Pedro António
Carvalho Abade

VR-Banway: deploying a body area network gateway on single-board computers and mesh networks

VR-Banway: implementação de um agregador local de dados de sensores em computadores de placa única e redes de malha

Universidade de Aveiro
2017

Departamento de Eletrónica,
Telecomunicações e Informática

Pedro António Carvalho Abade

VR-Banway: deploying a body area network gateway on single-board computers and mesh networks

VR-Banway: implementação de um agregador local de dados de sensores em computadores de placa única e redes de malha

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor José Maria Amaral Fernandes, Professor auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Ilídio Fernando de Castro Oliveira, Professor auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Prof. Doutor Arnaldo Silva Rodrigues de Oliveira
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Rui Pedro de Magalhães Claro Prior
Professor Auxiliar da Faculdade de Ciências da Universidade do Porto

Prof. Doutor José Maria Amaral Fernandes
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

agradecimentos/ acknowledgements

Ao meu orientador, Professor Doutor José Maria Amaral Fernandes, e ao meu co-orientador, Professor Doutor Ilídio Fernando de Castro Oliveira, pelo apoio, disponibilidade e suporte, tanto a nível técnico como pessoal, oferecidos durante a realização deste projeto.

À equipa do Departamento de Ciência de Computadores da Universidade do Porto, como forma de apreciação, pela disponibilidade e colaboração.

Em geral, a todos os meus colegas de laboratório pelo bom ambiente que proporcionaram no local de trabalho e pela disponibilidade e participação na discussão de alguns temas, assim como a todos os meus amigos, exteriores ao âmbito académico, pelo apoio e amizade proporcionada ao longo de todo este percurso.

À minha família, fonte de compreensão e apoio tanto nos bons como nos maus momentos.

À minha namorada e amiga Mariana Sousa que esteve sempre ao meu lado, independentemente dos momentos de maior indiferença e angústia, para me transmitir a energia e alegria necessária para concluir este projeto, agradeço fundamentalmente por tudo.

palavras-chave

Internet das coisas; Sistemas de monitorização; Computadores de placa única; Baixo consumo de energia; Redes Ad hoc.

resumo

A *Internet of Things* (IoT) é uma categoria genérica das arquiteturas de TIC que inclui o uso de sistemas baseados em sensores e comunicações. Um elemento comum das arquiteturas IoT é o agregador que recolhe dados de sensores nas proximidades e reencaminha-os para serviços remotos de mais alto nível. O projeto VR2Market, no qual este trabalho está integrado, usa duas implementações do agregador de dados, implementados em Android e RPI.

Com o novo módulo proposto, inspirado na IoT, é possível migrar os agregadores de dados para dispositivos mais pequenos e mais eficientes mantendo a abstração de programação de alto nível.

Neste trabalho, propomos e implementamos uma nova versão do agregador de dados, chamado VR-Banway, usando o módulo computacional Intel Edison, tendo em consideração a integração de novas camadas de serviços no VR2Market, especialmente no que diz respeito ao suporte de redes *Ad hoc*.

VR-Banway provou ser uma solução capaz de substituir o componente de agregador de dados existente no sistema VR2Market. A nova abordagem usa um módulo mais pequeno, reduz o consumo de energia e é mais portátil. VR-Banway foi usado no contexto de monitorização de bombeiros, mas está preparado para ser implementado noutros domínios.

keywords Internet of Things; Monitoring systems; Single-board computers; Low power consumption; Ad hoc networks.

abstract

Internet of Things (IoT) is a generic category of ICT architectures that includes the use of sensor-based, communication-enabled systems. A common architectural element in IoT is the sensors gateway that collects data from nearby sensors and relays them to higher-order remote services. The VR2Market project, in which this work is integrated, uses two implementations of the gateway, based on Android smartphones and RPI boards.

With the new proposed IoT-inspired computing module, it is possible to migrate gateways to a smaller, more efficient hardware, while retaining the high-level programming abstraction.

In this work, we propose and implement a new version of the gateway, named VR-Banway, using the Intel Edison compute module, taking into consideration the integration with additional service layers in VR2Market system, especially with respect to the required Ad hoc networks support.

VR-Banway proved to be a solution capable of replacing the existing gateway component in the VR2Market system. The new approach uses a smaller module, reduces power consumption and is more portable. VR-Banway has been used in the context of firefighters monitoring, but is ready to be deployed in other domains.

I. Contents

I. Contents i

II. List of figures iii

III. List of tables v

IV. List of Acronyms vii

1 Introduction 1

1.1 Objectives and Contributions 2

1.2 Dissertation Structure 3

2 State of the Art 5

2.1 Internet of Things overview 5

2.1.1 Typical architecture 6

2.1.2 Sensors Gateway 8

2.1.3 Efficient Messaging 10

2.2 Communication using Ad hoc networks 13

3 The VR2Market and new opportunities 15

3.1 Scenarios and workflows 15

3.2 VR2Market Architecture 17

3.2.1 Sensors 18

3.2.2 VRUnit: Android and RPI 21

3.2.3 Data transport and persistence 25

3.2.4 Front-End 26

3.3 New adaptations and opportunities 29

4 VR2Market System refactoring 31

4.1 What we are adding and changing 32

4.2 RPI VR-Unit components 32

4.3 Acquisition workflow 34

5 NetAPI service integration 37

5.1 NetAPI workflow 37

5.2 Adding bidirectional communication 40
5.3 The new scenarios and functionalities 41
6 VR-Banway: Refactoring for Intel Edison 43
6.1 The Intel Edison 43
6.2 VR-Banway: Intel Edison based VR-Unit 45
6.2.1 Internal broker refactoring 47
6.3 New opportunities by using Intel Edison 48
7 Other System changes 49
7.1 VR-Unit configuration method 49
7.2 Integrating new sensors and actuators 50
7.3 Front-End components changes 53
8 Refactored VR-Unit and VR-Banway evaluation 55
8.1 Aggregator comparison 55
8.2 Integration in the VR2Market system 57
8.3 Ad Hoc scenarios 61
9 Conclusion and future work 65
9.1 Future work 65
References 67
Appendices 71
A.1 Public Presentations 71

II. List of figures

Figure 1 – Layers of a typical Internet of Things architecture [11]. 6
Figure 2 – Components architecture of a typical Internet of Things scenario [18]. 8
Figure 3 – Example of IoT gateway usage in a smart house monitoring system [20]. 9
Figure 4 – IoT architectural components and communication protocols [25]. 10
Figure 5 – Broker messaging concept, publish-subscribe pattern [26]. 11
Figure 6 – Wireless Ad hoc nodes range [37]. 14
Figure 7 – Sensors attached to the firefighter clothes. 16
Figure 8 – High level architecture of VR2Market. 17
Figure 9 – Sensors examples:(A) FREMU, (B) HELMET, (C) Weather station, (D) G2RAYS. 20
Figure 10 – Vital Jacket sensor and corresponding cloth. 21
Figure 11 – VR-Unit external configuration. 22
Figure 12 – Android VR-Unit application. 23
Figure 13 – Raspberry Pi 3. Source: goo.gl/n4Jbzf. 25
Figure 14 – Sensors data flow from the VR-Unit to the Database. 26
Figure 15 – Project component to manage acquisitions (VR-Data). 27
Figure 16 – Project component to visualize past collected data (VR-Mission Review). 28
Figure 17 – Project component to visualize real-time collected data (VR-Commander). 28
Figure 18 – Component diagram for VR-Unit architecture. 33
Figure 19 – Activity diagram for the acquisition workflow. 35
Figure 20 – RabbitMQ required queues to use the NetAPI. 38
Figure 21 – NetAPI messages communication flow. 40
Figure 22 – Sensors data communication flow using Bridge application. 41
Figure 23 – Broadcasting commands to Ad hoc network nodes. 42
Figure 24 – Intel Edison Host CPU and MCU communication. Source: goo.gl/oVPPDb. 44
Figure 25 – Intel Edison and Breakout Board. Source: goo.gl/aCFsVE. 45
Figure 26 – VR-Banway architecture. 46
Figure 27 – Message distributions types supported in MQTT [45]. 47
Figure 28 – Intel Edison with battery module. Source: goo.gl/aBfTe4. 48
Figure 29 – BITalino BLE version with ECG sensor attached. 51
Figure 30 – BITalino Led Pulse Sensor. Source: goo.gl/yJCDEX. 52
Figure 31 – Adafruit Flora RGB Smart NeoPixel, size comparison. Source: goo.gl/on9ZVV. 52
Figure 32 – VR-Mission Review interactive timelines. 53
Figure 33 – VR-Mission Review login system. 54
Figure 34 – Size comparison between Intel Edison, RPI and Android. 56
Figure 35 – GPS route of the acquisitions. 58

Figure 36 – ECG signal acquired from Vital Jacket to the Android VR-Unit. 59
Figure 37 – ECG signal acquired from BITalino to the RPI VR-Unit. 59
Figure 38 – ECG signal acquired from BITalino to the Intel Edison based VR-Banway. 60
Figure 39 – Environmental temperature acquired to Android VR-Unit. 60
Figure 40 – Environmental temperature acquired to RPI VR-Unit. 61
Figure 41 – Environmental temperature acquired to Intel Edison VR-Banway. 61
Figure 42 – Ad hoc network know nodes. 62
Figure 43 – “Sync clock” command workflow. 62
Figure 44 – Check team command workflow. 63
Figure 45 – Poster of the VR-Banway system. Presented at students@deti 2017. 71

III. List of tables

Table 1 – Messaging protocols internal architecture comparison [31]. 13
Table 2 – Raspberry Pi 3 most important specifications. 24
Table 3 – Messages structure. 25
Table 4 – NetAPI most relevant commands. 38
Table 5 – Intel Edison most important specifications. 43
Table 6 – Feature comparison between VR-Unit implementations. 57

IV. List of Acronyms

ACC – Accelerometer

AMQP – Advanced Message Queuing Protocol

API – Application Programming Interface

BLE – Bluetooth Low Energy

BUZ – BITalino Buzzer

CoAp – Constrained Application Protocol

DDS – Distributed Data Service

ECG – Electrocardiography

EDA – Electrodermal Activity

EEG – Electroencephalography

EMG – Electromyography

GPIO – General Purpose Input/output

HTTP – Hypertext Transfer Protocol

IoT – Internet of Things

JSON – JavaScript Object Notation

LED – Light Emitting Diode

MAC – Media Access Control

MQTT – Message Queuing Telemetry Transport

OLSR – Optimized Link State Routing Protocol

OS – Operating System

P2P – Peer-to-peer

QoS – Quality of Service

REST – Representational State Transfer

RGB – Red Green Blue

RPI – Raspberry Pi

RTC – Real-Time Clock

SBC – Single-board Computer

STOMP – Simple/Streaming Text Oriented Messaging Protocol

UI – User Interface

WPA – Wi-Fi Protected Access

1 Introduction

Internet of Things (IoT) is a generic category of ICT architectures that includes the use of sensor-based, communication-enabled systems. A common architectural element in IoT is the sensors gateway that collects data from near-by sensors and relays them to higher-order remote services.

The VR2Market project [1], in which this work is integrated, uses two implementations of the gateway, based on Android smartphones and RPI boards. The VR2Market is a project that aims to monitor and support people and teams in dangerous professions, monitoring his vital signals, context and surrounding environment aspects. The project is a collaboration of a consortium involving several partners from technology to psychology.

The initial objective of this dissertation is to refactor VR2Market system to address known issues on transport layer and sensor integration in order to improve it and to prepare it to different areas and scenarios namely IoT scenarios.

Our main focus was on the personal data aggregator. The personal data aggregator (VR-Unit) is the data entry point to VR2Market incoming from team members in the field. The refactoring of an existing solution (Android and RPI) aims at the deployment of a personal aggregator that is more efficient, in terms of power consumption, and smaller and, also, the exploit of new single-board computers like Intel Edison were a step towards smaller and low power consumption solution. With new IoT-inspired computing modules, there was an opportunity to migrate the existing aggregators to smaller, more efficient hardware, while retaining the high-level programming abstraction.

However, some previously identified issues remained as objectives in the refactoring work, namely the lack of Bluetooth Low Energy sensors (for new sensor integration and more efficient power management) and the support of Ad-hoc network resources API which can be exploited also to add bidirectional communication between the aggregator and the rest of the system with QoS assurance.

1.1 Objectives and Contributions

The objectives of the current work are to refactor the existing personal aggregator unit based on RPI used in VR2Market to new requirements and technological evolutions:

- Integrate BLE sensors, namely BITalino.
- Add support for messaged based interface and integrate NetAPI, a new Ad hoc messaging transport layer API to the solution and, with it, add bidirectional communication support.
- Port the existing solution to other Linux based SBC - Intel Edison more precisely.

As a high-level objective, this refactoring will allow deploying a new version of the gateway, named VR-Banway, using the Intel Edison compute module, extending the existing usage scenarios of VR-Unit (the current VR2Market gateway solution) to more wide area of IoT, as well as, the addition of BLE support and port to Intel Edison allow a smaller footprint either in power requirements and overall form factors – when compared with existing RPI solution.

Contributions:

- Refactored VR-Unit: Personal aggregator deployed in RPI.
  - Added support to new BLE sensors.
  - Ad hoc communication service using NetAPI service.
- VR-Banway: Personal aggregator deployed in Intel Edison.

1.2 Dissertation Structure

This dissertation is composed by 9 chapters, including this one, all of them well divided by subjects:

- Chapter 2 describes the State of Art related to the work developed in this dissertation and the IoT.
- Chapter 3 describes the existing system, VR2Market, and alongside with it the changes and adaptations that this dissertation address.
- Chapter 4 describes the refactoring done to the VR-Unit, namely what was added, and the issues addressed as well as the new components and the overall workflow of the system.
- Chapter 5 describes the integration of a new communication service in VR-Unit that use Ad hoc networks. With this integration we explain the new scenarios and features added to the system.
- Chapter 6 describes the port of RPI based VR-Unit to a new single-board computer, Intel Edison, the VR-Banway.
- Chapter 7 describes other major changes that were made in several system components.
- Chapter 8 describes the evaluation of the refactored VR-Unit and new VR-Banway in comparison with the existing RPI based VR-Unit solution also considering the added features.
- Finally, Chapter 9 summarizes the conclusion descendant from all the work done and it is described some future work that can be done to improve the system.

2 State of the Art

The scope to the refactoring on the existing personal aggregator unit based on RPI is clearly within the scope of the broader area of Internet of Things (IoT) as most technical solutions we seek to improve are traditional concerns of IoT: low power consumption, data sharing, efficient data transport, sensor integration, and small footprint devices in which SBC can be included. Alongside this specifications, the constant evolution of the embedded systems (e.g. SBC) and communications technologies [2], [3], provide equipment and software capable of going along with the evolution of the monitor systems (e.g. VR2Market).

In this chapter we will address each of these concerns and review existing solutions that might be applicable in the evolution of the new personal aggregator for VR2Market project.

2.1 Internet of Things overview

The IoT concept starts to appear due to the growing number of machines deployed in every single activity or local plus the constant expansion of the internet coverage [4]. In IoT, deployed gadgets or devices start to connect between themselves using available Internet connection [5]. For this reason, most IoT enabled devices already have connection capabilities to allow remote access (e.g. for configuration) and to relay the data they collect [6]. IoT can be seen as enclosing the physical objects (controller, sensor, actuator or a combination of these), the data processing units (e.g. deployed software) and the network transport layer (with or without Internet), as stated in *Designing the Internet of Things* [7].

The typical devices in the IoT area are not the common desktop computers or servers, but small devices usually coupled to deployed sensors or actuators with design with focus on low power consumption, small size and network connectivity. A good example scenarios are *Smart Cities* [8] where the embedded systems (e.g. the devices used) are spread in several strategic locations to, typically, monitor actions and behaviours and act accordingly, managing and controlling an activity by inter-operating

between themselves without human interaction [9]. Thanks to this characteristics one big range of IoT scenarios relies in monitoring systems, where there are huge amounts of sensors, data processors and actuators [7] and, with that, the related acquired, processed and shared data over the network that needs to be processed and analysed, also, by some high-level applications to transform it in useful information about the scenario different contexts [10], from the information conclusions can be formed and act accordingly to improve life welfare and lifestyle.

2.1.1 Typical architecture

In a typical IoT scenario, the system architecture follows a layered approach where acquisition, processing, sharing and visualization the collected data is addressed – each layer addressing one of these concerns [11] as shown in the Figure 1.

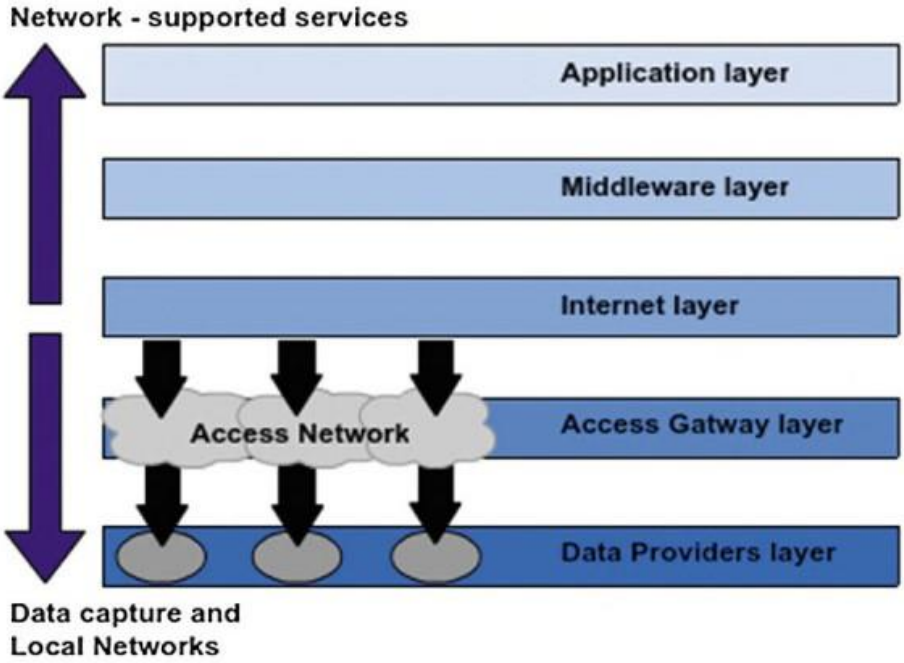


Figure 1 – Layers of a typical Internet of Things architecture [11].

