Evaluation of a Semantic IoT Platform for Reasoning and Big Data Analytics

Burak KILIC

Master in Innovation and Research in Informatics (MIRI)

Universitat Politècnica de Catalunya (UPC)

Supervisor

Prof. Fatos Xhafa

In partial fulfillment of the requirements for the degree of

Computer Networks and Distributed Systems

June 23, 2019

Acknowledgements

I would first like to thank thesis advisor Prof. Fatos Xhafa of the Barcelona School of Informatics, at the Polytechnic University of Catalonia. The door to Prof. Xhafa's office was always open whenever I had a question about my research or writing. He allowed this work to be my own and pointed me in the right direction whenever he presumed I needed it.

I would additionally like to thank experts who were involved in this research project, UPC academic and administrative staff, and all my professors. Without their support, the thesis could not have been successfully conducted.

I dedicate my thesis work to all my family and many friends who have always been a source of my motivation, dedication, and determination. A special feeling of gratitude to my loving parents, Nuri, Muzeyyen, and Busra KILIC for their words of encouragement and believing in me to achieve my true success during the entire master's program.

I also dedicate this dissertation to many professionals who have supported me throughout the process. I will always appreciate everything they have done for me. Especially, Enric Staromiejski Torregrosa for assisting me to develop my skills further, along with the (SEMBU) Team in everis. A very special thanks to Bigle Legal Family (Biglers) for their sincere support and motivation, and Prof. Fatos Xhafa once again, for the many hours of proofreading.

Last but not least, I would like to thank my best friend Burak Kengil, Patrick Schneider, Alhassan Ali, and Shams Methnani for being there for me throughout the entire master's program. All of you, have been my best cheerleaders.

Abstract

In recent years, the Internet of Things has been evolving and increasing as a trending technology. Since the idea of tagging things with RFID (Radio-Frequency Identification) and connecting them to the Internet, the technology has evolved at a swift pace, which has nowadays led to a multitude of IoT connected devices by various protocols. The IoT is a primary source of information generation thanks to the diversity of the available devices on the market and their rather low cost. Most notably, IoT devices are sources of data generation through sensing. Data processing in terms of both online and offline mode is a challenge due to the scale, data rate, and a variety of these devices. In this context, semantic processing is a useful means to enrich and aggregate the generated data for their processing and building intelligence from big data and/or big data streams.

However, semantically enriching the information requires additional processing steps and handling issues arising in big data and big data stream processing. Historically, managing big data could be a downside that has been tackled in Cloud computing. The idea of pushing all the information from IoT devices directly to the Cloud is no longer an option for many applications with demanding requirements on real-time response, low latency, and energy-aware processing. This is further propelled by the forthcoming 5G technologies, which bring

the capability of processing massive data streams at the edges of the Internet.

In this project, the aim is to spot tasks that could be offloaded from the Cloud Systems and pushed towards the edges of the Internet. To tackle this, the IoT-Edge-Cloud layered system that is conducted by four main layers is created where each layer has been distributed into different platforms. Thus, the real-life implementation scenario under the real-time requirements is studied for processing CAN (Control Area Network) bus dataset to detect potholes in the roads. From the given background, the anomaly detection algorithms are integrated with the machine learning approach. Also, a semantic enrichment process that generates various types of semantic annotations has been implemented. Moreover, in order to define a specific set of actions that need to be taken within the context of management of the system and reasoning over the existing data, the business rule engine is implemented.

In the context of business rule engine implementations, using reasoning techniques and fact-action based architecture is beneficial for both technical and business authorities. The usage of Machine Learning algorithm for prediction case scenarios has an exciting outcome as well. The results are promising in this study; However, they also come with unexpected results in some cases. It is pointed out that the differences between local and real-life scenario implementation constraints can create some challenges in terms of handling network connections, performance, and reliability. Hence, distributed platform architecture is beneficial to identify and solve these challenges. The model that is mentioned above has been tested on various operating systems and

hardware configurations to measure the performance and assess the feasibility of the designed architecture. As future work, to maximize the outcome of this research topic, the entire implementation can be switched to another development environment that is more suitable for creating IoT applications.

Contents

List of Figures

Chapter 1

Introduction

The number of IoT devices connected to the Internet in a multitude of applications is increasing exponentially. There is an ever-increasing need to process real-time data streams generated by such devices as well as to establish control mechanisms to increase the quality and consistency of the services. The aim, in this project, is to deploy an IoT data streaming processes in a real-life computing environment to test various features: interoperability, viability, and performance. Several layers from the IoT to an end user are considered to implement, and each layer is distributed in a real-life infrastructure under the real-time data processing requirements.

The motivation of the thesis comes by first acknowledging three main challenges from IoT systems:

1. The information that is generated by different devices can cause an intercommunication problem between them [1]. Our analysis focuses on this challenge and tries to provide an answer to this problem by enriching the data with semantic web technologies.

2. The resource-constrained IoT devices have restricted computing capability to process critical tasks. The second challenge is to take over the part of the

processing load from the Cloud-based systems and distribute it to the edges of the internet, where data is generated.

3. A distributed IoT Cloud system might generate high latency and high response time due to its physical distances between servers and IoT devices. Moreover, the latency might get higher as the network traffic increases. The goal is to keep latency as low as possible in the distributed system by using designed IoT layers.

The Internet of Things (IoT) enables an infrastructure for sensor deployment but also provides better communication among the connected sensors. The data that is generated by these sensors are enormous and continuously produced at a high rate. That requires mechanisms for continuous analysis in real-time in order to build better applications and services. We can classify sensors with three different types for data streaming, particularly, (i) Physical Sensors, (ii) Mobile and Wearable Sensors, (iii) Virtual Sensors and Social Media Streams [2]. Additionally, data streams are usually classified as single or multiple data streams depending on the number of sensors involved.

Among the above three categories, mobile and wearable sensors are harder to integrate within enterprise communication systems. It is not only due to technical integration issues and interoperability but also due to their dynamic nature and continuously changing context. Mobility and location-based sensory input might result in a higher level of unpredictability and a lower level of control over the distributed infrastructure that defines enterprise communication systems. These challenges are satisfied by new openings for IoT-enabled collaboration and communication systems to be designed in order to sense the context of a mobile user and take decisions according to dynamic sensory input [2].

The edge computing platform is taking advantage of its hardware resources to handle some of the processes in a distributed manner. In this case, they are

pre-processing and semantic enrichment. It is located near the terminal devices and supply services. The freshly added new hardware resources might balance the work of the system and improve computing capability.

Measurements of the system performance are one of the critical aspects of this project. The system should be inter-operable between different operating systems. Data loss should be minimal or better none during the data transfer session. The data stream rate should be defined with a specific limit so that the system can handle the workload without crashing.

1.1 Research Questions

In this project, we intend to answer three research questions:

- **How to implement real-time IoT data streaming and semantic data enrichment processes in a real-life infrastructure using edge devices?**
- **How to implement anomaly detection or detection of events of interest in IoT data streams in real time using edge devices?**
- **How do we measure the performance of data streaming and semantic data enrichment processes in an IoT-Cloud system?**

1.2 Thesis Structure

The rest of the thesis document is structured as follows. In chapter 2, we present our motivation to solve these problems, the scope and the objectives of this project. In chapter 3, the software requirements specifications are surveyed. Next, in chapter 4 the detailed architecture and implementation of each layer of the IoT-Cloud system have been studied. The deployment of the architecture in real-life infrastructure and the important concepts of this implementation have been studied in chapter 5. In chapter 6, some experiments are conducted in order to test the capabilities and limitations of the proposed system. We present the results obtained regarding these tests in the real infrastructure of RDLab [3] at the Department of CS, UPC. Chapter 7 presents the main conclusions of this work and Chapter 8 concludes the thesis with a discussion about future work.

Chapter 2

Motivation, Scope and Objectives

2.1 Motivation

Nowadays, many devices keep connecting to the Internet, and the number of connected devices will be even more in the very near future. IHS (Information Handling Services) forecasts that the IoT market will increase from an installed base of 15.4 in 2015 to 30.7 billion devices in 2020. The expected number for 2025 is 75.4 billion devices [4].

Several companies of the automotive and IT sectors such as DENSO Corporation, Ericsson, Intel Corporation, Nippon Telegraph (NTT), Toyota Info Technology Center and Toyota Motor Corporation predicted that the data size among vehicles and the cloud will reach to 100 petabytes per month around 2025, almost 10,000 times larger than the existing volume [5]. This increment will trigger the need for new architectures of network and computing infrastructure to promote distributed resources and topology-aware storage limit. These companies also launched the AECC (Automotive Edge Computing Consortium). One of the principal goals of the organization is to develop an ecosystem for connected cars to encourage emerging services such as intelligent driving, the production of maps

with real-time data and driving assistance based on cloud computing [5].

Regarding this thesis work, we intend to connect cars to the Internet that are communicating through mobile networks and sending data to edge computing devices [6], which are located near the roadways and highways. The reason why we use the edge devices is to reduce the workload over the cloud processing as it would be too complicated and slower to process all the information without pre-processing. Moreover, without using edge computing devices, the local processing power that car sensors have, might not be enough to handle the processing of the generated data continuously. We consider here the real problem of detecting potholes in the roads from the data stream received from cars. The cars will have a network connection to these receivers in order for their data to be processed by the edge platforms.

Nowadays, 4G technologies are dominating the interconnected devices with at least 40% coverage. The countries such as the United States of America, Japan, South Korea, Norway, and the Netherlands can reach over 90% coverage [7]. 5G technology is under experiment and poised to launch in some markets later this year. Mobile operators are anticipating with a mixture of resignation and expectation. They know that it will open opportunities to gain value from new 5G use cases and widespread confirmation of the Internet of Things (IoT) [8].

As an outcome, what motivates us to build an IoT ecosystem is to have significant control over real-time data streaming and processing stages. By doing this, it is conceivable to gain knowledge of how the standard IoT systems are performing with edge computing methodologies. It is also a part of the thesis motivation to see how semantic enrichment can add intelligence to systems by processing data streams.

2.2 Scope

The scope of this project contains several study cases regarding creating a real-life implementation of the IoT system architecture. These cases cover adding new algorithms such as anomaly detection, by implementing the Hierarchical Temporal Memory (HTM) algorithm [9] to predict patterns of the pothole occurrences, business rule engine implementation, creating a simple user web application to visualize the incoming data and performance measurements of the system.

In real-life implementations and testing cases, sometimes the expected results from the research outcomes might not match entirely. A system that is built and tested only in a local machine cannot give an accurate overview of the system behavior. In order to reduce the gap, it is crucial to test data sending and receiving performances in different scenarios to push the system and see if there is any compromise by considering a real-life infrastructure as an evaluation platform.

It should be noted that our real-life infrastructure does not include collecting data from the sensors within the car for anomaly detection. Instead, the generated real-life dataset is used for the study to simulate a data stream that is coming from the cars in order to simplify the setup.

There are requirements that are essential such as security, data privacy, and energy efficiency, which are very important in the context of IoT real-time data stream processing [10]. However, they are not in the scope of this thesis work.

2.3 Objectives

The project has several objectives that consist of functional improvements, adding new functionality and evaluating the IoT based system in a real-life infrastructure by using Raspberry Pi [11] and a cluster of RDLab UPC. Our starting point is a previous thesis work [10]. In this project, we extend the work in several directions:

(a) Pothole Example (Ref: Express)

(b) A Scenario of Crashing into the Pothole (Ref: OttawaCitizen)

Figure 2.1: Pothole Examples

(1) by implementing new functionality for anomaly detection in real time through the HTM algorithm; (2) by implementing semantic data enrichment and sending it to a server; (3) by implementing server-based modules that communicate the edge layer with a server node layer and (4) a real infrastructure deployment and evaluation of the whole system. The implementations and evaluations are to be done using Raspberry Pi and server nodes at RDLab of the Department of CS at UPC.

Based on the given constraints above, the problem statement definition is:

How to offload the components of the data processing and enrichment from Cloud platforms to lower levels of IoT, such as, to the edges of the Internet?

In a more technical view to explain; We aim to formulate the problem as the specification, design, and implementation of new layers in the architectural stack. From the IoT layer to the Cloud application layer, where part of the processing and reasoning over IoT data stream, can be distributed to the edge platforms to avoid performance bottlenecks, achieve efficiency and scalability under the real-time requirements.

There were some limitations of the previous thesis work in the context of efficiency and scalability, such as:

- The system was tested to work only on Windows OS which reduces efficiency.
- The system was bounded to one local computer and its hardware limitations. It is unavoidable that at some point, the limited hardware configuration would fail to process data when the system reaches the maximum processing capacity.

Finally, regarding the objective to study and evaluate the performance related parameters in a real-life infrastructure, these include:

1. Efficiency
2. Event-based Detection
3. Semantic Data Enrichment
4. Geolocation Information Generation
5. Car Sensory Data Process
6. Latency Measurement
7. Data Loss Measurement
8. Data Stream Rate Processing Capacity

In the following part, we explain the list of objectives in depth regarding their relevance to this thesis work.

Efficiency: In this case, the optimum usage of available hardware is taken into account.

In the context of efficiency, the goals are:

- Achieving efficient processing of incoming data streams at Raspberry Pi edge device.
- Establishing efficient communication among various parts of the system.
- Using distributed platforms to achieve better control and management of the system instead of a single local computer unit.

Event-based Detection: One of the main aspects of this project is the event processing. Therefore, a typical knowledge model (i.e., Semantic Technologies) is going to be conceived. Additionally, the system focuses on the process of the data in a semantic and event-based way. In order to define these events, the complex event process approach is used.

Semantic Data Enrichment: We aim to integrate semantic enrichment processes to enable reasoning over the data. In this case, the requirement is to generate new information from the given data is obligatory. The reasoning element is restricted to reason over streams in a specific time period. A rule engine is considered to implement in order to generate new information based on the existing data.

Geolocation Information: Car navigation systems that are based on GPS (Global Positioning Systems) can be used to generate the latitude and longitude information through satellite systems [12]. Nowadays, these systems are preferred by most of the car manufacturers, and as well as by third-party application developers in order to track the locations of the cars, in case of an accident, stolen car, or security concerns. In this project, we aim to measure the anomaly scores and the detection times to analyze the results. However, without the location

information, the system is limited. For instance, it wouldn't be possible to identify if the potholes occurred in a city, in a village or if it is in the higher traffic density area or not. Based on the location information, we can reason over the existing data to generate a specific set of actions that can be taken in case of a critical scenario, such as accidents. Therefore, besides the anomaly scores, and detection times, we aim to integrate geolocation information in the system to be able to track the location of the pothole occurrences.

Car Sensory Data Process: The use-case and one of the goals for this project is the processing of car's sensory data in order to detect potholes on the road. This implies some assumptions regarding both the data set and the environment of the sensor node. The sensor node is not a resource-constrained device, but as a power source that is readily available in a car. However, this is not a limitation for the study as any IoT sensory data can be handled through the edge platform. The nature of the data imposes a streaming context with heterogeneous data, as a car has a wide variety of sensors [10]. For this reason, the optimal usage of available hardware is considered. It is not the aim to optimize processing algorithms, nor is it to lower the overall processor usage of each component in the system. In this regard, edge computing devices that we mentioned in the motivation part, is crucial to enable the usage of all system components [13].

Latency Measurement: We aim to test the time differences between sending and receiving data. In mobile networks, the quality of the internet connection might vary depends on the antenna positions and location. For example, if the device location is outside of the 4G Mobile Broadband Coverage Area, the device will regress to a 3G Mobile Broadband connection, if available. When connected to 3G Mobile Broadband, the device does not encounter the same velocity or throughput as when connected to 4G Mobile Broadband [14]. There are several network delay aspects in an engineering discipline to consider. In this thesis work,

we tested the system through **processing delay** and **transmission delay** [13].

Data Loss Measurement: Another objective is to keep traffic and cars safe. A noticeable data loss in the context of pothole detection causes missing anomaly detection. This might create conflict between historical data and real-time data or even cause an accident, which is more critical than a conflict.

