João Emanuel Araújo Freire

Licenciado em Ciências da
Engenharia Electrotécnica e de Computadores

Transforming Home Appliances into IoT Devices

Dissertação para obtenção do Grau de Mestre em
Engenharia Electrotécnica e de Computadores

Orientador: Professor Doutor José António Barata de Oliveira,
Universidade Nova de Lisboa

Coorientador: Mestre André Dionísio Rocha,
CTS – Uninova

Júri:

Presidente: Professor Doutor Luís Augusto Bica Gomes de Oliveira

Arguente: Professor Doutor Tiago Oliveira Machado de Figueiredo Cardoso

Vogal: Professor Doutor José António Barata de Oliveira

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Transforming Home Appliances into IoT Devices

To my Family and Friends

Acknowledgments

I would like to thank all those who contributed in some way to the work described in this dissertation.

First and foremost, I thank my academic supervisor, PhD Professor José Barata, for giving me the opportunity to undertake this research study and to develop this dissertation in a subject of my interest.

Besides my supervisor, I would like to thank my co-supervisor, PhD Student André Rocha, for spending his time whenever I approached him and showing me the way ahead. His guidance helped me during the research and writing of this thesis.

I also want to take this opportunity to express gratitude to all my friends whose support and friendship proved essential.

Last but not least, I would like to thank my family for their love and support, especially my parents and sister for supporting me through my life in general.

Abstract

Home appliances were, until not very recently, isolated devices uncapable of communicate with others. With the recent technological advances is it possible nowadays to connect all these devices to the internet enabling them with the capacity to communicate between them and also the possibility to store and analyse the data generated making them more efficient.

Due to these advancements, terms like Internet of Things and Cloud Computing surfaced. These terms represent architectures that allow communications and interactions between devices. The potential of these technologies is substantial for the improvement of productivity and economic impact.

These advancements allowed home appliances to generate an enormous amount of data, and as repercussion, the need for structures able to receive, storage and process the data emerged.

The present document contains a proposition of an architecture that allows to collect and send home appliances' data to the Cloud, a way to store and make them available to other applications and a platform that allows the data's visualisation and analysis. The proposed architecture was tested with a refrigerator offered by Electrolux inserted in the ProSEcO project.

Keywords: Home appliances; Meta products; Internet of Things; Cloud Computing; Web-Services

Resumo

Os eletrodomésticos eram, até há bem pouco tempo, dispositivos isolados sem qualquer capacidade de comunicar com outros. Com os recentes avanços tecnológicos existe hoje em dia a possibilidade de ligar todos estes dispositivos à internet permitindo-os comunicar entre si e também a possibilidade de analisar os dados por eles gerados de modo a torná-los mais eficientes.

Devido a estes avanços começaram a surgir termos como *Internet of Things* e *Cloud Computing* que representam sistemas que permitem os dispositivos comunicar e interagir. Estas tecnologias apresentam um potencial substancial em termos de desenvolvimento de produtividade e impacto económico.

Este novo passo na evolução dos eletrodomésticos fez com que uma enorme quantidade de dados, provenientes destes dispositivos, começasse a ser gerada e, por conseguinte, a necessidade de estruturas capazes de receber, armazenar e tratar estes dados.

O presente documento contém uma proposta de arquitetura que permite recolher e enviar os dados dos eletrodomésticos para a *Cloud*, armazená-los e disponibilizá-los para outras aplicações assim como uma plataforma que permite a visualização e analise dos mesmos. A arquitetura proposta foi testada com um frigorífico disponibilizado pela Electrolux no âmbito do projeto ProSEcO.

Palavras-Chave: Eletrodomésticos*; Meta products*; *Internet of Things; Cloud Computing; Web-Services*

Table of Contents

Table of Figures

Table of Tables

Acronyms

ACID	-	Atomicity, Consistency, Isolation, Durability
API	-	Application Programming Interface
CC	-	Cloud Computing
CPS	-	Cyber-Physical Systems
CPU	-	Central Processing Unit
CRUD	-	Create, Read, Update, Delete
CSV	-	Comma Separated Values
GSM	-	Global System for Mobile communications
GUI	-	Graphical User Interface
HTTP	-	Hypertext Transfer Protocol
IaaS	-	Infrastructure as a Service
ID	-	Identifier
IoT	-	Internet of Things
IP	-	Internet Protocol
IPv4	-	Internet Protocol version 4
IPv6	-	Internet Protocol version 6
LAN	-	Local Area Network

LTE	-	Long Term Evolution
MP	-	Meta Products
NoSQL	-	Not Only SQL
PaaS	-	Platform as a Service
QoI	-	Quality of Information
QoS	-	Quality of Service
REST	-	Representational State Transfer
RFID	-	Radio Frequency IDentification
SaaS	-	Software as a Service
SOA	-	Service-Oriented Architecture
SOAP	-	Simple Object Access Protocol
SQL	-	Structured Query Language
TCP	-	Transmission Control Protocol
URI	-	Uniform Resource Identifier
URL	-	Uniform Resource Locator
WLAN	-	Wireless Local Area Network
WSDL	-	Web Services Description Language
WSNs	-	Wireless Sensor Networks
XML	-	Extensible Markup Language

1

Introduction

1.1 Problem

During the last decades, home appliances became indispensable in the households, saving people work and time. It is now unthinkable to go back to a time where a house did not have appliances like a washing machine or a refrigerator.

In the past, these home appliances were nothing more than isolated devices that helped people cook, clean, launder, store and cool food without communicating with any other devices. Nowadays with the evolution of the hardware, with the presence of wireless internet connection in almost every house and with technologies such as Machine Learning, Cloud Computing and Internet of Things (IoT) becoming more mature and present, a new approach to these home appliances started appearing. With this evolution is now possible to connect these appliances to the internet, making them able to communicate with each other, control them remotely and collect their data to analyse and ideally predict some behaviours in order to improve the users experience and reduce their work.

With all these appliances now connecting to the internet and sharing their sensors data the need for a structure capable of store and process the data quickly appeared.

For the vendors, the collection and analyses of these appliances' data are extremely valuable since it allows them to predict the users' behaviour and change the machines parameters during their lifecycle so they meet the users' needs more efficiently. With this data collection it

is also possible for the vendors to plan proactive maintenance, enhancing the costumers-company relationship.

1.2 Research Questions and Hypothesis

Taking in consideration the aspects previously presented and the benefits for the customer and the vendors in having their home appliances connected to the internet some research questions can occur:

- Which architecture is capable to collect the amount of data generated by these devices?
- How to make the extracted data available to be used by other tools and applications?

Regarding the first question, it is proposed in this document a modular solution based on services to collect the data from a home appliance and send it to a server to be stored. The author of this document proposes the use of a Web Services Architecture to achieve a quick data change and to guarantee a fast integration with multiple and different applications.

The use of a Web Services Architecture will also help to find an answer to the second question once it enables the possibility for different architectures to communicate easily with each other, making the collected data available for a variety of different tools.

1.3 Motivation

Nowadays it is possible to create new opportunities for home appliances and improve their capabilities. With this architecture the author aims to provide a valid answer for data acquisition from a non IoT device and its storage in a server capable of making data available for innumerable applications. The data can then be treated and presented in different outputs such as graphics and text.

This solution enables the possibility of making predictions based on behaviours and helps improving the user and the manufacturer overall experience with the devices. By analysing the collected data the devices can adapt their way of functioning to the users' needs. This offers the user and manufacturer enhanced services such as predictive maintenance and a way to accompany the device's evolution and performance throughout the user's experience.

1.4 Accomplished Work

The raised questions and the suggestions put forward, have provided the basis for a proposal of architecture, which aims to give home appliances a connection to the internet and a way to store the collected data and make it available to other applications.

The proposed architecture consists in three main layers:

- Data Acquisition Layer – acquire data from home appliances and send it to the core layer to be stored.
- Core Layer – receive data from the devices, store it properly and also make it available for other applications to analyse.
- Data Visualisation Layer – an application that makes use of the stored data in the core layer and presents a way to easily show and analyse the data collected from the home appliances.

All the layers are independent from each other since the communication with the core layer is made through web services. It is possible to have different data acquisition applications reporting data to the core layer and also more applications making use of the collected data.

All the data used in the proposed architecture was generated by a refrigerator, which belongs to Electrolux's case study of ProSEco European Project. The refrigerator was enabled with a device that collected all 130 sensors state and save it locally every predefined time.

The proposed architecture was implemented using Java, HTML and Javascript programming languages and some libraries such as Restlet and Google Charts.

1.5 Major Contributions

The major contributions of the current work coincide in presenting an architecture that can create new opportunities for home appliances and improve their functioning.

This architecture presents a possible solution for data acquisition from a non IoT device and its storage in a server capable of making that data available for innumerable applications that can, not only present the data but also, analyse it and predict behaviours.

Although this work was tested in a refrigerator as case study, the use of the suggested architecture may be generalized for retrieving the data of every home appliance, as long as the device is capable of generate and store its data.

2

State of the Art

This chapter is a theoretical research into the actual state of the art with respect to the Internet of Things and Meta Products focusing on methods, tools and technologies that are relevant to this dissertation.

2.1 Meta Products

Meta Products (MP) are, accordingly to (Hazenberg and Huisman, 2012), physical objects with their matching web elements. Usually, these objects have activator functions that are operated by the web elements and also sensorial elements that work as inputs for the system.

An IoT object is also a Meta Product since it is a real live object that has its virtual representation in an internet structure and it is uniquely identifiable. The term “Internet of Things” was introduced by Kevin Ashton in 1999 (Ashton, 2009).

The virtual representation is a virtual copy of the real object, it now has more value than its physical counterpart since it may undergo any data process such as storage and visualization.

Figure 2.1 is an illustration that represents a Meta Product, it shows its physical part, its sensors and actuators, as well as its web element.

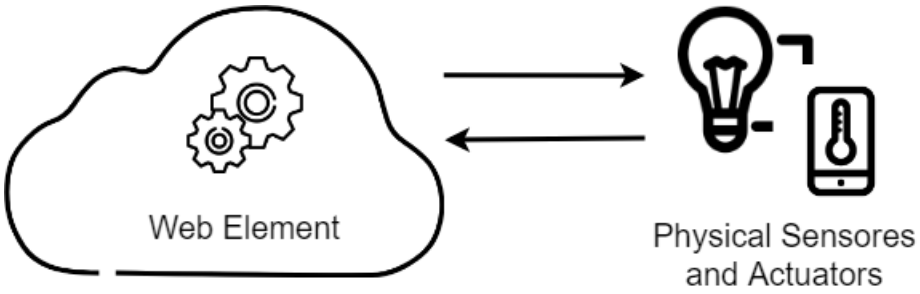

Figure 2.1 – Meta Products representation

Meta Products have the potential to transform the way that objects work, and how they can impact life. Any regular object can be converted as long as they are equipped with machine readable identifiers or identifying devices (Mattern and Floerkemeier, 2001).

An ontology for representing MP and their relationships in the IoT has been presented and it contains seven modules (Wang *et al.*, 2012):

- IoT service: means of interfacing the physical world with context intelligent applications and services;
- Service Test: proposed for testing functional and non-functional capabilities;
- QoS and QoI: important concepts in many areas, such as communications and networking;
- Deployment, System and Platform: provides information regarding resources and how they are organized and deployed;
- Observation and Measurement: contains the information collected from the real world;
- IoT Resources: focused on sensors and sensor networks;
- Entity of Interest and Physical Locations: object in the real world which is of interest for the user.

2.2 Internet of Things

The Internet of Things (IoT) concept is known for almost 20 years (Yashiro *et al.*, 2013). It was proposed in 1999 by Kevin Ashton as "uniquely identifiable interoperable connected objects with radio-frequency identification (RFID) technology ". Although there is not a standard definition for what IoT is (Li, Xu and Zhao, 2015), the main idea is "a world-wide network of interconnected objects uniquely addressable, based on standard communication protocols" (Atzori, Iera and Morabito, 2010; Botta *et al.*, 2016).

The IoT concept brought a lot of advantages to working and domestic fields like domotics, assisted living, automation, industrial manufacturing and logistics (Atzori, Iera and Morabito, 2010).

The RFID technology had a major role at the starting of the implementation of this concept once it allowed a normal object be automatically identified (Gubbi *et al.*, 2013). The passive RFID is composed by a tag and a reader. The tag is composed by an antenna and a microchip and it is not battery powered using the power of the signal sent by the reader to send the answer with its identification (ID) (Want, 2006). Due to its simplicity and the low cost of the tags this technology is widely used in retail and supply chain management and in public transportation and other places where there is the need of access control (Gubbi *et al.*, 2013).

The problem of having computer and communications power on everyday objects is that the size of the components mater, therefore, power consumption and processing power are limited (Risteska Stojkoska and Trivodaliev, 2017).

The IoT devices are generally know to be small objects in the real world, broadly distributed and with limited processing capacity and storage which normally create problems related to performance, reliability privacy and security (Botta *et al.*, 2016).

Cloud Computing has been used to provide the virtual infrastructures necessary to the IoT devices to work and communicate properly, creating the necessary environment and offering the processing power needed, giving the support needed to have utilities for integrate monitoring devices, storage devices data, analytics tools, visualization platforms and client delivery (Zhang, Cheng and Boutaba, 2010).

For the IoT devices to work properly it is necessary to have a Smart connectivity solution that works with the existing networks and have a context-aware computation using the network resources. “With the growing presence of WiFi and 4G-LTE wireless Internet access, the evolution towards ubiquitous information and communication networks is already evident” (Gubbi *et al.*, 2013).

These objects, acting as sensors or actuators, are able to interact with each other in order to reach a common goal.

According to (Joseph Bradley, Barbier and Handler, 2013) there are four main subjects that are in the core of IoT systems: RFID, Wireless sensor networks, Addressing and Middleware. As it was previously stated the RFID technology had a major role in the IoT world allowing common

object being unequivocally identified in a network. Wireless Sensor Networks (WSNs) can cooperate with RFID technology to better trach the Status of the object and also provide various useful data such as position, movement and temperature. “Uniqueness, reliability, persistence, and scalability represent critical features related to the creation of a unique addressing schema” that is needed for the success of the IoT. The addressing problem can be solved by protocols like the IPv4 and the recent IPv6 since it supports more devices.

As mentioned by (Joseph Bradley, Barbier and Handler, 2013) the main factors increasing the value of using IoT are:

- Increasing the return on research and innovation investments, reducing time to market, creating new business models and opportunities;
- Increasing the customer lifetime and adding more customers;
- Improve utility services such as supply chain and logistics, to a new and more efficient level;
- Improve business process with the expense on goods reduced;
- Increasing the employee productivity and efficiency.

2.3 Cloud Computing

Cloud computing (CC) emerges as a consequence of the rapid development in processing and storage technologies and the easy access to an internet connection. It is, therefore, a new computer model in which resources like CPU and Storage are provided as general utilities that can be rapidly provisioned and released through the Internet in an on-demand way (Zhang, Cheng and Boutaba, 2010). The National Institute of Standards and Technology (NIST) defined five essential characteristics, three service models, and four deployment models that compose the cloud computer model (Mell and Grance, 2011).

