iera, A.; Morabito, G. *The Internet of Things: A survey*. Computer Networks, Elsevier, Volume 54, pp. 2787-2805, 2010.

[52] Gubbi, J. et al. *Internet of Things (IoT): A vision, architectural elements, and future directions*. Future Generation Computer Systems, Elsevier, Volume 29, pp. 1645-1660, 2013.

[53] NIC. *Disruptive Civil Technologies - Six Technologies with Potential Impacts on US Interests Out to 2025*. National Intelligence Council (NIC), Conference Report CR 2008-07, 2008. https://irp.fas.org/nic/disruptive.pdf (accessed 23 November 2023).

[54] Della Mea, V. *What is e-Health (2): The death of telemedicine?* Journal of Medical Internet Research, JMIR Publications, Volume 3, Issue 2, pp. 1-2, 2001.

[55] Nacinovich, M. *Defining mHealth*. Journal of Communication in Healthcare, Taylor & Francis Group, Volume 4, Issue 1, pp. 1-3, 2011.

[56] Ketu, S.; Mishra, P.K. *Internet of Healthcare Things: A contemporary survey*. Journal of Network and Computer Applications, Elsevier, Volume 192, 31 pp., 2021.

[57] Mealy, G.H. *Another look at data*. In: Proceedings of the Fall Joint Computer Conference (AFIPS), ACM Digital Library, pp. 525-534, 1967. https://doi.org/10.1145/1465611.1465682 (accessed 27 November 2023)

[58] Gruber, T.R. *Toward principles for the design of ontologies used for knowledge sharing?*. International Journal of Human-Computer Studies, Elsevier, Volume 43, Issues 5-6, pp. 907-928, 1995.

[59] Gruber, T.R. *Ontology*. Encyclopedia of Database Systems, Ling Liu, M. Tamer Özsu (eds.), Springer, 03 pp., 2021. https://link.springer.com/content/pdf/10.1007/978-1-4899-7993-3_1318-2?pdf=chapter%20toc (accessed 27 November 2023).

[60] Smith, B.; Welty, C. *Ontology: Towards a New Synthesis*. In: Proceedings of the Second International Conference on Formal Ontology in Information Systems (FOIS), 07 pp., 2001. http://mba.eci.ufmg.br/downloads/recol/piii-foreword.pdf (accessed 27 November 2023).

[61] Guarino, N. *Formal Ontology and Information Systems*. Proceedings of the First International Conference on Formal Ontology in Information Systems (FOIS'98), N. Guarino (ed.), Frontiers in Artificial Intelligence and Applications, IOS Press, Vol. 46, pp. 3-15, 1998.

[62] W3C. *RDF 1.2 Schema*. W3C Working Draft, RDF-star Working Group, 2023. https://www.w3.org/TR/rdf12-schema/ (accessed 28 November 2023).

[63] W3C. *OWL 2 Web Ontology Language Document Overview (Second Edition)*. W3C Recommendation, OWL Working Group, 2012 https://www.w3.org/TR/2012/REC-owl2-overview-20121211/ (accessed 28 November 2023).

[64] W3C. *Semantic Sensor Network Ontology*. W3C Recommendation, 2017. https://www.w3.org/TR/vocab-ssn/ (accessed 28 November 2023).

[65] Janowicz, K. et al. *SOSA: A lightweight ontology for sensors, observations, samples, and actuators*. Journal of Web Semantics, Volume 56, pp. 1-10, 2019.

[66] Soldatos, J. et al. *OpenIoT: Open Source Internet-of-Things in the Cloud*. In: Proceedings of the Interoperability and Open Source Solutions for the Internet of Things, Lecture Notes in Computer Science, Springer, Volume 9001, pp. 13-25, 2015.

[67] ETSI. *SmartM2M; Smart Appliances; Reference Ontology and oneM2M Mapping*. ETSI TS 103 264 v2.1.1 (2017-03), European Telecommunications Standard Institute (ETSI), 26 pp., 2017. https://www.etsi.org/deliver/etsi_ts/103200_103299/103264/02.01.01_60/ts_103264v020101p.pdf (accessed 28 November 2023).

[68] ETSI. *SAREF4EHAW: an extension of SAREF for eHealth Ageing Well domain*. European Telecommunications Standards Institute (ETSI), 2020. https://saref.etsi.org/saref4ehaw/v1.1.1/ (accessed 28 November 2023).

[69] Kierkegaard, P. *Electronic health record: Wiring Europe's healthcare*. Computer Law & Security Review, Elsevier, Volume 27, Issue 5, pp. 503-515, 2011.