Data Stream Rate Processing Capacity: Regarding data loss measurements, if the data stream rate is too high, there can be a bottleneck or the data packages can be lost, or processing can be delayed. On the contrary, if the data stream rate is too slow, then the hardware cannot be used with maximum performance. Therefore, this objective aims to find an appropriate data stream rate that can be efficiently processed in the considered infrastructure.

Chapter 3

Software Requirements Specifications

3.1 Introduction

The need for the cars sensory information specifies a clear goal of the processing: Detection of potholes in the roads as well as the challenges of big data analysis and IoT real-time data streaming.

3.1.1 Purpose

In this section, we aim to provide a description of the software that is developed during this thesis work. The context will focus on the features of the software, the interfaces, what the software will do, and some of the constraints that it needs to operate.

3.1.2 Document Conventions

This section is created based on the IEEE template for System Requirement Specification Documents [15].

3.1.3 Intended Audience and Reading Suggestions

We can classify the target audience into three categories:

- Students, who want to research on semantic data enrichment processes, edge computing platforms, and real-time IoT data streaming.
- Advanced/Professional Users, such as engineers or business authorities, who want to use this software for testing or analysis.
- Programmers or future Master/Ph.D. thesis candidates who are interested in working on the project to develop it further or fix existing bugs.

  This thesis consists of both research and practical software implementation sections. As it is mentioned in the objectives of the project, the IoT real-time data processing, and semantic data enrichment topics are studied. Therefore, to have a clear understanding of this part of the thesis, it would be beneficial to have a background of software development, modular programming, and understanding the main principles of the back-end technologies(i.e., client-server communications, data transfer among different systems).

3.2 Overall Description

As it is previously mentioned before, this software is developed for anyone who is interested in semantic data enrichment processes, edge computing platforms, and real-time IoT data streaming.

It is an open-source project, and it is also open for feedbacks and reviews for anyone interested in contributing more. During the development process, several modifications have been applied in the software in terms of changing the libraries and operating system-dependent variables and their use cases. The

implementation of the software is more generalized to be able to work on various operating systems without any problems or strong dependencies. Hence, it works on Microsoft Windows 10 OS, Ubuntu 18.04 LTS, Raspbian OS, and Mac OS Mojave 10.14.4 which are the latest stable versions as the date of this project is being built. The reason why we chose to use these four operating systems is that they are commonly used operating systems as in 2019.

3.2.1 Design and Implementation Constraints

The principle of the software is to have a modular design where all the scripts are wrapped into separate modules. These modules are communicating with each other through an API Server. There are several APIs available to make the development easy and have great control over different parts of the application. The major hardware component of this architecture is a Raspberry Pi. Since the pre-processing and edge processing modules are carried in this component, there are some constraints that we need to take into account to deploy the system.

The implementation constraints are:

- **Synchronization:** The Open-SSH protocol is used to connect Raspberry Pi, Ubuntu, or Mac OS X compatible computers. These computers are built as the main platform, and also a virtual machine running on MS Windows OS to test with modifiable components both in terms of hardware and software where the data sending event resides.

- **Interoperability:** The system should be able to work on major operating systems as is indicated in the previous constraint, and regarding the project objectives.

- **Memory:** The edge device will have 1GB of a physical hard drive. The data that is going to be processed, cannot exceed this amount. The device

will have an SD card slot, and the software must be able to read data and write to that slot.

3.2.2 Assumptions and Dependencies

The software is developed by using Python programming language and therefore requires Python to be installed in the user's system. The latest stable version of this software requires Python version 3.5 or higher on Microsoft Windows OS, Ubuntu, and Raspbian. On Mac OS X, Python bundles with the application.

3.3 External Interface Requirements

3.3.1 Hardware Interfaces

Edge computing infrastructure and its functionality are significant parts of this project. We decided to use a Single Board Computer (SBC) as an edge platform carrier. There are few options to consider when it comes to SBC selection. These options might be Odroid XU4, UDOO Bold, Raspberry Pi, ASUS Tinker Board, and Banana Pi and many more [16]. In this project, it is considered to use Raspberry Pi as an edge computing platform regarding its size, processing power, cost, and user-friendly interface. It has a smooth interaction with other communication systems via SSH, Wi-Fi, and Bluetooth technologies. The idea of using microcomputer is to replicate another potential edge device capabilities that we use, such as cell phones. In recent days, cell phones can be much more powerful than a Raspberry Pi to act as an edge computing platform. Therefore, testing the environment and processing capabilities in Raspberry Pi gives us an excellent prospect of more powerful devices.

The minimum hardware requirements of real-time data streaming require

in-depth research and can be another research topic itself. It has a slight connection to the context of the project in terms of performance measurements and Raspberry Pi's data processing capabilities. However, the hardware requirements research is out of the scope of this thesis.

3.3.2 Software Interfaces

The dependencies of this software are pretty restricted since its written for research purposes. Hence, any paid or in another way of saying, licensed under the proprietary software usage is avoided.

As an operating system, MacOS Mojave is preferred for general software development. In the edge computing platform, the Raspbian operating system is used for the configurations and to create API communication with the server side. For the client-server application, RDLab UPC resources are used. Finally, to visualize the incoming results, IBM Node-RED application is used.

3.3.3 Communication Interfaces

The software uses MQTT [17] protocol to manage the communications with an edge computing platform and to enrich and aggregate the data by using MQTT Broker [18]. It has been built on top of the Eclipse Paho [19] Platform. MQTT stands for Message Queuing Telemetry Transport. It is a publish/subscribe, straightforward and lightweight messaging protocol, designed for resource-constrained devices as well as low-bandwidth, and unreliable networks. The MQTT design principles aim to reduce network bandwidth and resource demand of the device while also attempting to ensure reliability and some dignity of the promise of delivery. These principles make the protocol ideal for developing the Internet of Things systems, and for mobile applications where bandwidth and battery power are the main concerns.

The communication between server-to-client is performed through message passing over the IP network. From a technical point of view, TCP/IP as the transport protocol, where each server verifies a TCP connection to the network elements utilizing a well-known port number. Messages are sent bi-directionally between the server and network elements. All messages consist of a fixed length-header containing the cumulative data length and a request followed by an answer or an acknowledgment. Intercommunication between agents is performed by using HTTP.

3.4 Functional Requirements

3.4.1 Cars' CAN Data Stream Processing

The scenario of the use case is that multiple cars used to gather information regarding the road conditions to create an anomaly detection system by detecting the potholes on the roads. In order to obtain this information, CAN (Controller Area Network) data can be used for data streaming which has been standardized by Bosch in 1991 [20]. The most compelling reason to use this data set is; the CAN bus connects various Electronic Control Units (ECU) inside a vehicle. Its field of application differs from high-speed networks to low-cost multiplex processes. In the car's electronics, engine control units, and sensors are connected using CAN with bit rates up to 1 Mbit/s. At the same time, it is cost-effective to build CAN into vehicle body electronics to replace the wiring harness [20]. The valuable data about the state of the vehicle is presented on this bus, and it is useful to track drivers behavior, the health of the vehicle or analyzing the wheel's abnormal behaviors during the driving session. In reality, the CAN data is not directly accessible due to manufacturers specialized scripts and configuration files. Therefore, reverse engineering methodologies can be applied [21]. However,

the CAN data is considered to be obtained already, instead of applying these techniques due to the scope and the context of this project.

3.4.2 Efficiency and Reduced Latency

In the motivation section, we have mentioned 4G and 5G technologies briefly and their significance to this project. The generated data through sensors would be massive; therefore some concerns may occur in terms of network bandwidth sufficiency and carrying all the data wirelessly. In order to secure all the connections and reduce the latency significantly, 5G networks are considered to be used. The movement towards 5G from 4G networks means massive Machine Type Communications (m-MTC) will enable cities, transportation, and infrastructure to transfer real-time data for enhanced maintenance and higher operational efficiency [22].

When vehicles and other transportation-related components switched to 5G connectivity, it will reform the way we travel. Car-to-car and car-to-infrastructure communication will make roads safer, reliable, and more environmentally friendly while allowing public transportation to operate more efficiently. New services can be supported considering sensors embedded on roads, highways, and railways to communicate with each other or with smart vehicles [22].

3.4.3 Efficient Anomaly Detection Capability

Moreover, anomaly detection algorithms are considered to be implemented to predict future pothole occurrences and the anomaly scores. Regarding this context, Hierarchical Temporal Memory (HTM) Machine Learning algorithm is considered for research in this project. HTM aims to gather the structural and algorithmic properties of the neocortex [9]. The neocortex is a part of intelligent thought in the mammalian brain. In order to explain it more clearly, we can conclude the

neocortex as a controller of the set of actions that we perform such as moving, hearing, talking, and seeing.

HTM can also be seen as a sample of a neural network. By definition, any system that attempts to model the architectural aspects of the neocortex is a neural network. However, the term "neural network" is not entirely beneficial because it has been applied to a wide variety of systems. The neurons in the HTM's model which are called *cells*, and organized in columns, in layers, in sections, and in a hierarchy [9].

3.4.4 Rule based Reasoning Capability

Lastly, Business Rule Management System (BRMS) can be considered to be implemented. BRMS's have gained tremendous acceptance over the past decade. A recent IDC survey of IT managers showed that %55 were using BRMS technology and %16 of those using a BRMS were planning to increase their usage [23]. The BRMS market exhibited double-digit growth each year of the 2008 to 2010 period, making BRMS one of the leading growth areas across all application development and deployment tools [24]. It is expected that the market will grow from 847m $ in 2018 to 1.6b $ in 2024 [25].

A rule-based scheme consists of a set of **if-then** rules, a set of facts and some interpreter checking the application of the rules, given the facts. The goal of an expert system is to practice the information and encode it into a set of rules [23]. We discuss the BRMS in the following chapters in detail.

Finally, the functional requirements are listed as follows:

- The system should be able to handle heterogeneous streaming. Since the amount of incoming data is not the same, handling of this data should not cause any harm to the system such as crashing the system or slowing down the process.
- Different cars might identify the same potholes in different locations with different scores compare to other cars. Therefore, the detection of potholes must be precise.
- Hierarchical Temporal Memory (HTM) machine learning algorithm is considered to predict the anomaly scores from historical data measurements.
- The car's CAN data has to be filtered inside of the car, in order not to increase the latency and redundant data processing over the network.
- The throughput of the system has to handle up to a few Mb/s for each car.[10]
- In specific cases such as highly critical or too often occurrence of pothole detection, some set of actions have to be defined in order to manage the crisis. These set of actions should be understandable for both technical and business authorities in order not to increase the gap between two critical decision making authorities. Therefore, the BRMS has to be integrated.

3.5 Non-Functional Requirements

In addition to the prominent features and functions that we provide in our system, there are other requirements that do not do anything in technical terms. Nevertheless, it remains its essential characteristics. These are named as "Non-functional Requirements" or sometimes "Quality Attributes" [26].

Functional requirements are meaning by technical, but requirements also could be inherent to the software being produced or be triggered by external parts such as organizational, regulations, software licensing or software providers. In these cases, the requirements are incorporated as non-technical requirements [27].

The non-technical requirements of this software are:

Availability

- Other operating systems different than Microsoft Windows such as Ubuntu, Raspbian, and Mac OS X should be able to perform the same tasks.
- The architecture should be reproducible for any data set without significant changes or re-engineering. The principles of the design have to be global enough to apply to other similar cases.

Flexibility

The designed system should supply a specific degree of flexibility, to adapt to the change of dynamic settings and needs to be capable of feeding such changes.

Sustainability

Every library, framework, or tool that is used in the project has to be open source to be sustainable in the future.

Scalability

When the amount of data increases, the system should be able to scale itself in order to handle the workload.

Maintainability

The operational workflow of the system should be well documented. Hence, any other researcher, student, or professionals can understand the project without any assistant through technical documentation.

Performance

The system's performance parameters such as Throughput, Data Loss, and Latency should be measurable.

Data Integrity

- The handling of the source data must be in real-time. This means the detection time of the potholes should be equal to the current time in order to be able to analyze it more accurately.
- It has to be possible to manage the tasks of the system at a higher level [10].

Chapter 4

Architecture and Implementation

4.1 Overview

There are several layers and perks to deal with event processing in IoT. In this section, we go through over each layer of architecture and explain what is implemented, and the significance of the implementations to this thesis work.

4.2 IoT Layers

Two main blocks conduct the IoT layer. These blocks are responsible for managing the data collection, processing and sharing the results. A graphical representation of the architecture is shown in figure 4.1 below from the bottom layer of sensing to the top of the application.

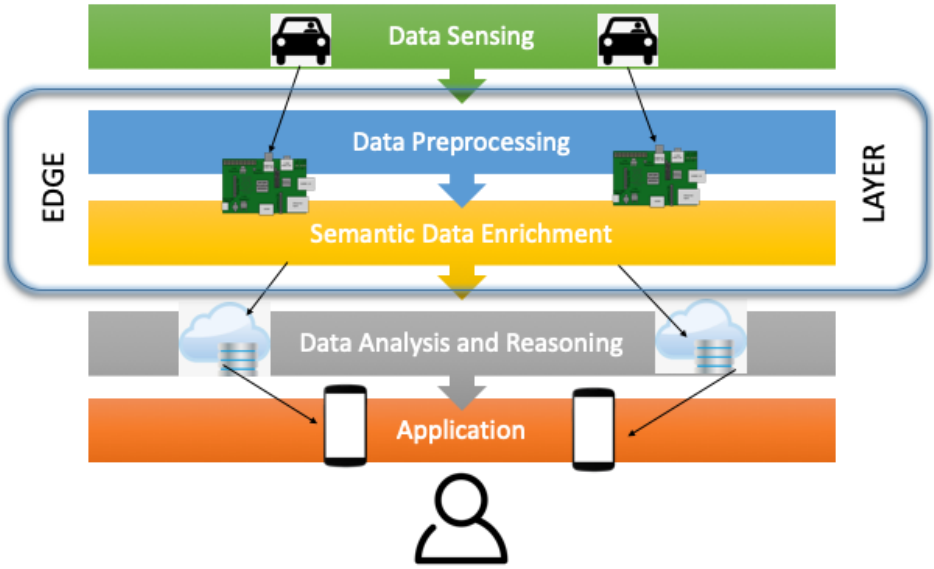

Figure 4.1: Architecture Overview

The IoT architecture is divided into four layers such as:

- Data Sensing Layer
- Edge Processing Layer
- Data Analysis and Reasoning Layer
- User Application Layer

4.2.1 Data Sensing Layer:

This is the lowest layer of the IoT architecture where we generate the data from the CAN bus data that is coming from the car sensors.

The generation layer is vital in determining the system's throughput characteristics, given that there is a data generation rate at this layer. It has to be

accommodated in experimental studies and real-life scenarios to ensure the events are not lost.

In order to have the maximum benefit of the edge computing characteristics and its processing capabilities, it is intended to keep the data sensing layer's capabilities at the lowest level. It means that only basic operations are applied in this layer corresponding to define which data will be sent, within which time frames, or by single or multiple measurements.

The figure 4.2 below indicates a small snapshot of a dataset. The columns show from left to right: message identifier, field length, eight columns of data values, the time-stamp indicator, the identifier for the next message, and its length[10].

1	8	21	25	35	46	34	23	45	23	2.09	246
2	8	24	25	40	50	55	35	65	15	2.1	246
2	8	45	40	55	34	80	70	75	45	2.11	246
3	8	17	24	33	56	34	23	44	11	2.12	246
4	8	67	33	51	31	11	14	24	15	2.13	246
5	8	22	25	64	65	67	18	24	18	2.14	246
6	8	85	81	65	45	59	29	49	33	2.15	246
7	8	5	12	20	14	16	19	33	21	2.16	246
8	8	51	49	7	81	23	7	54	22	2.17	246
10	8	90	83	58	81	83	87	10	42	2.18	246
11	8	16	89	89	24	7	24	70	80	2.19	246
12	8	33	2	29	17	7	46	18	88	2.2	246
13	8	1	24	47	55	4	91	9	18	2.21	246
14	8	98	56	52	24	70	84	58	51	2.22	246
15	8	47	15	90	45	71	29	75	93	2.23	246
16	8	55	87	54	27	30	37	79	52	2.24	246

Figure 4.2: An Example of a Dataset

4.2.2 Edge Processing Layer

The edge layer is the main distributed architecture component of the system. The data sensing layer is passing the data, which is coming from the car's sensors to edge layer for pre-processing, and semantic enrichment processes. The idea is to take advantage of the computation ability of middle-ware devices to reduce the upper layers processing amount, stress and time such as clouds or clusters. The Edge layer has two processing sublayers. These layers have a hierarchy

between them. The pre-processing always happens before the data enrichment, as it makes sense to enrich only over the qualified or appropriately selected data. In the following sections, we examine the capabilities of each layer in more depth to further analyze the streaming processes.