The essential characteristics are:

On-demand self-service – a consumer can unilaterally provision computer capabilities as needed automatically without requiring human interaction with each service provider.

Broad network access – the computer resources are available over the network and accessed through standard mechanisms that promote use by heterogeneous client platforms such as mobile phones and laptops.

Resource pooling - the provider's computing resources are pooled to serve multiple consumers with different physical and virtual resources dynamically assigned and reassigned according to consumer demand.

Rapid elasticity – computing resources can be obtained and released dynamically according to usage demand. The capabilities available often seem to be unlimited and can be taken in any quantity at any time.

Measured service – The system automatically controls and optimizes resources used, measuring them accordingly to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

The Service Models are provided as a way of controlling and using the resources available allowing different layers of control over the System.

The **Infrastructure as a Service** (IaaS) provide to the consumer fundamentals computing resources like storage, and processing and the ability to run software on the infrastructure like the Operating System and the applications.

The **Platform as a Service** (PaaS) provide the consumer with the ability to deploy his own applications and programs and some configurations settings but it does not allow the control of the infrastructure like the network and the operating system.

The **Software as a Service** (SaaS) provide the consumer the ability to use provider's applications that are running on the infrastructure using thin client interfaces like web browsers or a program interface. The consumer does not have any control of the infrastructure and is only able to change some limited application configuration settings.

The deployment models are Private Cloud, Community Cloud, Public Cloud and Hybrid Cloud and are implementing depending on the provider's interest in lowering the operation cost or have higher reliability and security(Zhang, Cheng and Boutaba, 2010).

Private Cloud is of exclusive use of an organization and it is usually owned and manage by the organization itself.

Community Cloud is of exclusive use of a community of consumers from organizations that have shared concerns and it is owned and maintain by one of them or for a third party.

Public Cloud is available for public in general being maintained by a business organization.

Hybrid Cloud is a composition of two or more distinct infrastructures (private, community, or public) running part of the services infrastructure in private clouds and the other part in public ones offering more flexibility than public and private clouds. This provide more control and security but also facilitates on-demand service expansion and contraction.

2.3.1 Storage

Data was typically stored in Relational Databases, widely known as SQL databases (Li and Manoharan, 2013). These kinds of databases are great at keeping data integrity using the properties known as ACID which stands for Atomicity, Consistency, Isolation and Durability (Binani, Gutti and Upadhyay, 2016).

Until now all kinds of data where stored in Relational Databases even if they were not adequate (Hecht and Jablonski, 2011). However, with the appearance of Big Data, this kind of storage became inappropriate since they could not guarantee fast responses and quick scalability (Binani, Gutti and Upadhyay, 2016). The quick responses were not possible due to the ACID implementation (Hecht and Jablonski, 2011) and the quick scalability was difficult once the Relational databases only allow vertical scaling.

To deal with these problems companies, like Google and Facebook, have to manage huge quantities of data start developing their own databases systems (Hecht and Jablonski, 2011). These systems are called NoSQL databases which means Not only SQL (Gudivada, Rao and Raghavan, 2014).

NoSQL databases became very popular because they give very quick responses, allow horizontal scaling with data partition and replications as built-in features (Gudivada, Rao and Raghavan, 2014) and have as main characteristic the fact of support unstructured data (Li and Manoharan, 2013).

Various architectures for NoSQL databases emerged, however, there are four groups that can contain most of the data models which are Key-value, Document, Column and Graph.

The **Key-value based** system a key is mapped to a value that can contain any type of data and it is not recognised by the databased system. Examples of these systems are Amazon DynamoDB, CouchDB and Membase (Binani, Gutti and Upadhyay, 2016).

The **Document based** systems like MongoDB and Riak store documents in JSON or in similar format that encapsulates key-value pairs. Opposed to key-value types, the values are recognised by the system and can be queried as well (Hecht and Jablonski, 2011).

The **Column Family** systems are inspired by a Google system called BigTable in which rows contain key-value data and columns have unique keys. Columns can be groups in column families which allows better data organization and partition (Hecht and Jablonski, 2011).

The **Graph type** is used to represent connected data and the relationship between the data. Once this type of system is layered it is used to implement authorization and access control. Google knowledge graph, Facebook Open graph and HypergraphDB are some of the most common databases systems of this kind (Binani, Gutti and Upadhyay, 2016).

On the Table 2.1 it is possible to see a little comparison between some factures regarding SQL databases and NoSQL databases (Binani, Gutti and Upadhyay, 2016).

Table 2.1 – Comparison of SQL and NoSQL

Distinguishing Feature	***SQL***	***NoSQL***
Relational	Yes	No
ACID	Yes	No
Scaling	No	Yes
Query Complexity	Low	High
Distributed	No	Yes

2.3.2 Web Services

Web Services offer us a way to communicate between applications implemented in different languages and in different operative systems since they use standard protocols like HTTP and the messages are encapsulated in XML documents (Alonso *et al.*, 2004). These Services are usually based on SOAP or REST principles(Belqasmi *et al.*, 2012).

SOAP Web Services are based on a service provider, a service registry and a service requestor. The communication between these entities is made with SOAP messages that are XML documents with a standard format (Belqasmi *et al.*, 2012) which is an envelope with a header and a body(Suda, 2003). The descriptions on the service registry are published using the Web Services Description Language (WSDL) and include a service description, the service methods and the binding information (Belqasmi *et al.*, 2012).

Rest Web Services have a client-server architecture and the messages are usually changed using the HTTP protocol. REST services are based on three principles: uniformity, addressability and stateless (Belqasmi *et al.*, 2012). The server offers various resources that are accessible by a URI (Zur Muehlen, Nickerson and Swenson, 2005) and only a few operations are allowed: Create, Read, Update and Delete (CRUD) that are implemented using Post, Get, Put and Delete (Belqasmi *et al.*, 2012). These have to do with addressability and uniformity. Stateless means that the request gives the server all the information it needs so it does not need any previous information to answer new requests.

Each implementation offers different benefits, SOAP web services can encapsulate complex operations and increase privacy. REST web services are lightweight which makes them better when a lot of requests are made (Zur Muehlen, Nickerson and Swenson, 2005).

On the Table 2.2 is possible to find some of the main characteristics of the SOAP and the REST web services.

Table 2.2 – SOAP and REST Web Services Characteristics

SOAP Web Services	***REST Web Services***
A library is necessary in the client side	No need for library once it uses the HTTP protocol
Not supported in some platforms	Do not expose the methods
Exposes the operations and methods calls	Support any content type such as JSON and XML
Use larger packets of data	Single resource for multiple actions
Can be stateless or stateful Have a WSDL	Usually uses HTTP Action verbs (CRUD)
More difficult to implement	Easier to implement
Only uses XML	

2.4 Cyber-Physical Systems

In the future, it is expected that all objects and structures will be embedded with computing and communication capabilities (Rajkumar *et al.*, 2010).

The recent developments have allowed a better availability and affordability of sensors and data acquisition technologies as well as better network systems leading to the growing use of sensors and networked machines. This result in a continuous generation of high volume data known as Big Data (Lee, Bagheri and Kao, 2015).

However a big gap between the cyber world and the physical world continued to exist. It was necessary to create a system capable of overcame the gap that exist between the cyber world, were information is changed and transformed, and the physical world (Sha *et al.*, 2009).

The Cyber-Physical Systems (CPS) were created with this goal of link the cyber world of computer and communications to the physical world so they can be described as "physical and engineered systems whose operations are monitored, coordinated, controlled and integrated by a computing and communication core" (Rajkumar *et al.*, 2010).

CPS are a new generation of systems with computational and physical capabilities that can interact with humans. The ability to interact with and expand the capabilities of the physical world using computation create a big opportunity to design and develop new kinds of systems such as fully autonomous cars and prostheses that allow brain signals to control objects (Baheti and Gill, 2011).

Some fields were CPS can be more relevant are medical devices and systems, transportation vehicles, robotic systems, aerospace systems, intelligent highways, defence systems, process control, factory automation, building and environmental control and smart spaces (Rajkumar *et al.*, 2010; ZHANG *et al.*, 2013).

CPS concept is very different from the traditional systems, therefor, they have some specific characteristics that, accordingly to the authors of (Shi *et al.*, 2011) are:

Closely integrated – CPS integrate computation and physical processes.

Cyber capability in every physical component and resource-constrained – Every component is embedded with software and the resources as computing power and network bandwidth are usually limited.

Networked at multiple and extreme scales – CPS networks can be wired or wireless networks making use of different technologies as Wireless LAN, Bluetooth, GSM, etc. to connect all the components. The systems scale and the devices category are extremely varied.

Dynamically reorganizing – CPS must have adaptive capabilities.

High degrees of automation, control loops must close – As CPS must have a convenient man-machine interaction advanced feedback control technologies should be applied.

Operation must be dependable, certified in some cases – As CPS tend to evolve to large scales and become more complex systems reliability and security measures are necessary.

CPS must operate dependably, safely, securely, efficiently and in real-time (Rajkumar *et al.*, 2010).

As stated by the authors of (Rajkumar *et al.*, 2010) as the world wide web have as the core enabling technologies the hypertext, communication protocols like TCP/IP and graphical interfaces the CPS have as their core the embedded systems, real-time systems, distributed sensor and control systems.

The CPS are a consequence or are originate or need by the raising of the IoT and the cloud computing. They bring together the discrete and powerful logic of computing to monitor and control the continuous dynamics of physical systems (Rajkumar *et al.*, 2010).

CPS aims to managing the Big data and control machines in order to create systems more intelligent and able to self-adapt. Factories can be transformed into an Industry 4.0 factory with significant economic potential if CPS are integrated in production, logistics and other services (Lee, Bagheri and Kao, 2015).

CPS is defined as transformative technologies for managing interconnected systems between its physical assets and computational capabilities (Lee, Bagheri and Kao, 2015).

Accordingly to the authors of (ZHANG *et al.*, 2013) CPS are difficult to analyse, design and validate because of two reasons:

- CPS, opposed to traditional systems, emphasize the holistic system view over both the cyber and physical parts. The engineering of a CPS must account for the interacting and inter-dependent behaviours of cyber and physical components to achieve system level functionalities;

- CPS need to deal with two distinct worlds: the discrete world of the cyber components usually model in discrete mathematics and the continuous world of the physical components usually modelled by differential equations. Due to these differences the integration and abstraction of these components are a major challenge.

To overcome the problem of integration that CPS systems have due to the existence of many kinds of operating systems, applications and other aspects, Service Oriented Architectures started to take place.

2.4.1 Service Oriented Architectures

The Service-oriented Architecture (SOA) is a concept of design, implement and assemble services that communicate with each other to achieve some goal (Barry and Dick, 2013). It have a huge importance in the fields of CPS once this architecture helps to overcome many challenges of computer distributed systems allowing the integration and communication between applications that have their own protocols and standards (Papazoglou and Van Den Heuvel, 2007).

The SOA are being promoted as a next evolutionary step to help organizations overcome more complex challenges by enhancing efficiency, agility, and productivity of an enterprise by positioning services as the building block.

There is the need of connecting this services to each other in some way and, although there is no need to be Web Services, it's developments and standards made in the recent years lead that these were chosen largely to make these connections in Service-oriented architectures (Papazoglou and Van Den Heuvel, 2007).

The main concept of this kind of architecture is the existence of a service provider and a service consumer (Barry and Dick, 2013). The services are software resources that encapsulate an operation and only offers an interface (Papazoglou and Van Den Heuvel, 2007).

Each service offered by a SOA application may implement a brand new function, use pieces of old applications that have been adapted and wrapped by the service implementation or it can implement a combination of legacy parts with new ones (Georgakopoulos and Papazoglou, 2008).

In a Service-oriented Architecture the services available should have the following characteristics (Channabasavaiah, Holley and Tuggle, 2004):

- All functions in an SOA are defined as services.
- All services are independent and their behaviour is unknown from the outside. They operate as “black boxes” and external components do not care how they perform their function, just expect a result.
- In the most general sense, the interfaces are invokable so, at an architectural level, it is irrelevant whether the services are local or remote. It does not matter what interconnect scheme or protocol is used to effect the invocation, nor which infrastructure components are required to make the connection.

Services in SOA should be designed in such a way that they become components that can be assembled in multiple ways (Papazoglou and Van Den Heuvel, 2007) to allow developers to quickly create and modify the services available to support or automate the business needs (Aljazzaf, Capretz and Perry, 2016). Due to the possibility of assembling services they are grouped in two types: atomic and composite (Barry and Dick, 2013).

The atomic service is a well-defined, self-contained function that does not depend on the context or state of other services.

The composite service is an assembly of atomic or other composite services. The ability to assemble services is referred to as composability. Composite services are also referred to as compound services.

The services should also be loosely coupled. This is a design concept where the internal implementation of one service is not known to another service. All that needs to be known is the external behaviour of the service. This way, the underlying programming of a service can be modified and, as long as external behaviour is the same, anything that uses that service continues to work as expected (Papazoglou and Van Den Heuvel, 2007; Barry and Dick, 2013) .

The architecture of this systems is composed by a service provider, which owns, implements, and controls access to the services, a service requestor, which is an application, service, or client who is searching and invoking a service and, although it was not always present, a service broker that groups all of the services together and maintains a registry of available services (Georgakopoulos and Papazoglou, 2008).

3

Architecture

As discussed on chapter 2 there are many technologies regarding the communication between different devices. The Rest Web Services and The Document Databases are possible solutions to solve and integrate these differences between the devices.

The proposed solution aims to transform non IoT devices into devices capable of communicating with the internet and therefore giving the system some intelligence.

3.1 Overview

3.1.1 The ProSEcO European Project

The ProSEcO solution enables the effective extension of products with Product Extensions Service solutions in different sectors (automotive, home appliances, automation equipment, machine tooling) by means of the innovative combination of Ambient Intelligence, Lean and Eco-design, Life-Cycle Assessment and Collaborative Knowledge Management techniques (Lima-Monteiro *et al.*, 2017).

The ProSEcO platform will also have the manufacture's specific learning algorithm that will generate predictive data regarding the devices' sensors that need to be stored and made available by this architecture (Di *et al.*, 2016).

3.1.2 The Interaction

The proposed architecture is designed to be integrated with the ProSEcO solution, creating the middleware between the ProSEcO platform and the devices offering also the possibility to analyse the retrieved data.

To develop this architecture a home appliance is provided with the capability to generate data and store it internally. The goal is to obtain that data and store it on the cloud making it available to the platform and other services.

The solution proposed was separated into three main layers. The core layer is responsible for receiving the data coming from the different sources and store it. It is also responsible for making the data available to other resources that may need it. Another layer is the data acquisition layer. This one is responsible for acquire the data from the device and send it to the core layer. The last layer is the data visualization layer, that allows the visualisation of the stored data and the analyses of it.