The architecture can, also, be represented using three main components, Hardware, Middleware and Presentation [12] where the Hardware component comprises the Data Providers, Access Gateway and Internet layers, the Middleware component comprises the Middleware layer and the Presentation component comprises the Application layer described in the previously described model.

The Hardware component, and respective layers, represent the physical data sensors, actuators, aggregator devices [13] and communication hardware responsible for collecting the sensed signals, aggregating and sharing them with the upper layers and components. The Middleware component acts as a bridge between the Hardware and Presentation components and it is responsible to manage the connected devices and organize the collected data [14] transforming it into information to provide it to the upper layers and components. The Presentation component represents the tools and applications that can exist in several platforms to access and present the collected information to the end users of the systems [15].

It is possible to approach IoT focused on the sensors (the things) and on the software components as in Figure 2. The sensors on left side (the Hardware) rely on “Sensor Gateway” also called aggregation gateway [16] (in Figure 2 depicted as the mobile device and “REST API” components) to gather the signals captured by the sensors. In this type of scenarios, and consequently the device used to fulfil this component features is called aggregator, responsible for feeding the Middleware and Presentation according to application scenario and specific requirements. This aggregator has an important role [17] in an IoT scenario.

Figure 2 – Components architecture of a typical Internet of Things scenario [18].

2.1.2 Sensors Gateway

In a typical IoT architecture, one particular component that plays an important role in interfacing the edge devices (e.g. sensors) with the Internet, the Cloud or other services available in networks [19] is the sensor gateway component. This component has been named sensors gateway, IoT gateway [20] or even data aggregator and it is deployed in a device that stays between the sensors and actuators and the rest of the system as shown in Figure 3.

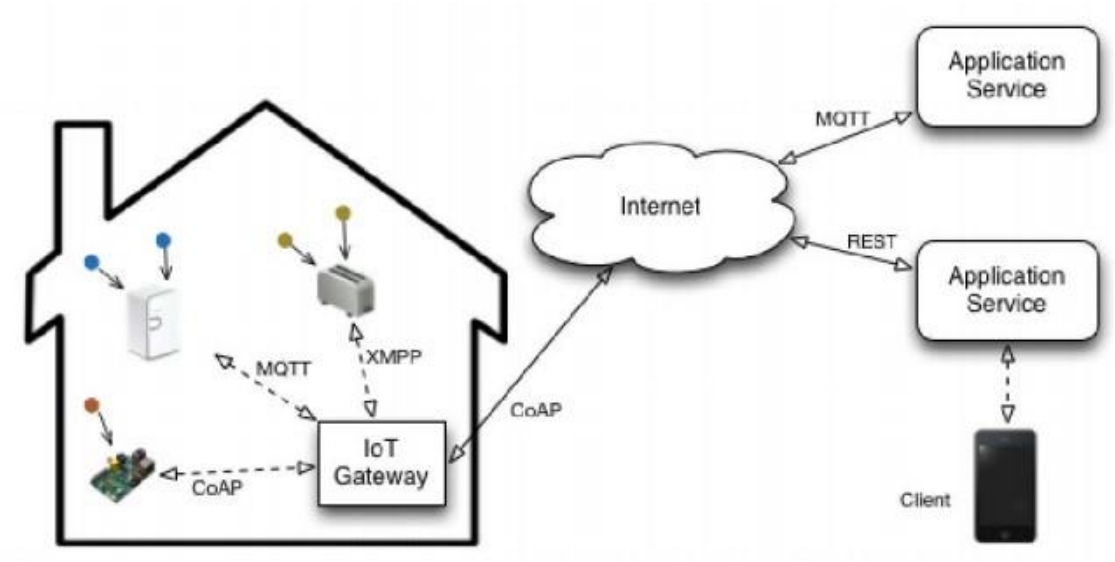


Figure 3 – Example of IoT gateway usage in a smart house monitoring system [20].

In terms of architecture, the sensors gateway makes the bridge and connectivity between the Data Providers and the Internet layers, so, it acts in the Access Gateway layer with the network role of being a central router point to interconnect all the smart devices together (sensors, actuators and even other aggregators) in order to collect and share the scenario adjacent data [21]. Aside from the internal network it also makes the bridge between the intranet and the internet [22] to allow the exterior accessibility to the internal hardware devices produced data and configure the devices behaviour.

The sensors gateways were traditionally devices with low processing capability [23] and its only responsibility was to route the data directly from the sensors to the other components but, nowadays, more and more, the systems are using smart devices that are in charge of the sensors connectivity, collected data processing, filtering, storage and security communication aspects among other tasks [24]. These devices are selected based in hardware and, also, software requirements to perform the stated required tasks and making an interface between communication technologies that act in different levels, as in Figure 4, because the sensors and actuators do not have OS so the communication protocols are at a lower level and the data need to be shared with higher-level devices and systems that use different technologies and protocols.

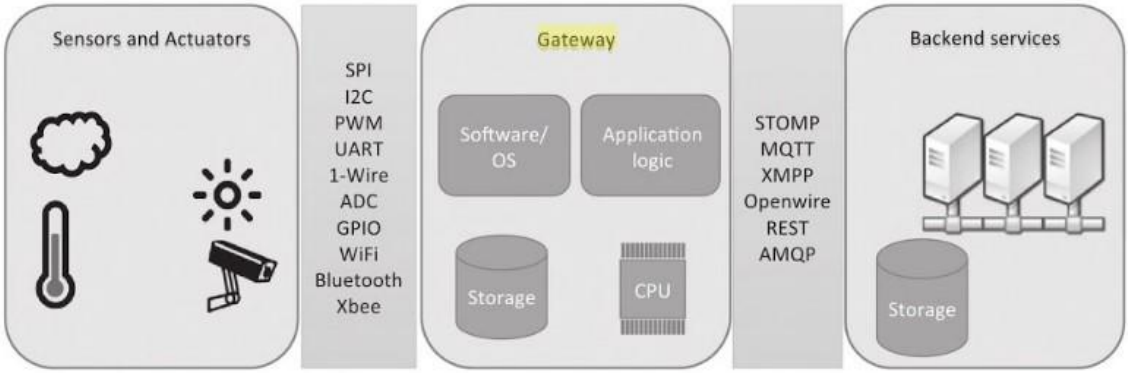


Figure 4 – IoT architectural components and communication protocols [25].

2.1.3 Efficient Messaging

In the IoT scenarios, the collected data, provided by the sensors and aggregated in the Sensors Gateway, needs to be shared and exchanged in order to make it useful (information) to be analyzed and become the source of decisions and acts to improve and correspond to the system objectives. The data communication can be done using a wide range of technologies, where it stands out the messaging protocols that, in general, are chosen for the specific system so that it obey the devices constraints [26] about communication efficiency and at the same time the battery, memory and processor saving of the Sensors Gateway.

The messaging purpose is to exchange messages using a broker to manage them, where the communication actors subscribe and publish in order to receive and send them respectively, as in Figure 5. There are several, most used and relevant, brokers like Kafka, RabbitMQ, Mosquitto and Kestrel [27], [28], [29], where Kafka and Kestrel are very similar, in terms of internal architecture and structure, and RabbitMQ and Mosquitto are in the same technological level both in terms of internal architecture and configuration parameters as of used messaging protocols. These last two, particularly, RabbitMQ have a large set of features [30] which allows to be more flexible and adaptable to the scenario requirements but that could be heavier, in terms of resources consumption, than the others, so, despite of the large range of scenarios where it can be used when there are resources restrictions and specific configuration requirements, other brokers, like Mosquitto, could be more suitable.

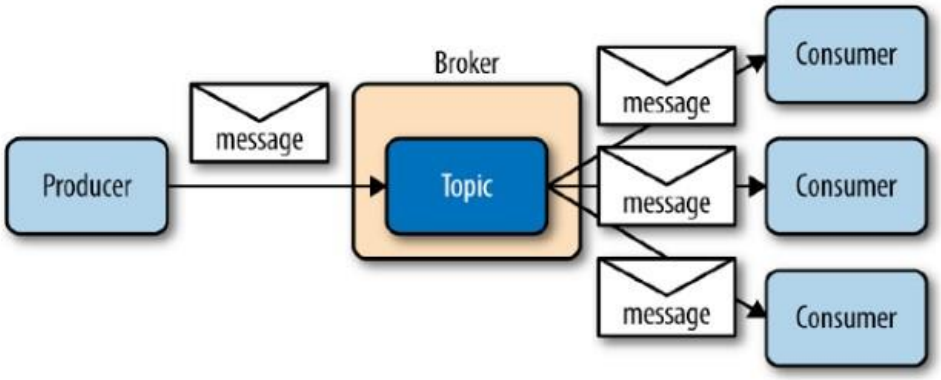


Figure 5 – Broker messaging concept, publish-subscribe pattern [26].

Aside of the brokers itself, all of them need to use communication protocols, that run over TCP or UDP protocols. There are many messaging protocols, designed for different operations and type of scenarios, among them some stand out for their recognition and evidence given, like, AMQP, MQTT, CoAp, STOMP and DDS [31]:

- AMQP: This protocol provides a large range of features in terms of reliable queuing, security, topic-based and flexible routing of the messages. Most of them are, also, supported because this protocol implements its own link and transport layers [31] over the TCP/IP standard ones. It is also programmable by the user in terms of access to the queues, their depth in terms of size, message headers, properties and annotations [32]. The internal management of the messages are through the publish-subscribe pattern using queues to maintain and share the messages. In conclusion is a protocol very comfortable in terms of configuration, reliable message delivery and security, although, it is more expensive in terms of processor and memory usage, so, sometimes it is not suitable for an IoT scenario.
- MQTT: This protocol was designed to be simpler, and consequently lighter, than AMQP [32]. It was implemented over TCP and Web sockets, so, it has also reliability and assure QoS [28] internally and from the TCP protocol in the message delivering. The internal structure, to manage the messages communication, is based in the publish-subscribe pattern using a topic-based filtering [28]. It was developed to be mainly deployed in constrained-resource devices, so, with the stated fact, this protocol is more suitable in IoT scenarios where the network signal can be weak, the hardware is minimum and with low power supply.

- CoAp: This protocol was designed and developed with the awareness of the constraints in the devices and networks in the IoT scenarios and similar [33]. It works over the UDP protocol so there is no reliability and QoS assurance in the message delivery, although, by using the HTTP REST model (only a modified subset of the REST API) it can mark the packages [31] in order to have some notification about the delivery status. As stated, using modified REST commands, this protocol does not follow the publish-subscribe pattern, instead it uses a request-respond pattern.
- STOMP: This protocol is text-based, analogous to HTTP [32], so it is labeled as a protocol that works in an higher-level than the ones stated before and, consequently, not appropriate to be used as a message broker in an IoT scenario [31], even being simpler and lighter than AMQP. Although it is a broker suitable and adaptable for many scenarios due to its simplicity accepting client connections from different technologies and by not using queues and topics and, instead, use semantic commands like HTTP [32].
- DDS: This protocol is different from the others in terms of its base architecture being data oriented instead of message oriented [31], that is, the client nodes post its own data and the other nodes post what type of data they want, in order to consume it [34].

Through the analysis of the described messaging protocols it is clear that for a specific scenario and their requirements the chosen protocol and broker have a significant impact on the behavior and efficiency of the overall system workflow, so there is no perfect protocol or broker, it just need to be chosen based on an evaluation of constraints and requirements. In Table 1, adapted from *EEJournal - All About Messaging Protocols*, by *Bryon Moyer* [31], is a brief comparison between the stated protocols in the general most important features to take into consideration when choosing the protocol to be used.

Protocol	Message pattern	QoS	Security	Suitable for constrained devices
AMQP	P/S	Sophisticated	TLS and SASL	No
MQTT	P/S	Some levels	Manually by best practices	Yes
CoAp	R/R	Not native. Can use HTTP REST features	DTLS	Yes
STOMP	P/S	Server-specific	None	No
DDS	P/S	Sophisticated	No standard	Yes

Table 1 – Messaging protocols internal architecture comparison [31].

2.2 Communication using Ad hoc networks

In the VR2Market project, one relevant feature is to support Ad hoc communications between system nodes to overcome the absence of a traditional centralized network topology. A good example are the scenarios of critical response, disasters or hazardous professions where typical network resources may be absent, namely, due to infrastructures hazards (damage or destruction) [35], compromising the overall system communications.

In the deployment of technological systems where data communication is present, and sometimes crucial, the respective communication topology infrastructure has several hardware components that represent a point of failure if it was damaged or destroyed, compromising all other components and communication channels. For example, if the communication technology is Wi-Fi based, the packets “routing” is made in a central point device [36], making it a central point that makes the connections between all the users and, consequently, a point of total failure.

With the use of a wireless Ad hoc network the system does not need a central infrastructure, it assumes a decentralized topology [37], there is no central failure point that compromises the communication and the distance problem is also resolved by the base architecture of Ad hoc technology regarding to the used routing protocols to properly know the network nodes. In this type of networks there are many protocols like AODV (On-Demand Distance Vector), OLSR (Optimized Link-State Routing), RPL (Routing Protocol for Low power and lossy networks) and SPIN (Sensor Protocol for

Information via Negotiation) [38]. With the use of these protocols it is possible to know the neighbors nodes and calculate and decide the path to take in order to reach the non-neighbors nodes by the maintained routing table [39]. With this routing mechanism, it is assured that the communication between any two nodes is guaranteed as long as there is a path between the nodes that routes the messages between the sender and the receiver.

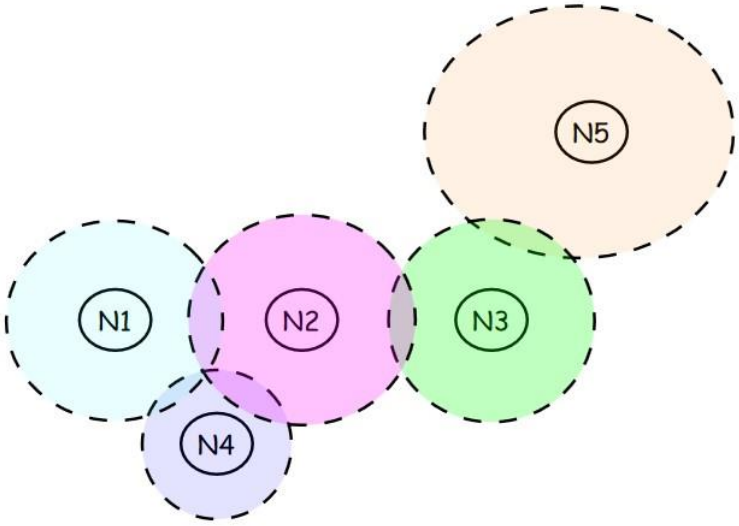


Figure 6 – Wireless Ad hoc nodes range [37].

3 The VR2Market and new opportunities

The VR2Market is a project that aims to monitor and support people and teams in dangerous professions, monitoring their vital signals, context and surrounding environment aspects. The project is a collaboration of a consortium involving several partners from technology to psychology [1].

VR2Market is oriented to serve the firefighters in their missions, whether in fire or in accident and rescue scenarios. By applying this system the firefighters themselves can have a feedback about their body response in the several situations of the missions but also the commander or team chiefs have the perception about the welfare of the rest of the team members.

In VR2Market system [1] the assumption is that by monitoring both environmental and physiological status of the operational in the operation theater, it is possible to have a better management, at personal and team level, when they are at a danger situation and, also, in post operation, to track possible markers of conditions that demand attention and action on the long run (e.g. psychological support, physical deficits, etc.) as result of continuous exposition to hazardous conditions along the time.

3.1 Scenarios and workflows

The original VR2Market scenario is focused on firefight scenarios, where the firefighters have the sensors in their clothes and body. The broader scenario also includes support monitoring operational in several types of hazard jobs and scenarios. In either scenario each operational has an aggregator and several sensors, as in Figure 7, that collects, processes, stores and sends the information to the Cloud and to the running external services if there are any.

Figure 7 – Sensors attached to the firefighter clothes.

In a real fire scenario, VR2Market, through web UI, allows to show, the commander or team supervisor, the status of the firefighters in the field [1]. The information can be visualized in real time because the aggregator sends the data to the external service, as long as it has connection to the same network as the service. By doing this the supervisor or commander has a real-time vision of the firefighters location, their heart rate and body temperature and the environmental status present near the firefighter team. The collected information can also be visualized afterwards to analyze the ambient and fire evolution and the behavior of the firefighters in response to those changes.