4.2.2.1 Data Pre-processing:

The pre-processing is a layer that blocks of events can be done by the computational power of its fundamental units. These fundamental units depend on the complexity and type of the collected data. Data-preprocessing is also acknowledged as data preparation [28]. It settles some techniques not limited to but concerned with processing raw data collection, data filtering, integration, transformation, and reduction. [28]. For instance, redundant data elimination can be done before the data passes to the upper layers. If there is enough computational power in the existing structure, some of the pre-processes can be handled in local devices, before sending them to any network. If network bandwidth is not a concern, or there is not enough computational power in the existing structure, moving pre-processes to the edge platform could be a good option in order not to increase the stress over the local devices. In this project, the pre-processing is handled inside the edge device (Raspberry Pi) to match with the objectives. Some data pre-processing options are considered to be implemented in this project, such as:

- **Filtering:** The classical filtering of the data can be applied based on the preferences. The context of the data can be selected manually, automatically, or some of them can be discarded. The other filtering options are error detection and filtering out unclassified data for specific environments.
- **Data Reduction:** This step aims to present a reduced representation of the data. In some cases, the amount of data can be huge. Therefore it is

necessary to be able to reduce the utilized data without losing any of its accuracies. In another way, we can reduce the intensity or data stream rate for the measurements. This might sound like reducing the data stream rate would automatically drop the quality. However, in some cases, it might be useful to reduce the unnecessary amount of data if the high data stream rate is not the principal objective.

- **Anomaly Detection**: The detection of the anomaly scores are based on the statistical approach. Furthermore, this approach can be extended with the advance of machine learning algorithms that can state a prediction score to detect anomalies. For research purposes, we focus on Hierarchical Temporal Memory (HTM) Algorithm to train the dataset. We review the HTM algorithm in depth in the following subsection.

- **Event Detection:** Any event stream process assumes that some functions will be enforced to find the limits of an atomic event, that is, crucial once the event begins and when it finishes at the intervals of the stream. These could be any unexpected environmental changes or autonomous events that are affecting the results of the measurements.

- **Data Analysis** Nowadays, microchips are so powerful to do some operations to perform small tasks and analysis. We control many applications and wearable devices through our smart-phones. Thus, extracting the part of the data analysis into these platforms is becoming the new trend in engineering and research areas [29]. The reason for that is the data that is generated by these mobile devices are gigantic, and they need to be classified or pre-analyzed before doing more complex operations.

4.2.2.2 Hierarchical Temporal Memory (HTM) Algorithm

The HTM algorithm is getting popular since Numenta Org. [9] was initialized in 2005. It is founded with two primary goals which are reverse engineering of the neocortex and applying that knowledge to create machine intelligence.

HTM provides a theoretical framework that tries to replicate how the neocortex works. Still, human neocortex is extremely complex and it is still a mystery of how it learns from the detailed world by modeling it [30].

In this part of the thesis, we implement the HTM algorithm to our preprocessing block to detect anomalies and see the efficiency of the algorithm. Regarding this, two main goals are preserved:

- The initial goal is to determine whether the algorithm can read the anomaly scores, learn from the patterns, and predicts the pothole occurrences.
- The second goal is to test the performance of the algorithms in relatively small computational units such as Raspberry Pi.

Principles of an HTM Algorithm

Before we move to the implementation part, there are several principles of the HTM algorithm to reflect on how it is built. For instance: How important is a hierarchical organization? How regions are structured? Why the time information is crucial? and Why HTM uses sparse distributed representations? [9].

- **Hierarchy:** The HTM network contains regions that are built in a hierarchy. The regions are the central unit of the memory and prediction process[9].

  One of the most important benefits of the hierarchy is efficiency. It reduces training time and memory usage because the patterns acquired at each

level of the hierarchy are reused when they are connected in unique ways at higher levels, just like how our brain stores the information. [9].

- **Regions:** The term region comes from the actual neocortex. The human neocortex is divided into different areas called regions. These regions are interconnected in a way that some of them are involved with the input data directly and the others receive it after several interactions. The figure 4.3 below shows a simplified diagram of HTM regions established in a four-level of hierarchy, communicating with each other within levels to transfer data (represented with the arrows), among levels, and from outside of the hierarchy [14].

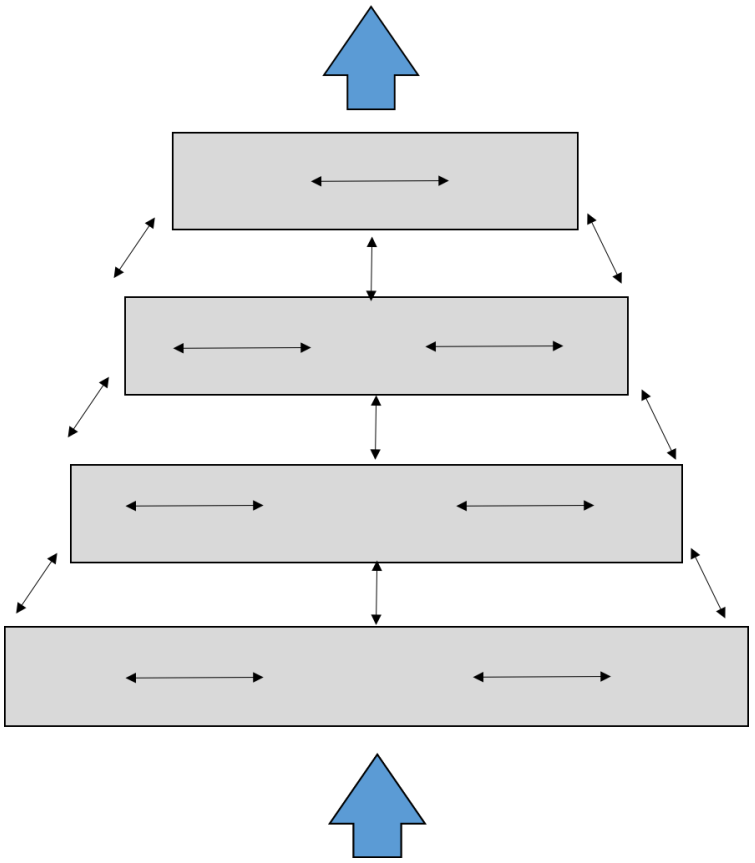

Figure 4.3: A Simple HTM Hierachy Arcihtecture (Ref: Numenta Org.)

- **Sparse Distributed Representation:** It uses inhibitory neurons to involve a partial amount of active neurons that are interconnected to each other to represent the information. It is a set of processes for replicating our brains communication methodology [14]. Every neuron is equal to a bit that carries a piece of information. These bits are classified as active or passive with values of 1 representing the active state, and 0 representing the passive state. It is essential because HTM regions are depending on this representation. Thus, regions transfer the input data in the SDR format.

- **The Role of Time:** The HTM network does not need to train an enormous amount of data to learn and predict. However, providing more data increases the accuracy and overall prediction quality. This fact requires some helpful information for an algorithm to recognize the patterns more efficiently. In this case, time plays an important role in analyzing the patterns so that the algorithm can start to justify the learning curve after a while. It is the same relation between our neocortex and the real world objects. When we are blindfolded and try to identify any object by touching it, we can get more information in time and start to feel more patterns over it [9]. There are a few parameters for time management as it can be seen from figure 4.4. These parameters can be extended and configured based on the dataset and the measurements. For instance, there can be a hierarchy between different time indicators. Hence, the HTM algorithm can learn more, and predict more accurately with the hour based prediction compared to second based prediction. This is due to the fact that synapses activation processes rely on the time parameter. If there are more parameters related to time, the inference can be improved.

```
{'aggregationInfo': {'days': 0,
                     'fields': [(u'timestamp', 'first'), (u'value', 'sum')],
                     'hours': 1,
                     'microseconds': 0,
                     'milliseconds': 0,
                     'minutes': 0,
                     'months': 0,
                     'seconds': 0,
                     'weeks': 0,
                     'years': 0},
 'model': 'HTMPrediction',
```

Figure 4.4: HTM Algorithm Time Parameters (Ref: NuPIC)

Learning, Inference, and Prediction

HTM learns everything by finding patterns in the input data. Regions do not know what the data means. It looks for the specific bit combination occurrences to find the relationship statistically, which are called Spatial Patterns. Then, it identifies how these patterns occur in a time period, which are called Temporal Patterns.

When the learning of the patterns is done, to infer, the algorithm matches new input information with the previously trained spatial patterns. Every region stores a small number of patterns inside. When the data arrives, it states a prediction about the next incoming data. The predictions are continues and occur at every level of the hierarchy. Since every region has a prediction, if the predicted data does not match with the next one, it states that there is an anomaly.

Implementation of the HTM Algorithm

The implementation of the HTM is based on the Numenta Platform Intelligent Computing (NuPIC) [31] libraries. The core of the NuPIC implementation is written in C++ language. It also supports Python bindings on top of C++, Java, and Closure.

We intend to develop an HTM algorithm in Python language since it matches

with our development environment. However, the HTM algorithm is built in Python 2.x version, and there is not a compatible stable version for Python 3.x environment. Therefore, for the sake of stability and to take advantage of already available resources, the migration between the versions has been taken out from the scope of this project.

The architecture of the NuPIC API [31] is conducted with three main interfaces:

- **Online Prediction Framework:** It is procuring predictions from online training algorithms, including HTM. It is designed to work with both robust architectures [31], as well as in a standalone mode. It mainly consists of HTM Prediction Model, an Interface, Model Result, and Inference Shifter along with many other components as well.

- **Network API** A lower-level API to create hierarchical structures of nodes. The structure can be modified based on the requirements of each prediction process. It defines regions that are meant to retrieve the input and carry the output and linking them together.

- **Algorithms:** The execution process starts with the data encoding into the form of SDRs (Sparse Distributed Representations). It consists of massive arrays of bits that each one of them is carrying a semantic meaning to identify the overlapping bits. Any data format can be converted to SDR to feed the HTM system.

  The second step is to pass the encoded data over Spatial Pooler. Spatial Pooler is responsible for relations between the columns of a region and the input bits.

  The third step is to execute the Temporal Memory over the active Spatial Pooler columns. The algorithm trains over the sequences that are formed

by Spatial Pooler and predicts the next SDR value.

The fourth step is to extract predictions with SDR Classifier. It accepts the binary input pattern and data from the encoders. Then, maps those patterns to class labels and shows the output as a probability distribution.

The final step is deriving the anomaly detection and anomaly likely hood values. Additional to these two variables, time-stamp, anomaly scores, and predicted anomaly scores are saved in a CSV file for further needs and analysis.

Model parameters that are in figure 4.5 are created before the executions to specify the algorithms and will be used to train the data set. However, it would be helpful to explain some of the parameters and their goal in the prediction process. There are several parameters that implement both Spatial Pooler and Temporal Memory classes [31]. In the Spatial Pooler class, the **globalInhibition** is helping to select the winning columns as the most active columns during the inhibition. If false, then winning columns are selected based on their local neighborhood. The **numActiveColumnPerInhArea** is a way of controlling the active column density, along with the local area Density. The **potentialPct** represents the percentage of the input data that is in the column's potential area. It is used for giving an authority for each column to connect every input that is in its area. The **synPermActiveInc** value represents the increment of an active synapses in each round. The **synPermConnected** value is a threshold of connected synapses to contribute the cells triggering. Which means that the synapses must contribute to the process as at least its value. Regarding to Temporal Memory implementation, the **activationThreshold** represents the active connected synapses. The **globalDecay** is responsible for deleting the non-active synapses. The **InitialPerm** is setting up the initial permanence for constructed

synapses. The **pamLength** prevents the algorithm to fall back to the previously predicted state.

```
'spEnable': True,
'spParams': {'boostStrength': 0.0,
             'columnCount': 2048,
             'globalInhibition': False,
             'inputWidth': 0,
             'numActiveColumnsPerInhArea': 40,
             'potentialPct': 0.8,
             'seed': 1956,
             'spVerbosity': 0,
             'spatialImp': 'cpp',
             'synPermActiveInc': 0.05,
             'synPermConnected': 0.1,
             'synPermInactiveDec': 0.08475386478617661},
```

(a) Spatial Pooler Parameters Example

```
'tmEnable': True,
'tmParams': {'activationThreshold': 13,
             'cellsPerColumn': 32,
             'columnCount': 2048,
             'globalDecay': 0.0,
             'initialPerm': 0.21,
             'inputWidth': 2048,
             'maxAge': 0,
             'maxSegmentsPerCell': 128,
             'maxSynapsesPerSegment': 32,
             'minThreshold': 10,
             'newSynapseCount': 20,
             'outputType': 'normal',
             'pamLength': 1,
             'permanenceDec': 0.1,
             'permanenceInc': 0.1,
             'seed': 1960,
             'temporalImp': 'cpp',
             'verbosity': 0},
```

(b) Temporal Memory Parameters Example

Figure 4.5: Model Parameters Examples (Ref:NuPIC)

In figure 4.6 a small fraction of results regarding the predictions of the HTM algorithm can be seen. From left to right, the values are: The identifier of processed data entries, time-stamp, actual anomaly score that is obtained by statistical approach, HTM's anomaly prediction score, detected anomaly *Boolean* indicator, and anomaly likelihood value. The anomalies are detected if the predicted score is higher than the next actual anomaly score, and the anomaly likelihood is higher than the given threshold.In this case, the selected threshold was chosen as *0.98*, considering the max value was *1.0*. Each anomaly score value has been multiplied by 100 in order not to get confused with anomaly likelihood value.

1112	12/6/19 17:45	57	98	0	0.95836922
1113	12/6/19 17:50	76	98.9999999	0	0.95836922
1114	12/6/19 17:55	68	84.0000023	0	0.95836922
1115	12/6/19 18:00	77	77	0	0.95836922
1116	12/6/19 18:05	99	99	1	0.99997344
1117	12/6/19 18:10	87	87	1	1
1118	12/6/19 18:15	72	72	1	0.999
1119	12/6/19 18:20	9	87	0	0.999

Figure 4.6: The Anomaly Score Prediction Results of HTM Algorithm

The figure 4.7 below shows the resulting plot of 3000 streamed CAN data that is processed by an HTM algorithm line by line. The blue areas indicate the detected anomalies.

The results have consistency in terms of overall prediction capabilities except for some cases. The reason for that is, the simulated CAN data set consists of basic values for testing purposes. Thus, the algorithm was having some trouble to find the patterns between the entries.

As an outcome of this experiment, our first goal for the HTM algorithm implementation regarding reading, learning and predicting the anomaly scores has been covered successfully as it is indicated at the beginning of this section.