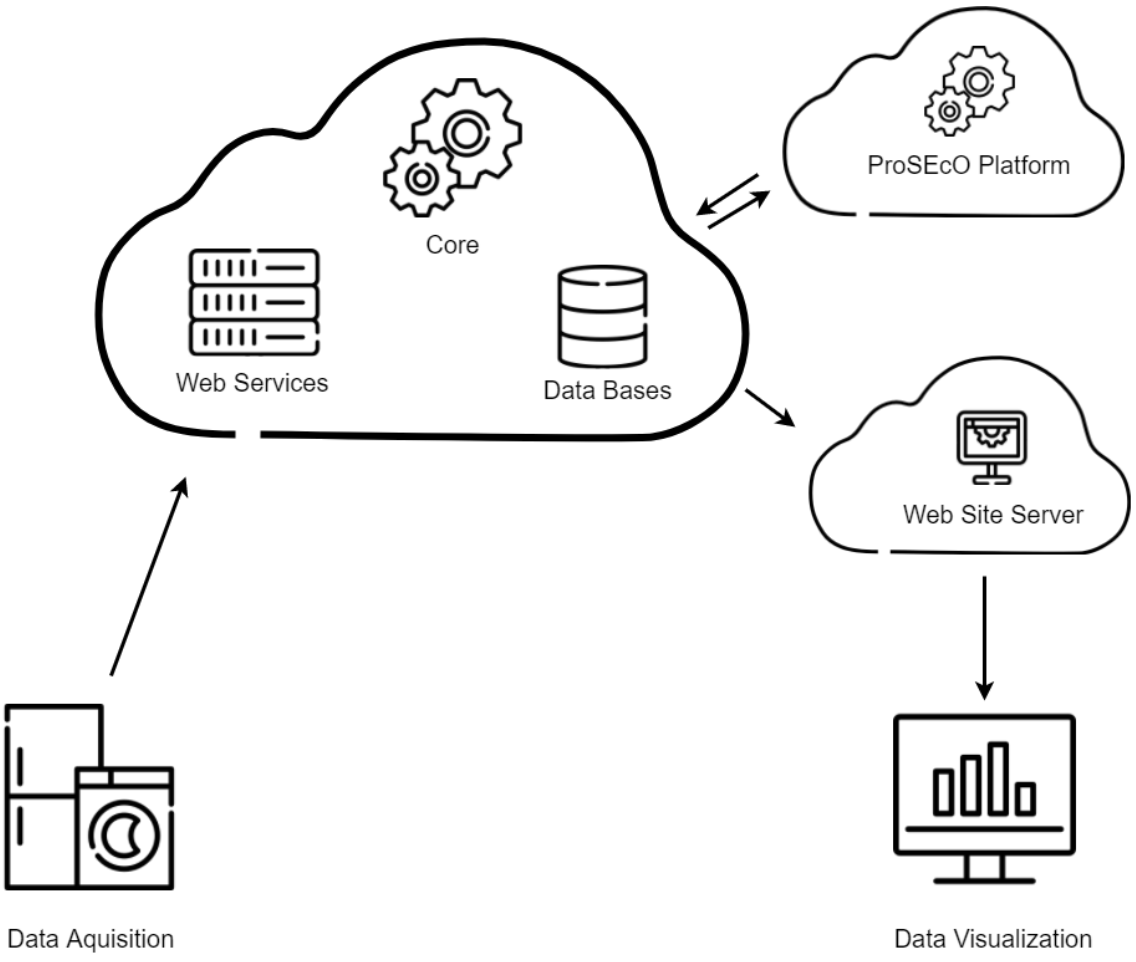

Figure 3.1 - Proposed Architecture Overview

The Figure 3.1 represents an overview of the proposed architecture where it is possible to see how all the elements of the system connect and interact.

3.2 Data Acquisition Layer

The acquisition layer is the layer responsible for collecting the data from the home appliance and send it to the core layer to be stored. This layer will allow the devices that normally do not have an internet connection to become IoT devices and communicate their status to a platform online. For this architecture the used home appliances need to have the capacity to collect sensorial information regarding their status and save it locally. This layer will read that information and send it to the core layer.

In the proposed architecture, the user uses a Graphical User Interface (GUI) that has been implemented in order to allow a more user-friendly management of the system. Although this layer is developed to acquire data from a specific type of home appliances a replication or adaptation of it can be done since, that in the end, the sensors' information is sent through a web service to the core layer.

The flow chart on Figure 3.2 describes the architecture used to retrieve data from a device.

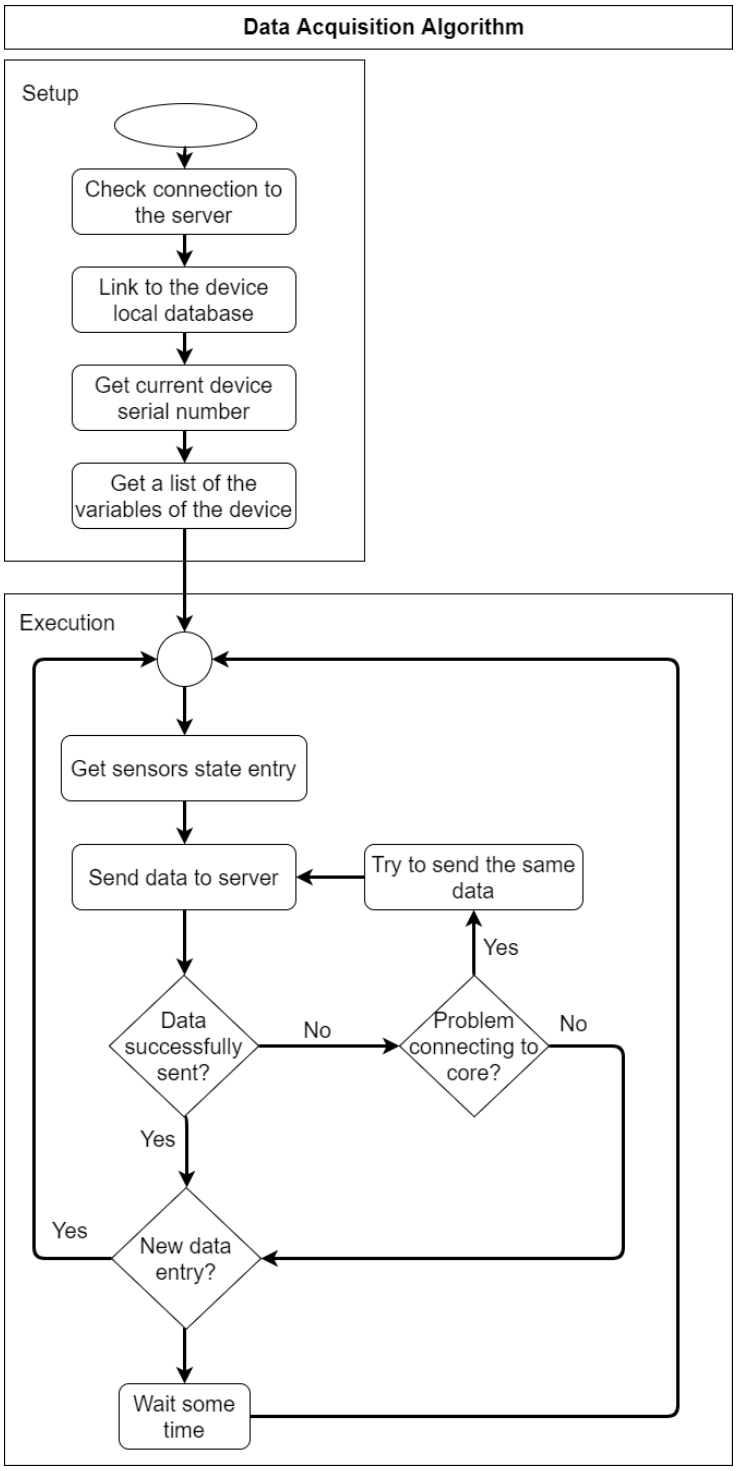

Figure 3.2 – Data Acquisition Architecture

This architecture was developed having in mind that the home appliances used could locally save the state of their sensors and keep a history of them.

This application starts by getting a link to the file where the device stores its sensors data. When this link is successful it proceeds to get a list of the variables stored and checks if it matches

the number of variables that it was supposed to have. If the number of variables matches as expected, the application will start to send one entry of data to the core system. If it delivers the data successfully it will get another reading set, otherwise will try to send the previous set again. The set of readings is sent with the addition of the device serial number to inform the core layer who generated that data.

When there are no more sets of data to send the application will wait some predefined time until it checks for new data again.

3.3 Core Layer

The core layer is composed by different modules that when together make the connection between all the devices and the different layers.

These modules are responsible for receiving data, organizing it, storing it and make it available for other layers. It can be divided in 3 main blocks:

- Data Communication
- Data Processing
- Data Storage

The Data Communication will be supported by web services because this approach, as discussed in Chapter 2, is the most efficient to use in systems that work on different programming languages. This module will allow the data to be retrieved and sent from the Core Layer and will be responsible for taking care of different types of requests, such as, requests to save data that is coming from the different devices, give a list of devices that are saved on the system, give the list of variables saved for each device and give the requested data of a certain device.

The Data Processing module will be responsible for processing the incoming data so it can be properly stored in the database. It will also be responsible for gathering the necessary data from the databases to respond to the requests made to the core layer.

The Data Storage module will be composed by the databases in which the data will be saved. As it was discussed on the Chapter 2 a NoSQL database will be used as it is more efficient to work with huge amounts of data.

3.3.1 Data Storage Module

The data storage module is composed by the databases server and the driver in the main application that allows the communication between the different components. For this architecture a NoSQL database will be used since it can achieve better speeds and can grow horizontally (Gudivada, Rao and Raghavan, 2014) which can fit better the needs of this solution since it will have to deal with a lot of requests from the various devices and the storage used will increase rapidly.

This solution has two databases, one to store the data from the devices and another one to store the predicted sensor data that come from other applications such as the ProSEcO platform.

The driver in the main application needs to provide to the other core elements methods that allow the communication with the database. For that, regarding each database, the driver needs to have a method to open and close the connections to the database, one to store the data received by the devices and another one to store the data sent by the ProSEcO learning algorithms. It also needs a method to find the desired data using the sensors' name and a date.

3.3.2 Data Communication Module

The data communication module will be composed by a Web Services architecture that will allow the interaction between all the different devices, applications and the core layer. In this architecture the REST principle will be used since it can offer more combability between different kinds of devices and is more lightweight, which can be an important aspect since many devices will be communicating (Zur Muehlen, Nickerson and Swenson, 2005).

This module represents the server side of the web services architecture and it is responsible for receiving the incoming requests and route them to the appropriate data processing method. For the proposed architecture to work it is necessary to offer a specific set of end-points to allow the communication between the devices and the server as well as external applications to retrieve the data. It is also necessary to make available end points for the ProSEcO platform to store the predicted data created by its learning algorithms and a way for external applications to show that data.

The necessary end-points to deal with the data generated by the devices need to:

- Receive the data from a device;
- Get a list of devices stored;
- Get values from a specific device and time interval;
- Get values from a specific device in real time.

The data collected from the different devices can be analysed by the ProSEcO platform and the predictions made will be stored in this application core so there is also the need to create services that allow the ProSEcO platform to store de data in this database. This data can then be consulted in the same way as the generated data from the devices but will be a prediction of what it is expected for the sensors values to be. Therefore, it is, once again, necessary to also have a service to:

- Store the data generated by the algorithm;
- Get a list of the devices stored;
- Get values from a specific device and time interval;
- Get values from a specific device in real time.

The real time end-points offer the possibility to request the last hour of data, so the user can have an overview of the data evolution, and also the possibility to request the new data updates since the latest request.

3.3.3 Data Processing Module

The data processing module is composed of the functions that will interpret the requests coming from the data communication module and will call the appropriate methods, from the data storage module, to store or retrieve the desired data.

The functions that deal with the data storage need to validate the incoming data and call the right method from the data storage module.

The functions that deal with the requests of data need to check how many dates are given to call the appropriate function. If one date is given, they request the data generated from that date and onward, if two dates are given they request the data generated between them.

Regarding the real time data requests, the functions needs to check if the request is for the last hour of data or for the new entries and retrieve the correct data.

In Figure 3.3 it is possible to see a sequence of how the modules of the core layer communicate between them when a request from an external application is made.

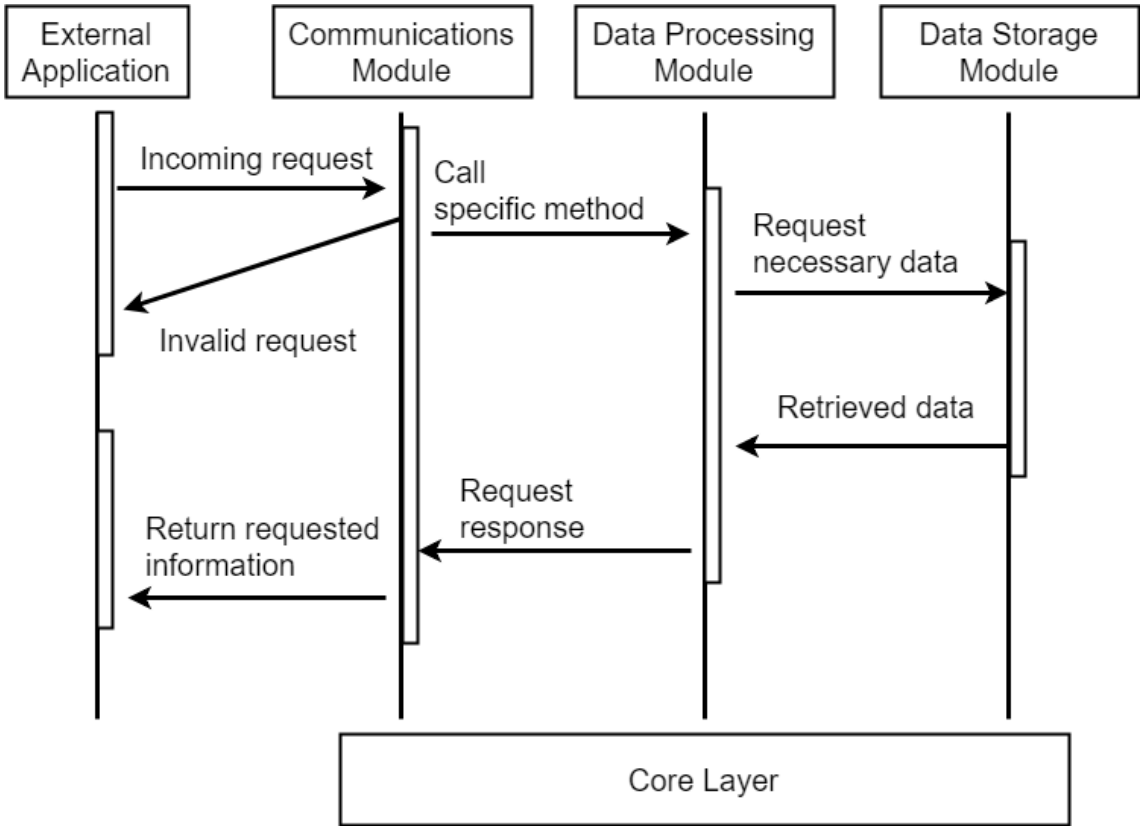

Figure 3.3 – Core Layer Sequence Diagram

3.4 Data Visualization Layer

The data visualization layer aims to provide a way for the users to see the retrieved data from the devices, as well as, the data generated by the learning algorithms. This layer also provides a way to visually analyse the data generated by the devices. With that in mind, this solution proposes a platform where it is possible to see the collected data in time series charts.