All the data is collected from the sensors equipped in the firefighter clothes as shown in Figure 7, where it is possible to see the several used gear, e.g., Vital Jacket shirt (1), GPS sensor (2), FREMU sensor (3), VR-Unit Android phone (4), HELMET sensor (7), it is used also two localization sensors, already used by the firefighters that are not included in the project but the data can be used to validate the collected values from the project sensors (5, 6).

Besides just visualizing the physiological and environmental information, it is possible to receive alerts and alarms from the aggregator equipped in the subject. This type of warnings are showed as a more relevant event that needs more attention and proper response.

This monitoring scenario is already implemented in the VR2Market project where our system acts as the aggregator or Sensor Gateway. There is a real-time application to visualize the information in live mode and another application to analyze the collected information afterwards.

3.2 VR2Market Architecture

In the VR2Market architecture we clearly distinguish two scopes (Figure 8): data collection & monitoring and data processing & review. On the left side, the data collection in operational theater relies on VR-Unit (personal data aggregator) to gather information from the equipped sensors. VR-Unit relies the incoming data to the Data Collector component (mission level aggregator) that gathers all information incoming from a specific operation theater. It serves as bridge to the data handling and monitoring backoffice, present in the right side, that besides supporting online monitoring (VR-Commander) also allows offline mission review (VR-Mission Review) and provide the data to analytics and third-party applications.

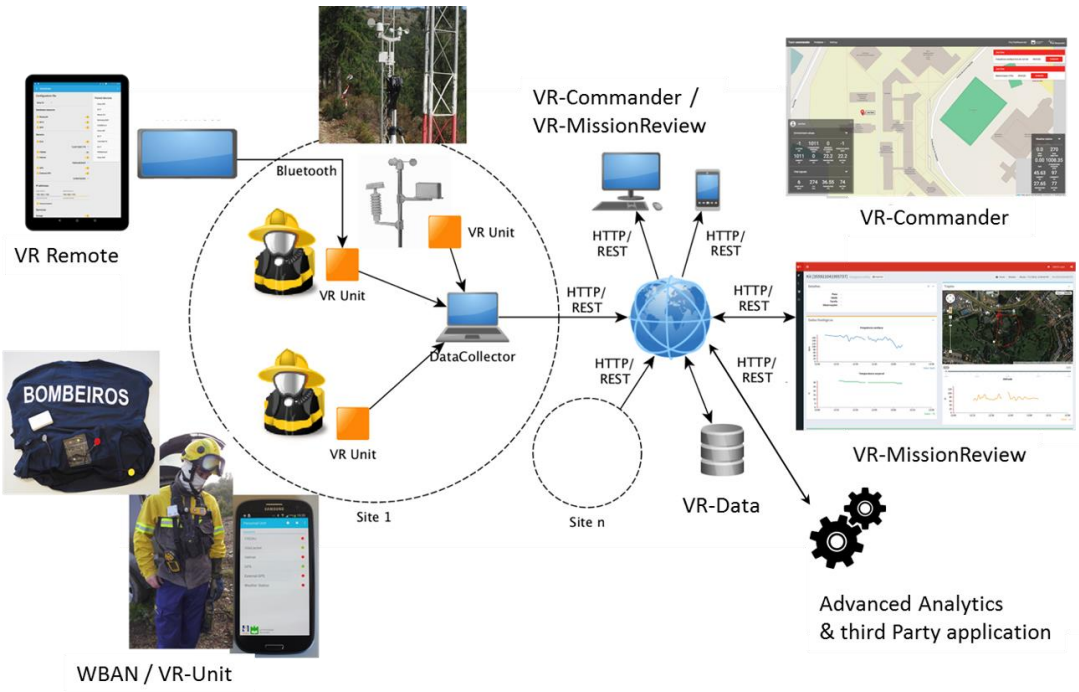


Figure 8 – High level architecture of VR2Market.

Briefly, the main components present in the VR2Market system are:

- VR-Unit responsible for acquiring the data from the sensors.

- VR-Remote responsible for the configuration of the VR-Units.
- Data Collector responsible for the aggregation of the data from all the VR-Units.
- VR-Data responsible for storing the collected data and providing access to it.
- VR-Mission Review responsible for visualizing and analyzing the data after the acquisition.
- VR-Commander responsible for visualizing the data in real time during the acquisition.

The VR-Unit is the aggregator, present in each firefighter, that gathers and process all the data collected from the sensors attached to the firefighter equipment and it can be external configured by the VR-Remote component. It is an end point of the system because it is responsible for establishing the connection with the data sensors and acquire its data to aggregate, filter, store and send it to the rest of the system. Also, the VR-Mission Review and the VR-Commander are endpoints because their task is to provide a UI of the collected information to the end users of the system, the first one is for review the collected information after the acquisition while the second one is focused on online monitoring in real-time.

The Data Collector and the VR-Data are responsible for providing access to the collected data. The first one aggregates all of it and provides it to the second one that makes it persistent, saving it in a database, and exposing it to other applications or components through a REST API.

3.2.1 Sensors

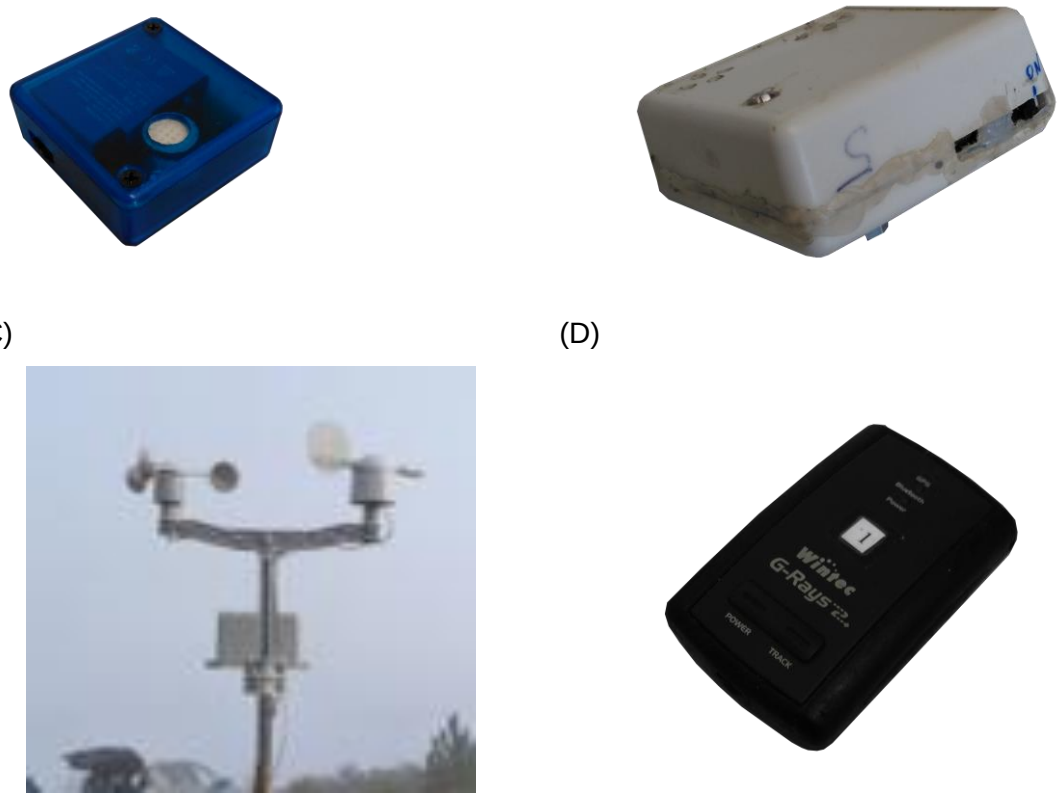
The VR2Market system collects data of diverse types (environment, location and physiological data). The values are provided by external sensors via Bluetooth:

- **Environmental data**
  - FREMU sensor.
  - HELMET sensor.
  - Weather Station sensor.
- **Location data**

    - GPS sensor.
    - G2RAYS sensor.
- **Physiological data**
    - Vital Jacket sensor.

The environmental data collected by the sensors is very important in this system because it provides information about the surrounding environmental aspects of the subject with the sensors. In this type of scenarios, it is important to be aware of these factors in order to associate the environmental data and the corresponding reactions of the subject and, with that, also, preserve the welfare of the subject by warning him about any surrounding danger. The FREMU device (Figure 9 A) is capable of sensing and measure ambient temperature, carbon monoxide, atmospheric pressure and relative altitude. The HELMET device (Figure 9 B), is like a newer version of the FREMU due to the acquisition signals being the same plus the humidity, luminosity, nitrogen dioxide and sensor battery. These two sensors are made to be placed directly on the professional subjects, but, there is also a fixed sensor, the Weather Station (Figure 9 C), capable of sensing and measure ambient temperature, rain, wind direction, wind speed, atmospheric pressure, humidity, luminosity and sensor battery.

(A) (B)



(C) (D)

Figure 9 – Sensors examples:(A) FREMU, (B) HELMET, (C) Weather station, (D) G2RAYS.

The location data is collected from the internal aggregator GPS and the G2RAYS (Figure 9 D), these devices are capable of sensing and measuring location, time and altitude. This type of data is used to relate and analyze the correlation between the location adversity versus the subject reactions at every moment.

The physiological data is a crucial element in the system due to is direct relation with the welfare of the subjects, obtaining physiological signals there could be created a relation between all other collected data and take conclusions about the firefighters responses to the several present stimuli in a mission. The Vital Jacket, shown in Figure 10, is capable of sensing and measure electrocardiography (ECG), body temperature and accelerometer (ACC) variations. Through the ECG it is also possible to calculate and determine the heart rate at each instant.

Figure 10 – Vital Jacket sensor and corresponding cloth.

The current VR-Unit only support sensor communications via normal Bluetooth and the stated sensors use only this technology. In the application developed in this dissertation there is also support for Bluetooth Low Energy (BLE) communications and consequently there some new sensors that use this technology that can be integrated in the system.

3.2.2 VRUnit: Android and RPI

In VR2Market the VR-Unit aggregator, based on two implementations, Android and RPI platforms, is responsible for acquiring and processing the collected data by the sensors before transmitting it to the operation theater data collector (Data Collector). The main responsibilities of the aggregator are:

- Find and make a connection with the sensors to receive their data.
- Store locally the processed data.
- Stream the processed data to the Data Collector component.
- Send the processed data to remote location.
- If possible, show feedback of the actual received data.

The VR-Unit architecture relies on a messaging paradigm having a broker as the main gateway to broadcast the collected data to internal components that process, filter, store and share the data. In the Android implementation, this approach is assured by using Broadcast Receivers, intents making it possible to share the collected data between the different activities and services. The RPI implementation is similar, but the broker mechanism relies on a RabbitMQ messaging server sharing the collected data to the different running threads or processes, equivalent to the Android services.

The aggregator can be configured by an external application, as in Figure 11, the VR-Remote, or in the aggregator itself using its graphical interface. The application is able to listen for Bluetooth connections from external devices that configure the active sensors, network addresses, acquisition metadata and used hardware resources.

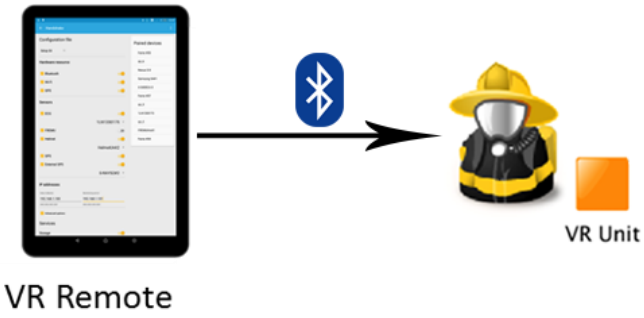


Figure 11 – VR-Unit external configuration.

The Android application (Figure 12) supports all the features required for the system and has the advantage of having a graphic user interface in the aggregator itself which allows the user, in certain scenarios, to configure and visualize the collected data directly on the aggregator. This version was already tested in the field by the professional firefighters in real scenarios and it has proved to be a stable version for acquiring data that can be analyzed afterwards.

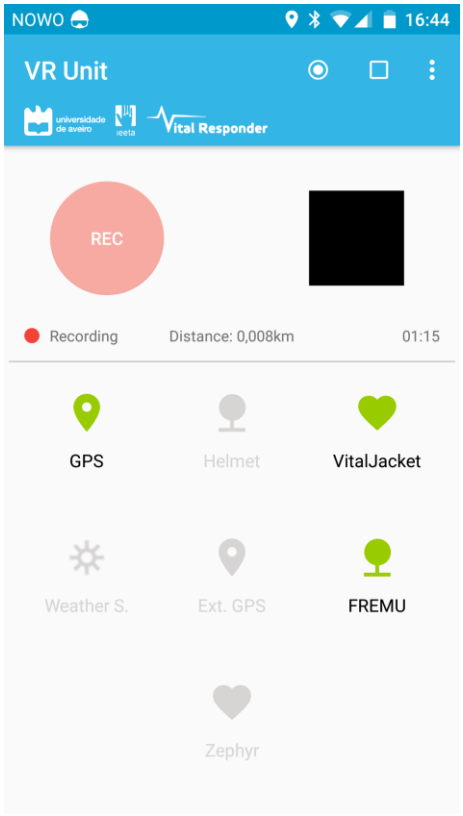


Figure 12 – Android VR-Unit application.

The application developed to run in the RPI also supports most of the system required features but misses the capability of sending the aggregated data to the cloud and the internal storage of it for post-viewing by the rest of the system components. Besides, this version was only developed and used in laboratory experiences, it was not tested in real scenarios but it also processes and produces good quality data, so, it is a solution that can also be refactored to be fully integrated in the system.

These two implementations, despite being applicable in the actual system, do not fit in the objective of this dissertation due to their size and power consumption. Even the RPI implementation, being already a SBC, is still too big and consumes a considerable power from the battery supply.

3.2.2.1 RPI

The Raspberry Pi (RPI) device is a small single board computer i.e. all computer components within one small size board. Besides typical computer it provides some onboard integrated resources: Bluetooth, Wi-Fi modules and provides configurable GPIO ports that allows developers to connect and control several electrical components.

There are several available models, beginning in the model B to the RPI 3 (Figure 13) that was released in 2016, this last one was chosen because in our system we need some memory to store the data and a good processor that can accompany the communication rates of the sensors and, at the same time, do the rest of the data processing. By having on-board Wi-Fi and Bluetooth adapters, which are necessary in our system, this model is the most suitable to our requirements. In the following table, Table 2, we can check the RPI 3 specifications that proves to be more suitable for our system.

Component		**Specification**
Processor	**Core**	Cortex-A53 64-bit Quad Core
	Clock	1.2 GHz
GPU		VideoCore IV
RAM		1 GB
SD / MMC		microSD
GPIO		40 ports
Wi-Fi		802.11n
Bluetooth		4.1
Power consumption		800 mA
Power source		microUSB or GPIO

Table 2 – Raspberry Pi 3 most important specifications.

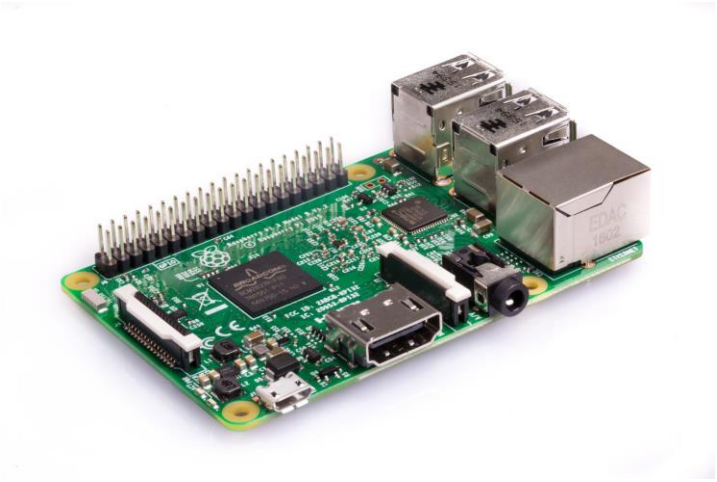


Figure 13 – Raspberry Pi 3. Source: goo.gl/n4Jbzf.

3.2.3 Data transport and persistence

Data Collector is the VR2Market logical broker on the operational theater and acts as the gateway for the VR2Market backoffice. This component is the central point for data collection incoming from VR-Units. As in VR-Unit, the communication between VR-Unit and Data Collector relies on a messaged based paradigm where the Data Collector receives the information and routes it VR-Data that is responsible for its storage and provide the information to online and offline components monitoring and analytics.

Currently, communication between VR-Unit and Data Collector has two modes: P2P and messaging. P2P are supported on TCP or UDP sockets [40], messaging uses the AMQP protocol using RabbitMQ as message broker. Both modes are supported in both Android and RPI VR-Units and relay on a message based API to send and receive text based messages. The text messages with the sensors collected values follow a simple structure as showed in Table 3.

Timestamp	**Aggregator**	**Headquarter**	**Team**	**Message Hash**	**Data**
248, timestamp	109, add. ID	150, headq. ID	151, team nº	149, hash	…

Table 3 – Messages structure.

After receiving the message, the Data Collector filters and validates the messages routing them to VR-Data through REST API. In the VR-Data component, the received messages from the Data Collector are parsed, to get the sensors values, aggregator and acquisition information, and stored in a PostgreSQL database using the programming language Java and its Java Persistence API. After the data is persisted it becomes available to all components and services that require it through the provided REST API, so, this component, in terms of information, is the core component of the system where the validated, useful and organized information is stored and provided to other system components and third-party applications.