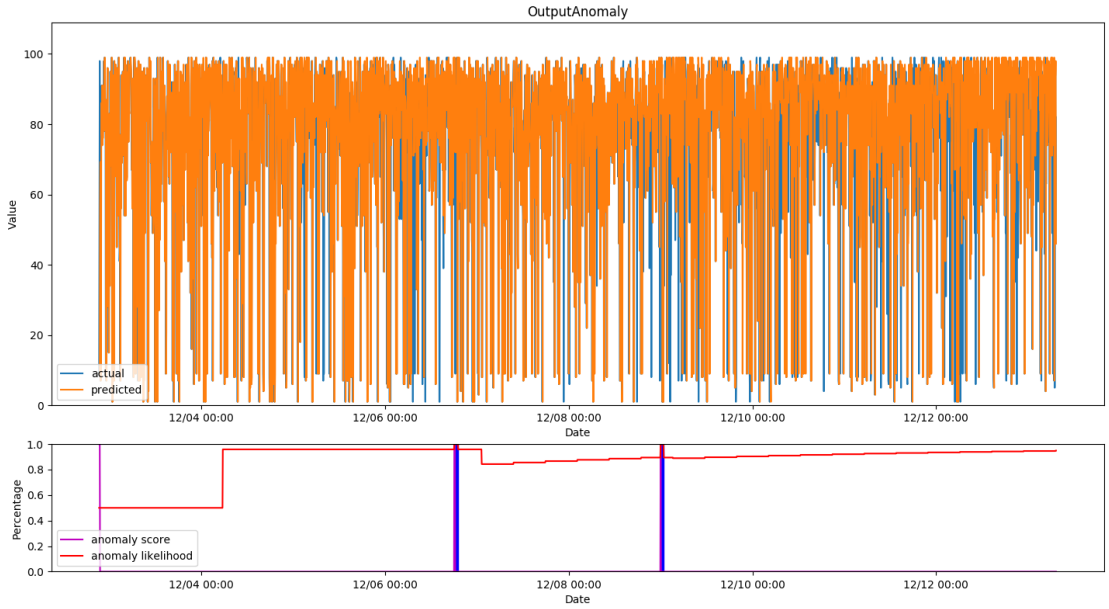

Figure 4.7: HTM Algorithm Comparison and Anomaly Detection Results

The second goal that is regarding testing the HTM algorithm inside the Raspberry Pi, could not be covered due to the implementation constraints. The HTM algorithm requires a 64-bits operating system in order for its libraries to be compatible. The Raspbian OS is packed with built-in libraries to reduce the configuration effort after the installation. Yet, they do not provide 64-bits operating system support for Raspbian. There are some other available light-weight 64-bits supported operating systems for Raspberry Pi such as Raspex [1] or Ubuntu MATE [2] in an experimental state. However, the speed of the installations and the performance of the device drops significantly due to some misconfigured packages, resulting in crashing the entire system consequently.

[1] www.raspex.eston.se
[2] www.ubuntu-mate.org

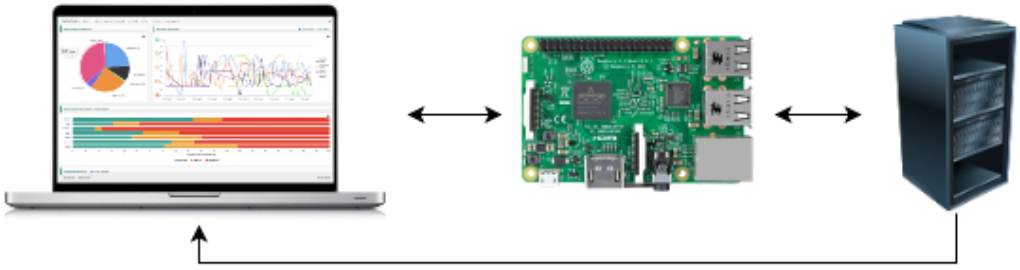

Figure 4.8: Semantic Enrichment in Raspberry Pi Model 3 B+

4.2.2.3 Semantic Data Enrichment

Semantic Data Enrichment process is about adding additional content on top of the original data for further process steps or classification, based on the specific preferences [32]. A sample would be adding the data generation time and date for ordering purposes and maintenance planning.

In most cases, we might not have enough resources to activate the data enrichment process by using pre-processing inside the cars. Even if we have, in order not to put too much stress on the system and save the resources for other possible required purposes, the edge layer is preferable.

The key functions of the edge platform are data aggregation and data enrichment for further processing. The idea of having multiple cars(in the thesis scenario will be multiple raspberry pi receivers) and identifying them based on their UUID and create a smart streaming service for each with a queuing system.

4.2.3 Data Analysis and Reasoning Layer

One of the crucial functions of the data analysis and reasoning layer is to ensure the communications between the edge layer and the client. Additionally, this layer is responsible for gathering the incoming data from the lower layers. This data is referring to pre-processed and semantically enriched data that has been

tackled in the edge layer. During the reasoning process, it is important to generate additional data on top of the original data to increase usability. In this section, semantic web technologies and their significance to this project are presented.

The semantic web extends the web with machine-interpretable meaning. Hence, it establishes data integration, sharing, and interoperability among interconnected machines. The Semantic Web-scheme is based on the Resource Description Framework (RDF), which allows connecting and joining of relationships between entities from multiple resources on the web through Internationalized Resource Identifiers (IRI). RDF Schema (RDFS) and ontologies offer a vocabulary for modeling and representing RDF data. Semantic technologies can reconstruct things into smart objects that are qualified in communicating intelligently with each other in IoT [10].

Data representation: Uniform Semantic Web data representations, such as RDF, which can be unambiguously interpreted on the Internet, are appealing candidates for data transfer formats in IoT. Nevertheless, resource-constraints and latency requirements introduce challenges for implementing these technologies. RDF data can be expressed in various data formats for publishing and exchanging semantic data. RDF/XML, Turtle, and N-Triples are alternative W3C standard representations for RDF [10]. Notation3 (N3) is another expressive format of RDF, which can also express rules with N3 Logic and RDF properties. All of those are based on the triple structure but differ in expressive power. These RDF syntaxes are designed for Web applications. However, resource usage is critical for IoT but was not emphasized when these formats were designed. JSON for Linked Data (JSON-LD), Entity Notation (EN) and Header-Dictionary-Triples (HDT) is more compact, lightweight representations for RDF. HDT is designed for compressed RDF data storage rather than a lightweight data exchange for IoT. Sensor Markup Language (SML) is a data format for representing sensor

measurements and device parameters but is not based on RDF, although it has the necessary capabilities to annotate the type of data.

In recent years, a notable number of technologies that facilitate real-time and linked stream processing have also risen. Linked Stream Data is an annex of the SPARQL query language. It has a semantic engine to deal with stream sensor data and enrich them with the Linked Open Data cloud. SPARQL is an RDF query language and protocol produced by the W3C RDF Data Access Working Group (DAWG) [33]. SPARQL is widely used in Semantic Web associations and was published as a W3C Recommendation in 2008. C-SPARQL was an initial proposal of the streaming SPARQL system [2].

Reasoning: Reasoning is about making conclusions and deriving new facts that do not exist in the knowledge base. Reasoning with rules is typically based on first-order predicate logic or Description Logic (DL) to make conclusions from a sequence of statements (premises) derived by predefined rules.

A reasoning engine (i.e., a reasoner) is a software tool that ensures reasoning with rules [32]. Current reasoners can handle a broad set of RDFS and OWL vocabularies and most RDF data formats. A reasoner concludes facts from semantic data and ontology-based on predefined rules. Common reasoning and inference engines such as the Jena Inference subsystem, Pellet, RacerPro, HermiT, RIF4J, and Fact++ are based on different rule languages and have support for ontologies and OWL.

Some of the reasoners support SWRL (Semantic Web Rule Language) and RIF (Rule Interchange Format) rule languages, whereas others have implemented their human-readable rule syntax.

Distributed reasoning: IoT introduces additional challenges for reasoning; for example, reasoning can occur at any stage of the data delivery process, from the sensor node to back-end knowledge repositories. Distributing reasoning tasks

can physically improve reasoning latency with large data sets [10].

4.2.3.1 Rule Based Engines

The title "rule engine" is considerably obscure that it can be any system that uses rules, and it can be implemented for any form of the data to produce outcomes [34]. This entails modest systems, like form validation and dynamic interpretation engines.

The control scheme of the rule engines can be orchestrated by two fundamental concepts which are respectively *forward chaining* for data-oriented reasoning and *backward chaining* for goal-oriented reasoning [35]. Although, some systems prefer to use the *hybrid reasoning systems (HRS)* to tackle some problems such as imperfect reasoning and defeasible logic [36]. In the rule-based systems, the inference engine commonly goes through a simple recognize-assert cycle. It seeks to understand first by starting from input data. This can be in between or final outputs by using the encoded rules. There are numerous methods of reasoning that can be used:

- **Deduction:** It is also known as *modus ponens* [37] in mathematics.

$$(B; B \rightarrow K) \Longrightarrow K$$

  meaning that if K is true, and that a rule $B \rightarrow K$ is also true, thus we conclude by deduction that B is also true. An example of deduction can be; *if you have a car, you can drive.*You have a car, therefore *you can drive*

- **Abduction:** It can be considered as an opposite meaning of the modus ponens known as *modus tollens* [37].

$$(K; B \rightarrow K) \Longrightarrow B$$

meaning that if B is true, and we have a rule stating that $B \rightarrow K$, then we obtain by abduction that K is true.

- **Transitivity:** This is an intermediate action process between two rules.

$$(B \rightarrow F; F \rightarrow K) \Longrightarrow (B \rightarrow K)$$

meaning that, if we have two different rules, the first ends with F and the second is starting from F, hence, by transitivity, a new rule can be gathered, which is covering both B and K.

To implement such logical statements, we review several Rule Engines and their performances and significance to this project. For doing that, it would be helpful to look over the expert knowledge system definition and briefly examine why it is crucial for any IT application.

In the 1990s, it was realized that expert and knowledge-based systems were efficiently automating the business rules in organizations used to automate decisions. The outcome was an aim for software developers to embed these business rules into Business Rule Engine (BRE) to manage the business logic in their applications or system processes [38].

In order to efficiently describe the BRE's, we need to dive into expert knowledge base systems. Expert's knowledge and methods needed for these knowledge-based systems.

A knowledge-based expert system definition is:

"An expert system is computer software that attempts to act like a human expert on a particular subject area. It applies a knowledge base of human expertise for problem-solving, or to clarify uncertainties where typically a human expert would need to be consulted [39]."

However, even though the BMRS and EKS (Expert-Knowledge System) share the same mechanics to solve problems, they tackle different problems [40]. In addition to that, the EKS's and BMRS'target different problem segments as well. For instance, the BMRS's are commonly used for real-time decisioning systems, straight-through processing (STP) systems, and compliance systems. On the other hand, EKS's are used for advising systems, expert systems, and knowledge automation[41]. Nowadays, the differences matter most in the implementation style. Some of the programs are named as expert systems, but it is possible to integrate them as business rule engines, as it is mentioned before thanks to the same mechanics that they share.

For instance, EKS's tackle:

- Difficult engineering or medical problems with no exact solution and it requires heuristic knowledge [40].
- Scarcity of expertise. Which means for complex problems, more than one expert and co-operation are required to find an answer [39].

On the other hand, BMRS tackle different challenges with respect to common business knowledge:

- Simple business rule definitions with broad logic [41].
- Externalization (explicitation). This means, for the production rules, the business and the technical logic could be separated in order for business people to be easily integrated into projects [40].

A research [38] was led out into rule execution algorithms and languages to support for effective execution. Notably, the rule conditions could expertly be compiled into efficient pattern matching mechanisms by using some algorithms

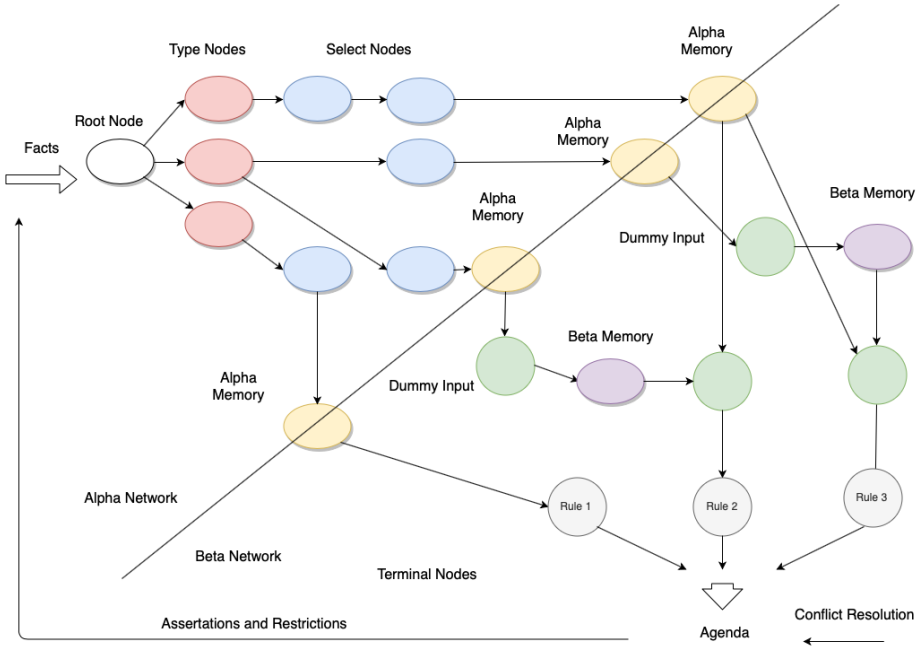

Figure 4.9: Rete Algorithm Overview (Ref: The Improvement for Rete Algorithm)

such as the Rete algorithm. Many of the Business Rule Engines are mostly Rete-based and works in a data-driven, and forward chaining manner.

RETE Algorithm

The RETE algorithm [42] [4.9] is originally designed and published by Dr.Charles L. Forgy in 1974 as a working paper, and later on, elaborated it in his Ph.D. in 1979.

It is an efficiently designed pattern matching algorithm for implementing production rule systems, used to settle which of the production rules should fire based on its data store [43].

Several versions of Rete is developed since its released [44]. Rete is the original name of its release and then Rete II, Rete III and finally in 2010, Rete-NT. The latest version of the Rete algorithm is 500 times faster than the first release [45].

In the part of the functional requirements, we discussed how the BRMS mar-

ket is growing so fast and why BRMS is essential for any IT application briefly. There are many open source rule-based engines available for implementation. However, considering the compliance of the system, we choose the one that designed explicitly for Python programming language.

CLIPS Language

CLIPS [46] is a RETE based expert system tool and a programming language [47] initially developed by the Software Technology Branch (STB), NASA/Lyndon B. Johnson Space Center in 1986. It has endured continual clarification and improvement. Thousands of people around the world use it.

There are three ways to describe the knowledge in CLIPS:

- Rules, which are designed initially for heuristic knowledge based on experience.
- With the help of deffunctions which are designed to define inside rule functions and generic functions.
- Object-oriented programming, also originally intended for procedural knowledge. The five commonly accepted characteristics of object-oriented programming are supported: classes, message-handlers, abstraction, polymorphism, encapsulation, and inheritance. The rules can pattern match on objects and facts. This concept is very important to be able to use the other functions variables or a result of the function as a variable inside the deffuctions.

PyKnow

PyKnow [48] is a CLIPS based Python library. It is aimed to be an alternative to CLIPS but also to be as compatible as possible. Moreover, since it is implemented in Python, it also supports Python2 and Python3 versions. The design of the PyKnow is very straight forward and easy to understand for a person

who is familiar with any other Rete-based rule engines. As it is described in the architecture [48], PyKnow is capable of matching up a collection of facts with a collection of rules to those facts and execute some actions based on the matching rules. We examine the characteristics of the PyKnow in the following part.

Facts [49] are the initial and necessary unit components of PyKnow. They are used by the functions to create reasoning facts regarding the problem. On the other hand, Rules [49] have two strong components, which are LHS (left-hand-side) and RHS (right-hand-side).

LHS describes (using patterns) the conditions on which the rule should be executed (or fired).

RHS is the set of actions to execute when the rule is fired. In order Fact to match a pattern, all pattern restrictions must be 'True' when the Fact is evaluated against it. Besides, for a Rule to be useful, it must be a method of a "KnowledgeEngine" 4.10 subclass.

The difference between Facts and Patterns is not significant. Patterns are just Facts including Pattern Conditional Elements (PCE) instead of regular data. They are accepted only in the LHS of a rule [49]. If we don't provide the content of a pattern as a PCE, PyKnow will enclose the value in a LiteralPCE automatically. Also, it is not possible to declare any Fact containing a PCE, in case if we do it, it returns an exception back.