This layer also works as an external application since it gets the necessary data using only the available end-points provided by the core layer.

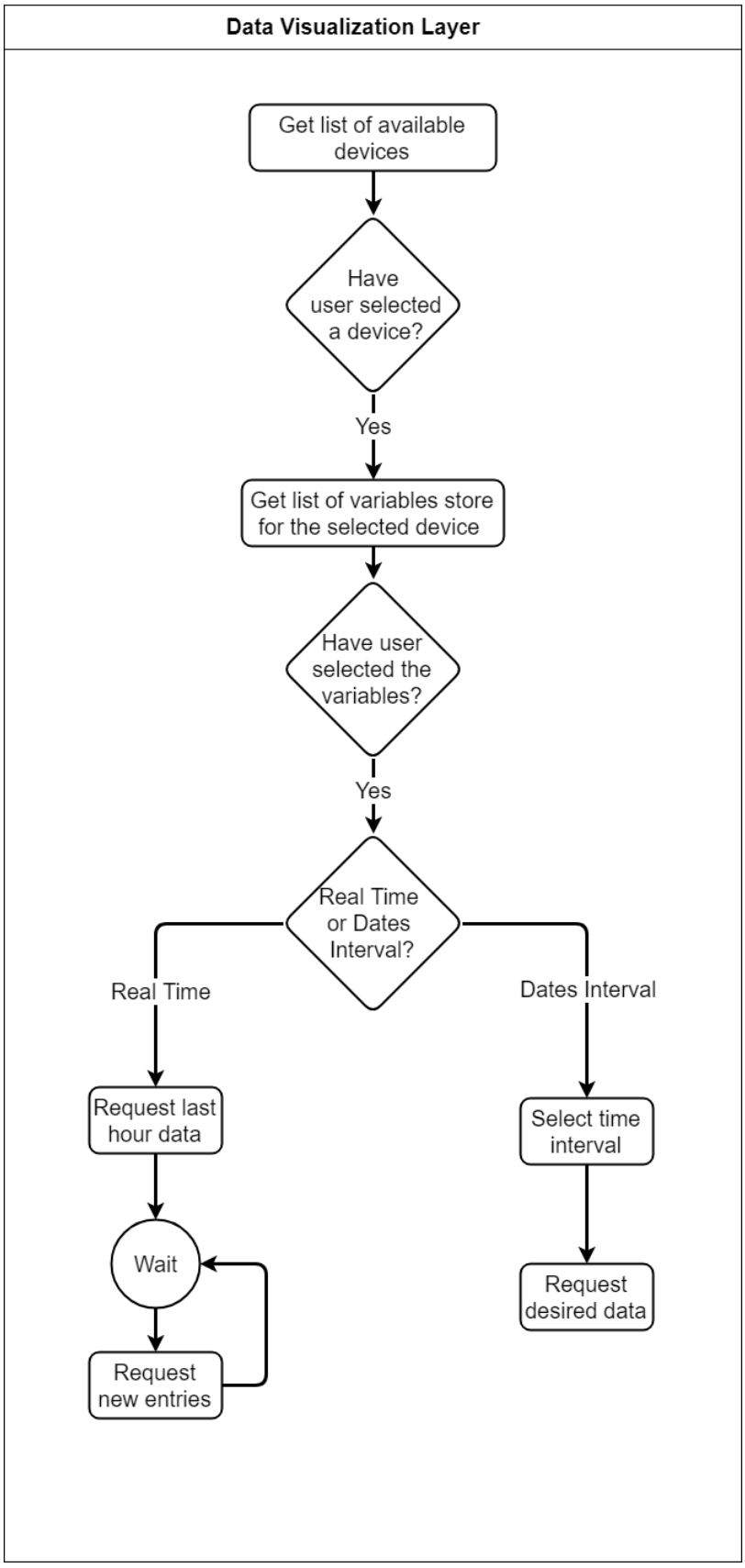

Figure 3.4 – Data Visualisation Layer main architecture

In Figure 3.4 the main behaviour of this layer is presented, although it offers more options that will be explained bellow.

The visualisation platform starts by requesting all the serial numbers available and gives the user the possibility to choose one. After that, it will request to the core layer all the variables available for that device and present them to the user. After the user select the desired sensors the platform will, by default, show a time series chart with the real time data. It starts by requesting the last hour data from that device and showing it on the chart, then, every predefined time (a few seconds), it will request the new entries for that device. The user has the possibility to stop this request, stopping the chart from changing, allowing him to better analyse some data.

If the user wants to see a specific time frame of data, it has an option to switch the type of chart. When that happens, the application will show a field to choose a start and an end date and will validate if the dates chosen by the user are correct (the start date is before the end date). If the dates are correct it will request the core layers for the desired data and show it in the chart.

The platform also gives the user the possibility to change the environment from the data collected from the device to the data predicted by the external algorithms. When this option is chosen the new list of available variables is requested and shown to the user. Then the user has the same options related to the charts that had regarding the data collected from the devices.

4

Implementation

This chapter will describe in detail the implementation of the proposed architecture explained in Chapter 3, it will maintain the same format separating the architecture in three main layers, data acquisition layer, core layer and data visualisation layer.

4.1 Data Acquisition Layer

The data acquisition layer will collect the data generated by the home appliance and will send it to the Cloud. The home appliance in use is a refrigerator from Electrolux capable of writing a CSV (Comma Separated Values) file with the state of the 130 sensors every 3 seconds.

A simple Java Graphic Interface was created to help the user start the application on the gateway connected to the refrigerator, test the link to the Cloud Server and choose the right CSV file.

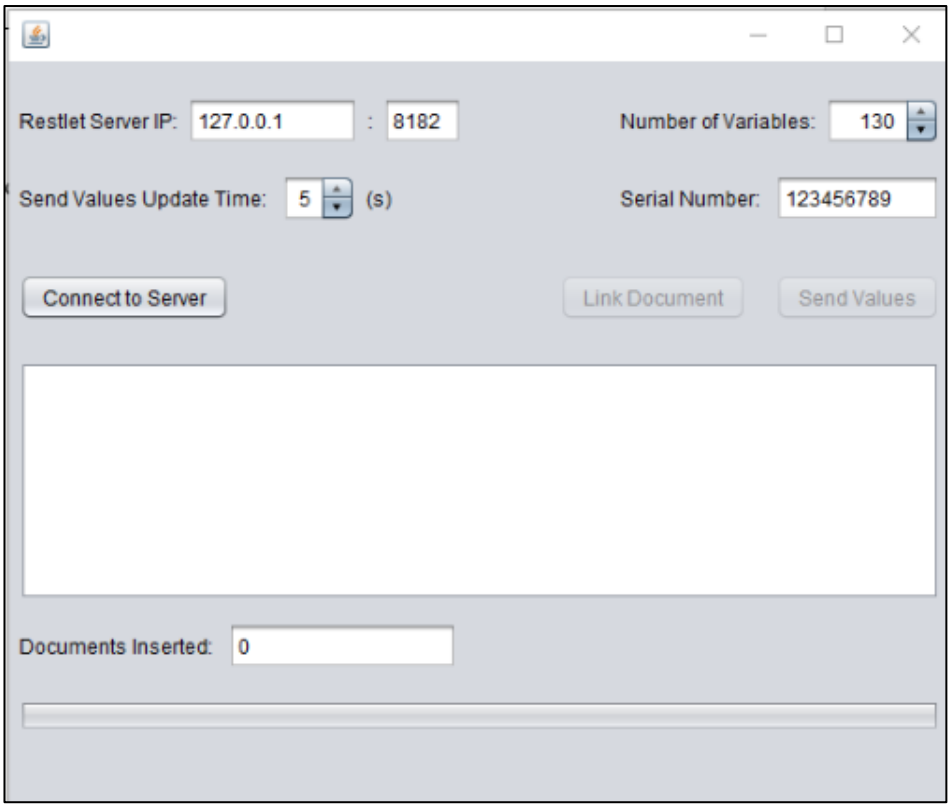

Figure 4.1 – Data Acquisition Graphical User Interface

The Figure 4.1 shows the GUI available for the user to start the data acquisition application. On this screen it is possible to set the server's IP address and port as well as the expected number of variables, the Serial Number of the home appliance ant the update time.

After all these parameters are set correctly the Connect to Server button can be pressed which will call the *connectToServer* function. As it can be seen in the Figure 4.2 the base URL for the server is defined using the IP and Port defined by the user. A new *ClientResource* is created and stored in the variable Client to be used during the flow of the application once it has the information needed to communicate to the server. The *CallGet* function is then called and it will perform a REST request Get method to check if the connection is working properly.

```
basePath = "http://" + IP + ":" + Port;

Client = new ClientResource(basePath);

return CallGet("/Electrolux");
```

Figure 4.2 – Code of *connectToServer* function

If the connection is valid the Link Document Button will be enabled and the user can select the desired document. After selecting the document the *OpenFile* function will check if the document meets the desired format.

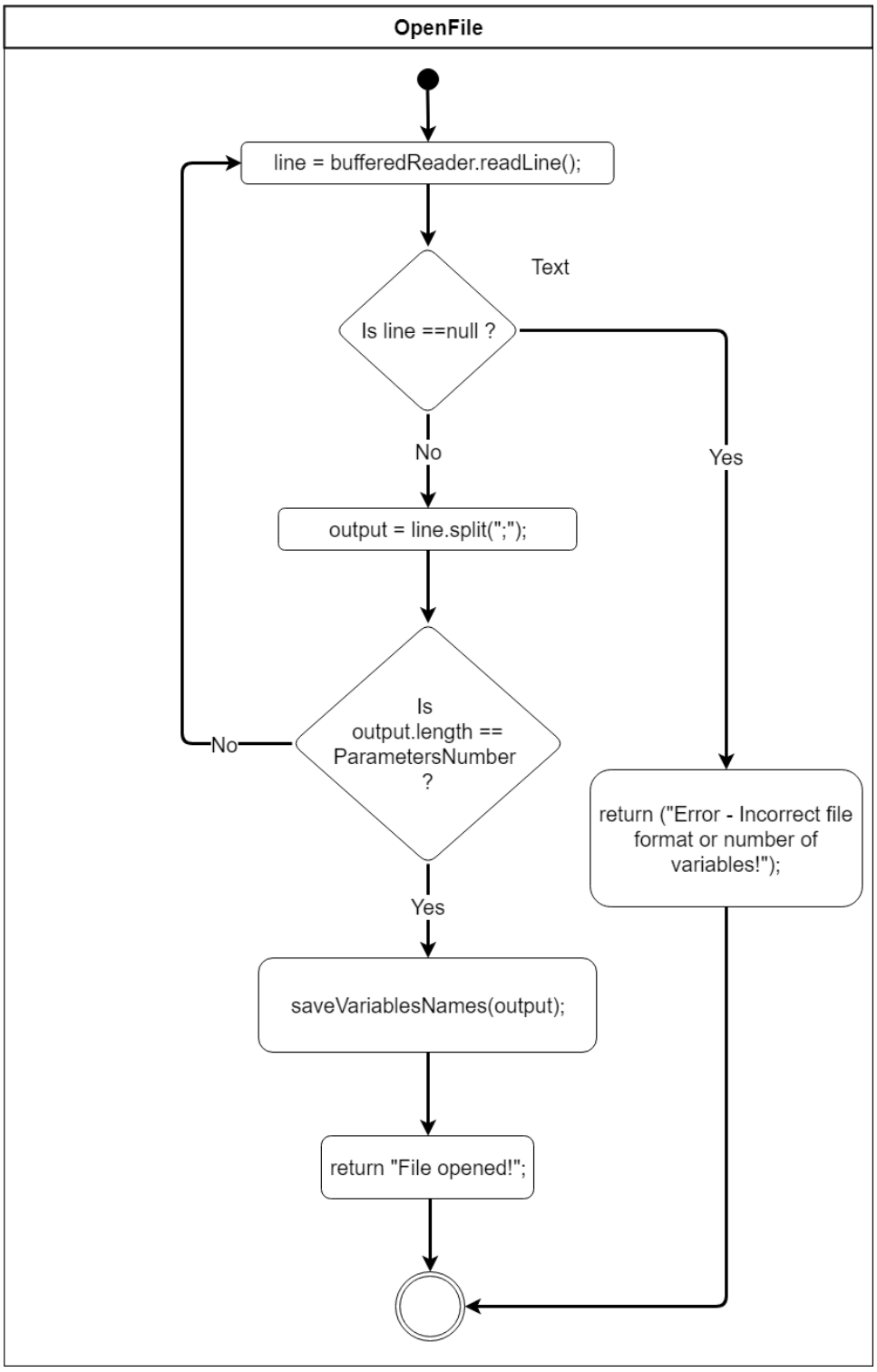

Figure 4.3 – OpenFile Implementation

The *OpenFile* function starts by opening the file given using the *FileReader* Java Library and then using the *BuffuredReader* Lybrary it reads one line from the file. If the line is equal to *null* then the file is invalid, if not, it will try to split the line by the ";" character. If it is split in the same amount given by the user in the Number of Variables the file is assumed as correct and a list of the variables name, including the serial number is stored in a local variable. If the Number of Variables is not met, then a new line from the file is read and the conditions tested again. If the end of the file is reached the *readLine* function will return *null* and the file is considered invalid. If the file is valid then the Send Values button will be enabled. When pressed by the user the *run* function from the *FileHandler* class is called. This function has a *while* loop that will run until the end of the program.