[70] BioPortal. *Open Biological and Biomedical Ontology (OBO) Foundry*. OBO Technical WG, 2022. http://obofoundry.org/ (accessed 28 November 2023).

[71] BioPortal. *International Classification of Diseases, Version 9 - Clinical Modification*. National Center for Biomedical Ontology, 2023. https://bioportal.bioontology.org/ontologies/ICD9CM/ (accessed 28 November 2023).

[72] BioPortal. *International Classification of Diseases, Version 10 - Clinical Modification*. National Center for Biomedical Ontology, 2023. https://bioportal.bioontology.org/ontologies/ICD10CM/ (accessed 28 November 2023).

[73] BioPortal. *Body System Terms from ICD11*. National Center for Biomedical Ontology, 2010. https://bioportal.bioontology.org/ontologies/ICD11-BODYSYSTEM/ (accessed 28 November 2023).

[74] BioPortal. *SNOMED CT*. National Center for Biomedical Ontology, 2023. https://bioportal.bioontology.org/ontologies/SNOMEDCT (accessed 28 November 2023).

[75] WHO. *ICD-11 - International Classification of Diseases 11th Revision*. World Health Organization, 2022. https://icd.who.int/en (accessed 28 November 2023).

[76] HL7. *HL7 Standards*. HL7 International, 2923. http://www.hl7.org/ (accessed 28 November 2023).

[77] OpenEHR. *The future of health & care is open*. openEHR International, 2023. https://openehr.org/ (accessed 28 November 2023).

[78] Lewis, J.; Fowler, M. *Microservices: a definition of this new architectural term*. martinFowler.com, 23 pp., 2014. https://martinfowler.com/articles/microservices.html (accessed 01 December 2023).

[79] Velepucha, V.; Flores, P. *A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges*. IEEE Access, Volume 11, pp. 88339-88358, 2023.

[80] Richardson, C. *Microservices Patterns: With Examples in Java*. Manning Publications Co., 522 pp., 2019.

[81] Richardson, C. *Pattern: Decompose by business capability*. Microservice Architecture, 03 pp., 2023. https://microservices.io/patterns/decomposition/decompose-by-business-capability.html (accessed 05 December 2023).

[82] Richardson, C. *Pattern: Decompose by subdomain*. Microservice Architecture, 04 pp., 2023. https://microservices.io/patterns/decomposition/decompose-by-subdomain.html (accessed 05 December 2023).

[83] Abdelmoneem, R.M. et al. *A Cloud-Fog based Architecture for IoT Applications Dedicated to Healthcare*. In: Proceedings of the 2019 IEEE International Conference on Communications (ICC), pp. 1-6, 2019.

[84] Debauche, O. et al. *Fog IoT for Health: A new Architecture for Patients and Elderly Monitoring*. In: Proceedings of the 9th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare, Procedia Computer Science, Elsevier, Volume 160, pp. 289-297, 2019.

[85] Heydon, R. *Bluetooth Low Energy: The Developer's Handbook*. Pearson Education Inc., Prentice Hall, 328 pp., 2013.

[86] Rahman, H.; Hussain, M.I. *Fog-based semantic model for supporting interoperability in IoT*. IET Communications, Volume 13, Issue 11, pp. 1651-1661, 2019.

[87] Castaneda, D. et al. A. *A review on wearable photoplethysmography sensors and their potential future applications in health care*. International Journal of Biosensors & Bioelectronics, Volume 4, Issue 4, pp. 195-202, 2018.

[88] Jeon, D.-c.; LIUHAOYANG; Hwang, H. *Design of Hybrid Application Based on GraphQL for Efficient Query for PHR*. In: 2019 International Conference on Information and Communication Technology Convergence (ICTC), pp. 381-383, 2019.

[89] Brito, G.; Valente, M.T. *REST vs GraphQL: A Controlled Experiment*. In: 2020 IEEE International Conference on Software Architecture (ICSA), pp. 81-91, 2020.

[90] Chitra, L.P.; Satapathy, R. *Performance comparison and evaluation of Node.js and traditional web server (IIS)*. In: 2017 International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET), pp. 1-4, 2017.

[91] Lei, K.; Ma, Y.; Tan, Z. *Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js*. In: 2014 IEEE 17th International Conference on Computational Science and Engineering, pp. 661-668, 2014.

[92] Halili, E.H. *Apache JMeter: A practical beginner's guide to automated testing and performance measurement for your websites*. Packt Publishing, 138 pp., 2008.