KnowledgeEngine is an initiating part of the PyKnow execution. The first step is to make a subclass of itself and use Rules to decorate its methods. After this step, we can instantiate it, populate it with facts, and finally execute it.

The usual process of the execution of "KnowledgeEngine" can be defined as:

1. A class must be initiated with the "KnowledgeEngine".

2. Calling the reset method:

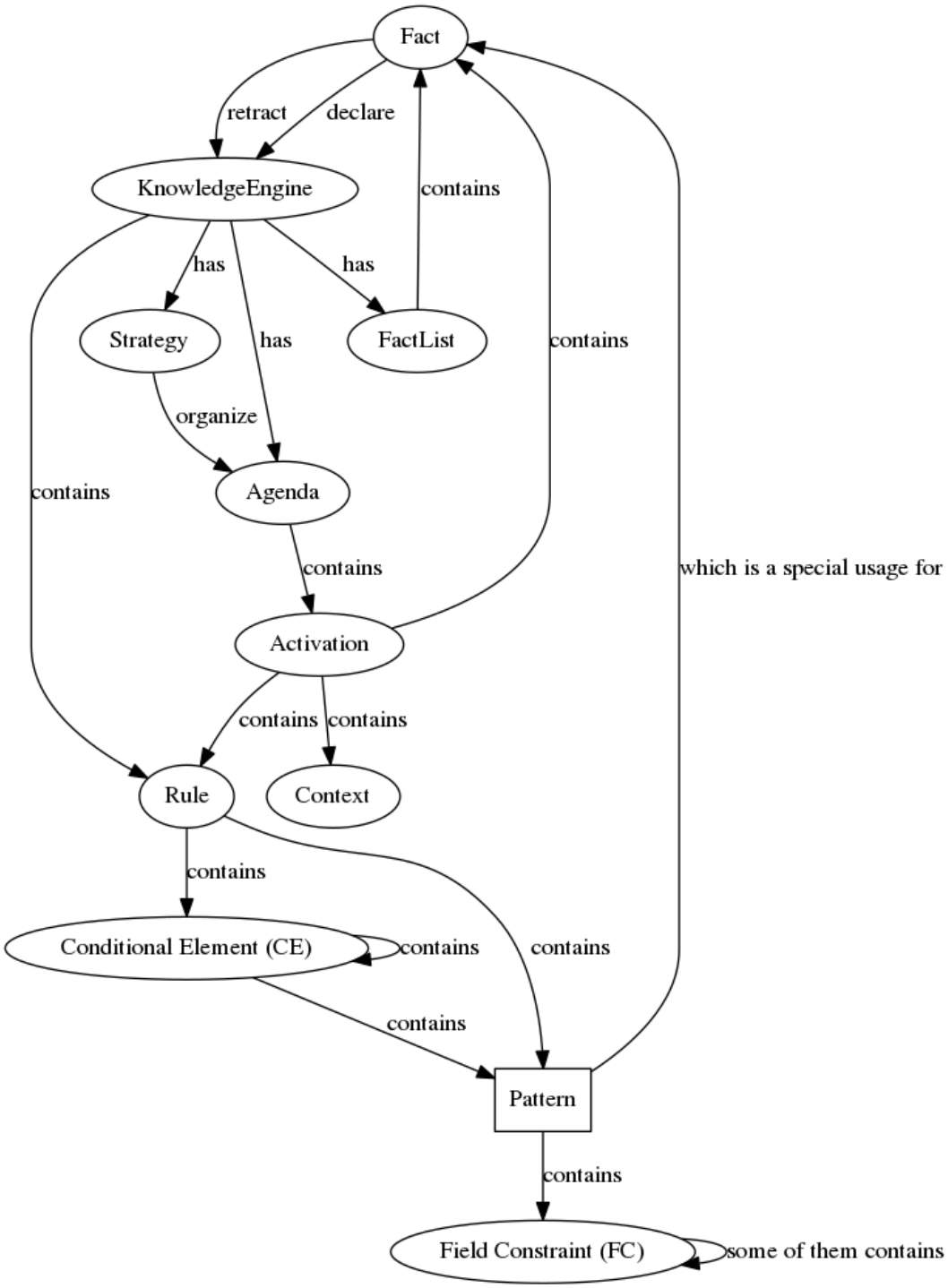

Figure 4.10: PyKnow Architecture (Ref: PyKnow Documentation)

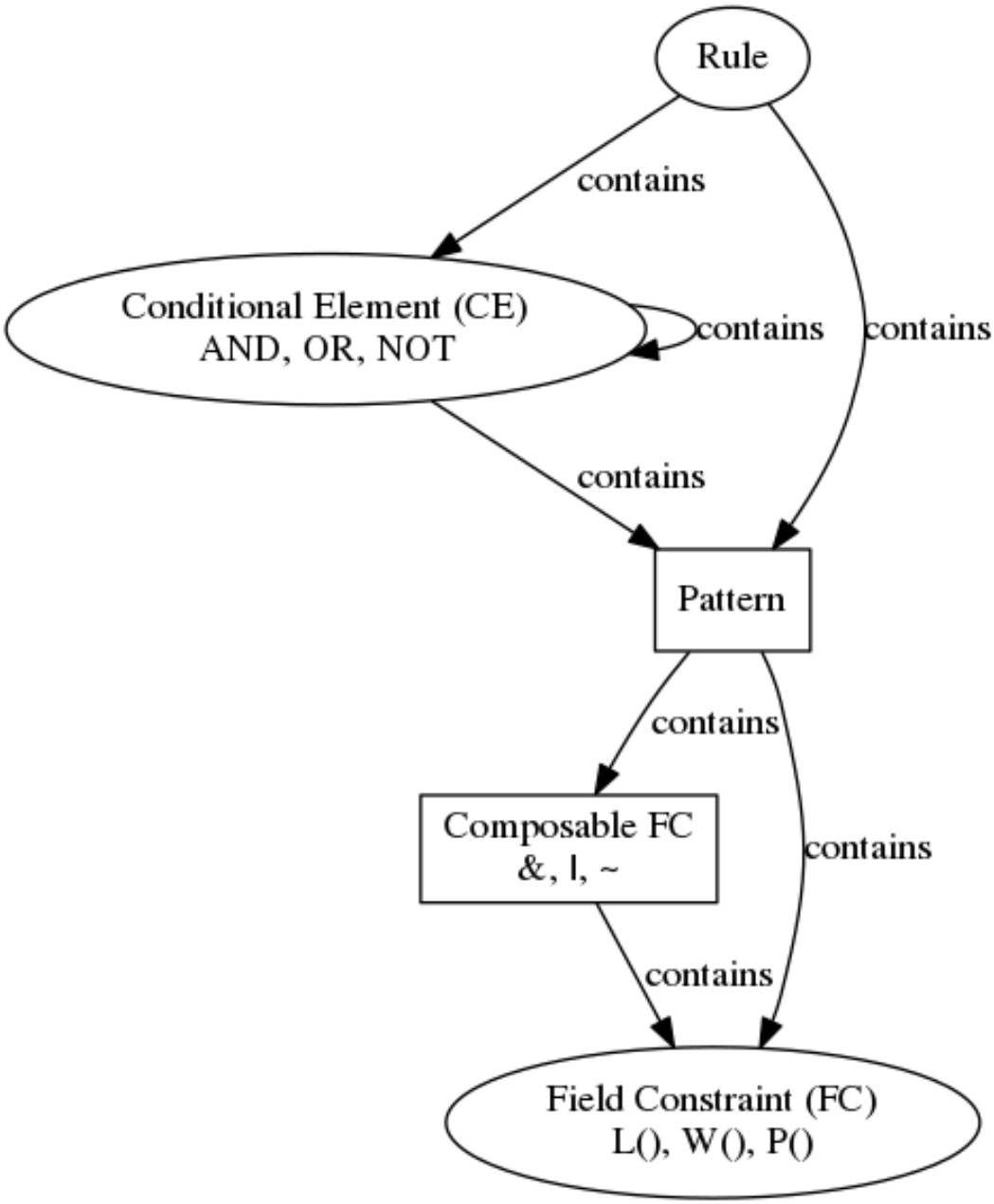

Figure 4.11: PyKnow Rule Architecture (Ref: PyKnow Documentation)

- This indicates the special fact **InitialFact**. It is needed for some rules to work properly.
- State all the facts yielded by the methods decorated with *@DefFacts*.

3. Finally, calling the run method. This starts the cycle of execution.

As a final step, we look into the cycle of execution. In PyKnow the program flow does not need to be determined explicitly. The knowledge (Rules) and the data (Facts) are divided, and the KnowledgeEngine is used to implement the knowledge to the data [49].

1. When the rule firing limit reaches to its limits, the execution stops.
2. The top rule is selected for execution. If there are no rules on the schedule, the execution stops.
3. The RHS actions of the chosen rule are executed. As a result, rules may be activated or deactivated. Activated rules are placed on the schedule. The deployment on the schedule is determined by the notability of the rule and the current conflict resolution strategy. Deactivated rules are removed from the schedule.

   Based on the theoretical information above, the implementation of the PyKnow rule engine has been completed with several business rules.

   The rules are related to anomaly scores, distances, and detection times. The relationship between these three variables can be used to create effective rules to reason over already existing data. This means that we can generate new information out of these three variables in order to create actions or to take precautions for dangerous situations. The rules are respectively:

- The anomaly scores are categorized into three different scenarios. The scenarios are sending info, warning or error messages depending on the anomaly score value. For instance, if the anomaly score is between [0.0, 0.6] it means that the car is in a safe area. Therefore, the info message is sent to the client in order to inform the driver about the road conditions. If the score is between [0.6, 0.8], the car is the medium level danger area. Thus, the warning message is sent to the client to be aware of the changes. If the score is between [0.8, 1.0], it means that the car is in the critical zone. Therefore, the system automatically creates a report to send it to the authorities.
- The anomaly score is higher than 0.8 and the distance between two locations is lower than or equal to 2 meters. Therefore, it is very likely that two anomalies are detected by different wheels of the same car.
- The anomaly detection time is higher than 16 and the anomaly score is higher than 0.8. The emergency case must be active due to out of working hours.
- The distances between the two anomalies are lower than 500m and the detection time difference is higher than 2 hours. The time format of the car is not valid.

These business rules are created for the project in order to represent the real cases and examples. However, since the real rule set is not given for the project, the rules are very simple and they are kept as an experimental state. The integration of the engines is not automated due to time constraints.

4.2.4 User Application Layer

The user application layer has several functions to ensure communications between the edge platform and the client, as we mentioned in communication interfaces section of the software specifications and requirements. The functions in this layer are responsible for representing the incoming data to the client. This data is referring to pre-processed and semantically enriched data that is coming from the edge layer.

After the gathering process, the algorithm encodes the incoming messages to deliver it to the client in a more pleasant and human-readable format.

A pure text notification system is preferred in the integrated development environment to notify the developers or contributors for viewing the results. However, in reality, this notification system cannot be reached or visualized by the end user meaningfully without any user-interface interaction. For this reason, we aim to implement a Node-RED application to visualize the IoT data in real-time which is specified in the following chapter.

Chapter 5

Deployment in Real Infrastructure

5.1 Overview

In this section of the thesis work, the real infrastructure is used to reproduce the results, which means that the scripts are distributed depending on their functionality to get the maximum benefit of edge computing and as well as for testing the infrastructure.

As we discussed in the previous sections, the local implementations and integration of any system might only give a general overview of how the system behaves. When all the components of the system are initiated in a single computer, we naturally eliminate the external factors that might affect the communication between the layers, data transfer speed and success rate, and single failure point of the system. Thus, it is important to test these implementations in a more realistic environment in terms of different infrastructures, within different hardware and software configurations.

The real infrastructure deployment of the system is divided into three cat-

egories. These steps are covering Raspberry Pi integration, Ubuntu Machine in RDlab UPC to store and execute the server side scripts and IBM Node-RED server to visualize the incoming results on both desktop and mobile view.

5.1.1 Edge Platform Integration in Raspberry Pi

The integration of the edge platform is handled by Raspberry Pi 3 model B+ as it is the current stable final release of the hardware. The new model has some advantages over the previous models in terms of quantities of the pre-installed software, 5G connectivity, Virtual Network Computing (VNC) to be able to access Raspberry Pi remotely with GUI. There are many VNC providers available in the market for commercial use. Also, Raspberry Pi comes with a built-in VNC software which is called tightvncserver [50]. However, choosing a VNC option might cause enormous connectivity problems, slow progressing, ineffective command processing when the usage time increases. There is also an alternative way to visualize the Raspberry Pi to connect it to an external monitor via HDMI cable. Thus, middle-ware connection problems can be eliminated in this way.

Raspberry Pi comes with a Raspbian OS which is based on Debian and it is optimized for the Raspberry Pi's hardware configuration.The command line options, and software updates are similar with original Debian.

Based on the implementation constraints that are discussed in the software requirements specifications section, the software integration to Raspberry Pi has several steps to complete the deployment process. The API Server is a handler of the Server-Edge communications. MQTT protocol has been integrated into Rasberry Pi to execute the edge processing and sending data to the server via MQTT Broker. The communication port is defined as 1884 in the edge platform different than the default API Server protocol which is 1883. Generally, MQTT

Broker is able to bind the ports and do not prevent the connections from external sources. However, using two different ports reduce the possibility of failure and increases the maintainability. Thus, even if the edge platform fails, the API Server would be still online and keep listening for the new incoming connections.

As it is mentioned in the previous sections, the pre-processing block is responsible for retrieving and parsing the incoming CAN data files. In this case, instead of assuming the data is already arrived at the edge computing platform to be processed, we implement a simple File Transfer Protocol server to send the *CarData.csv* file to the Raspberry Pi. Thus, we have all the components to initiate the pothole detection process.

The figure 5.1 and figure 5.2 below shows the examples of a semantic annotation.