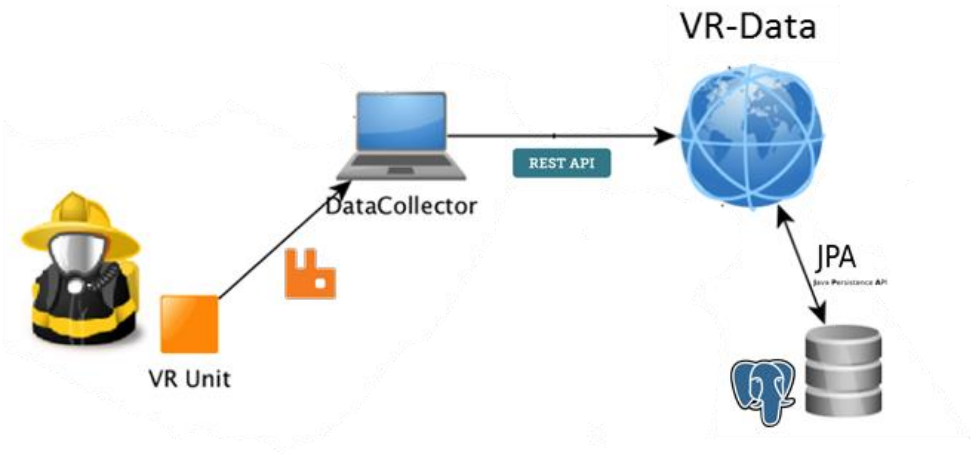


Figure 14 – Sensors data flow from the VR-Unit to the Database.

3.2.4 Front-End

The current front-end components present in the VR2Market system are based in Java EE framework [41] applications deployed in a WildFly [42] server. The three front end components are the VR-Data, which has also a backend processing role, the VR-Mission Review and the VR-Commander.

The VR-Data front end (Figure 15) interface allows the end user to manage the acquisitions loaded and stored in the system, import, visualize, delete and export acquisitions to be used by other applications or systems. Besides the interface, this

component is also responsible for supporting a REST API which provides access to the stored information.

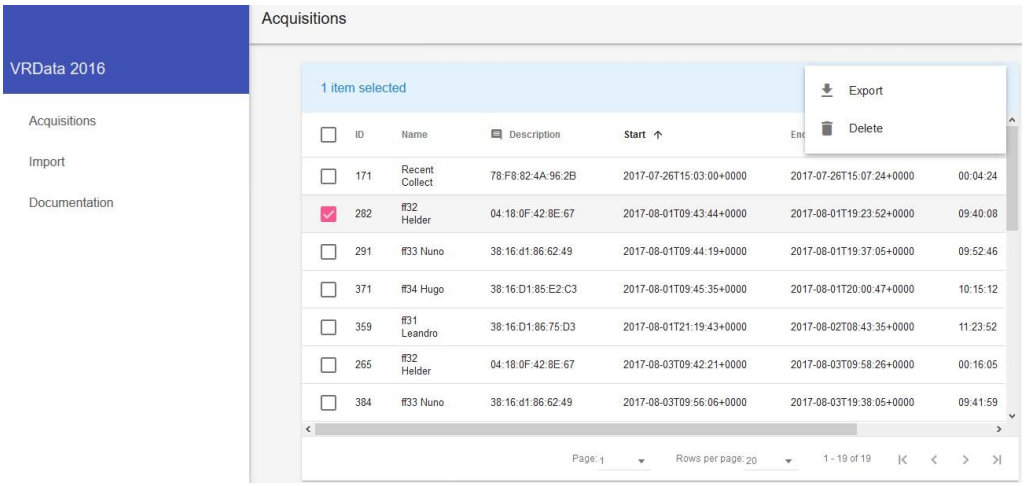


Figure 15 – Project component to manage acquisitions (VR-Data).

The main UI components are the VR-Mission Review and the VR-Commander, despite some specific requirements, both rely on JavaScript programming language to present the acquisitions information provided by the VR-Data component. The VR-Mission Review (Figure 16) presents the information based on graphics and a map to trace the firefighter route. This visualization is static because all the information presented is from a past, ended acquisition, but the user can interact with the interface in order to navigate through the timeline of the mission and visualize the acquired values at a specific moment with more detail.

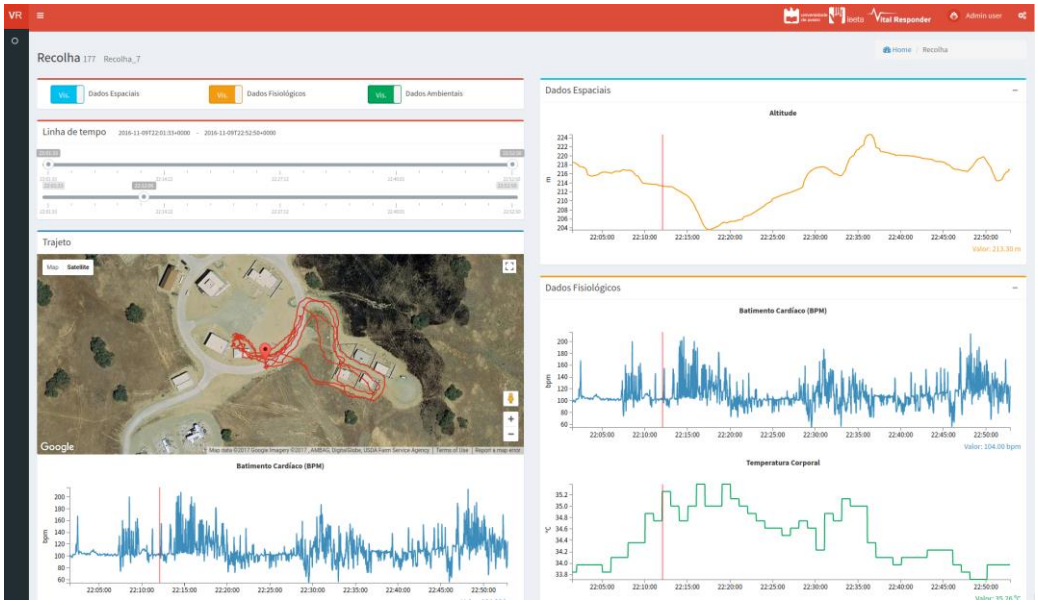


Figure 16 – Project component to visualize past collected data (VR-Mission Review).

The VR-Commander (Figure 17) is the online monitoring solution to visualize the acquired information in real-time. Despite being a real-time interaction, the information flow is unidirectional, because this component only receives the incoming data streaming. The user can check, in an integrated Google Maps, the location of the firefighters and, for each one, visualize the sensors data being collected.

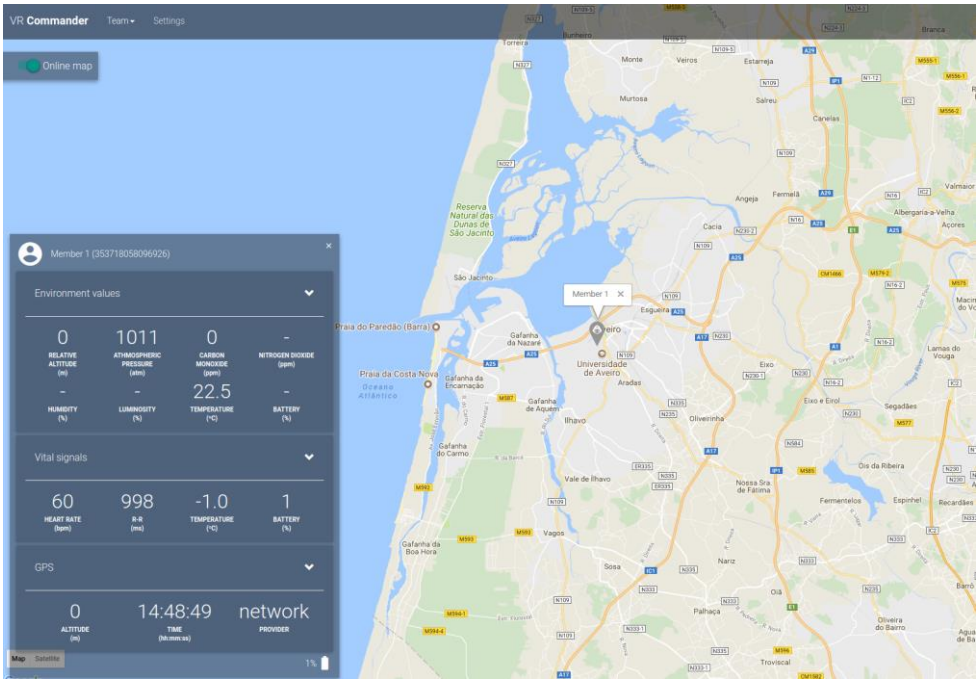


Figure 17 – Project component to visualize real-time collected data (VR-Commander).

3.3 New adaptations and opportunities

In the current implementation, there are some aspects that can be improved and some that need to be introduced for a better performance monitoring and adaptation to other types of scenarios. We focus our work in the reengineering and improvement of the VR-Unit, more specifically the RPI implementation, due to its orientation for IoT scenarios. We also adapt it to work in another SBC, Intel Edison, due to its properties of being as capable of processing as RPI but with some more features and small size with lower power consumption.

By preparing the actual system to be used in other IoT scenarios, there are some features that clearly need to be enhanced in the system and other aspects that we saw as an opportunity to be valuable to integrate in a system like this:

- Integration of software to work with BLE communications and, with that, integration of new sensors like BITalino.
- Allow the aggregator configuration from external devices through Bluetooth. This feature is fulfilled for the Android system but not properly adjusted in RPI.
- Ensure the collected data export to the Cloud and running external systems in the RPI implementation.

Besides the adaptation of the RPI implementation, the new system opens the exploit of other solutions oriented to IoT scenarios and, for that, the need of new objectives and features to better suit in a scenario of that type, namely, small devices with lower power consumption, without compromising the system performance and features. With this innovative approach we decided to adapt the RPI implementation to run on Intel Edison, that satisfies the new requirements.

In relation to the communication of the collected data from the aggregator to other system components or even to other aggregators, the NetAPI service, supported by Ad hoc networks, is an opportunity to have a network without any central point of failure or messaging distributor. Integrating this service, the system becomes independent of the Wi-Fi networks restrictions, and we, also, used this change to introduce bidirectional communication between the aggregator and the rest of the system components. With the stated communication flow, the system can now interrogate the aggregators in order to

check for the users welfare, trigger alarms and warnings and even maintain awareness about the connection status.

With the stated specifications we are able to adapt the existing components, namely the VR-Unit, to explore new IoT scenarios and at the same time continue to use the new system in the existing platforms with a better performance and enhancement.

4 VR2Market System refactoring

The Android VR-Unit already satisfies several of the requirements as aggregator of the project, namely, by support several sensors, providing internal storage and sharing the collected data to the Cloud. It also provides a graphic user interface that allows the user to configure it directly in the device. However, with Android evolutions, the native support of Ad hoc networks was lost, with Android OS changes, became unrealistic due to logistics and even changes to the APIs. The current RPI solution, is Linux based and while losing the UI available in Android devices, has the added value of being more easily extended to use Ad hoc networks and new sensors, namely those using BLE. The objective of the current work is to refactor the existing personal aggregator unit based on RPI used in VR2Market to new requirements and technological evolutions:

- Integrate BLE sensors, namely, BITalino.
- Support data sharing with the Cloud.
- Add support for messaged based interface and Integrate NetAPI service, a new Ad hoc messaging transport layer API to the solution.
- Add bidirectional communication support.
- Port the existing solution to other Linux based SBC - Intel Edison more precisely.

Exploiting SBC's like Intel Edison seem is an interesting opportunity to both reduce the size and power consumption of the VR-Unit with similar computational power. This could be an opportunity to explore other scenarios either by exploring new SBC features or by exploring their smaller size.

4.1 What we are adding and changing

The refactoring of the RPI implementation allowed identifying some extra features already deployed in Android version and port them to RPI version besides the initial objectives of this work:

- Refactor the organization and structure of the data files in the local storage so that the files can be imported to the system backend.
- The refactoring led to some options in technological changes like replacing the aggregator broker, RabbitMQ, to use a lighter one based on MQTT (Mosquitto) already proven in IoT scenarios.

4.2 RPI VR-Unit components

The RPI VR-Unit architecture is composed by the following components, as shown in Figure 18:

- The manager component.
- The storage component.
- The configuration component.
- The sensors drivers component.
- The external services component.
- The NetAPI management component.

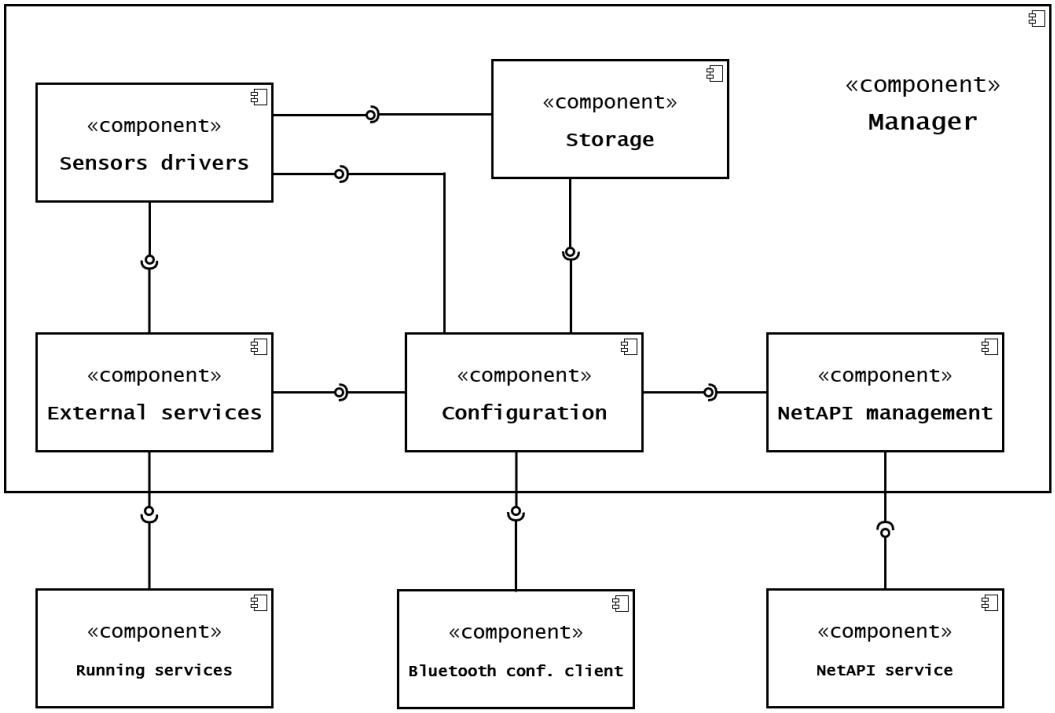


Figure 18 – Component diagram for VR-Unit architecture.

The manager component is responsible for controlling the execution of the aggregator application namely start and stop the overall acquisition workflow.

The storage component is responsible managing the internal data storage including files organization and structure and handling the data flow either those from sensors or those when exporting to the Cloud.

The configuration component is responsible for the advertising of the aggregator service via Bluetooth and receive external configuration from the user, adapting the acquisition workflow correspondingly. It also supports monitoring the VR-Unit components for configuration, execution and status review.

The sensors drivers component are responsible for managing the communication and the data flows from the sensors and relaying it to the storage component and other components requiring it.

The external services component is responsible for managing the communication and sensors data relay to external running services that require the collected data. These services include, namely, the rest of the VR2Market system backend.

The NetAPI component is responsible for handling the interaction with the NetAPI Ad hoc network resources service. This includes, besides managing the messaging from

and to the VR-Unit, register and configure the aggregator running application in the NetAPI services in order to participate as a node of the Ad hoc network.

4.3 **Acquisition workflow**

When the aggregator is powered on it starts the execution of the application using the default configuration, detailed in the internal storage configuration file. The application starts by initiating the manager component that subscribes the aggregator status queue of the internal broker and decide, based on aggregator status parameter, the start, stop or restart the acquisition. This component is also responsible for initiating the execution of two i.e. the storage component and the configuration component.

The storage component, when initiated, creates the data and status files in the internal storage. By managing that data, it stores it in the internal storage and in the end of the acquisition it creates a .zip file with all the created files and, if there is internet connection, send it to the Cloud (Dropbox).

The configuration component, when initiated, read the default configuration file, already in the aggregator, and initiates the demanded components listed in the configuration file. After that it makes the advertise of the aggregator service via Bluetooth so the clients can initiate a connection with it in order to configure the collection parameters of the aggregator by sending other configuration file. If a configuration file is received, it reads it and initiates the demanded components. If the client wants to use an external service, a connection to the network of that service is required, so the aggregator need to receive another file from the client via Bluetooth that have the configuration to be used by the WPA supplicant service and connect to the specified network.

When initiated, the sensor driver component, correspondent to the specified sensor, begins to search for the Bluetooth service advertised by the sensor to stablish a connection with it in order to read its collected data, validate it and then write it in the internal broker queue to be used by the rest of the components.

When initiated, the external services component, subscribe the internal broker respective queues in order to access the collected data from the sensors. Then it establishes the demanded connections with the running external services, it can be through RabbitMQ, Mosquitto MQTT and TCP or UDP, in order to share with them the collected data.