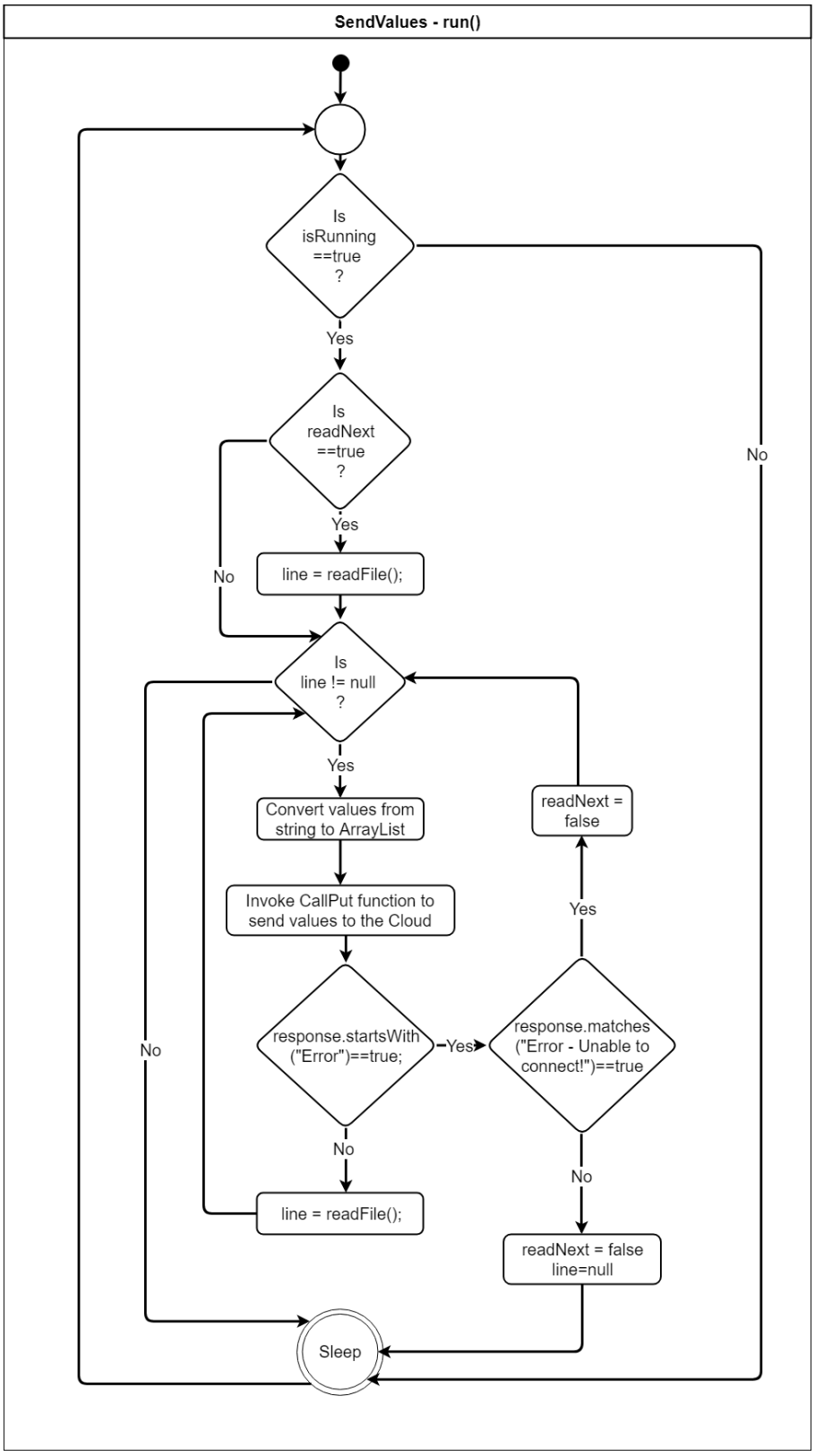

Figure 4.4 – SendValues – *run*() Implementation

The loop implemented by the *run()* function is a java thread that is sleeping during the specified time interval defined by the user on the interface. When the thread awakes it checks if the isRunning variable is true, that represents if the Send Values button was selected or not. If it is true then it checks if it should read a line from the file or not. If the *line* variable is different from null then it is converted to an *ArrayList* and it is sent to the Cloud using the *CallPut* function. If no errors occur, another line is read from the file and sent to the Cloud until there are no more lines (*line == null*). When the line variable is equal to null the thread goes back to sleep.

4.2 Core Layer

As it was explained in chapter 3 the Core Layer, although it should be seen as just one component of this architecture, can be separated in 3 main modules: Data Storage, Data Communication and Data Processing. The Core Layer will work as a server and is implemented as a Service Oriented Architecture that allows the easy communication with other applications.

To implement the server the Java programming language will be used, as well as, a MongoBD database to store the data. The Web Services used are based on the REST principle and will be implemented with the help of the Restlet Framework. This application is also made with the use of the Maven tool that works as the project's dependency management system.

4.2.1 Data Storage Module

The Data Storage Module is composed by a MongoDB database and some functions in the Server application. The connection between the Java Application and the MongoBD database is made with the help of the MongoDB Java Driver.

As it was explained in the Chapter 3 this architecture has two separate databases: one is called *Electrolux* and is used to store the data received from the devices. The other one is named *ElectroluxProseco* and is used to store the predicted data generated by the learning services of Electrolux.

Although there have been created two Java Classes to deal with each database only the functions that deal with the *Electrolux* database will be explained since the ones used with *ElectroluxProseco* work in the same way.

To work with the database it is firstly necessary to open a connection between the Java application and the MongoDB server. For that, the ***OpenConnection*** function is used. As it can

be seen on Figure 4.5 the function will try to connect to the database that, in this case, will be running in the same host, then it will try to acquire the desired database, *"Electrolux"*.

```
mongoClient = new MongoClient("localhost", 27017);
try {
    mongoClient.getAddress();

    // Now connect to your databases
    db = mongoClient.getDatabase("Electrolux");

    return true;
} catch (Exception e) {

    return false;
}
```

Figure 4.5 – OpenConnection implementation

With a connection now opened it is possible to store and retrieve data from the database. The MongoDB database stores a Document object that has a key-value structure. Each document can have as many pairs of key-values as it is needed and are stored under groups called collections.

For this solution each collection will represent a device, using its serial number as its name. The documents in each collection will represent a reading from all the sensors having as many pairs of key-value as the number of variables of the home appliance.

The ***InsertDocument*** function is used to store a reading in the database, receiving as parameters the serial number, a list with the variables name and a list with the respective values. The function starts by checking if the list of variables and the list of values have the same size. If they do, it creates a document with the data and stores it in the proper collection. If no, it returns an error.

The ***FoundDocuments*** function receives a serial number and two dates and returns an *ArrayList <Document>* with all the documents that it found between the two dates.

There is another ***FoundDocuments*** function that only receives one date with the difference being that it returns all the documents found after that date.

The ***getSerials*** function returns a list with all the serial numbers found in the database which corresponds to the name of the collections.

The ***getVariables*** receives a serial number and returns a list with the names of all the variables available for that specific equipment. For that, it retrieves one document of the given serial number and returns the name of all the key fields present.

To deal with the real time requests explained in chapter 3 it was necessary to implement the next two functions: *getLastHour* and *getNewEntries*.

The ***getLastHour*** function receives a serial number and a client key, that should be unique for each client requesting data to the server, and return a list with all the documents found. This function starts by retrieving the date and time of the last document saved for the specified serial number and then using the *FoundDocuments* function it retrieves all the documents between that hour and an previous one. The date of the last document is then saved in a *HashMap* along with the client key for future use.

The ***getNewEntries*** function also receives as parameter a client key and a serial number and returns a list with all the documents found. This function should be used after the *getLastHour* function since it returns the documents whose date is greater than the one stored in the *HashMap* for that client key. It will return *null* if the client key is not found in the *HashMap.*

4.2.2 Data Communication Module

This module is composed by the server that will receive the requests from external applications. To implement this server the Restlet Framework will be used which helps to easily implement REST APIs (Application programming interface) in a java application.

The Restlet server is composed by a Restlet Component that includes virtual hosts and applications. In Figure 2.1 it is possible to see how all these elements are combined.

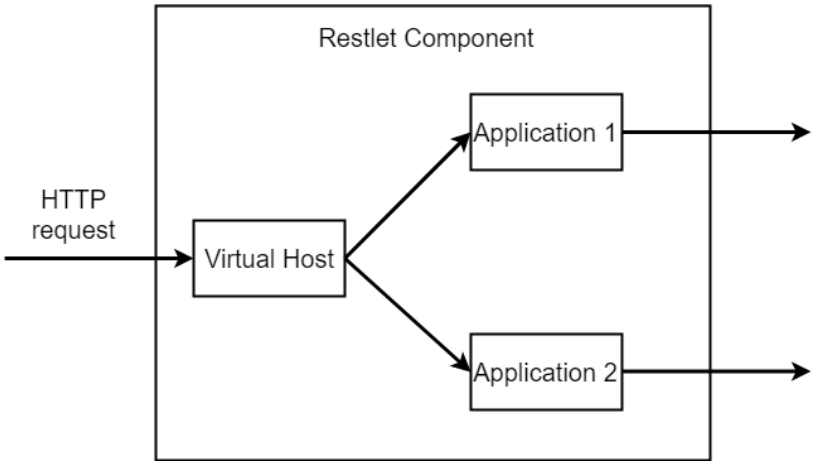

Figure 4.6 – Restlet Server Overview

The Restlet component will create the necessary environment for the other elements. In this case it will receive HTTP requests on the port 8182 and send it to the virtual host to be analysed which will then forward it to the right application.

For this architecture two applications are necessary, one to process the requests that deal with the *Electrolux* database, where the data from the devices is stored and another one to process the requests that deal with the *ElectroluxProseco* database, where the data generated by prediction algorithms is stored.

These applications work as a router forwarding the incoming requests to the right data processing algorithm. In each application it is defined the available APIs and to which *ServerResource* class are they forward.

All the HTTP requests made to this server have to start with the following format:

http://[RestletServerIP]:8182/Electrolux/…..

or

http://[RestletServerIP]:8182/Proseco/…..

The virtual host will forward the requests starting like the first case to the *Electrolux* application and the ones starting like the second case to the *ElectroluxProseco* application.

The *Electrolux* application has the following APIs implemented:

- http://[RestletServerIP]:8182/Electrolux/**StoreValues/{serial}**

  {serial} – The serial number of the device

This is a POST request method and the body should be a **Form** with the pair key-values to store.

- http://[RestletServerIP]:8182/Electrolux /**GetValues/{key}/{serial}/{type}**

  {key} – Client unique key

  {serial} – The serial number of the device

  {type} – The value can only be Variables or Sensors

This is a GET request method. If the {type} is Variables it returns an Array with all the variables available for that device. If the {type} is Sensors then it is necessary to add to the URL the desired Sensor's names and the dates or date. If only one date is given the system will return

all the data that is from date onward, if two dates are given then the data returned will be from the time between them.

These options should be added as parameters with the following format:

Date=yyyy-MM-dd_HH:mm:ss

Sensor=sensorName

The response of this request is an Array of Strings and with the String having the following format:

sensorName=value1;value2;…..;valueN;

As an example, if the desire is to get the values of the sensor *Cooler_Sensor_Temp* and the respective timestamps obtained after the date 1-02-2018 the URL would be:

http://[RestletServerIP]:8182/Electrolux/GetValues/{key}/{serial}/Sensors?Sensor=date&Sensor=Cooler_Sensor_Temp&Date=2018-02-01_00:00:00

- http://[RestletServerIP]:8182/Electrolux/**GetValuesRealTime/{key}/{serial}/{operation}**

  {key} – Client unique key

  {serial} – The serial number of the device

  {operation} – lastHour or update

This is a GET request method and needs as URL parameters the names of the sensors desired with the following format:

Sensor=sensorName

The response returned by this request is an Array of Strings and with the String having the following format:

sensorName=value1;value2;…..;valueN;

If the {operation} value used is lastHour the system will retrieve the values of the desired sensors regarding the last hour available in the system. The variable update should only be used after one request with the lastHour being made since that it will return new values in the system whose timestamp is greater than the last timestamp sent in the lastHour request.

- http://[RestletServerIP]:8182/Electrolux/**GetSerials**

This request is a GET method that will return an Array with all the available serial numbers in the system.

The endpoints implemented to work with the *ElectroluxProseco* database are equal to the ones described above having, as stated before, their beginning as the only difference.

4.2.3 Data Processing Module

The data processing module is composed by the implementation of the various java classes that deal with the Rest requests. Due to the Restlet Framework each endpoint is associated to a java class that extends the *ServerResource*. In each of these classes it is possible to create different functions to support the different requests methods Post, Get, Put, Delete, etc. For the framework to know which one to use it is only necessary to annotate it with the right type, e.g. @Put. In this solution it is only implemented the desired method for each endpoint having one class for each.

Has it happened with the previous modules only the classes that work with the Electrolux endpoints will be explained since the ones used to deal with the ElectroluxProseco worked in the same way with the exception that call the functions that deal with the other database.

The function *store* is responsible for processing the *StoreValues* request. As it can be seen the Figure 4.7 the function starts by retrieving from the HTTP request the serial number of the device and the form with the values to store. Then it stores all the sensors names in one *ArrayList* and all the values in another and sends them as parameters of the *InsertDocument* function that will store them in the database. If the operations succeeds a message saying that the file was stored is returned, if the operation is unsuccessful an error message is returned.

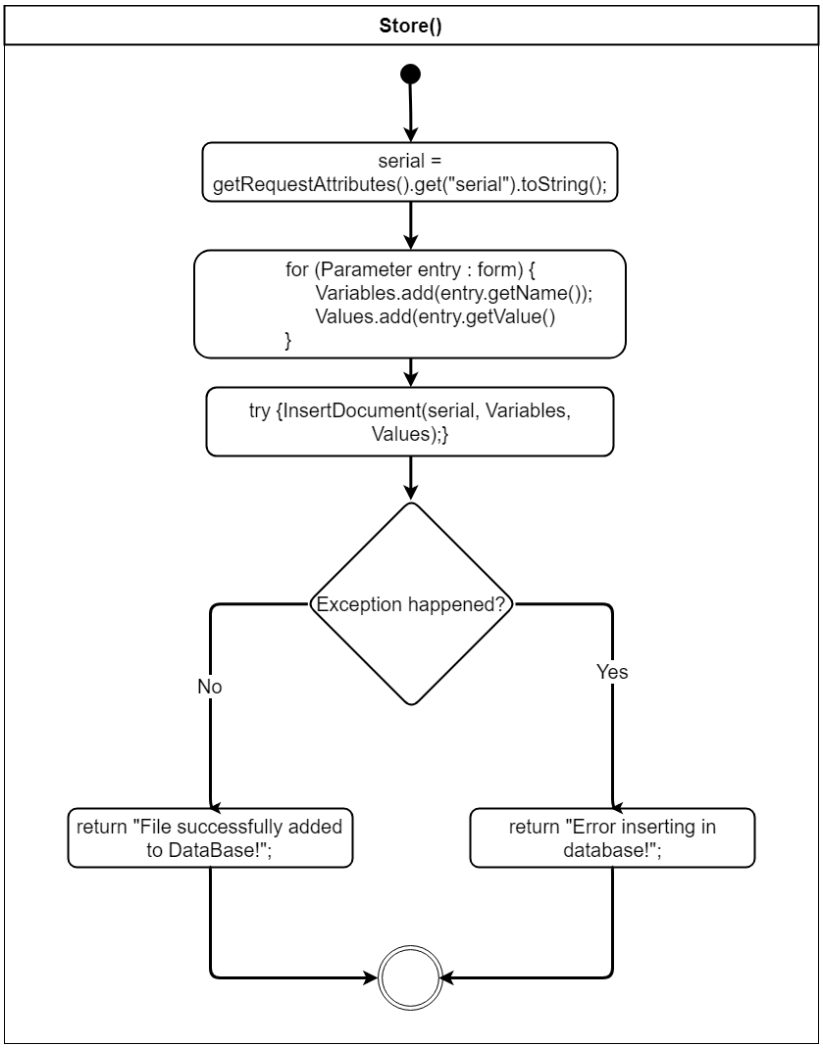

Figure 4.7 – *Store* function implementation

The *getVariables* function, described in the Figure 4.8, is responsible for processing the *GetValues* request. It starts by getting the serial number, client key and the type from the requests parameters. Then checks if the type given is “Variables”. If it is, the function will retrieve all the names of the sensors available for the device with the given serial name by calling the getVariables function. If the type given is “Sensors” the function will get the sensors’ names and the dates given as parameters in the URL of the request and then analyse the number of dates given. If the number of dates given is one it will call the *FoundDocuments* that only receives one date as parameter and that will return all the values found for the desired sensors whose dates are from after the given date. If two dates are given the *FoundDocuments* function that receives two dates will be called and will return all the values found between them. After this process the *getVariables* function will return an ArrayList of Strings that will contain the name of all the sensors or the values for the requested sensors.