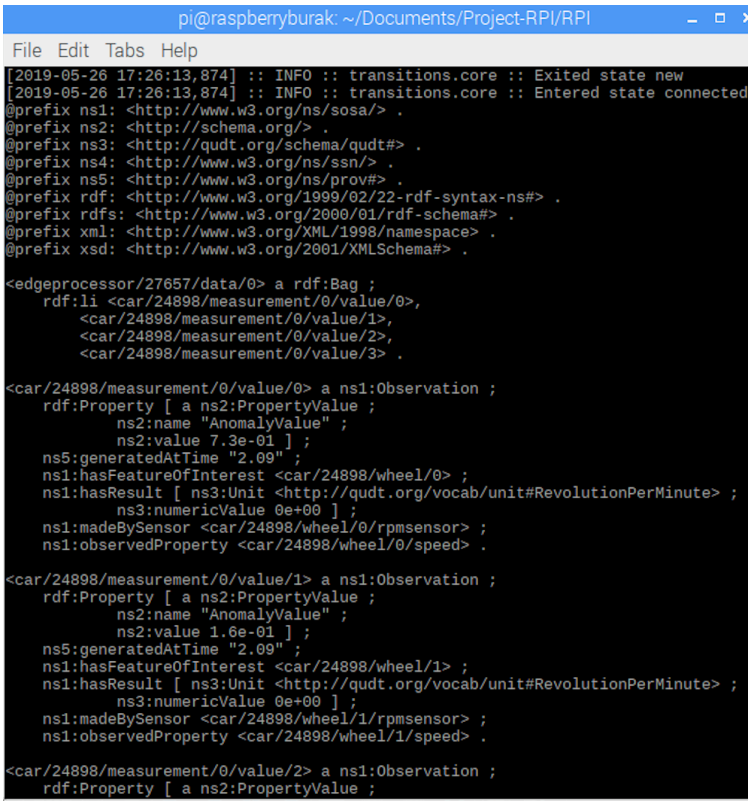

Figure 5.1: Sample of an Enriched Data - Serialized in Turtle Format - RPI

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
    xmlns:ns1="http://schema.org/"
    xmlns:ns2="http://www.w3.org/ns/sosa/"
    xmlns:ns3="http://www.w3.org/ns/ssn/"
    xmlns:ns4="http://www.w3.org/ns/prov#"
    xmlns:ns5="http://qudt.org/schema/qudt#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
>
  <rdf:Description rdf:nodeID="N98d88ed698f845d4af6607e69add8948">
    <ns1:value rdf:datatype="http://www.w3.org/2001/XMLSchema#double">0.2</ns1:value>
    <rdf:type rdf:resource="http://schema.org/PropertyValue"/>
    <ns1:name>AnomalyValue</ns1:name>
  </rdf:Description>
  <rdf:Description rdf:about="car/12144/measurement/0/value/1">
    <ns2:hasResult rdf:nodeID="result1"/>
    <ns2:madeBySensor rdf:resource="car/12144/wheel/1/rpmsensor"/>
    <rdf:Property rdf:nodeID="N98d88ed698f845d4af6607e69add8948"/>
    <ns2:observedProperty rdf:resource="car/12144/wheel/1/speed"/>
    <rdf:type rdf:resource="http://www.w3.org/ns/sosa/Observation"/>
    <ns2:hasFeatureOfInterest rdf:resource="car/12144/wheel/1"/>
    <ns4:generatedAtTime rdf:datatype="http://www.w3.org/2001/XMLSchema#datetime">
      2019-06-08T18:38:08.957512</ns4:generatedAtTime>
  </rdf:Description>
  <rdf:Description rdf:about="car/12144/wheel/0/speed">
    <rdfs:label>the rotation speed of wheel 0 of car 12144</rdfs:label>
    <rdf:type rdf:resource="http://www.w3.org/ns/sosa/ObservableProperty"/>
  </rdf:Description>
  <rdf:Description rdf:about="car/12144/measurement/0/value/2">
    <ns2:madeBySensor rdf:resource="car/12144/wheel/2/rpmsensor"/>
    <rdf:type rdf:resource="http://www.w3.org/ns/sosa/Observation"/>
    <ns4:generatedAtTime rdf:datatype="http://www.w3.org/2001/XMLSchema#datetime">
      2019-06-08T18:38:08.957512</ns4:generatedAtTime>
    <ns2:hasFeatureOfInterest rdf:resource="car/12144/wheel/2"/>
    <ns2:hasResult rdf:nodeID="result2"/>
```

Figure 5.2: Sample of an Enriched Data - Serialized in XML Format - RPI

5.1.2 Server Integration in RDLab UPC

RDLab is funded for promoting research and development of IT projects at the Polytechnic University of Catalunya at the end of 2010 [3]. It has over 160 physical servers, 1000 CPU cores, 3TB RAM and 10 Gbit high-speed network.

There are three main reasons to deploy the application inside the RDLab machine. In research projects, it is crucial to keep the current state of the progress and resources accessible. Secondly, RDLab can offer much more enhanced computing resources compare to commercial computers. Moreover, this idea also gives us the possibility to test the system behavior within a more distributed architecture. It means that each component of the system can be analyzed and

monitored individually in terms of performance and reliability.

5.1.3 Application Integration in Node-RED

The final integration step is to implement a User Interface (UI) application for the software to enable the accessibility for the commercial users.

A good UI design requires consideration of several important characteristics as it is a final interaction of any application with end-users. As an example, some of the design goals can be summarized as; Eye pleasing look, clarity, consistency, familiarity, and forgiveness [51].

Regarding the characteristics above, the goal of this section is to present an interactive user interface application to visualize the data and outcomes of the back-end implementations of our IoT system via graph-based tools. Nowadays, there are plenty of cloud-based services [52] to build and deploy an application such as AWS (Amazon Web Services), Microsoft Azure, Google Cloud, Oracle Cloud, IBM Cloud, and many more. The typical architecture of these systems is a compilation of integrated connected hardware, software, and internet infrastructure. In order to be more compatible with the tasks and requirements, Node-RED [53] is chosen for the deployment of the application. A flow-based programming tool for connecting hardware devices, APIs, and online services, that is developed by IBM's Emerging Technology Services in 2013. It describes the application's behavior as a network of boxes, or "nodes" as it is called in the node-RED environment.

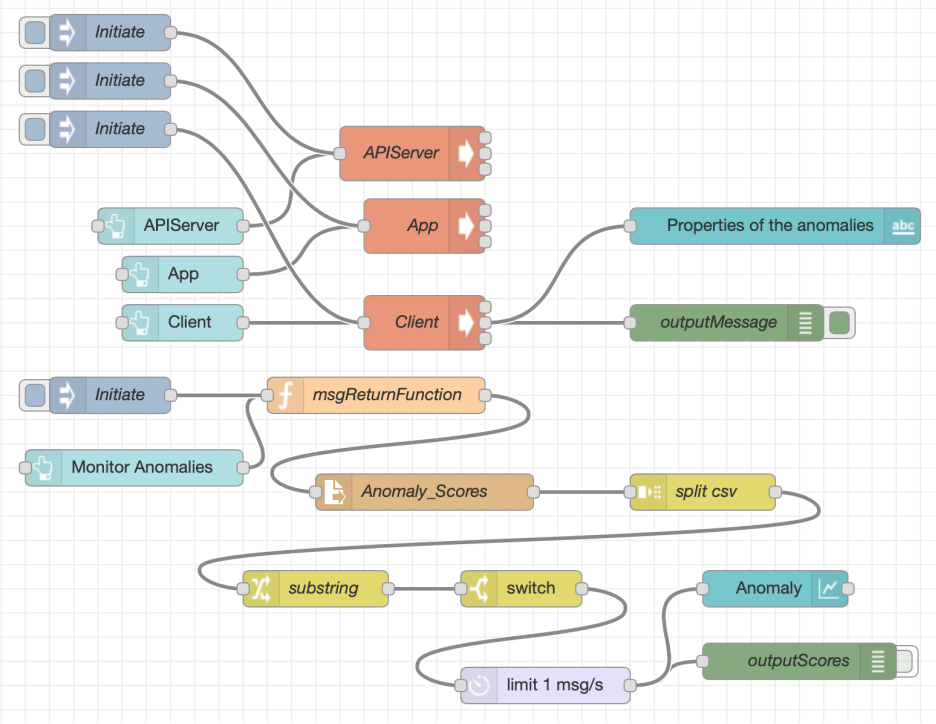

Figure 5.3: Node-RED Architecture Example

There are many nodes in the Node-RED environment that is related to create dashboard elements, networking components, raspberry pi, social media. There are plenty of functions to parse, modify, and interact with other nodes as well. The system is built entirely in NodeJS [54]. Hence, diversity can be increased by simply installing new additional nodes via the node package manager(npm) tool. It also comes with its own CLI. Therefore, it is easy to monitor the flow of the interactions and errors through debugging console as well.

Moreover, the Node-RED is supporting both desktop and mobile platforms. There is not an additional development for mobile deployment. The principle is to develop once, deploy and view everywhere.

Finally, the flows of the architecture that is shown above, can be visualized via built-in dashboard elements. All the simple UI elements such as coloring, text editing, size, and the position of the elements can be defined with built-in control panels as well.

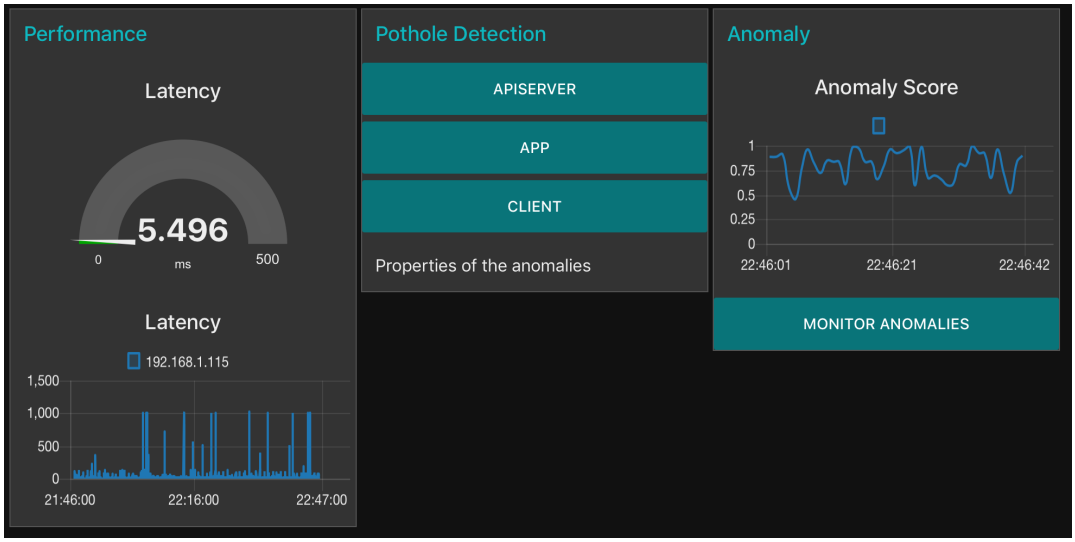

Figure 5.4: Node-RED User Interface Example

Chapter 6

Experimental Study

6.1 Overview

In this part of the project, we aim to test the implementations that are mentioned. The ideal testing environment requires the full deployment of the visualization of the layered architecture model. The RDlab machine is not included for some test cases in order to simplify the setup and eliminate the remote connection problems.

6.2 Testing of the Implementations

In this part of the project, we aim to test the implementations that are mentioned. The ideal testing environment requires the full deployment of the visualization of the layered architecture model. The RDlab machine is not included for some test cases in order to simplify the setup and eliminate the remote connection problems.

6.3 Testing of the Implementations

The goal of the testing part is to test and share the results of the system performance step by step, as it is indicated in the project objectives. In the initial testing plan, we use the common 2.4Ghz speed Wi-Fi network, a Raspberry Pi that is connecting to the server through MQTT broker with port number 1883, and the edge processing MQTT broker with port number 1884.

The tests cover interoperability between different operating systems, execution times, memory usage, network performance, and data stream rate. Moreover, geolocation information is added for testing purposes to the CAN dataset. The location information is crucial to determine the pothole occurrences. By knowing the location of the detections and the detection time together, we would be able to analyze, classify, and enrich the data more realistically.

- **Processor:** Intel Core i7-4770HQ 2.2 Ghz Clock Speed up to 2.7 Ghz Turbo Speed 6MB cache
- **Memory:** 16GB DDR3 1600 Mhz RAM
- **Storage:** 256 GB SSD
- **Networking:** Broadcom BCM43xx 1.0, 802.11 a/b/g/n/ac.
- **Operating System:** MacOS Mojave 10.14.4

The single local computer is used to set up and test the edge device connection and data transfer.

- **Processor:** Broadcom BCM2837B0 Quad-Core A53 (ARMv8) 64-bit @ 1.4GHz.
- **Memory:** 1GB LPDDR2 SDRAM.

- **Storage:** Micro-SD
- **Networking:** Gigabit Ethernet (via USB channel), 2.4GHz and 5GHz 802.11b/g/n/ac Wi-Fi
- **Ports:** HDMI, 3.5mm analogue audio-video jack, 4x USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)
- **Operating System:** Raspbian (Debian based) 4.4 Kernel

RDlab Ubuntu Machine is used to test the server integration and the network speed.

- **Processor:** Intel Quad-Core x86_64 2.3 Ghz Clock Speed
- **Memory:** Red-Hat Virtual Ram Memory 4GB
- **Storage:** 200 GB Hard Disk
- **Networking:** Gigabit Ethernet (via USB channel), 2.4GHz and 5GHz 802.11b/g/n/ac Wi-Fi
- **Operating System:** Ubuntu 18.04 LTS

6.3.1 Interoperability Test

In order to execute these tests, we created an image of the system in each operating system, which are respectively Windows 10, MacOS Mojave, and Raspbian.

As it is indicated in the above sections, the previous thesis project was developed within the Microsoft Windows platform due to its context and the time restrictions. Respecting that, some considerations need to be taken into account in this project for the users perspective. Still, this statement does not mean that any software is written in a specific platform would not work in others properly.

What is proposed and focused here is that platform independence [55] that is a widely accepted universal principle that software applications should be adjusted to be able to work on many operating systems and hardware platforms, without too much modification. The platform independence is relevant to the concepts of portability, re-usability, universality and financial viability. Portability, multiple usage, and financial viability can be considered as primary objectives for the achievement of platform-independent software architectures [55].

During the pre-system tests, several observations and some difficulties are noticed such as:

- Due to the administrative privileges and security concerns in Unix based operating systems, such as MacOS or Debian, the integrated development environment must run with 'root' privileges.
- For the external communication protocols usage, such as MQTT and TCP/IP, might cause connection refused error from the system for the security reasons [56]. It may happen mostly due to administrative privileges, wrong public key, broken ports, or socket programming bugs.
- A firewall might block the port that is intended to connect. Therefore, it is advisable to leave the firewall in off status, during the executions and tests. Obviously, this solution would not be suitable in real-life implementation scenarios, as it would bring high-security breaches and vulnerabilities to the system such as malware injection. However, the security-related concerns and implementations are out of this thesis context.