When initiated, the NetAPI component, subscribes and create the necessary queues in the RabbitMQ broker server to use the NetAPI functionalities, registers the aggregator application in the NetAPI service to be recognized as a new node in the Ad hoc network and then it waits for interactions with other nodes and answer it with the appropriate behavior.

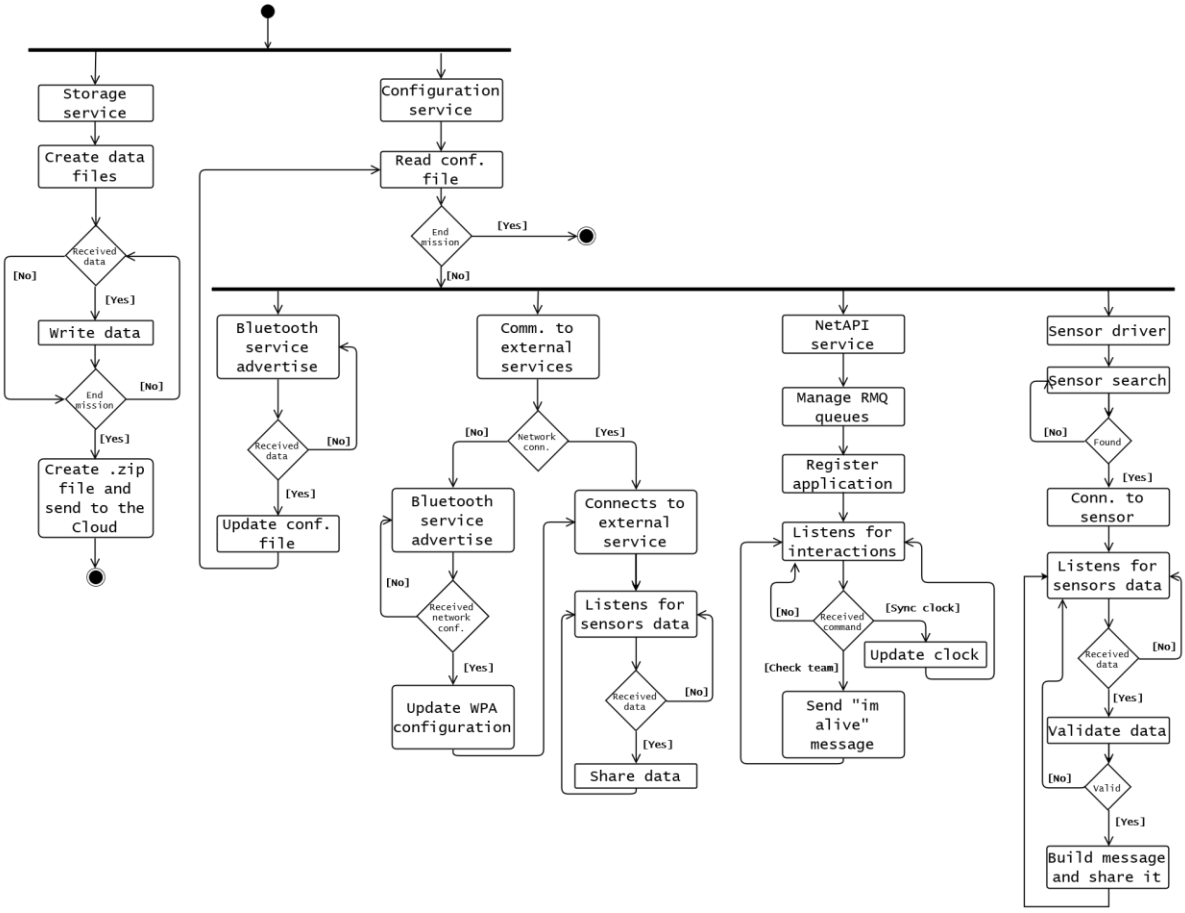


Figure 19 – Activity diagram for the acquisition workflow.

5 NetAPI service integration

The current RPI VR-Unit already supports Wi-Fi based connection, this allows the VR-Unit to use the typical network based on a central device, usually a router, so it has a centralized topology. The initial RPI version had, also, an Ad hoc interface that relied on a low-level programming level wrapping of socket based communication, over UDP, delegating to the programmers all the application level semantics. Such specifications are not available in the Android platforms because it does not have official support due to administration restrictions, in order to connect and use an Ad hoc network in a mobile platform, the user needs to have root permissions and that is forbidden in the commercialized mobiles [43].

Recently a new API based on a message based paradigm was created to access the stated Ad hoc interface in the RPI and participate in the network, including basic QoS, the NetAPI.

5.1 NetAPI workflow

NetAPI is an API that provides application access to Ad hoc network resources. In this service, applications can be registered to be known by the other nodes in the network, as applications that can be contacted to exchange messages of any kind as long as it follows a defined structure. NetAPI uses a broker mediated API based on RabbitMQ, i.e., main interaction with the Ad hoc network is made using messaging, from the application to the NetAPI service, and received, from the NetAPI service to the application, through this broker. To ensure the communication, there are some steps that the application needs to do:

- The application must create and subscribe two queues in the RabbitMQ server, the "id_data" queue, where "id" is a unique identifier of the application, that is used to receive data messages and the "id_control" that is used to receive control messages. To send messages the application must write the message in one of two queues present in the RabbitMQ server, the "network_data" queue that is used to send data messages and the "network_control" that is

used to send control messages. This process is shown in Figure 20**Error! Reference source not found.**.

- The application must register in the NetAPI service sending a proper control command ("register_application"). Besides this command, there are others as listed in Table 4.

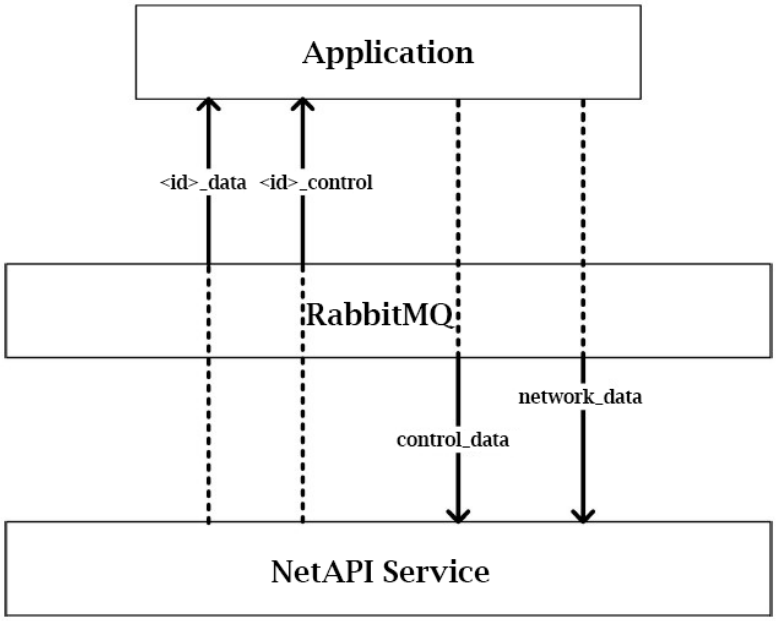


Figure 20 – RabbitMQ required queues to use the NetAPI.

Commands	Function
network_status	Discover if network is created or not.
known_nodes	Discover network registered nodes.
network	Turn network on or off.
register_application	Register application to use the NetAPI service.

Table 4 – NetAPI most relevant commands.

The communication between the application and the NetAPI service or other applications in other nodes of the network, as in Figure 21, it is done by exchanging messages written in the RabbitMQ service queues, those messages need to follow a certain structure in order to be interpreted by the API. When it is intended to send a control message, which is a message with a NetAPI command associated, the application must serialize a JSON message with the following keys and values:

- src_app_id: [integer] unique identifier of the application.
- what: [string] command to execute.
- args: [string] command arguments if needed.
- id: [string] message identifier that would be present in the response of the command.

The response message of the executed command has the following format:

- what: [string] executed command.
- args: [string] argument used in the request.
- result: command result, the type depends on the command.
- id: [string] message identifier used in the request.

When it is intended to send a data message to another node the application must serialize a JSON message with the following keys and values:

- src_app_id: [integer] unique identifier of the source application.
- destination_name: [string] name or IP address of the destination node.
- dest_app_id: [integer] unique identifier of the destination application.
- qos: [integer] quality communication.
- data: [bytes] data to be send.
- mesg_id: [string] message identifier that would be present in the send process message if quality communication is set.

When a node receives a message it has the following format:

- src_address: [string] IP address of the source node.
- src_app_id: [string] unique identifier of the source application.
- data: [bytes] received data.
- qos: [integer] quality communication.

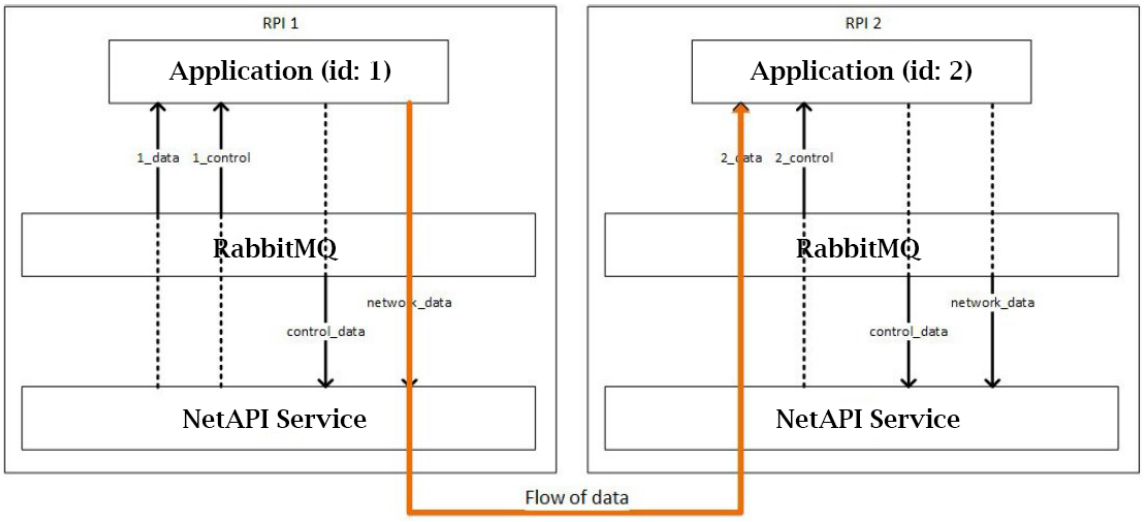


Figure 21 – NetAPI messages communication flow.

5.2 **Adding bidirectional communication**

In the original system, the data communication has only one direction, i.e., the aggregator sends the collected data to the external services (e.g. Data Collector). With the integration of the NetAPI service we also exploit and add the capability, for the system, to communicate with the aggregator units becoming a bidirectional communication.

To explore the bidirectional capabilities of NetAPI, a new application had to be added to the VR2Market architecture on the server side. This application acted as a gateway for relaying server-side messages to the VR-Unit (this flow was not required in initial scenarios). The developed application was deployed in the side of the data processing system that was deployed in the Data Collector component. It is a python application to serve as a bridge between the Data Collector existing Java application and the integrated NetAPI service, as in Figure 22. This application registers itself with the NetAPI service to participate in the Ad hoc network acting as a collector node to be contacted by the other nodes to receive its collected data and to broadcast commands to the other nodes. By adding this bridge application, the bidirectional communication is possible, so, we add the possibility of integrating new features to the existing system and to expand the range of scenarios where the system can be deployed and used.

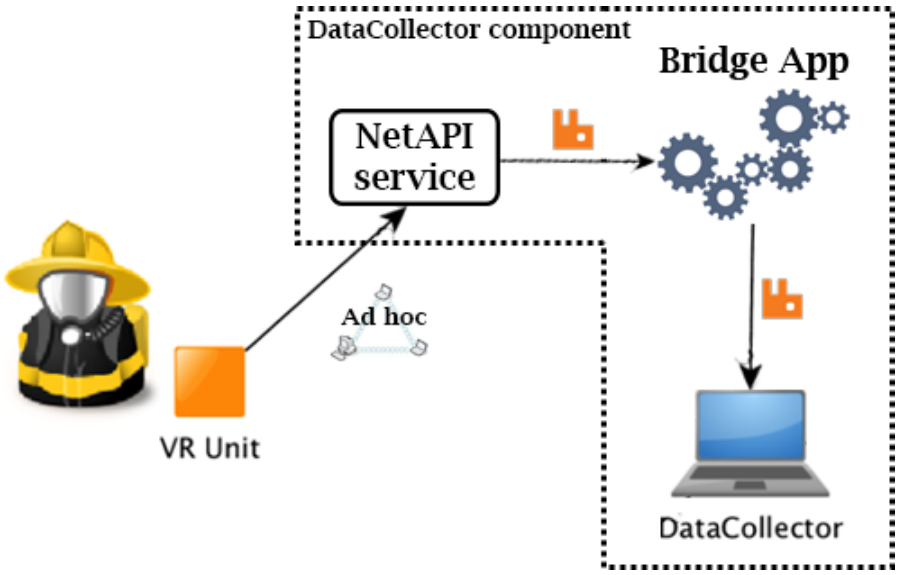


Figure 22 – Sensors data communication flow using Bridge application.

5.3 The new scenarios and functionalities

With the introduce of bidirectional communication between the aggregators and the rest of the system we had some new functionalities that involve the overall data management and visualization system components and the aggregators. These entities can now exchange messages in a request and reply scenario, where the collector node (processing system side) sends commands to the aggregator nodes (data collection system side), as exemplified in Figure 23.

In a fire fight scenario, it is important to maintain the knowledge of the welfare of the fireman, for that it was implemented a method to check the team status that works in a basic way of sending a message and get a response from the nodes in the network (e.g. fireman aggregator). This method request, by the collector node, to the NetAPI a list of the known nodes and send them a data message with a specific tag to be recognized in the nodes as a check team message. When the nodes receive the message, they send a reply message so the processing system application knows that the node is reachable and active.

Another system feature is related to the synchronization of the aggregators internal clocks. RPIs do not have an RTC so its internal clock is only correct if it has connection to the internet to fetch the actual date, otherwise, when it is switched off the value of the date is lost and the next time it is switched on it will use the default date. Due to this restriction, it is necessary to send the processing system actual date to all nodes in the

Ad hoc network in order for the timestamps associated with the sensors data collected to be synchronized. The processing system needs to do the following steps to send its date to the nodes registered in the NetAPI:

- Read the system date.
- Construct a data message with the read date.
- Request to the API a list of the known nodes.
- Send the constructed message to all the nodes in the requested list.

When the nodes receive the message, their internal clocks are updated with the received date and with that the reachable nodes maintain their clocks synchronized, useful for the data analysis after the mission is done.

These new functionalities are supported by a QoS feature integrated in the NetAPI service which allows to be aware of the messages delivery status. By using this feature we have the capability to know if the messages exchanged between the network nodes are delivered or not, and with that take conclusions about the welfare of the aggregator subject (e.g. is reachable).

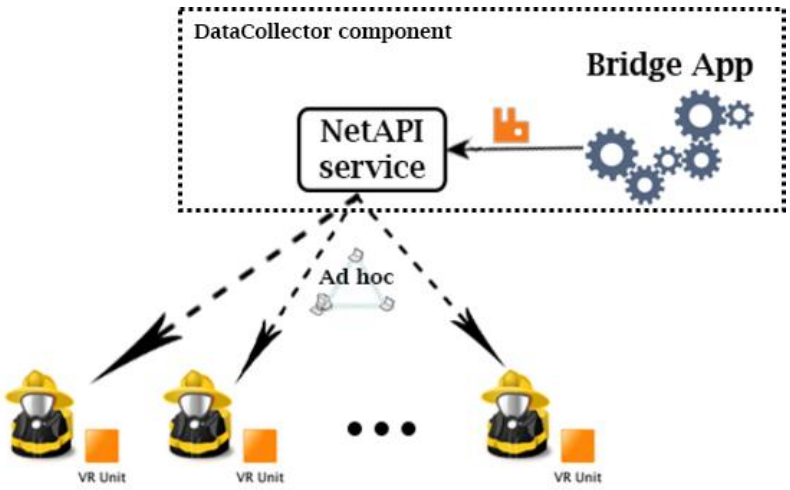


Figure 23 – Broadcasting commands to Ad hoc network nodes.

6 VR-Banway: Refactoring for Intel Edison

One of the objectives of this dissertation was to make a refactoring to the existing aggregator device, that was already developed and used in real scenarios, with the aim of reducing the overall size and power consumption without greatly impairing the processing capability of the application and extend the range of scenarios, where the system can be deployed, approaching the system to the IoT area using other more recent SBC that suits this kind of scenarios and requirements.

The current system has a unit developed to run in the RPI, so, by reusing the existing implementation and adapt it for the new requirements the actual system continues to use the RPI as an aggregator and we also introduce the use of the Intel Edison board.

6.1 The Intel Edison

Intel Edison [44] is a very small SBC that supports several OS and already have integrated modules like flash memory, Wi-Fi and Bluetooth 4.0 adapters besides the dual-core processor, main memory and a separated microcontroller processor. These characteristics accompanied by the reduced size and very low power consumption make it very useful for IoT projects and scenarios.

Component	Specification
Processor	Intel Atom CPU 32-bit 500 MHz Dual Core
Microcontroller Unit	Intel Quark 100 MHz
RAM	1 GB DDR3
Flash Storage	4 GB
GPIO	40 ports
Wi-Fi	802.11 a/b/g/n
Bluetooth	4.0 LE

Table 5 – Intel Edison most important specifications.