[93] Vohra, N.; Manuaba, I.B.K. *Implementation of REST API vs GraphQL in Microservice Architecture*. In: 2022 International Conference on Information Management and Technology (ICIMTech), pp. 45-50, 2022.

[94] Health Bureau. *How to measure blood pressure using digital monitors*. The Government of the Hong Kong Special Administrative Region of the People's Republic of China. https://www.healthbureau.gov.hk/pho/files/How-to-measure-blood-pressure-using-digital-mon.pdf (accessed 29 April 2023).

[95] Stergiou, G.S. et al. *A Universal Standard for the Validation of Blood Pressure Measuring Devices: Association for the Advancement of Medical Instrumentation/European Society of Hypertension/International Organization for Standardization (AAMI/ESH/ISO) Collaboration Statement*. Hypertension, Volume 71, Issue 3, pp. 368-374, 2018.

Chapter 4

[96] Rahman, H.; Ahmed, N.; Hussain, M.I. *A QoS-aware hybrid data aggregation scheme for internet of things*. In: Annals Telecommunications, Springer, Volume 73, pp. 475-486, 2018.

[97] Rahman, H.; Hussain, M.I. *A comprehensive survey on semantic interoperability for Internet of Things: State-of-the-art and research challenges*. Transactions on Emerging Telecommunications Technologies, Wiley Online Library, Volume 31, Issue 12, 25 pp., 2020.

[98] Buneman, P.; Staworko, S. *RDF Graph Alignment with Bisimulation*. In: Proceedings of the VLDB Endowment, Volume 9, Issue 2, pp. 1149-1160, 2016.

[99] W3C. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. W3C, 2004. https://www.w3.org/submissions/SWRL/ (accessed 09 October 2023).

[100] W3C. *SPARQL 1.1 Overview*. W3C Recommendation, SPARQL Working Group, 2013. https://www.w3.org/TR/sparql11-overview/ (accessed 06 October 2023).

[101] Gyrard, A.M.; Serrano, M.; Atemezing, G.A. *Semantic Web Methodologies, Best Practices and Ontology Engineering Applied to Internet of Things*. In: Proceedings of the 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), pp. 412-417, 2015.

[102] Moreira, J.L.R. *SEMIoTICS: Semantic Model-driven Development for IoT Interoperability of Emergency Services: Improving the Semantic Interoperability of IoT Early Warning Systems*. PhD Thesis, University of Twente (UT), Enschede, the Netherlands, 305 pp., 2019.

[103] Portisch, J.; Hladik, M.; Paulheim, H. *Background Knowledge in Ontology Matching: A Survey*. Semantic Web, Volume Pre-press, Issue Pre-press, pp. 1-55, 2022. https://doi.org/10.3233/SW-223085 (accessed 06 October 2023).

[104] Euzenat, J.; Shvaiko, P. *Ontology Matching*. Second Edition, Springer, 512 pp., 2013.

[105] Lamy J.-B. *Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies*. Artificial Intelligence In Medicine, Volume 80, pp.11-28, 2017.

[106] ETSI. *SmartM2M; Extension to SAREF; Part 8: eHealth/Ageing-well Domain*. ETSI TS 103 410-8 V1.1.1 (2020-07) Technical Specification, 2020. https://www.etsi.org/deliver/etsi_ts/103400_103499/10341008/01.01.01_60/ts_10341008v010101p.pdf (accessed 06 October 2023).

[107] Torab-Miandoab, A. et al. *Interoperability of heterogeneous health information systems: a systematic literature review*. BMC Medical Informatics and Decision Making, Volume 23, Issue 18, pp. 1-13, 2023.

[108] Protégé. *A free, open-source ontology editor and framework for building intelligent systems*. Protégé Version 5.5.0, 2019. https://protege.stanford.edu/(accessed 07 October 2023).

[109] Ashouri, M. et al. *Edge Computing Simulators for IoT System Design: An Analysis of Qualities and Metrics*. Future Internet, Volume 11, Issue 11, 235, 12 pp., November 2019.

[110] Gupta, H. et al. *iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Environments*. Journal of Software: Practice and Experience, Volume 47, Issue 9, pp. 1275-1296, September 2017.

[111] Calheiros, R.N. et al. *CloudSim: a toolkit for modelling and simulation of cloud computing environments and evaluation of resource provisioning algorithms*. Journal of Software: Practice and Experience, Volume 41, Issue 1, pp. 23-50, August 2010.