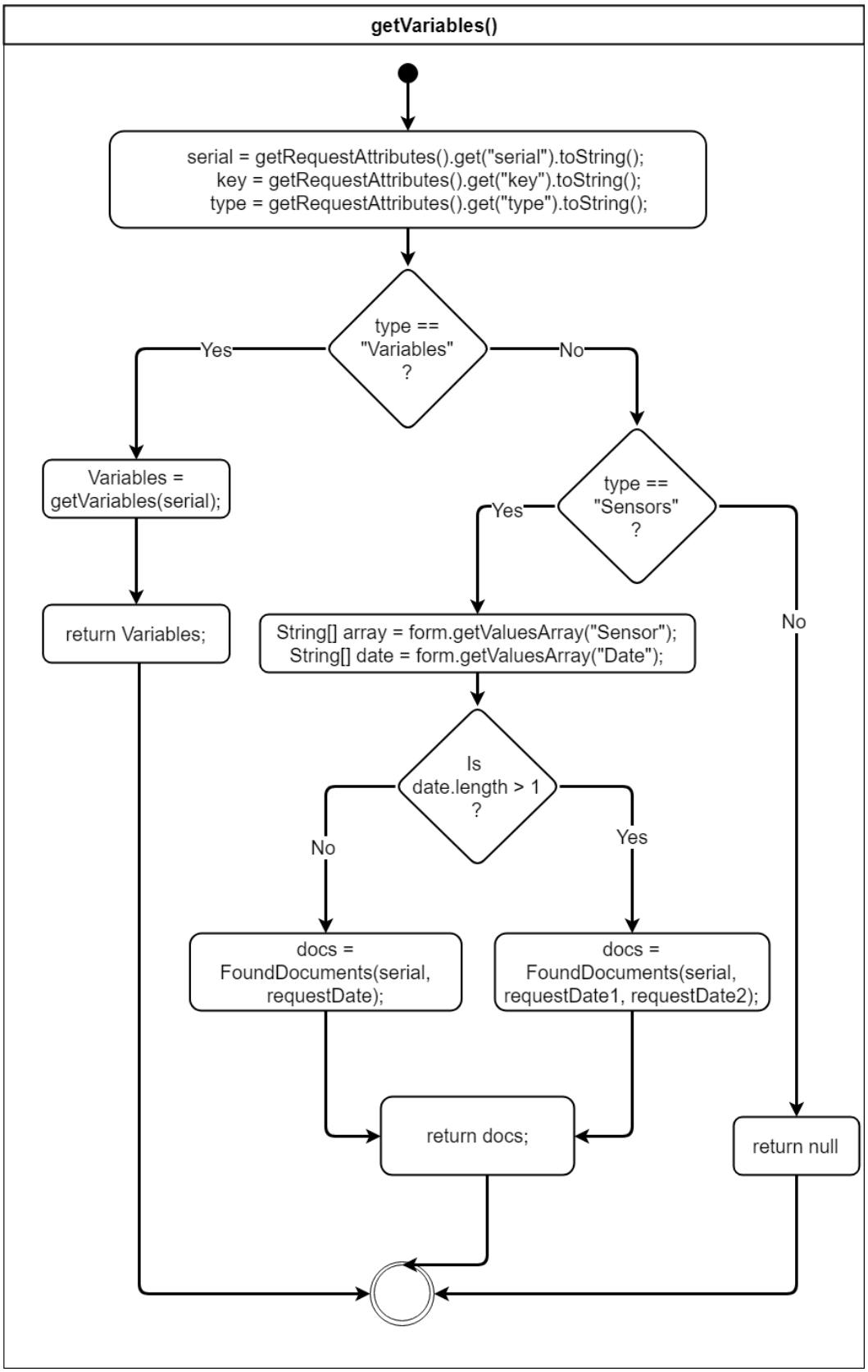

Figure 4.8 – *getVariables* function implementation

The *getValues* function is responsible to process the data that comes from the *GetValuesRealTime* request and its behaviour is described in the Figure 4.9. It starts by getting the serial number, the client key, the operation type and the sensors' names from the requests parameter. Then it will check if the operation given is "lastHour" or "update", if it is "lastHour" it will call the *getLastHour* function that will return the last hour of recorded data of the requested sensors, if the operation is "update" it will call the *getNewEntries* function that will return new data obtained since the date of the last value sent by the *getLastHour* function. This function will then return an ArrayList of Strings that will contain the values for the requested sensors.

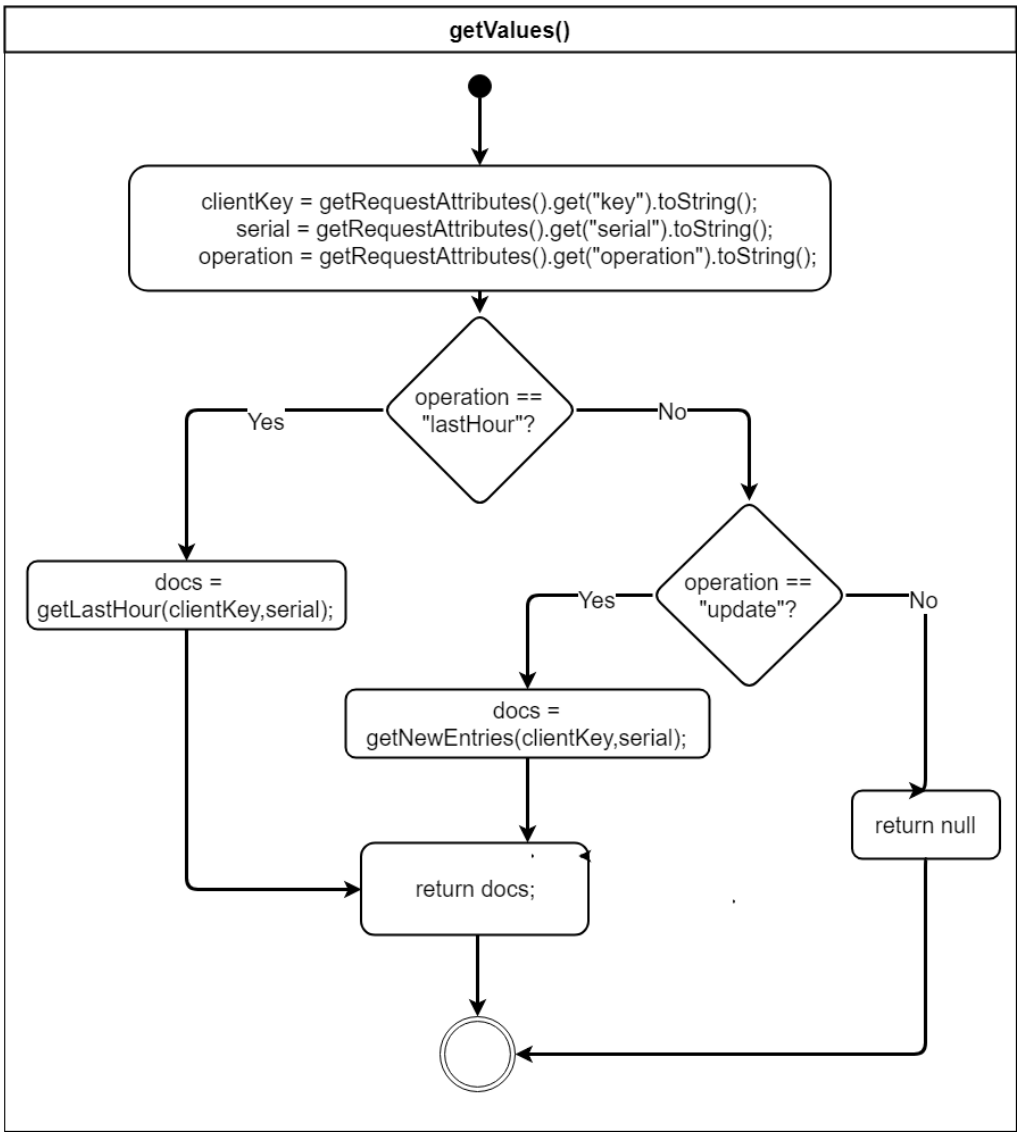

Figure 4.9 – *getValues* function implementation

There is also the *getSerials* function that is responsible for processing the *GetSerials* request. This function only needs to call the *getSerials* function from the Data Storage Module that will retrieve all the serial numbers stored. Then this function returns an ArrayList of Strings with all the serial numbers found.

4.3 Data Visualisation Layer

The data visualisation layer works on this architecture as an external application that will present the collected data in graphical forms, so the user can analyse is and maybe take some conclusions.

This layer is implemented as a web site, so it could be accessed online from different types of devices. This web site has three main pages: one to choose the desired device, one to see the data collected from the devices and another one to see the data generated from the data mining tools of the ProSEcO platform.

This web site was programmed in Java using JSP (Java Server Pages), HTML, JavaScript, and some CSS (Cascading Style Sheets) libraries such as Bootstrap and the Google Charts. The GlassFish Server will be used to host this web site.

The usage of JSP pages became very useful since they allow java code to be embedded in the HTML that is then processed on the server side of the application allowing better dynamic pages.

When the user enters this site for the first time it will be prompted to choose a home appliance from a dropdown list as it can be seen in the Figure 4.10 .

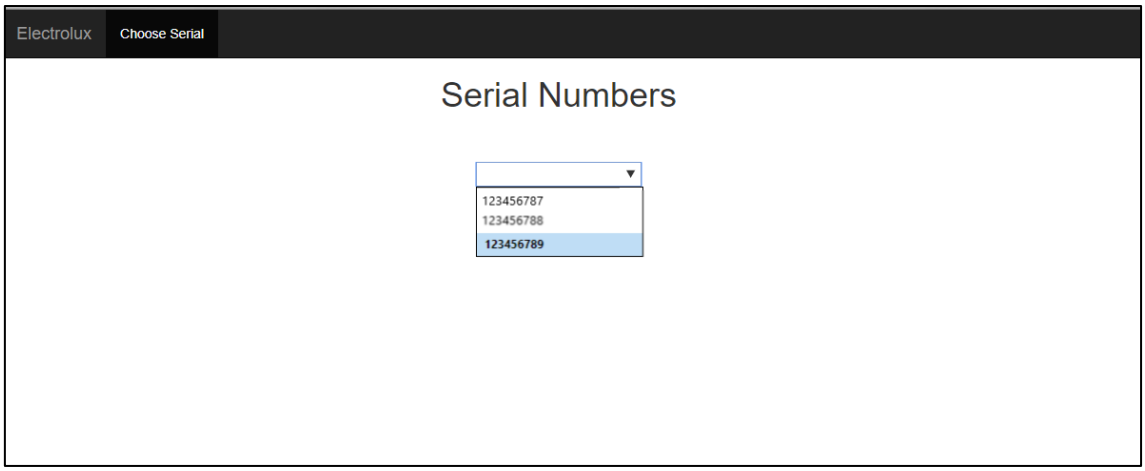

Figure 4.10 – Web page to choose the device

The platform presents this page because it searched for a cookie called *Serial* and did not found it. The list of serial numbers is requested using the appropriate API exposed by the Core Layer.

After the user selects a serial number the main page of the web site is shown as it can be seen on the Figure 4.11.

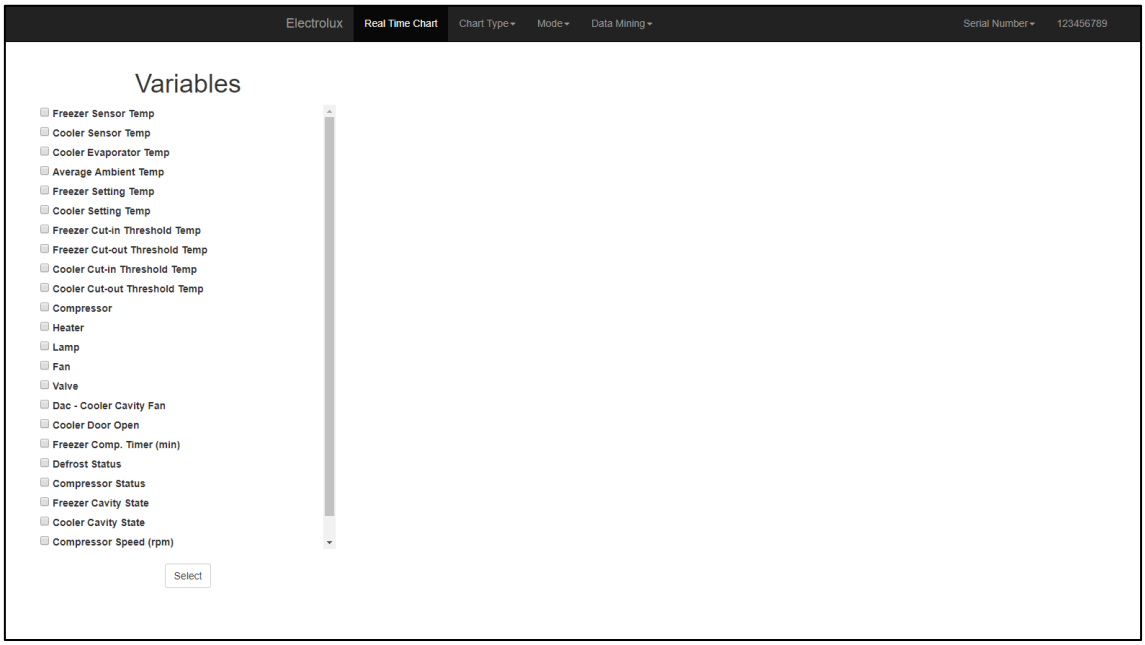

Figure 4.11 – Web Site main page

This page is composed in three main parts:

- Navigation Bar – Allows the user to change the type of graphic, alternate between the collected data and the predicted data and change the device;
- Variables List – Allows the user to select the variables of the device that he wishes to see on the chart;
- Chart Area – the area on the right were the charts required will appear.