The separated Microcontroller Unit, that processes alongside with the main processor, as shown in Figure 24, contain real-time peripheral control features for GPIO ports and serial bus interfaces (UART, SPI, I^2C, I^2S, IPC) and have system calls to control interrupts, manage memory and thread scheduling. With these capabilities we have, available at the same execution time and in the same board, a Microprocessor with OS and a Microcontroller where these two can process different code in parallel and the threads can communicate with each other between the two types of processors through the IPC serial bus interface.

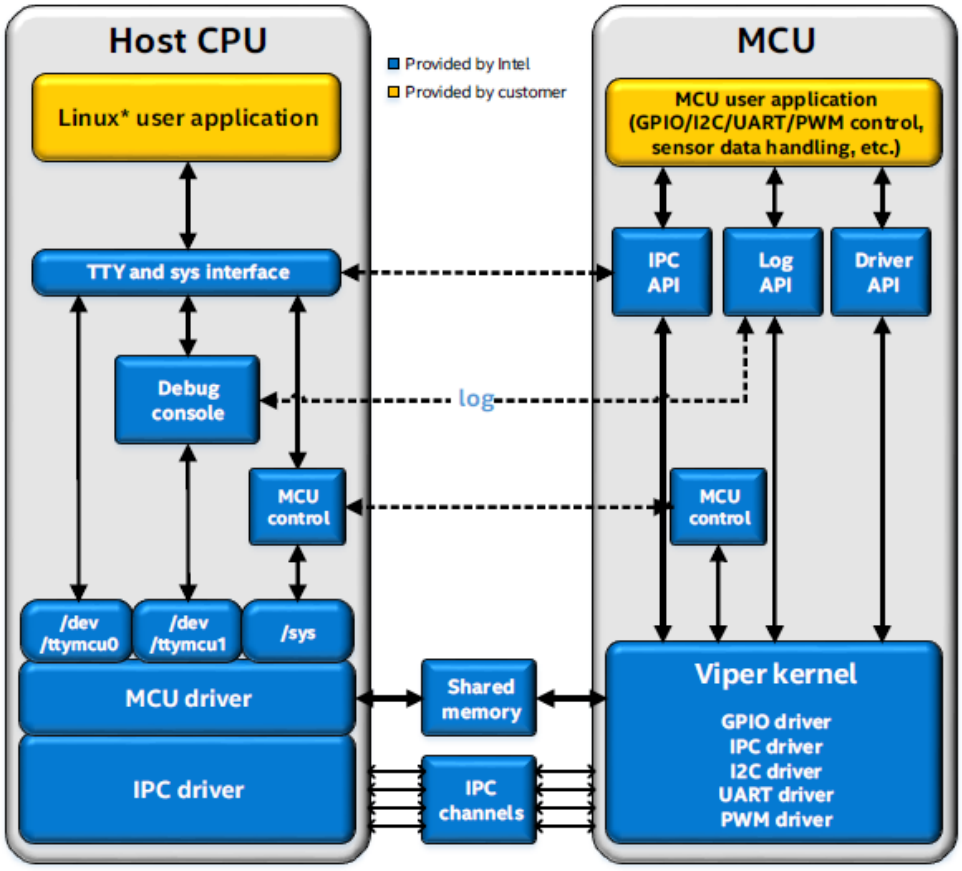


Figure 24 – Intel Edison Host CPU and MCU communication. Source: goo.gl/oVPPDb.

This single board computer can be attached to a Breakout Board, as in Figure 25, created by Intel too, which allow the users to have, besides the main board features, available programmable GPIO ports, microUSB port and power supply module. With the main board attached to the stated Breakout Board we have an aggregator with an acceptable processing capacity and same specifications as in the implementation in the RPI, but with a reduced size and lower power consumption, because of that, we also choose the use of Intel Edison as the system aggregator approaching the solution to an IoT scenario.

Figure 25 – Intel Edison and Breakout Board. Source: goo.gl/aCFsVE.

6.2 **VR-Banway: Intel Edison based VR-Unit**

With the refactoring of the VR-Unit component to be deployed in an Intel Edison board, our objectives of having a smaller and more portable device, still useful in the actual scenario and guided to new IoT scenarios, were achieved. That said, we are adding an Intel Edison based VR-Unit, VR-Banway, to the existing system, working alongside the Android and RPI based solutions.

With this aggregator device migration, the main concern is to make sure that this change does not compromise the actual system functionalities and performance, so, even being presenting a new aggregator it needs to fulfil all the requirements stated for the RPI implementation. The used programming language remains the same, and the application workflow continues to follow the same structure, as well as the way of storing

the collected data in the internal storage and sharing it to the external services and to the Cloud.

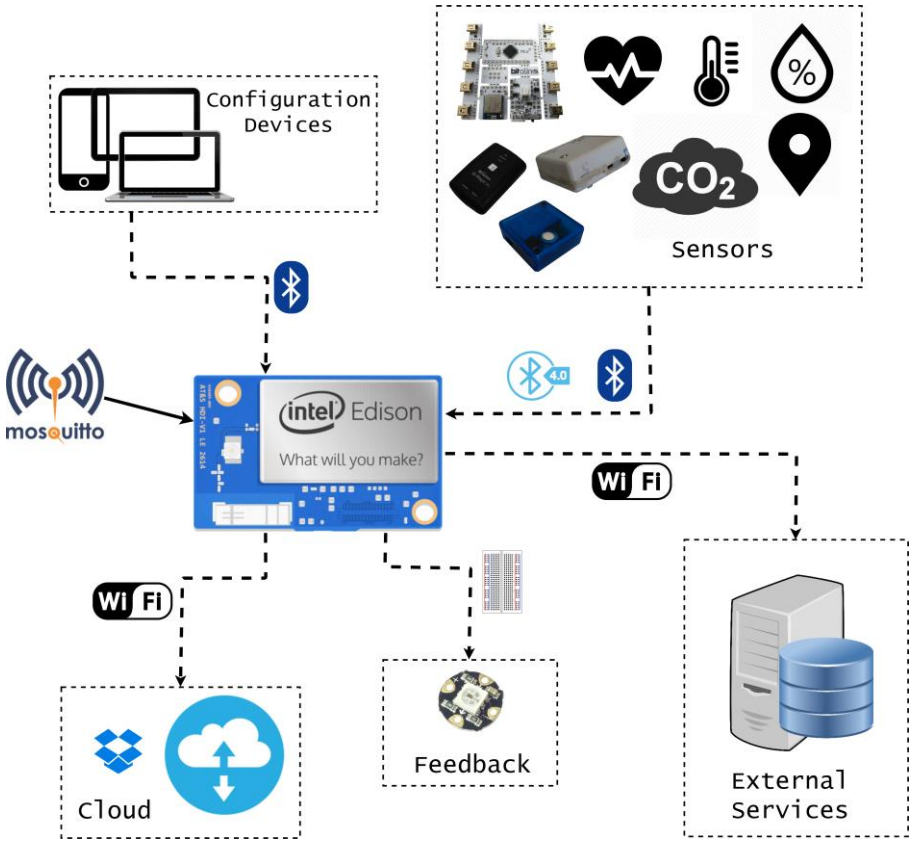


Figure 26 – VR-Banway architecture.

In the network communication of the collected data there were some constraints imposed by the use of this device, since the internal broker was changed, as explained below, from RabbitMQ to Mosquitto, due to requirements of the RabbitMQ server deployment, so by eliminating the RabbitMQ server we could not integrate the NetAPI service in VR-Banway due to its dependency with the RabbitMQ broker. Nevertheless, this constraint does not compromise the overall system workflow since it, originally, does not use this service.

6.2.1 Internal broker refactoring

The RPI based VR-Unit internal broker relies on RabbitMQ broker, technology that includes three common messaging protocols (AMQP, MQTT and STOMP) and use, by default, AMQP, although when the server is installed in a device all the features and dependencies [30] are installed, even if it would not be all used. By doing this alongside with the installation of the RabbitMQ server it is also required to mainly install the Erlang/OTP libraries, which is a programming language and an execution environment, a java client library which requires also an installation of the Java Runtime Environment (JRE) among other dependencies.

All the stated requirements make RabbitMQ a very heavy choice for our system, since we are trying to use Intel Edison, so, we replaced the internal broker for one that uses only the MQTT messaging protocol and has a lighter internal architecture. In order to have an API to manage the messages exchange with the MQTT protocol it was installed an Eclipse Mosquitto server which is a light server suitable for IoT scenarios, having minimal installation dependencies, low power consumption and memory usage, but, besides that, granting the same, or similar, capacities in terms of number of clients and connections, as shown in Figure 27, and quality of service [45].

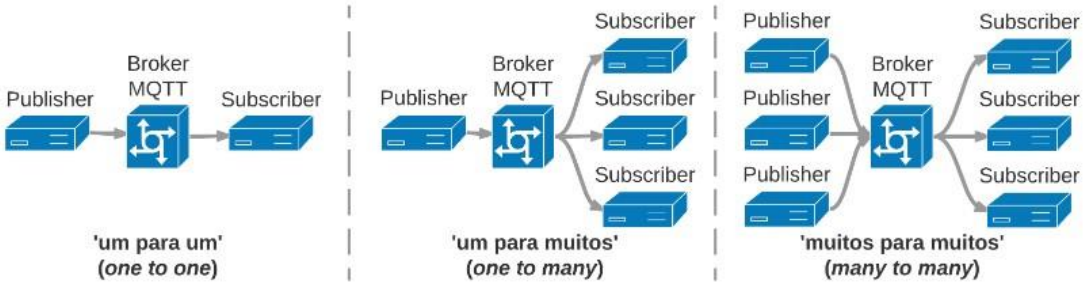


Figure 27 – Message distributions types supported in MQTT [45].

With the replacement of the existent internal broker for the Eclipse Mosquitto server we ensured that the internal messaging, of the collected data, was done without losing any functionalities in the system aggregator, both in Intel Edison and in RPI implementations.

6.3 New opportunities by using Intel Edison

With the use of SBC's in this project, specially, with the integration of the Intel Edison, the range of acting scenarios can be expanded and the approximation of the system to IoT scenarios is more remarkable due to the device size, by being much smaller it fits in a larger range of objects, like wearables, and the low power consumption, its battery lasts an acceptable time for the acquisitions and good process power that satisfies the projects purposes. This way the system becomes more portable and flexible towards the scenario specifications and conditions.

Although not implemented some new possible scenarios were discussed where VR-Banway can be deployed. The aggregator being so small (Intel Edison with battery module, as in Figure 28), with low power consumption and with the communication and process resources necessary to work with other devices (e.g. sensors), it can be deployed in several places with reduced size, since a packet of tobacco to a teddy bear, with, for instance, a accelerometer sensor. In this example scenario, the system can be deployed with all the technology (e.g. devices) hidden from the users, so the experiments done can be more realistic due to the user not being distracted or influenced by the presence of such technology.

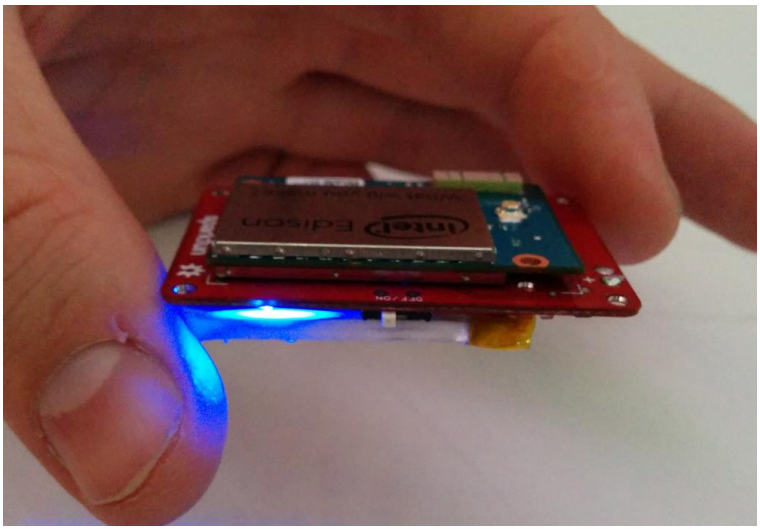


Figure 28 – Intel Edison with battery module. Source: goo.gl/aBfTe4.

7 Other System changes

Despite the focus of this dissertation being mainly the reengineering and refactoring of the VR-Unit component and associated requirements, there were also some changes in other system components, namely, the integration of new sensors and actuators, a new way to configure the VR-Units and in the data analysis and visualization components the addition of some new features and refactor the presentation of the acquired signals and events.

7.1 VR-Unit configuration method

The system VR-Units need to be configured in order to specify the acquisition information, used sensors and required external connections to share the collected data:

- Headquarter name.
- Team number.
- Data Collector IP.
- Enable local storage of the collected data.
- Used sensors settings.

In the Android VR-Unit this configuration is done in the aggregator itself due to the existence of a graphical interface, or, the configuration can be transmitted from an remote Android application, the VR-Remote.

In the refactored VR-Unit (RPI) and VR-Banway implementations there is no graphical interface available to configure the collection parameters in the aggregator itself, so, the configuration must be done from an external application. For this, the aggregator advertises his service over Bluetooth and, along with the overall processing, it waits for the receiving of an JSON file containing the collection configurable parameters:

- Headquarter name.
- Team number.

- Data Collector IP.
- External services settings to share the collected data.
- Used sensors settings.
- NetAPI service settings.
- Commands to control the acquisition flow.

When the aggregator wants to send the collected data to some of the external services, namely the Data Collector, it needs to be connected to the same network, and the network connection configuration cannot be done directly in the aggregator as it was done in the Android system. Therefore, the configuration user needs to send another file over Bluetooth to the aggregator, containing the configuration of the WPA supplicant service that is running in the aggregator OS to manage wireless network connections. The configuration user is responsible to correctly describe the WPA supplicant configuration file in order to establish the connection with the desired network.

7.2 Integrating new sensors and actuators

As stated before we introduce, with this dissertation, the use of some new sensors and actuators, this is possible because of the changes in the communication technologies, with the use of BLE we add to the system the possibility to work with some new vital signal sensors and by using SBC's we now have some liberty to explore actuators that require electronic connections to work properly.

For that purpose, the actual system now works with BITalino for acquiring physiological signals, adding the possibility to acquire, besides the traditional ECG, EEG, EDA and EMG signals.

The BITalino device [46] used in this system is the BLE version, as in Figure 29, which allows a lower power consumption in the device and a faster and asynchronous communication of the data. Using this physical device, the system, in comparison with the old one, is capable of sensing more vital signals besides the heart and the body temperature, such as:

- EEG: The electroencephalography sensor senses the electrical activity of the brain.

- EDA: The electrodermal activity sensor senses the variation in the electrical characteristics of the skin.
- ECG: The electrocardiography senses the variation of the electrical activity generated by the heart.
- EMG: The electromyography sensor senses the electrical activity of the muscles.

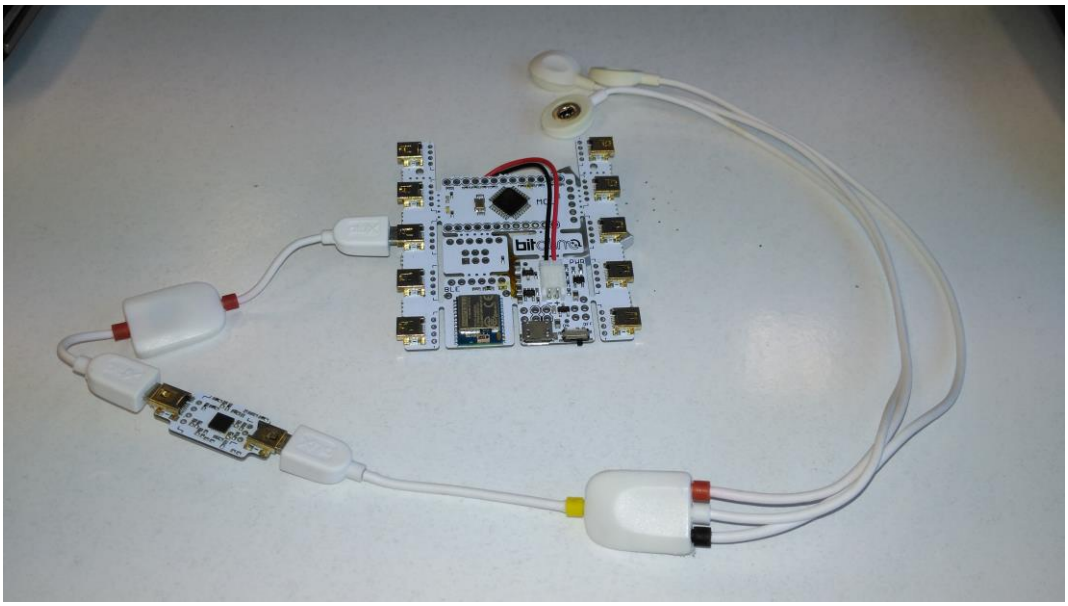


Figure 29 – BITalino BLE version with ECG sensor attached.

Besides the physiological signals referred, the actual system uses also the Pulse Sensor in order to get the heart rate directly, instead of obtaining it through calculations from the ECG. Pulse Sensor is a heart rate sensor created to work with multiple SBC's, like Arduino, RPI and others. It was also adapted by Plux (BITalino creators) to work in BITalino so it can be used alongside with the use of other physiological signals.