[112] Mahmud, R. et al. *iFogSim2: An extended iFogSim simulator for mobility, clustering, and microservice management in edge and fog computing environment*. The Journal of Systems & Software, Volume 190, Issue C, 17 pp., August 2022.

[113] Rahman, F.H. et al. *A Performance Study of High-End Fog and Fog Cluster in iFogSim*. In Advances in Intelligent Systems and Computing, Volume 888, 10 pp;.2019.

[114] Surantha, N.; Utomo, O.K.; Lionel, E.M.; Gozali, I.D.; Isa, S.M. *Intelligent Sleep Monitoring System Based on Microservices and Event-Driven Architecture*. IEEE Access, Volume 10, pp. 42069-42080, 2022.

[115] Patti, E.; Donatelli, M.; Macii, E.; Acquaviva, A. *IoT Software Infrastructure for Remote Monitoring of Patients with Chronic Metabolic Disorders*. In: 2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud), pp. 311-317, 2018.

[116] Sujatha Kumari, B.A; Shreyas, K.S.; Skanda, M.S.; Manoj Kumar; Prajwal, C.D. *IOT-Based Remote Patient Monitoring System Using Microservices Architecture*. In: Soft Computing for Security Applications (Proceedings of ICSCS 2021), Advances in Intelligent Systems and Computing 1397, Springer, pp. 365-38, 2022.

[117] Wickramasinghe, S. Data Denormalization: The Complete Guide. Slunk, 2023. https://www.splunk.com/en_us/blog/learn/data-denormalization.html (accessed 12 March 2024).

[118] VMware Tanzu. Spring Projects. Spring, 2005-20024. https://spring.io/projects (accessed 12 March 2024).

[119] Vogel, M.; Weber, S.; Zirpins, C. *Experiences on Migrating RESTful Web Services to GraphQL*. In: Service-Oriented Computing - ICSOC 2017, Lecture Notes in Computer Science, Springer, Volume 10797, pp. 283-295, 2018.

[120] Lawi, A.; Panggabean, B.L.E.; Yoshida, T. *Evaluating GraphQL and REST API Services Performance in a Massive and Intensive Accessible Information System*. Computers, Volume 10, Issue 138, 16 pp., 2021.

[121] Quiña-Mera, A. et al. *REST, GraphQL, and GraphQL Wrapper APIs Evaluation. A Computational Laboratory Experiment*. In: Proceedings of International Conference on Information Technology and Applications, Lecture Notes in Networks and Systems, Springer, Volume 614, pp. 397-407, 2023

[122] PerfMon. *Servers Performance Monitoring plugin for jMeter*. Version 2.1, 2024. https://jmeter-plugins.org/wiki/PerfMon/ (accessed 01 May 2024)

[123] Docker stats. *A Docker command to live stream a container's runtime metrics.* Docker Version 4.20, 2023. https://docs.docker.com/config/containers/runmetrics/ (accessed 01 May 2024).

[124] Kazanavičius, J.; Mažeika, D. *The Evaluation of Microservice Communication While Decomposing Monoliths*. Computing and Informatics, Volume 42, Issue 1, pp. 1–36, 2023.

[125] Blinowski, G.; Ojdowska, A.; Przybyłek, A. *Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation*. IEEE Access, Volume 10, pp. 20357-20374, 2022.

[126] Al-Debagy, O.; Martinek, P.A. *Comparative Review of Microservices and Monolithic Architectures*. In: Proceedings of the 2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI), pp. 149-154, 2018.

[127] Ross, D.T. *Structured Analysis (SA): A Language for Communicating Ideas*. IEEE Transactions on Software Engineering, IEEE Press, Volume SE-3, Issue 1, pp. 16-34, January 1977.

[128] Lopes de Souza, P.; Lopes de Souza, W.; Ferreira Pires, L. *ScrumOntoBDD: Agile software development based on scrum, ontologies and behaviour-driven development*. Journal of the Brazilian Computer Society (JBCS), Elsevier, Volume 27, Issue 10, 45 pp., June 2021.

[129] Person, S. *9 Best Business Process Modelling Techniques (With Examples)*. TllyFy Inc. https://tallyfy.com/business-process-modeling-techniques/ (accessed 23 January 2024).

[130] OMG. *About Business Process Model and Notation Specification Version 2.0*. Object Management Group. https://www.omg.org/spec/BPMN/2.0 (accessed 23 January, 2024).

[131] North, D. *What's in a Story?*. Dan North & Associates, 2007. https://dannorth.net/whats-in-a-story/ (accessed 24 January, 2024).