By default, the type of graphic that will appear is the real time chart. When the user chooses the desired variables and press the Select button an HTTP request is made to the core layer requesting the last hour of data for that sensors and a chart is displayed with that. After that the browser will send a request for new entries of that sensors every 5 seconds.

If the user wishes to pause the new data income to better analyse the presented data, an option on the "Chart Type" tab will stop the browser from continuing to request new data until the user activates the function again.

As it was said, the user can also change the type of graphic switching from a Real Time chart to a Between Dates chart. When that option is chosen a bottom area will appear for the user to select the desired dates as it can be seen on Figure 4.12.

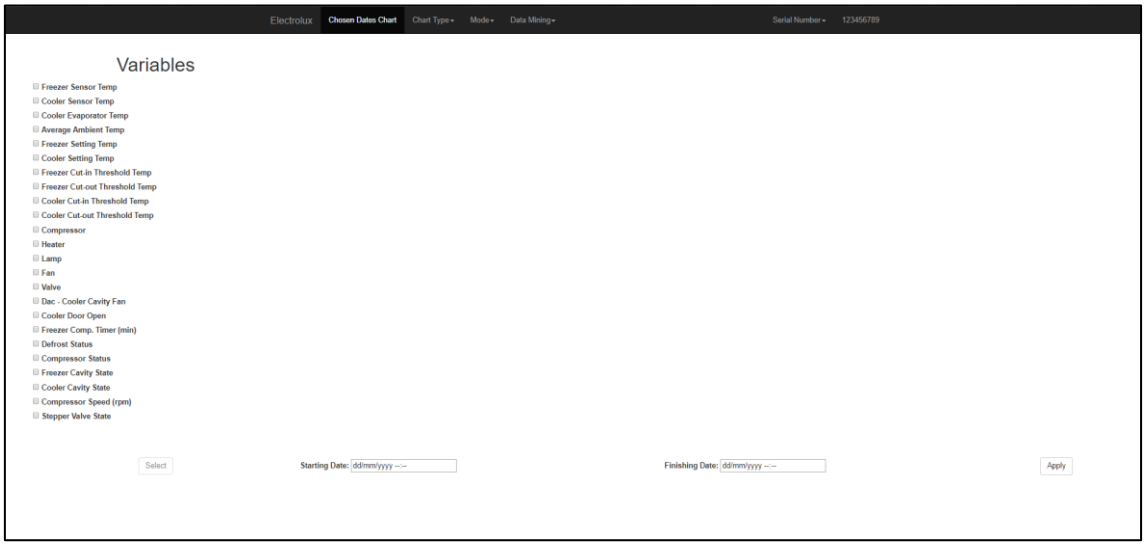

Figure 4.12 – Page Layout when between dates chart type is selected

For the application to request the data, the user has to select the two dates and press the Apply button that will check if the starting date is before the finishing date. If the dates are valid

the user can press the select button and the desired data will be request to the core layer using the Get Values request.

In the Figure 4.13 it is possible to see an example of the implemented chart that will present the requested data in three different ways:

- The data that comes with a decimal point is considered to be data coming from analog sensors and will be presented in the chart as a continue line;
- The data that comes without a decimal point is considered to be data coming from boolean digital sensors, that only return 0 or 1, and will be presented as a continue line with the area billow painted;
- The data that comes as text is presented bellow the chart and only the last value received is presented.

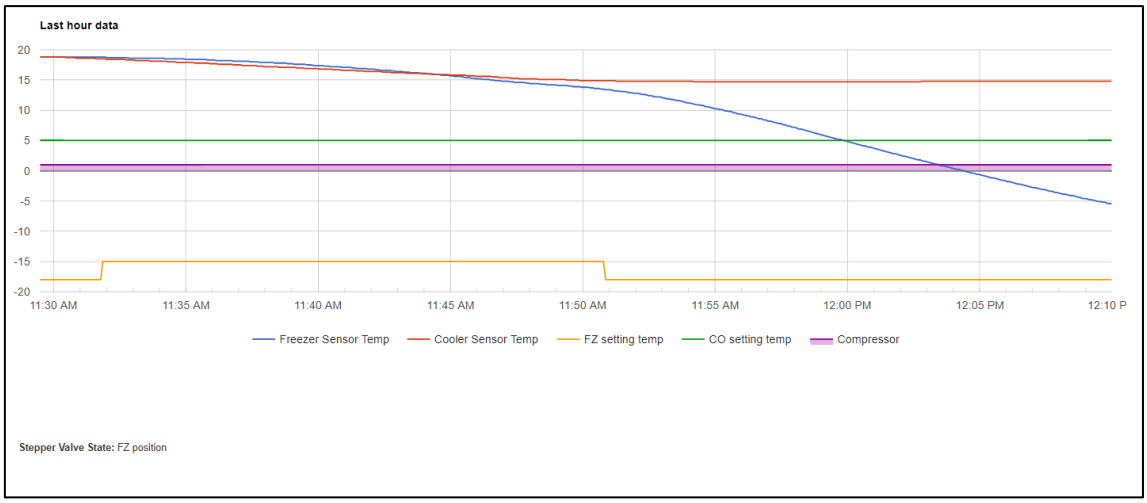

Figure 4.13 – Example of the implemented chart

On the Navigation bar is also possible to switch to the called Data Mining mode as it can be seen in the Figure 4.14. When the user turns on this mode a cookie is set that will indicate the application that the requests made should be for the data predictions end-points. When this happen a new list of variables is requested using the appropriate request. The user then will have the same functions as the ones described above but the data retrieved will be regarding the data predicted by the learning algorithms.

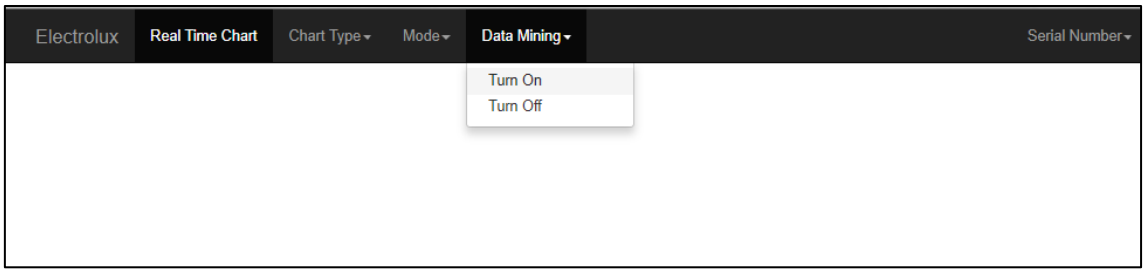

Figure 4.14 – Data Mining mode switching

5

Results and Analysis

In this chapter, the results of the implementation of the proposed architecture in the previous chapters will be displayed. To do so, images of the data visualisation layer will be displayed showing the data stored in the Cloud that was collected from a refrigerator with the data acquisition layer.

It is possible to see that the architecture does not rely on a specific type of home appliance, other appliances besides the one presented can be used in this system without changing the architecture, being the only condition that they have to report the data to the core layer as described in chapter 5.

The tests presented in this chapter were performed with an AEG refrigerator, model S83930CMX2, supplied by Electrolux that monitors and saves the state of 130 variables. To have access to these variables it is necessary to use a specific device to connect the refrigerator to a computer and a software that will process the data received by the device and generate a CSV File with that information.

The device used is called an Appliance Mini Interface (AMI) and it is a USB module that connects the appliance to the PC. It allows the communication between them using the DAAS (Domestic Appliance Acquisition System) communication protocol from Electrolux.

The Figure 5.1 shows the flow from the data since it is generated by the refrigerator until being saved on the CSV file.

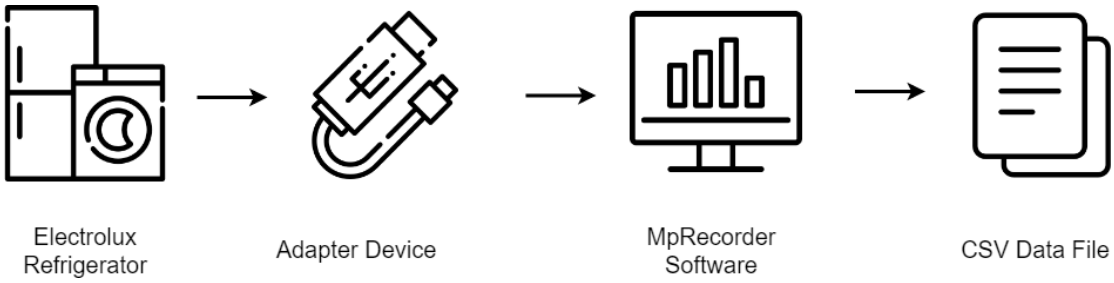

Figure 5.1 - Components needed to retrieve data from the fridge

For this case study we will look and test only six of these variables:

- Freezer Sensor Temperature – represents the real temperature of the freezer compartment (in Celsius);
- Freezer Setting Temperature – represents the temperature set for the freezer compartment (in Celsius);
- Cooler Sensor Temperature – represents the real temperature of the cooler compartment (in Celsius);
- Cooler Setting Temperature – represents the temperature set for the cooler compartment (in Celsius);
- Compressor State – represents the state of the compressor, 0 when it is off, 1 when it is on;
- Cooler Door State – represents the state of the cooler door, 0 when it is closed, 1 when it is open;

The data generated by this refrigerator will be collected and sent to the Server by the implemented Data Acquisition Application, the Server will store the data and make it available. The proposed web site will be used to present the collected data in order to check that the status of the fridge was properly captured and stored.

The first step was to start the Server that will expose the Web Services by using the Interface provided by the implemented application as it is shown on Figure 5.2.

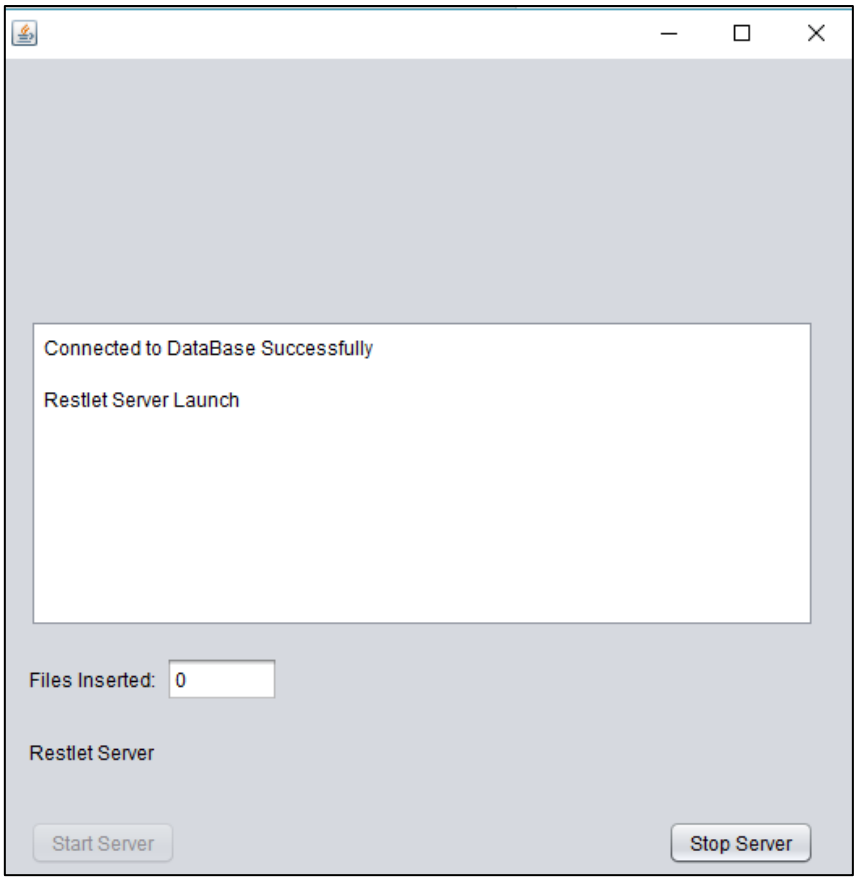

Figure 5.2 – Server Application window

The next step was running the Data Acquisition Application, shown in Figure 5.3, and using the provided interface connect it to the server and chose the right file created by the refrigerator so the application could read and send the data.

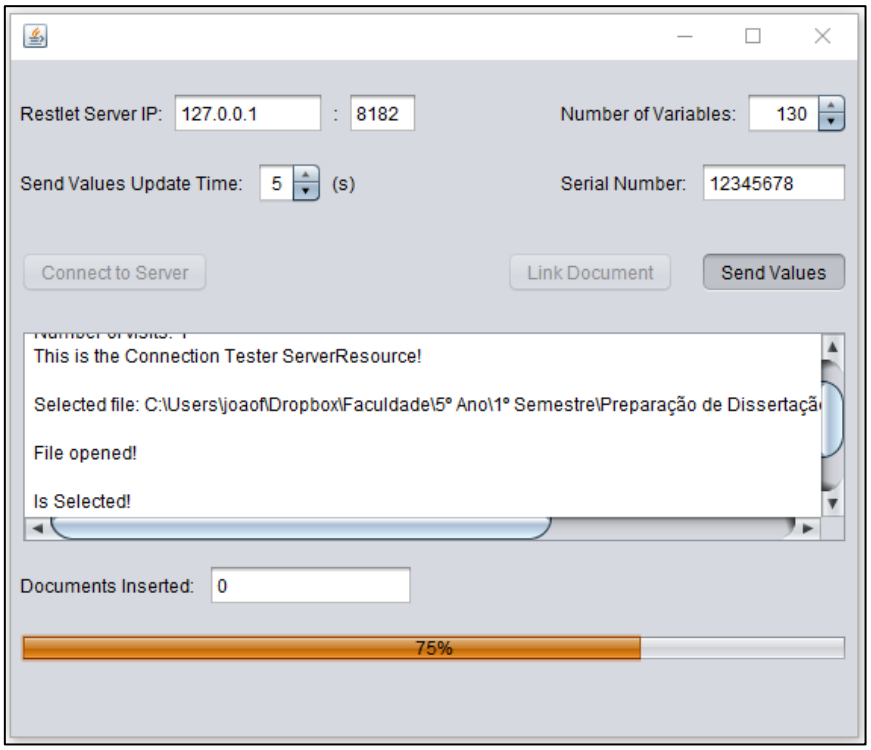

Figure 5.3 – Data Acquisition Application window

The next step was to start the Data Visualization Application and select the device in use. With all the tree layers of this architecture running some tests were made to check if the solution was working as expected.