The use of the Pulse Sensor is suitable when the objective is just get a fast reading of the heart rate without the need to stick electrodes in the person's body due to its non-sticky method, it only uses a LED and a light detector. The LED sends his light to the skin surface and the light detector receives the light reflected with a certain intensity, though the intensity it is aware if it is a heartbeat or not [47].

Figure 30 – BITalino Led Pulse Sensor. Source: goo.gl/yJCDEX.

As for the actuators, to retrieve some feedback to the sensors user we integrate, in the VR-Unit component, the Flora RGB Smart NeoPixel version 2, as shown in Figure 31, designed to fit in Wearable and IoT scenarios due to its low power consumption, reduced size and the configuration simplicity [48]. We decide to correlate the collected heart rate with the showed colour in the RGB LED sending feedback to the user about his heart rate (heart rate between 50 and 80 shows green light, between 80 and 150 shows yellow light, below 50 and above 150 shows red light).

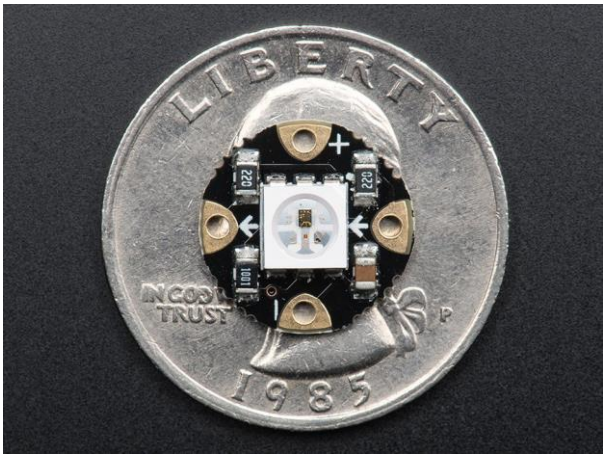


Figure 31 – Adafruit Flora RGB Smart NeoPixel, size comparison. Source: goo.gl/on9ZVV.

7.3 **Front-End components changes**

In the VR-Mission Review component, in aesthetic terms there were some changes in the web page navigation method and in the layout organization of the information to facilitate the platform usage by the firefighters. Also related to the information display we had some filter options to the displayed information so it can be filtered by several categories, like acquisition headquarter, team and date. Other important filter is about the timeline representing the duration of the acquisitions. We added the possibility, alongside with the time navigation already implemented as showed in Figure 32 on the bottom timeline, to define the time range displayed in the page, filtering the information to show only the data acquired in that time range, as showed in Figure 32 on the top timeline. This last filter is very important to the end users when the acquisition duration is extensive, giving the user the opportunity to select the desired time range to analyse the data with more detail in a certain interval of interest.

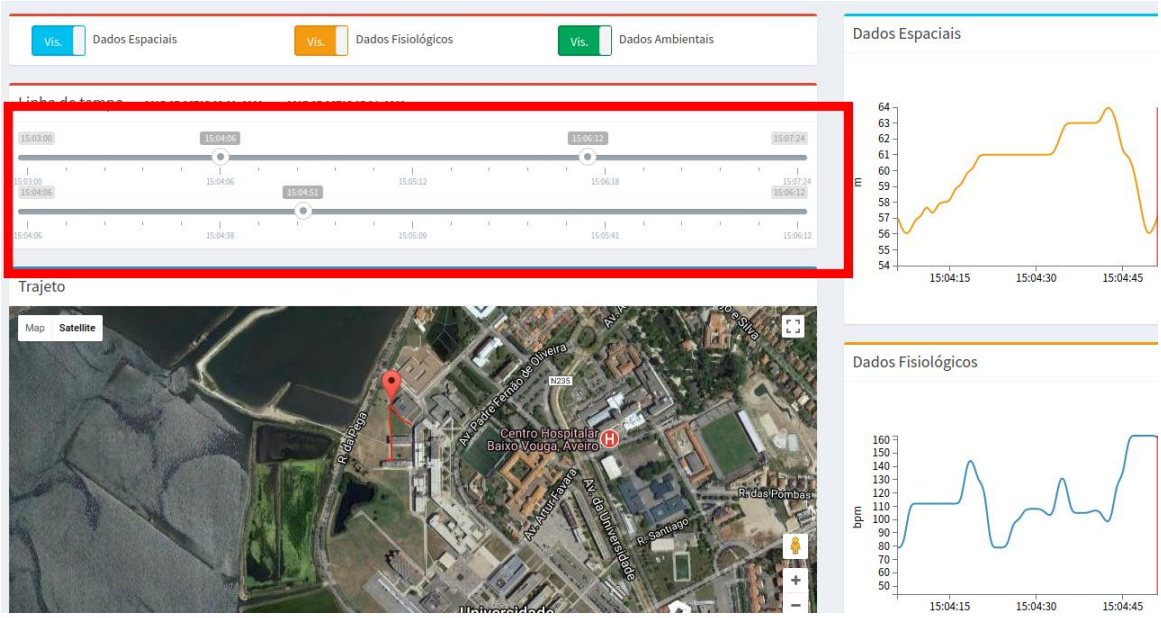


Figure 32 – VR-Mission Review interactive timelines.

Still related to the information filters, we added a simple login system, as shown in Figure 33, to access the sensible or confidential information of the sensors users, like the collected ECG, making the web page to have different views, one with all the information and other with only spatial and environmental information.

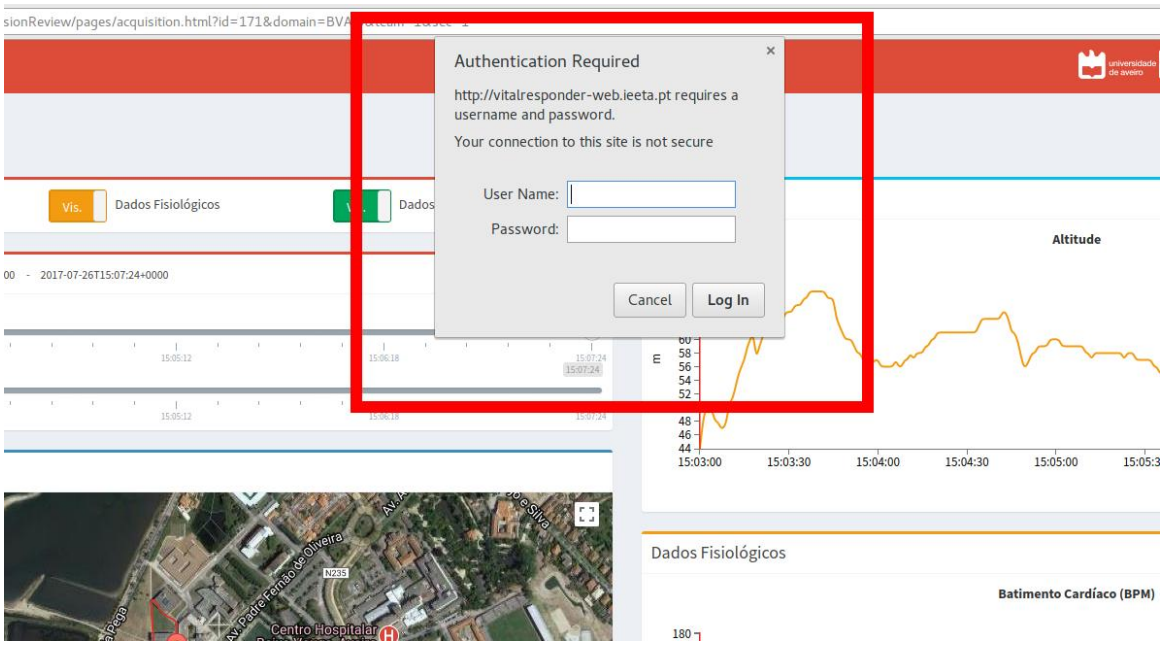


Figure 33 – VR-Mission Review login system.

The stated changes, mainly the created filters, are very important in this type of scenarios and, in general, in all kind of IoT scenarios due to the large quantity of collected data and displayed information. The data needs to be processed and the information organized in order to facilitate the users to visualize it and take more easy and accurate conclusions.

In the VR-Commander web page, there were not relevant aesthetic changes because the interface was already very user friendly and the necessary filters were already implemented. However, in this type of scenarios the quick response to an emergency is crucial, so we added a major change in the backend workflow of the data related to events and alarms triggered by the VR-Units.

Previously, when an alarm or event was triggered, the generated message in the aggregator was communicated via RabbitMQ to the Data Collector, then it was routed via REST API to the VR-Data, processed in this component, and only then passed to the VR-Commander, via REST API, to be displayed in the main window as an emergency alarm. With our backend changes in the communication technologies and workflow, when the message arrives to the Data Collector it is directly sent to the VR-Commander via RabbitMQ, so the time elapsed between the trigger of the alarm, in the VR-Unit, and the corresponding display in the VR-Commander is smaller due to the messaging efficiency and the route taken by the message.

8 Refactored VR-Unit and VR-Banway evaluation

The main objective of this work was to refactor the existing VR-Unit so our evaluation was focused in verifying if any existing features were still functional in the new version – either RPI (Refactored VR-Unit) and/or Intel Edison based (VR-Banway). Our evaluation consisted in test the three implementations of the VR-Unit (original, RPI refactored and Intel Edison based) and assert if, at least, all initial features were still functional.

Aside from the aggregator direct comparison in terms of specifications, we evaluate the effectiveness in the collection of physiological and environmental data, using the new integrated sensors.

8.1 Aggregator comparison

With this dissertation, the general project, VR2Market, ends up having three types of aggregators, each one for different purposes and scenarios. It cannot be explicitly said that one is better than other because all the three possible implementations of the aggregator are suitable, depending on the solution and requirements wanted for a specific scenario.

In general, we can compare the three implementations by enumerate the main differences between them:

- The Android solution has a graphic interface in the aggregator itself, but, it cannot exchange messages through the NetAPI service due to its lack of support for Ad hoc networks.
- Refactored VR-Unit - the RPI solution do not have a graphic interface to configure the collect parameters, it is done through a file sent via Bluetooth. The aggregator device can be attached with electrical components through the programmable GPIO ports. The board is cheaper than a mobile device that supports Android OS. By having Linux OS it is possible to use the NetAPI service to exchange messages between the Ad hoc network nodes.

- VR-Banway: The Intel Edison solution, also, has no graphic interface so the configuration must be done in the same way as in the RPI solution. The aggregator device is much smaller than the other devices and have a lower power consumption which is more appropriate for an IoT scenario. It also has the programmable GPIO ports and has an exclusive microcontroller that works regardless of the main processor tasks and schedule of them. The NetAPI service is not used because the process cost of the RabbitMQ server.

Figure 34 – Size comparison between Intel Edison, RPI and Android.

The following table (Table 6) demonstrates, in a more summarized way, the main features required for the VRUnit solution and the adequacy of all the three different implementations.

	Android	RPI	Intel Edison
Graphic Interface	Yes	No	No
Bluetooth LE	No	Yes	Yes
NetAPI	No	Yes	No
GPIO	No	Yes	Yes
Size	Big	Medium	Small
Power consumption	High	Low	Low
Processor velocity	1.4 GHz (Nexus 5X)	1.2 GHz (RPI3)	500 MHz
Cloud services	Yes	Yes	Yes
Cost	High	Low	Medium
Remote configuration	Yes	Yes	Yes

Table 6 – Feature comparison between VR-Unit implementations.

8.2 Integration in the VR2Market system

With the new implementations we guarantee that the past features were also running in the new deployments without compromising performance status, plus, the addition of the new sensors and system features.

By changing the aggregator, the interface methods and technologies that support the communication and data exchange between the VR-Units and the rest of the system must remain functional and follow the already standard methods and technologies, to guarantee that, even if the used technologies in the aggregator reengineering are different, we ensured to create a middle layer between the aggregator and the system, so, the technologies used in the aggregator are transparent.

To have a proof of concept we have made some acquisitions with both systems and compared if the collected data in terms of quality, consistency and sample rate was acceptable in comparison to the Android implementation. Some aspects are also related with the sensors, like sample rate, spatial aspects and isolation of external interferences, but, besides that the data can be analysed and validated. Besides the collected data the aggregator performance is also a comparison parameter. We compare the overall performance in terms of quantity of data, capacity to share that data to the rest of the system, to another external service and to the Cloud.

For the acquisition of the physiological and environment signals we make three acquisitions, each one with the different implementations of the VR-Unit, around the same location, as shown in Figure 35, to have not many variations in the environmental conditions and in the physical effort of the subject.

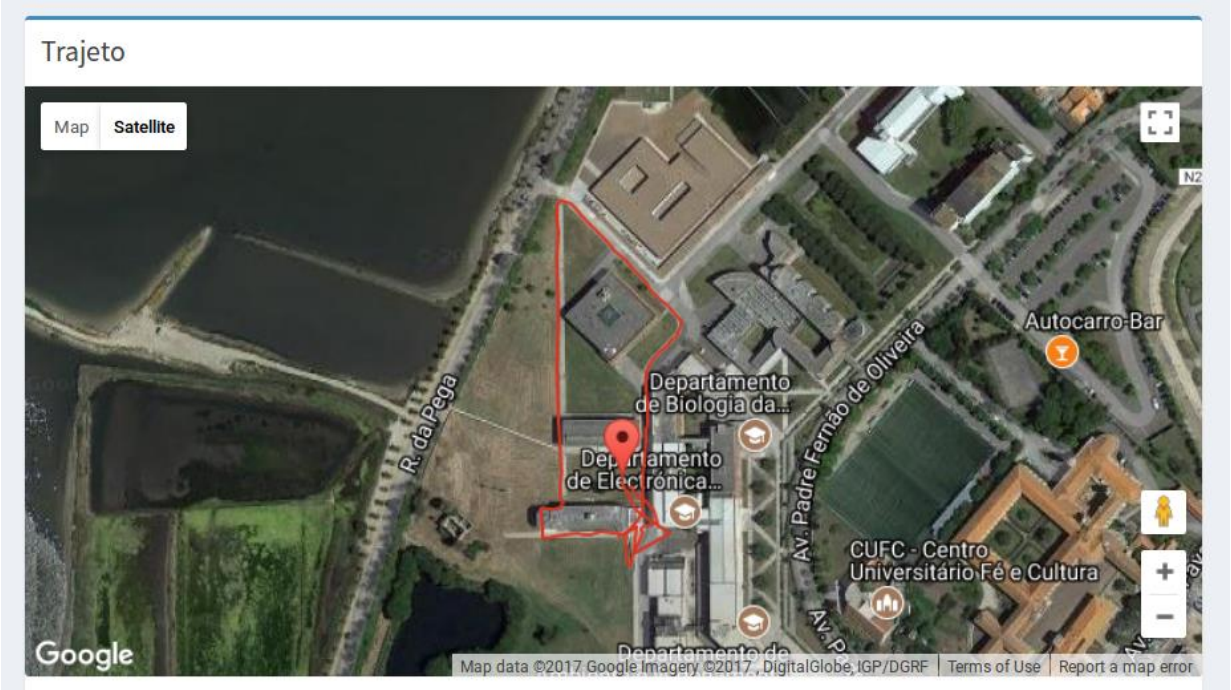


Figure 35 – GPS route of the acquisitions.

In the ECG signal comparison, we used the Vital Jacket sensor in the Android system and the BITalino sensor in the RPI and Intel Edison based, VR-Banway. We can see in the Figure 36 part of the ECG signal acquired from the Vital Jacket to the Android aggregator, the Vital Jacket works with samples at 500 Hz, and in Figure 37 and Figure 38 is the ECG signals acquired from the BITalino that works with samples as 1000 Hz, to the RPI and Intel Edison implementations correspondingly. We conclude that the refactored VR-Unit and VR-Banway implementations can acquire and process this type of signals at the defined sample rate.

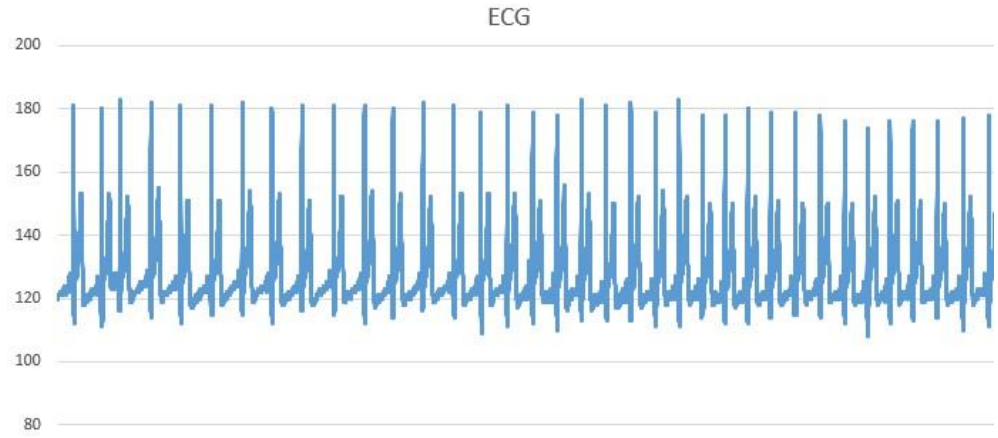


Figure 36 – ECG signal acquired from Vital Jacket to the Android VR-Unit.