[132] Guizzardi, G. *Ontological Foundations for Structural Conceptual Models*. PhD Thesis, University of Twente, Enschede, the Netherlands, 441 pp., 2005. https://ris.utwente.nl/ws/portalfiles/portal/6042428/thesis_Guizzardi.pdf (accessed 06 February 2024).

[133] OntoUML Community. *Tooling OLED (OntoUML lightweight editor)*. https://ontouml.org/ontouml/tooling/ (accessed 06 February 2024).

[134] Gangemi, A. et al. *Modelling ontology evaluation and validation*. The Semantic Web: Research and Applications, Lecture Notes in

Computer Science (LNCS), Springer, Volume 4011, pp. 140-154, June 2006.

[135] Balalaie, A.; Heydarnoori, A. *Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture*. IEEE Software, Volume 33, Issue 3, pp. 42-52, May-June 2016.

[136] Patil, C.; Chaware, A. *Heart (Pulse Rate) Monitoring using Pulse Rate Sensor, Piezo Electric Sensor and NodeMCU*. In: Proceedings of the 8th International Conference on Computing for Sustainable Global Development (INDIACom), pp. 337-340, 2021.

[137] Kirtana, R.N.; Lokeswari, Y.V. *An IoT based remote HRV monitoring system for hypertensive patients*. In: Proceedings of the IEEE International Conference on Computer, Communication and Signal Processing, 2017, pp. 1-6.

[138] Putra, R.D.; Wibisono, G. *System Design and Implementation of Machine-to-Machine (M2M) for Hypertension Patients*. In: Proceedings of the IEEE 85th Vehicular Technology Conference, pp. 1-5, 2017.

[139] Ghoshachandra, P.; Limkriengkrai, C.; Wimonsakcharoen, P.; Tangsripairoj, S. *oHealth: A self-care android application for senior citizens with hypertension*. In: Proceedings of the 6th ICT International Student Project Conference, pp. 1-5, 2017.

[140] Sood, S.K.; Mahajan, I. *IoT-Fog based Healthcare Framework to Identify and Control Hypertension Attack*. IEEE Internet of Things Journal, IEEE, Volume 6, Issue 2, pp. 1920-1927, 2018.

[141] Mishra, S.K.; Sarkar, A. *Service-oriented architecture for Internet of Things: A semantic approach*. Journal of King Saud University - Computer and Information Sciences, ScienceDirect, Volume 34, Issue 10, pp. 8765-8776, 2022.

[142] Jaleel, A. et al. *Towards Medical Data Interoperability Through Collaboration of Healthcare Devices*. IEEE Access, IEEE, Volume 8, pp. 132302-132319, 2020.

[143] Elhadj, H. B. et al. *Do-Care: A dynamic ontology reasoning based healthcare monitoring system*. Future Generation Computer Systems, Elsevier, Volume 118, pp. 417-431, 2021.

[144] Lamprinakos, G.C. et al. *An integrated remote monitoring platform towards Telehealth and Telecare services interoperability*. Information Sciences, Elsevier, Volume 308, pp. 23-37, 2015.

[145] Rhayem, A.; Mhiri, M.B.A.; Salah, M.B.; Gargouri, F. *Ontology-based system for patient monitoring with connected objects*. Procedia Computer Science, Elsevier, Volume 112, pp. 683-692, 2017.

[146] Ianculescu, M.; Alexandru, A.; Neagu, G.; Pop, F. *Microservice-Based Approach to Enforce an IoHT Oriented Architecture*. In: Proceedings of the 7th IEEE International Conference on E-Health and Bioengineering Conference (EHB), pp. 1-4, 2019.

[147] Garcia-Moreno, F.M. et al. *A Microservices e-Health System for Ecological Frailty Assessment using Wearables*. Sensors, Multidisciplinary Digital Publishing Institute (MDPI), Volume 20, Issue 12, 3427, pp. 1-23, 2020.

[148] Falbo, R. A. *SABiO: Systematic Approach for Building Ontologies*. In: Proceedings of the 1st Joint Workshop ONTO.COM/ODISE on Ontologies in Conceptual Modeling and Information Systems Engineering, CEUR Workshop Proceedings, Volume 1301, pp. 1-14, 2014.

[149] Qi, J. et al. *Examining sensor-based physical activity recognition and monitoring for healthcare using Internet of Things: A systematic review*. Journal of Biomedical Informatics, Volume 87, pp. 138-153, 2018.