- Another critical control point could be, checking all the connection ports, whether they are in use or not.
- Eclipse Mosquitto [18] is an open-source message broker that implements the MQTT protocol. The configuration file of Mosquitto, which is called

mosquitto.conf, has a parameter called **allow_anonymous**. This parameter is controlling some of the features of MQTT respecting to [**read-write-rewrite**]. To establish the connection successfully, allow_anonymous must be True. The reason is that MQTT uses port number 1883 as a default. Thus, apart from the default connection, anything else is considered anonymous. In this project, two different port numbers are used to differ edge to server and server to client communications.

Observations

Both semantic enrichment and pre-processing blocks are executed on Raspbian OS. The RDLab Machine is used to test the Ubuntu 18.04 OS for the server integration tests. On the other hand, MacOS Mojave based computer is used to replicate the same results for the server integration as well. As a result of this experiment, the interoperability test between each operating system has been covered successfully to see the behaviors of the different operating systems regarding the integration of different processes.

6.3.2 Execution Time Performance Test

In this section, we intend to test the execution time differences between Raspberry Pi and the local computer that is used to simulate the semantic enrichment process. Since the computational powers are significantly different due to limited and relatively lower resources that are used in Raspberry Pi, it is expected to see slower execution times in Raspberry Pi, compared to the local computer.

Figure 6.1: Semantic Enrichment Execution Time Comparison - Serialized in Turtle Format

Observations

The test indicates that Raspberry Pi delivers nearly 10-14 times slower processing compare to more resourceful machines 6.1. Even though, the task that is given to be computed is not demanding huge resources. Still, in terms of real-life expectations, between [0.10-0.20] secs of processing time for each entry, could be a downside depends on the data set. In the example, only a small fraction of the dataset is used, which are 2000 entries that are sent with 1 entry/sec data stream rate to simulate the processing power. If we consider having a few hundred thousands of instances to be computed, or higher data streaming rate, the difference would be enormous. Additionally, after a few executions, Raspberry Pi starts to show some thermal problems as the heat goes up and gets very slow that is not operable enough to control the unit properly.

6.3.3 Memory Usage Test

In this testing case, each function has been analyzed related to the pre-processing 6.3, and semantic enrichment 6.2 unit in the edge platform which is held in Raspberry Pi. Starting from connection establishment, processing the information, and the final delivery of the results.

In the case of the semantic enrichment process, in order to get precise measurement results and comparisons, each type of syntax regarding RDF goes through the same process under the same conditions in the edge unit by just specifying the file format.

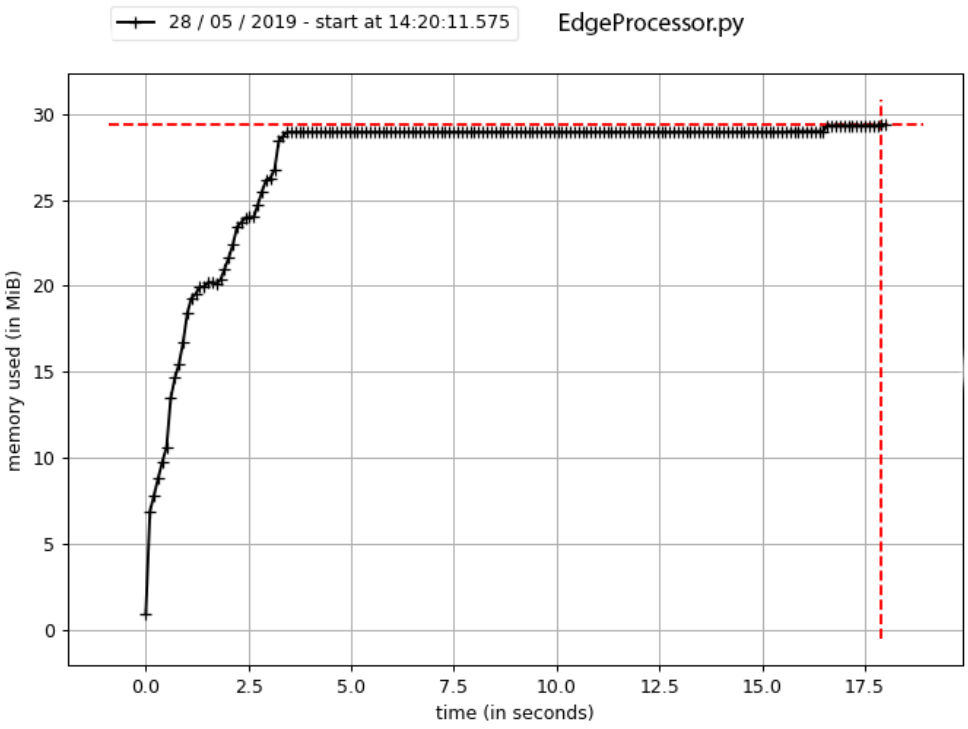

Figure 6.2: Memory Usage of Semantic Enrichment Unit in Turtle Format

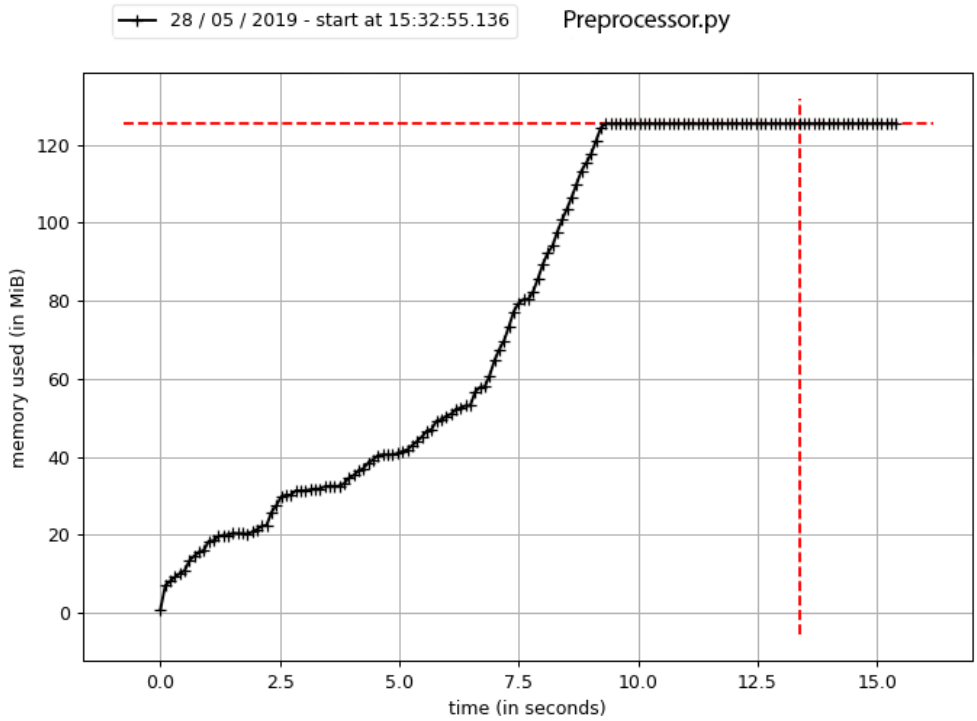

Figure 6.3: Memory Usage in Pre-processing Unit

Observations

As we can see from the graphs, the memory usage of semantic enrichment is 30 MiB [1] that is equal to approximately 3.3% of the total amount. The pre-processing usage is equal to 12% of the total amount. However, the memory usage depends on the number of incoming data. In case the memory cannot provide enough processing power for all the incoming data, more than one Raspberry Pi with parallel programming techniques can be considered. Hence, the distribution of the workload would give some benefits to handle the memory bound and the stress over the system. Yet, it is not one of the goals of this project.

[1] 1 MiB = 1.048576 Mb

6.3.4 Network Performance Test

The testing of network performance is divided into two categories. We intend to measure the latency and Round Trip Time of the data streaming process with different network configurations. Due to a lack of sim-card injection ability devices, the network performance is simulated by 2.4 Ghz and 5Ghz Wi-Fi speed networks.

Before testing, every possible resource demanding applications such as online data streaming, and downloads are eliminated in network connection to obtain more accurate results. The tests are obtained by processing 2000 CAN bus data entries and send over the network to the RDLab server. In figure 6.4 the comparison between 5G and 2.4G speed performance can be seen.

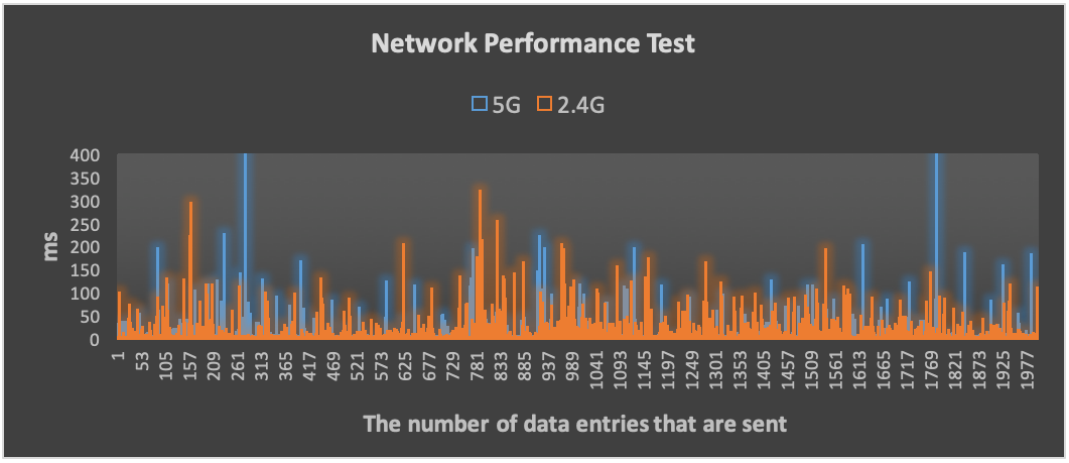

Figure 6.4: Latency Test with 5G vs 2.4G Example

In figure 6.5 below, it is shown that how the latency between the RDlab server, Raspberry Pi, a local computer behaves. The idea of doing this test is to have an idea of the Raspberry Pi's built-in 5G Wi-Fi hardware and software performance, compared to more advanced configurations.

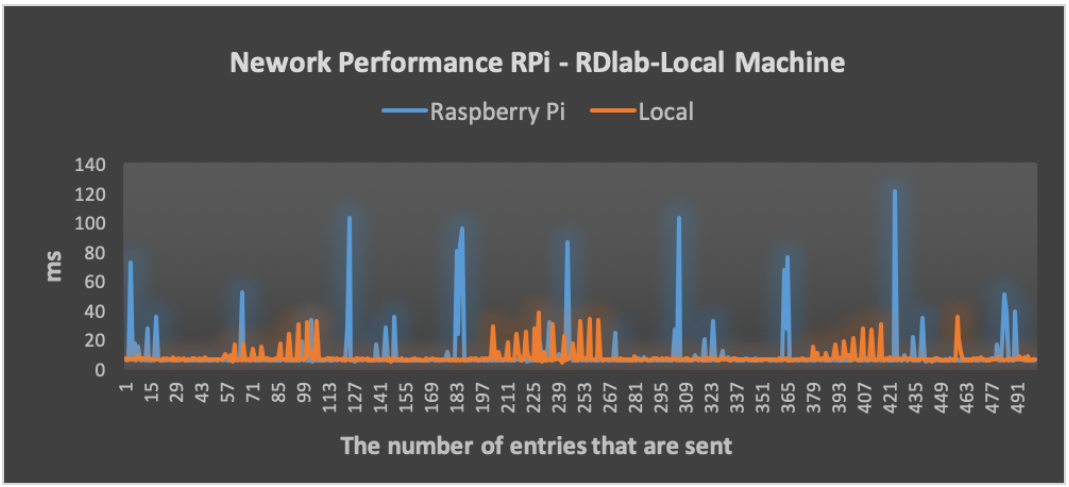

Figure 6.5: Latency Test Example - 5G

The final networking performance test refers to the Round Trip Time (RTT) between the Raspberry Pi and the RDlab Server with different Wi-Fi speeds. In Figure 6.6 the results of RTT measurements can be seen.

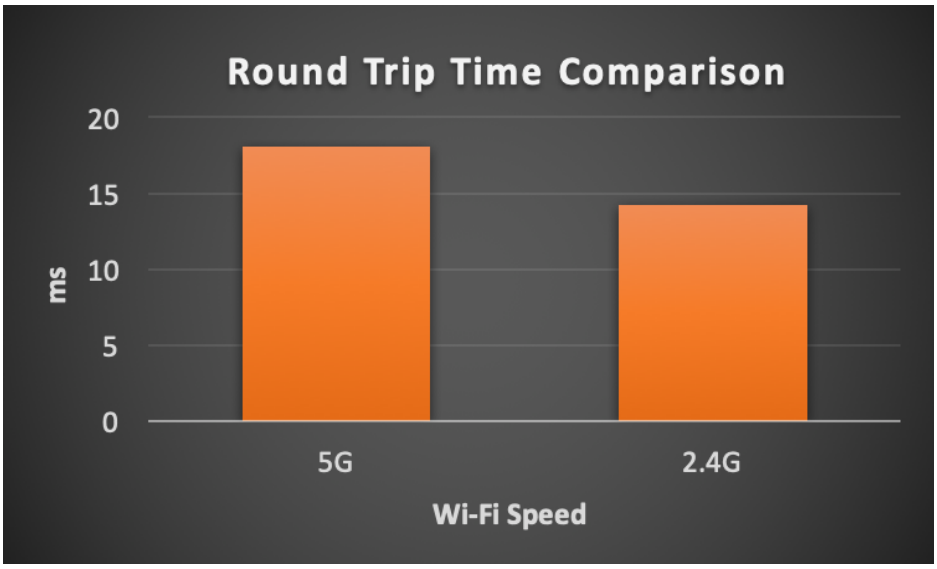

Figure 6.6: Round Trip Time Measurement - Raspberry Pi and RDLab

Observations

Despite known as high-speed and accurate, 5G networks came out with unstable results as we can see in figures above. This may vary due to the established quality of 5G, the quality of the antennas and the distance, and the infrastructure quality. During the tests for the 5G measurements, the average latency was observed between [6-18] ms with an average of 16ms and only 300 data entries have sent with lower than 6ms. On the contrary, 2.4 Ghz was provided quite acceptable results with this dataset. Most of the packets were sent between [5-14] ms with an average of 12ms, and over 1000 data entries are sent lower than 6ms by using 2.4G networks. However, it should be noted that for higher data streaming rates and resource demanding tasks, the bandwidth of the 2.4 Ghz may not be enough.

6.3.5 Data Stream Rate Test

Data streams might have dramatic spikes in data volume, such as high event rates during critical states in pothole data processes. The peak load can be higher than typical loads in the areas that the traffic load is relatively higher. Usually, it is impractical to provide resources to handle the spike load adequately [54]. However, accurate data stream processing is most crucial in such circumstances of high data load. There are two approaches to determine this:

- Provide estimated processing results instead of accurate to assure high performance by shedding a controllable fraction of input stream data.
- Another approach is to provide accurate results by buffering overload. However, this option carries a risk of failure to keep up with the input rate.

Our tests over the data stream rate is based on the second approach since it is important for us to keep the quality of the measurements higher. It means

that we want to reach the minimal or in ideal conditions zero data loss in the data streaming process when we increase the data streaming rate. The tests are covering the data streaming process through a CSV file by reading it with multiple lines in one second, instead of just one as it is on a regular basis. The MQTT Broker was a connection provider to transfer all the information to the server-side. Hence, we would be able to see for how long the MQTT Broker and the Raspberry Pi hardware resources can hold its state as stable and working. In order to achieve these tasks, we have created multiple test scenarios with different streaming rates. The total amount of data entries were up to 2000. Each entry is representing a single **msg.payload** that is a size of 4000 bytes. The test case is created by reading 15, 20, 25, 30, 50, 100 entries in a second that has been processed, and continuously sent from Raspberry Pi to the server.

Observations

According to our test results, it is convincing that the MQTT Broker cannot handle multiple entries simultaneously. The streaming up to 15 lines per second, was the most stable result without losing connection or data, despite the execution time increase. After this threshold, the MQTT has started to struggle to keep its state and failed at the end. However, with the appropriate data stream rate, there wasn't any data loss during the tests since MQTT operates on top of the TCP/IP protocol, and it guarantees the transmission of the packets. In the figure below, the performance of each streaming rate and the total amount of processed lines that are achieved by the MQTT Broker has shown before the failure.

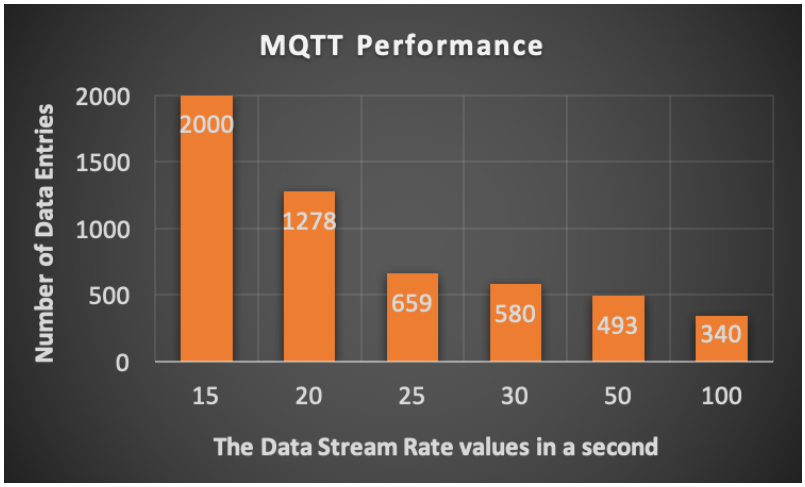

Figure 6.7: MQTT Performance Comparisons

In figure 6.8 below, the execution performances of each streaming rate are analyzed. The numbers are selected according to the lowest number of entries that are processed by the 100 lines/sec data rate streaming.

Figure 6.8: Data Streaming Rate Execution Time Comparisons

As we can see from the graph above, while the data streaming rate was increasing, Raspberry Pi was getting unstable. Also, it has crashed several times after 4-5 execution cycles due to workload and thermal problems that forced us to restart and cooldown the machine.

6.4 Semantic Enrichment with Geolocation Information

The enrichment of the data with location information is an experimental study in the context of this thesis work. The generated locations are not a part of the original CAN dataset. Thus, the latitude and longitude information has been generated randomly via python script inside the edge platform to replicate the process. In order to be as close as possible to the real case scenario, the location information has been chosen for the Barcelona city. The latitude information respect to this information is between [41.40 - 41.41] and the longitude information is between [2.09-2.10]. The figure 6.9 below shows an example of a location enrichment.