For the first test the Freezer Setting Temperature will be changed on the refrigerator. The desired variable is selected on the web-page and the real time chart is requested. After obtained the request chart the temperature setting is changed from -15ºC to -18ªC and some seconds later the changed is reflected on the chart. On Figure 5.4 it can be seen the change made.

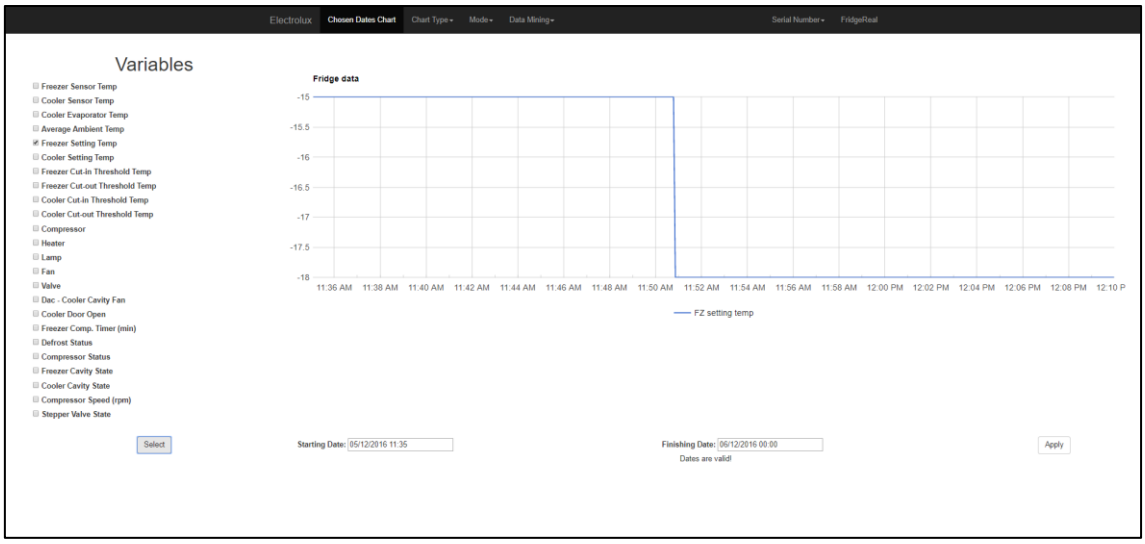

Figure 5.4 – Change Freezer Temperature test

The next test made was to open and close the door and check if the changes were properly collected by the system. The *Cooler Door Open* variable was selected, and the chart was requested. After that the refrigerator's cooler door was open and closed after a few seconds. After some time the changes were reflected on the chart. in Figure 5.5 it is possible to see that change on the door sensor as well as that the application recognized the variable as a boolean sensor and it is show with a painted area bellow the line.

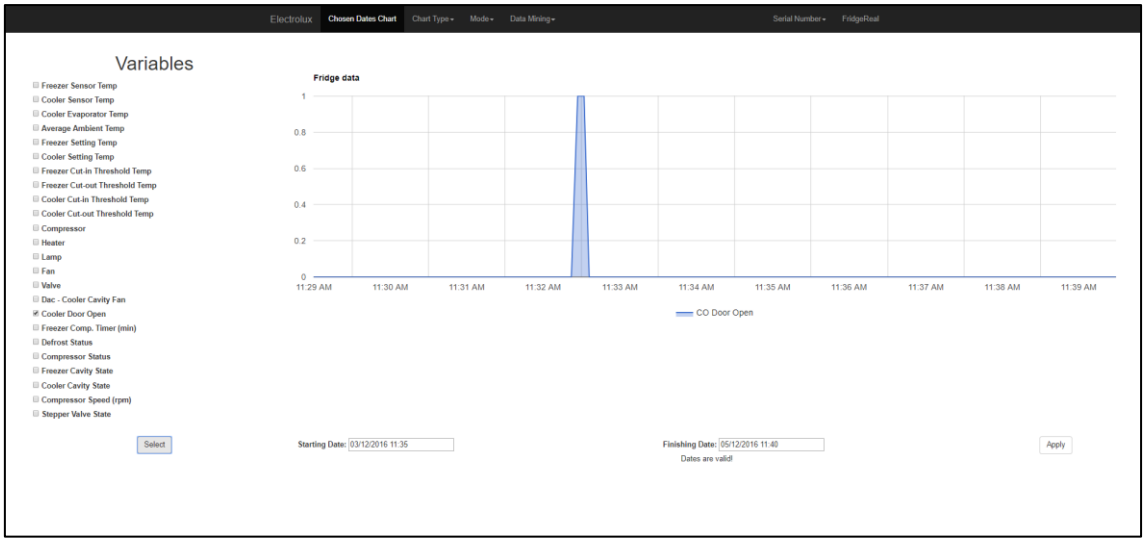

Figure 5.5 – Open Door test

For the next test the refrigerator was left working for some time without making any change to it in order to see the normal behaviour of the appliance. Then the *Freezer Sensor Temperature, Freezer Setting Temperature, Cooler Sensor Temperature, Cooler Setting Temperature* and *Compressor State* variables were selected and a chart requested for a certain period of time. As it can be seen in Figure 5.6 is possible to see that when one of the temperatures went higher the accepted interval the compressor is turned on and both temperatures drop for the set temperature.

Figure 5.6 – Normal behaviour of the refrigerator test

With the tests made it was possible to conclude that the proposed solution worked as expected. As anticipated, it was necessary to wait some time for the real time data to appear on the chart, this is due to the sampling rates and refresh requests. In the worst cases the data can take 13 seconds to be displayed in the chart.

6

Conclusions and Further Work

6.1 Conclusions

As shown in chapter 5 the architecture presented in this document proved to be an effective solution to gather, process and display data generated from home appliances, creating new ways to improve user usage and optimize the products features.

This architecture effectively turns an unintelligent home appliance into an IoT device with the use of a simple gateway, capable of communicate with the *Cloud* through a Service Oriented Architecture that shown capability of receive, process and store the data as well as making it available to other services. The data analysing through graphic charts in the data visualisation layer also proved to be an effective way to understand certain behaviours of the appliances.

The results displayed in Chapter 5 were successfully obtained using the architecture presented in Chapter 3 and the implementation described in Chapter 4.

The work done on this document, contributed the writing of a conference article, presented in the International Conference of Industrial Informatics (INDIN'2017) conference which has been published in the end of 2017 (Lima-Monteiro *et al.*, 2017).

6.2 Further Work

This solution has presented a way to turn unintelligent home appliance into an IoT device and have the data collected for posterior analysis.

For further work, solutions to exploit these data can be made. The goal is to create patterns to develop usage profiles and to notify the user of possible delays and obstacles the appliance may run into but also to help the manufacturer, who can better optimize the product and its functionalities.

With the collected data we can discover potential problems and malfunctions and make a more predictive maintenance to solve or prevent them from happening. It is possible to discover unreliable parts of the machine that are causing more problems and thus change the production method and parts if necessary. With the user data it is possible to adapt the appliance functions to match the user lifestyle, for example, the refrigerator's defrost cycle can change accordingly to usage patterns, choosing a less used time to execute.

References

Aljazzaf, Z. M., Capretz, M. A. M. and Perry, M. (2016) 'Trust-based Service-Oriented Architecture', *Journal of King Saud University - Computer and Information Sciences*. King Saud University, 28(4), pp. 470–480. doi: 10.1016/j.jksuci.2015.12.003.

Alonso, G. *et al.* (2004) 'Web Services', in *Web Services*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 123–149. doi: 10.1007/978-3-662-10876-5_5.

Ashton, K. (2009) 'That "internet of things" thing', *RFID journal*, 22(7), pp. 97–114.

Atzori, L., Iera, A. and Morabito, G. (2010) 'The Internet of Things: A survey', *Computer Networks*. Elsevier B.V., 54(15), pp. 2787–2805. doi: 10.1016/j.comnet.2010.05.010.

Baheti, R. and Gill, H. (2011) 'Cyber-physical Systems', *The Impact of Control Technology*, (1), pp. 161–166. doi: 10.1145/1795194.1795205.

Barry, D. K. and Dick, D. (2013) *Web services, service-oriented architectures, and cloud computing : the savvy manager's guide*. Morgan Kaufmann. Available at: http://www.sciencedirect.com/science/book/9780123983572 (Accessed: 27 January 2017).

Belqasmi, F. *et al.* (2012) 'SOAP-Based vs. RESTful Web Services: A Case Study for Multimedia Conferencing', *IEEE Internet Computing*, 16(4), pp. 54–63. doi: 10.1109/MIC.2012.62.

Binani, S., Gutti, A. and Upadhyay, S. (2016) 'SQL vs. NoSQL vs. NewSQL- A Comparative Study', 6(1), pp. 43–46.

Botta, A. *et al.* (2016) 'Integration of Cloud computing and Internet of Things: A survey', *Future Generation Computer Systems*. Elsevier B.V., 56, pp. 684–700. doi: 10.1016/j.future.2015.09.021.

Channabasavaiah, K., Holley, K. and Tuggle, E. M. (2004) 'Migrating to a service-oriented architecture', *IBM DeveloperWorks*, (April), pp. 1–22. doi: 10.1109/ICWS.2004.1314715.

Di, G. *et al.* (2016) 'A Platform to Support the Product Servitization', *International Journal of*

Advanced Computer Science and Applications, 7(2), pp. 392–400. doi: 10.14569/IJACSA.2016.070254.

Georgakopoulos, D. and Papazoglou, M. P. (2008) *Service-Oriented Computing*, *Service-Oriented Computing*. The MIT Press.

Gubbi, J. *et al.* (2013) 'Internet of Things (IoT): A vision, architectural elements, and future directions', *Future Generation Computer Systems*. Elsevier B.V., 29(7), pp. 1645–1660. doi: 10.1016/j.future.2013.01.010.

Gudivada, V. N., Rao, D. and Raghavan, V. V. (2014) 'NoSQL Systems for Big Data Management', in *2014 IEEE World Congress on Services*. IEEE, pp. 190–197. doi: 10.1109/SERVICES.2014.42.

Hazenberg, W. and Huisman, M. (2012) 'Meta Products: Building the Internet of Things', p. 160.

Hecht, R. and Jablonski, S. (2011) 'NoSQL evaluation: A use case oriented survey', *Proceedings - 2011 International Conference on Cloud and Service Computing, CSC 2011*. IEEE, (December), pp. 336–341. doi: 10.1109/CSC.2011.6138544.

Joseph Bradley, Barbier, J. and Handler, D. (2013) 'Embracing the Internet of Everything To Capture Your Share of $ 14 . 4 Trillion', *Cisco Ibsg Group*, p. 2013. Available at: http://www.cisco.com/web/about/ac79/docs/innov/IoE_Economy.pdf.

Lee, J., Bagheri, B. and Kao, H. A. (2015) 'A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems', *Manufacturing Letters*. Society of Manufacturing Engineers (SME), 3, pp. 18–23. doi: 10.1016/j.mfglet.2014.12.001.

Li, S., Xu, L. Da and Zhao, S. (2015) 'The internet of things: a survey', *Information Systems Frontiers*, 17(2), pp. 243–259. doi: 10.1007/s10796-014-9492-7.

Li, Y. and Manoharan, S. (2013) 'A performance comparison of SQL and NoSQL databases', *IEEE Pacific RIM Conference on Communications, Computers, and Signal Processing - Proceedings*, (November), pp. 15–19. doi: 10.1109/PACRIM.2013.6625441.

Lima-Monteiro, P. *et al.* (2017) 'ProSEco as a data processing platform: A service-oriented architecture for data analysis', in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, pp. 1205–1210. doi: 10.1109/INDIN.2017.8104945.

Mattern, F. and Floerkemeier, C. (2001) 'From the internet of computers to the internet of things', *From Active Data Management to Event-Based Systems and More. Papers in Honor of Alejandro Buchmann on the Occasion of His 60th Birthday*, pp. 242–259. doi: 10.1007/3-540-68339-9_34.

Mell, P. and Grance, T. (2011) 'The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology', *National Institute of Standards and Technology*, p. 7. doi: 10.1136/emj.2010.096966.

Zur Muehlen, M., Nickerson, J. V. and Swenson, K. D. (2005) 'Developing web services choreography standards - The case of REST vs. SOAP', *Decision Support Systems*, 40(1 SPEC. ISS.), pp. 9–29. doi: 10.1016/j.dss.2004.04.008.

Papazoglou, M. P. and Van Den Heuvel, W. J. (2007) 'Service oriented architectures: Approaches, technologies and research issues', *VLDB Journal*, 16(3), pp. 389–415. doi: 10.1007/s00778-007-0044-3.

Rajkumar, R. *et al.* (2010) 'Cyber-physical systems: The next computing revolution', *47th ACM/IEEE Design Automation Conference (DAC)*, pp. 731–736. doi: 10.1145/1837274.1837461.

Risteska Stojkoska, B. L. and Trivodaliev, K. V. (2017) 'A review of Internet of Things for smart home: Challenges and solutions', *Journal of Cleaner Production*. Elsevier Ltd, 140, pp. 1454–1464. doi: 10.1016/j.jclepro.2016.10.006.

Sha, L. *et al.* (2009) 'Cyber-Physical Systems: A New Frontier', in *Machine Learning in Cyber Trust*. Boston, MA: Springer US, pp. 3–13. doi: 10.1007/978-0-387-88735-7_1.

Shi, J. *et al.* (2011) 'A survey of Cyber-Physical Systems', *2011 International Conference on Wireless Communications and Signal Processing, WCSP 2011*. doi: 10.1109/WCSP.2011.6096958.

Suda, B. (2003) 'SOAP Web Services', *Retrieved June*. Available at: http://pop.suda.co.uk/publications/MSc/brian.suda.thesis.pdf.

Wang, W. *et al.* (2012) 'A Comprehensive Ontology for Knowledge Representation in the Internet of Things', in *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 1793–1798. doi: 10.1109/TrustCom.2012.20.

Want, R. (2006) 'An Introduction to RFID Technology', *IEEE Pervasive Computing*, 5(1), pp. 25–33. doi: 10.1109/MPRV.2006.2.

Yashiro, T. *et al.* (2013) 'An Internet of Things (IoT) architecture for embedded appliances', *2013 IEEE Region 10 Humanitarian Technology Conference, R10-HTC 2013*, pp. 314–319. doi: 10.1109/R10-HTC.2013.6669062.

Zhang, Q., Cheng, L. and Boutaba, R. (2010) 'Cloud computing: State-of-the-art and research challenges', *Journal of Internet Services and Applications*, 1(1), pp. 7–18. doi: 10.1007/s13174-010-0007-6.

ZHANG, Y. *et al.* (2013) 'High Fidelity Virtualization of Cyber-Physical Systems', *International Journal of Modeling, Simulation, and Scientific Computing*, 04(02), p. 1340005. doi: 10.1142/S1793962313400059.