```
<rdf:Description rdf:nodeID="Nf07d88a958344d9ea4e20f1640022fce">
  <ns4:value rdf:datatype="http://www.w3.org/2001/XMLSchema#double">0.35</ns4:value>
  <ns4:name>AnomalyValue</ns4:name>
  <rdf:type rdf:resource="http://schema.org/PropertyValue"/>
</rdf:Description>
<rdf:Description rdf:about="car/19033/wheel/2/speed">
  <rdfs:label>the rotation speed of wheel 2 of car 19033</rdfs:label>
  <rdf:type rdf:resource="http://www.w3.org/ns/sosa/ObservableProperty"/>
</rdf:Description>
<rdf:Description rdf:about="edgeprocessor/2603/data/0">
  <rdf:li rdf:resource="car/19033/measurement/0/value/2"/>
  <ns3:generatedAtTime rdf:datatype="http://www.w3.org/2001/XMLSchema#datetime">
    2019-06-10T22:11:49.385246</ns3:generatedAtTime>
  <ns3:generatedAtCoordinates rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    41.40401057035725,2.0926754729294688</ns3:generatedAtCoordinates>
  <rdf:li rdf:resource="car/19033/measurement/0/value/0"/>
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag"/>
  <rdf:li rdf:resource="car/19033/measurement/0/value/1"/>
  <rdf:li rdf:resource="car/19033/measurement/0/value/3"/>
</rdf:Description>
<rdf:Description rdf:about="car/19033/wheel/2/rpmsensor">
  <rdfs:label>a sensor observing the RPM of a cars wheel</rdfs:label>
  <rdf:type rdf:resource="http://www.w3.org/ns/sosa/Sensor"/>
</rdf:Description>
<rdf:Description rdf:about="car/19033/wheel/3/rpmsensor">
  <rdfs:label>a sensor observing the RPM of a cars wheel</rdfs:label>
  <rdf:type rdf:resource="http://www.w3.org/ns/sosa/Sensor"/>
</rdf:Description>
```

Figure 6.9: Enriched Data with the Geolocation Information

Chapter 7

Conclusions

In this thesis, the challenges regarding implementations and evaluation of real-time IoT data streaming and processing in a real-life infrastructure, under the real-time requirements are studied to answer the research questions stated in the introduction about the **problem of detecting potholes in roads from cars data streams using edge processing devices**.

With the emergence of Edge and Fog Computing as the next layers close to the IoT layer, it is now commonly admitted that it is not a feasible solution to push all the data to Cloud-based systems anymore without pre-processing. Despite the huge computing power of Cloud-based systems, putting all the processing load over these systems could bring high latency in processing data and considerable storage and energy costs while performance requirements might not be met. Instead, it is considered as an alternative to use the processing power of IoT and edge devices as the microchip technologies have reached the point that they can provide enough computational power to do some operations before sending data to Cloud.

In this context, this thesis uses a **layered architecture model** that consists of four main layers: the IoT layer consisting of computing devices in the cars, edge layer consisting of Raspberry Pi and server layer consisting of server nodes

at RDLab of CS Department of UPC. The tasks that are allocated in the Raspberry Pi were data sensing, filtering, anomaly detection, semantic annotation generation, and semantic enrichment. As a dataset, the simulated and simplified CAN bus data was used due to the limitations of the actual car driving event to obtain this data realistically. Besides, geolocation information was generated. Since this information was not represented in the original dataset, the coordinates are randomly generated. Adding the geolocation information brought significant value to the enrichment process as it would make sense to know contextual information about detected events (in our problem, the exact location of the pothole occurrences in the roads) to classify them in a better way.

Considering the fact that, even our cell phones are capable of doing much more than basic operations these days, including AI and Machine Learning applications are feasible. In this regard, **Hierarchical Temporal Memory (HTM) algorithm** was implemented to predict the event anomaly detection scores. The results of the HTM algorithm are promising as it is on a continually evolving phase. However, despite HTM algorithm differs from other types of machine learning algorithms regarding less demand for massive data entries, it is beneficial to provide more than a certain amount of data to get accurate and reliable results. This amount can be different depending on the provided dataset, configuration of HTM parameters, and the target outcome such as accurate or faster. Moreover, with the given information of the simplified dataset, the HTM algorithm struggled to find the patterns and stating accurate predictions in some cases. The idea of using the processing power of the Raspberry Pi made a positive impact on the results despite its limited resources.

Furthermore, for the **reasoning**, a business rule engine was implemented with an *if-then* and *fact-action* based logical statements. For the research purposes, PyKnow rule engine is implemented to define these logical statements. As a result

of this experiment, four statements were created to define some set of actions that need to be taken in specific cases to improve the control and manageability of the system. Also, it is foreseen that it creates a bridge between technical and business authorities.

A **user interface** application was created in the Node-RED programming tool for visualization. The reason for selecting this tool is that the Node-RED is specifically created for IoT environments with node-based architecture. The outcome of this experiment has significant benefits since nodes are designed to work with a drag and drop method, it offers creating customized dashboards in a shorter time compared to conventional web development techniques. Also, IBM adds new nodes to the system regularly, which brings scalability to the system.

Last but not least, the **performance measurements** of the entire system within the context of given parameters for the IoT real-time data streaming were studied. Respectively, interoperability, execution time, data stream rate, memory usage, and network performance. The tests are essential assets of this thesis work to be able to realize the real-life infrastructure implementation constraints. The system has been tested in various operating systems to asses availability. The Raspberry Pi was able to provide enough CPU, and memory for the executions, and it was able to handle higher date stream rates within certain limits. During the network performance tests, it is observed that the current quality of the 5G implementation is not fully reliable yet. However, this is an observation, not a conclusive statement as only a small area was used where these tests are obtained.

In conclusion, we believe that this thesis has achieved the proposed objectives by presenting the studies, the proposal, implementation, results and, finally, the conclusions based on them.

Chapter 8

Extensions and Future Work

The designed architecture and implementations show some new results within the case of a real-life application. However, some issues remain as this is a proof of concept and due to several difficulties encountered during the implementation. The generated semantic annotations in multiple formats weren't directly accessible. Therefore, this data can be stored in semantic repositories such as Ontotext to improve persistence.

The edge layer can be improved by adding more than one Raspberry Pi and used as a parallel computing platform to improve the performance of data stream processing and therefore increasing the stream data rate in the input.

The CAN dataset that is used for the car's data streaming can be regenerated to be extended with the actual car driving event to obtain more realistic data entries. Therefore, the anomaly detection of the HTM algorithm would be expected to provide more accurate results.

Moreover, the mobile or desktop application for drivers can be re-designed at an advanced level to offer more controls and awareness of the real-time road conditions. For instance, a real-time geolocation integration with the anomalies, weather conditions, traffic information, can be integrated into one of the open-source maps that can be done available to users for a better overview of the entire

road management system.

Improvements can also be done to create a more comprehensive system with the aim to support a variety of final users (drivers, administration, road control authorities, etc.). The technology stack that is used during this project can be broadened by considering other technologies, communication protocols and data caching techniques at various layers of the architecture. Finally, implementing more intelligent functions at the edge computing layer based on event complex processing would improve the usefulness of the project.

Acronyms

- **Pothole:** A depression or hollow in a road surface caused by wear or subsidence.
- **IEEE:** Institute of Electrical and Electronics Engineers
- **MQTT:**. Message Queuing Telemetry Transport)
- **IoT:** Internet of Things
- **OS:** Operating System
- **API:** Application Programming Interface
- **TCP:** Transmission Control Protocol
- **IP:** Internet Protocol
- **HTTP:** Hypertext Transfer Protocol
- **SSH:** Secure Socket Shell
- **CAN:** Controller Area Network
- **ECU:** Electronic Car Units
- **HTM:** Hierarchical Temporal Memory
- **API:** Application Programming Interface
- **RDlab:** Resarch and Development Laboratory
- **m-MTC:** massive Machine Type Communications
- **BRMS:** Business Rule Management System
- **AI:** Artificial Intelligence

References

[1] Yolanda Blanco-Fernández, José J. Pazos-Arias, Alberto Gil-Solla, Manuel Ramos-Cabrer, and Martín López-Nores. Semantic reasoning: A path to new possibilities of personalization. pages 720–735, 2008.

[2] Muhammad Intizar Ali, Naomi Ono, Mahedi Kaysar, Zia Ush-Shamszaman, Thu-Le Pham, Feng Gao, Keith Griffin, and Alessandra Mileo. Real-time data analytics and event detection for iot-enabled communication systems. *J. Web Semant.*, 42:19–37, 2017.

[3] RDLab UPC. History. *Web Page*, 2019.

[4] IHS Markit Ltd. IoT platforms: Enabling the Internet of Things. *White Paper*, pages 4–5, 2016.

[5] Automotive Edge Computing Consortium. General Principle and Vision. *White Paper*, page 1, 2018.

[6] Blesson Varghese, Nan Wang, Sakil Barbhuiya, Peter Kilpatrick, and Dimitrios S. Nikolopoulos. *Challenges and Opportunities in Edge Computing.* IEEE Computer Society, New York, NY, USA, 2016.

[7] Opensignal Inc. The State of LTE (February 2018). *Article*, 2018.

[8] Halldor Sigurdsson Ferry Grijpink, Alexandre Ménard and Nemanja Vucevic. The road to 5g: The inevitable growth of infrastructure cost. *Article*, 2018.

[9] Numenta Inc 2011. Hierarchical Temporal Memory Including HTM Cortical Learning Algorithms. *White Paper*, page 4, 2011.

[10] Reinout Van Hille, Fatos Xhafa. Semantic IoT for Reasoning and Big Data Analytics. *Master Thesis*, 2018.

[11] Raspberry Pi. What is a Raspberry Pi? *Web Page*, 2019.

[12] W. Rahiman and Z. Zainal. An overview of development gps navigation for autonomous car. pages 1112–1118, June 2013.

[13] Bo Zhang, Tze Sing Eugene Ng, Animesh Nandi, Rudolf H. Riedi, Peter Druschel, and Guohui Wang. Measurement-based analysis, modeling, and synthesis of the internet delay space. *IEEE/ACM Trans. Netw.*, 18(1):229–242, 2010.

[14] Bo Zhang, Eugene Ng, Animesh Nandi, Rudolf Riedi. Determine Signal Type/Strength - VZAccess. *Web Page*, 2018.

[15] Institute of Electrical and Electronics Engineers. IEEE Guide for Software Requirements Specifications. *Documentation*, 1984.

[16] Moe Long. Best Raspberry Pi Alternatives 2019. *Web Article*, 2019.

[17] Eclipse Foundation. Eclipse Paho. *Web Page*, 2019.

[18] Mosquitto Org. Eclipse Mosquitto, MQTT Broker. *Web Page*, 2019.

[19] MQTT Org. MQTT. *Web Page*, 2019.

[20] Robert Bosch GmbH. CAN Specification. *Documentation*, 1991.

[21] Thomas Huybrechts, Yon Vanommeslaeghe, Dries Blontrock, Gregory Van Barel, and Peter Hellinckx. Automatic reverse engineering of CAN bus data using machine learning techniques. pages 751–761, 2017.

[22] Telefonaktiebolaget LM Ericsson (publ). Smart vehicles and transport. *Web Article*, 2018.

[23] Crina Grosan, Ajith Abraham. *Rule-Based Expert Systems*. Springer, Berlin, Heidelberg, 2009 - 2020.

[24] Stephen D. Hendrick, Kathleen E. Hendrick. The Business Value of Business Rules Management Systems. *White Paper*, 2012.

[25] TechSci Research. Global Business Rules Management System Market By Component , By Organization Size , By Deployment Mode , By Vertical, By Region, Competition, Forecast & Opportunities, 2024. *Article*, 2019.

[26] Dan Stearns. Non-functional Requirements. *University Lecture*, 1998.

[27] David Ameller, Xavier Franch. Non-Functional Requirements as drivers of Software Architecture Design. *PhD Thesis*, January 2014.

[28] Mortadha Hamad and Banaz Anwer Qader. Data pre-processing for knowledge discovery. *Tikrit Jourual of Pure Science*, 19:143–148, 01 2014.

[29] He Jiang, Xin Chen, Shuwei Zhang, Xin Zhang, Weiqiang Kong, and Tao Zhang. Software for wearable devices: Challenges and opportunities. pages 592–597, 2015.

[30] Jeff Hawkins, Marcus Lewis, Mirko Klukas, Scott Purdy, and Subutai Ahmad. A framework for intelligence and cortical function based on grid cells in the neocortex. *Frontiers in Neural Circuits*, 12, 01 2019.

[31] Numenta Platform for Intelligent Computing (NuPIC). NuPIC 1.0.5 API Documentation. *Technical Documentation*, 2017.

[32] Altti Ilari Maarala, Xiang Su, and Jukka Riekki. Semantic reasoning for context-aware internet of things applications. *IEEE Internet of Things Journal*, 4(2):461–473, 2017.

[33] World Wide Web Consortium (W3C). SPARQL 1.1 Query Language. *Documentation*, 2012.

[34] JBoss Org. JBoss Community Documentation. *Documentation*.

[35] Richard C. Dorf. *The Electrical Engineering Handbook, Third Edition - 6 Volume Set (Electrical Engineering Handbook)*. CRC Press, Inc., Boca Raton, FL, USA, 2006.

[36] Horatiu Cirstea, Claude Kirchner, Michael Moossen, Pierre-Etienne Moreau. Production Systems and Rete Algorithm Formalisation. *Paper*, page 3, 2008.

[37] K.H. Rosen. *Discrete Mathematics and Its Applications with MathZone*. McGraw-Hill Higher Education, 2006.

[38] Paul Vincent, Co-Chair OMG PRR FTF TIBCO Software. OCEB White Paper on Business Rules, Decisions, and PRR. *White Paper*, 2008.

[39] Nwigbo Stella, Agbo Okechuku Chuks. Expert System: A Catalyst in Educational Development in Nigeria. *Paper*, 2011.

[40] Ronald G. Ross. Business Rules vs. Expert Systems: Same or Different? *Article*, page 1, 2012.

[41] BizRules Blog. What is the Difference Between Data-based, Rule-based, and Knowledge-based Systems? . *Blog*, page 1, 2006.

[42] Bala Subramanyam Lanka. Rete Algorithm. *Web Page*, 2016.

[43] Ms. Pallavi M S, Mr. Vaisakh P, Ms. Reshna N P. Implementation of RETE Algorithm Using Course Finder System. *Paper*, pages 2–3, 2016.

[44] Carole-Ann Berlioz. How the Rete Algorithm Works? *Web Page*, 2011.

[45] Bala Subramanyam Lanka. Different Versions of Rete Algorithm. *Blog*, 2016.

[46] Joseph C. Giarratano, Ph.D. CLIPS 6.4‘ User's Guide. *Article*, pages 3–4, 2016.

[47] James L. Crowley. Intelligent Systems: Reasoning and Recognition. *Lecture*, pages 5–7, 2012.

[48] Roberto Abdelkader Martínez Pérez. Introduction to PyKnow. *Article*, page 1, 2018.

[49] Roberto Abdelkader Martínez Pérez. PyKnow Documentation. *Book*, pages 4–5, 2018.

[50] Constantin Kaplinsky. TightVNC Software. *Documentation*, 2006.

[51] Uday Bhaskar, P Prathap Naidu, S.R. Ravi Chandra Babu, and P.Govindarajulu . Principles of good screen design in websites. *International Journal of Human Computer Interaction*, 2:48–57, 11 2011.

[52] Priyanshu Srivastava and Rizwan Khan. A review paper on cloud computing. *International Journal of Advanced Research in Computer Science and Software Engineering*, 8:17, 06 2018.

[53] Node-RED Org. Node-RED Documentation. *Documentation.*

[54] Anja Klein, Gregor Hackenbroich, and Wolfgang Lehner. How to screen a data stream - quality-driven load shedding in sensor data streams. pages 76–90, 2009.

[55] Siniša Vlajić, Ilija Antović, and Dusan Savic. General concept of platform independency model. pages 1–2, 06 2016.

[56] BMC Communities. BMC Analytics for BSM. *Article 000116254*, page 1, 2017.

[57] Martin Serrano, Amelie Gyrard. A Review of Tools for IoT Semantics and Data Streaming Analytics. *Paper*, pages 4–5, 2018.