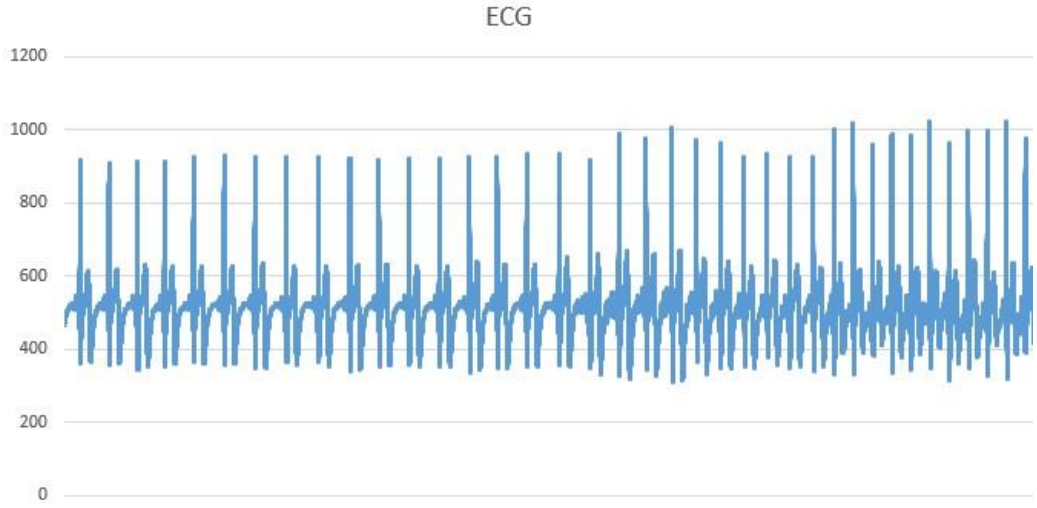


Figure 37 – ECG signal acquired from BITalino to the RPI VR-Unit.

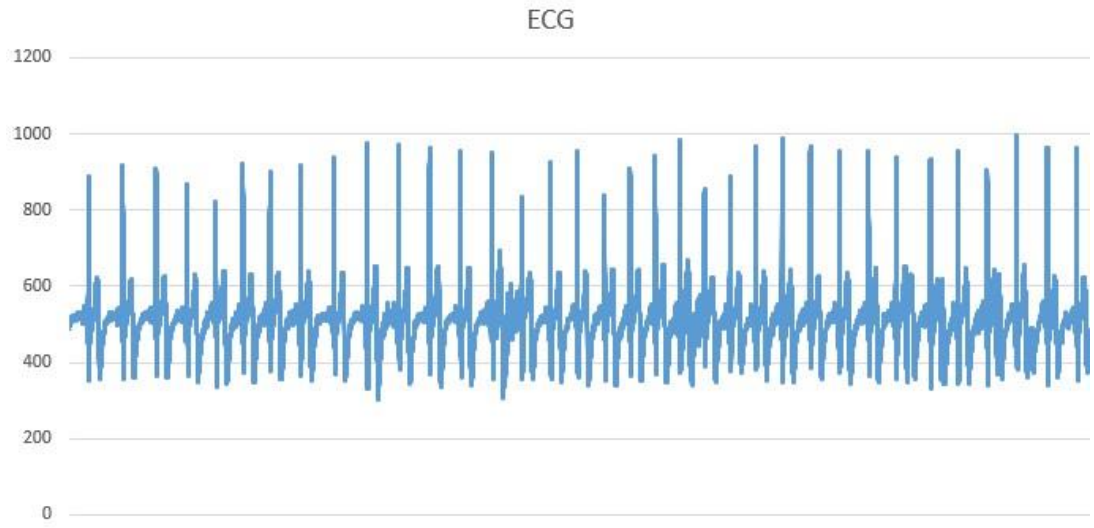


Figure 38 – ECG signal acquired from BITalino to the Intel Edison based VR-Banway.

The environmental variables were acquired from the FREMU. Here the difference could be only in the aggregator, because the sensor used was the same in the three systems so the only constraint or bottleneck that could interfere in the results was the aggregator capacity of receive and process the data fast enough in order to do not lose any data communication from the sensor. For this test purposes we focus on the environmental temperature signal and we can see in the Figure 39, Figure 40 and Figure 41 the signals acquired to the Android, RPI VR-Unit and Intel Edison VR-Banway correspondingly.

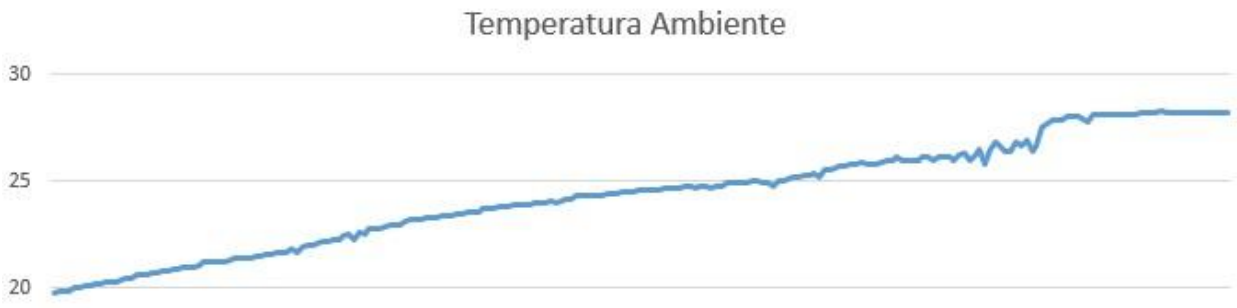


Figure 39 – Environmental temperature acquired to Android VR-Unit.

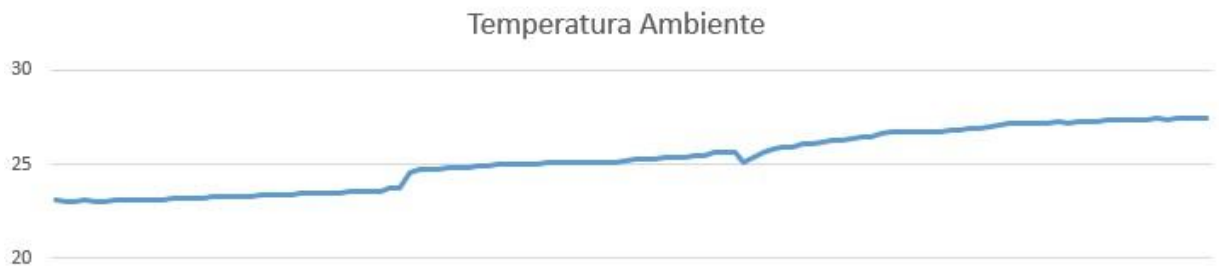


Figure 40 – Environmental temperature acquired to RPI VR-Unit.

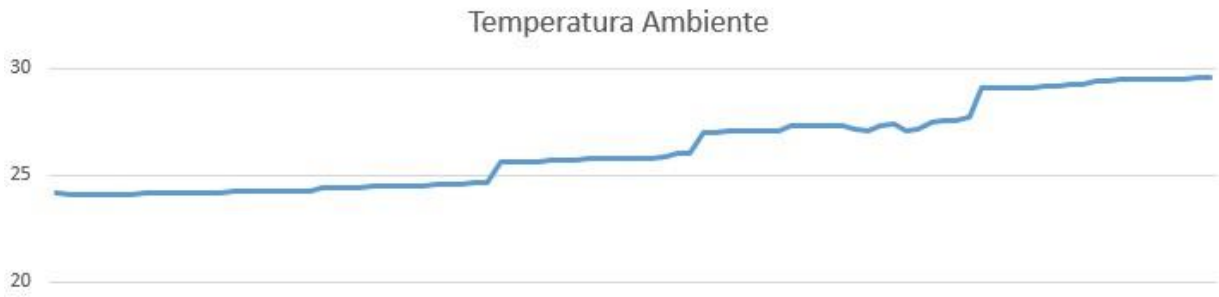


Figure 41 – Environmental temperature acquired to Intel Edison VR-Banway.

8.3 **Ad Hoc scenarios**

With the use of the NetAPI service, we introduced the bidirectional communication and consequently new functionalities such as the synchronization of the RPIs internal clocks and the send of a “check team” command to check the activity and response of the aggregators in the real scenario field.

In the implementation of the bridge between the NetAPI service and the Data Collector component, the commands are sent by this bridge running application.

To synchronize the RPIs internal clocks the bridge application requests to the NetAPI service for the known nodes in the Ad hoc network, as shown in Figure 42, and sends a message to all the known nodes indicating the actual date of the system (Figure 43 A). When the nodes receive the message, their internal clock is updated to the received date and the synchronization is complete (Figure 43 A, B).

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.1.1     0.0.0.0         UG    100    0        0 enp0s3
10.0.0.0        0.0.0.0         255.0.0.0       U     0      0        0 wlx74da3804f529
10.176.64.39    10.176.64.39    255.255.255.255 UGH   2      0        0 wlx74da3804f529
10.234.155.63   10.234.155.63   255.255.255.255 UGH   2      0        0 wlx74da3804f529
```

Figure 42 – Ad hoc network know nodes.

(A)

```
[NetAPI Gateway] Message sent to 10.176.64.39 : Sync Clock_Mon Nov 20 16:06:57 WET 2017
[NetAPI Gateway] Message sent to 10.234.155.63 : Sync Clock_Mon Nov 20 16:06:57 WET 2017
```

(B)

```
[NetAPI Service] Data message from 10.108.104.246 - 1 : Sync Clock_Mon Nov 20 16:06:57 WET 2017
Mon Nov 20 16:06:57 WET 2017
```

(C)

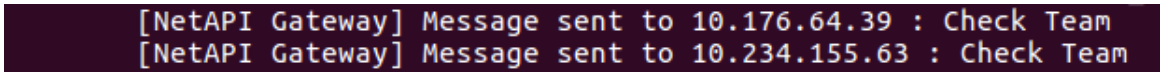
```
[NetAPI Service] Data message from 10.108.104.246 - 1 : Sync Clock_Mon Nov 20 16:06:57 WET 2017
Mon Nov 20 16:06:57 WET 2017
```

Figure 43 – “Sync clock” command workflow.

As shown in Figure 42, there are two know nodes with IP’s 10.176.64.39 (pt.aveiro.aveiro1.equipa1.x1 is the corresponding name resolution) and 10.234.155.63 (pt.aveiro.aveiro1.equipa1.x2 is the corresponding name resolution). In the clocks synchronization the collector node sends the message “Sync Clock_Mon Nov 20 16:06:57 WET 2017” to the two known nodes, as in Figure 43 A). When the nodes receive the message from the collector node (10.108.104.246 with the resolution name pt.aveiro.aveiro1.collector) their clocks are updated to “Mon Nov 20 16:06:57 WET 2017” as shown in the Figure 43 B), C).

The “check team” functionality represents fully the bidirectional communication due to the request and response scenario where the bridge application broadcasts a request message to all the known nodes (Figure 44 A), signalling that the aggregators need to respond to confirm their welfare and connectivity. When the aggregator nodes receive the message they send a response message (Figure 44 B, C), indicating that they are reachable in the network (Figure 44 D).

(A)



(B)

(C)

(D)

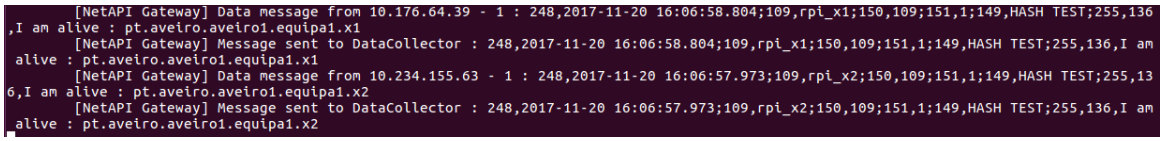


Figure 44 – Check team command workflow.

The know nodes are the same as in the “Sync Clock” command (10.176.64.39 and 10.234.155.63), the collector node sends the message “Check Team” to the nodes, as in Figure 44 A), when received the nodes respond with the messages “248,2017-11-20 16:06:58.804;109,rpi_x1;150,109;151,1;149, HASH TEST;255,136,I am alive : pt.aveiro.aveiro1.equipa1.x1” and “248,2017-11-20 16:06:58.804;109,rpi_x2;150,109;151,1;149, HASH TEST;255,136,I am alive : pt.aveiro.aveiro1.equipa1.x2” respectively, as in Figure 44 B) C), the message structure follows the general data messages of the system. With the receiving of the nodes response, the collector forwards the message to the DataCollector, as in Figure 44 D) (“Message sent to DataCollector : ...”).

9 Conclusion and future work

In this dissertation we aimed to refactor the VR2Market system gateway (VR-Unit) and migrate it to SBCs in order to have smaller devices with lower power consumption without compromising the already functional requirements. The VR-Unit refactoring involved incorporating a new service, called NetAPI, for Ad hoc network support and integrating new sensors with BLE interface.

This refactoring also allowed deploying a new version of the gateway, named VR-Banway, using the Intel Edison compute module – the reengineering of the original VR-Unit also had impact on the size and processing footprint due to the adoption of more flexible brokering solution. This allowed extending the existing usage scenarios of VR-Unit (the current VR2Market gateway solution) to more wide area of IoT as the addition of BLE support and port to Intel Edison allow a smaller footprint either in power requirements and overall form factors – when compared with existing RPI solution.

9.1 **Future work**

Despite of the work done in this dissertation, we are aware of some functionalities that can be exploited in the future for the improvement of the overall system usability.

A relevant improvement to be done about the bidirectional communication is related to the user interface, namely, in the VR-Commander application component, where it should be added a button with the functionality to make the “Check Team” command. Although the functionality is implemented, through the NetAPI bridge component, there are no graphic interface to execute the command, so, in the VR-Commander this button would add the possibility to execute that command. Beyond the stated command, other type of commands and features can be added to the system to dialogue with the aggregators.

In relation to the NetAPI service, to integrate this service in more IoT scenarios the dependency with the RabbitMQ broker should be replaced by a lighter one, like Mosquitto, used in this project, although it can be already used in many scenarios with that refactoring the range of IoT scenarios to be deployed would be wider.

To help the end-users in the interactions with our system, namely, in the configuration of the refactored RPI VR-Unit and VR-Banway there should be developed an application similar to VR-Remote, or even adapt this one, to have an UI to configure the acquisition parameters of the VR-Units.

References

[1] C. INESC TEC, IT, UA, “VR2MARKET.” [Online]. Available: https://www.inesctec.pt/cber/projetos/projectos-em-destaque/vr2market/. [Accessed: 12-Jul-2017].

[2] R. Budihal, “EETimes - Emerging trends in embedded systems and applications,” 2010. [Online]. Available: https://www.eetimes.com/author.asp?section_id=36&doc_id=1287560. [Accessed: 05-Aug-2017].

[3] A. Usman and S. H. Shami, “Evolution of communication technologies for smart grid applications,” *Renew. Sustain. Energy Rev.*, vol. 19, pp. 191–199, 2013.

[4] O. León, J. Hernández-Serrano, and M. Soriano, “Securing cognitive radio networks,” *Int. J. Commun. Syst.*, vol. 23, no. 5, pp. 633–652, 2010.

[5] A. Meola, “What is the Internet of Things (IoT)?,” 2016. [Online]. Available: http://www.businessinsider.com/what-is-the-internet-of-things-definition-2016-8. [Accessed: 21-Aug-2017].

[6] J. Morgan, “A Simple Explanation Of ‘The Internet Of Things,’” 2014. [Online]. Available: https://www.forbes.com/sites/jacobmorgan/2014/05/13/simple-explanation-internet-things-that-anyone-can-understand/#331e25c31d09. [Accessed: 21-Aug-2017].

[7] A. McEwen and H. Cassimally, *Designing the Internet of Things*. John Wiley & Sons, 2013.

[8] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, “Internet of Things for Smart Cities,” *IEEE Internet Things J.*, vol. 1, no. 1, pp. 22–32, 2014.

[9] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, “Future internet: The internet of things architecture, possible applications and key challenges,” *Proc. - 10th Int. Conf. Front. Inf. Technol. FIT 2012*, pp. 257–260, 2012.

[10] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Context Aware Computing for The Internet of Things: A Survey,” vol. 16, no. 1, pp. 414–454,

2013.

[11] D. The *et al.*, *Beyond the Internet of Things*. Springer, 2016.

[12] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of Things (IoT): A vision, architectural elements, and future directions,” *Futur. Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.

[13] S. Haller, “The Things in the Internet of Things,” *Symp. A Q. J. Mod. Foreign Lit.*, no. September, pp. 97–129, 2010.

[14] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta, “Role of Middleware for Internet of Things: a Study,” *Int. J. Comput. Sci. Eng. Surv.*, vol. 2, no. 3, pp. 94–105, 2011.

[15] D. Pavithra and R. Balakrishnan, “IoT based Monitoring and Control System for Home Automation,” *Proc. 2015 Glob. Conf. Commun. Technol.*, vol. 35, no. 1, p. 24, 2015.

[16] S. K. Datta, C. Bonnet, and N. Nikaein, “An IoT gateway centric architecture to provide novel M2M services,” *2014 IEEE World Forum Internet Things, WF-IoT 2014*, pp. 514–519, 2014.

[17] M. Aazam and E. N. Huh, “Fog computing and smart gateway based communication for cloud of things,” *Proc. - 2014 Int. Conf. Futur. Internet Things Cloud, FiCloud 2014*, pp. 464–470, 2014.

[18] C. Doukas and I. Maglogiannis, “Bringing IoT and cloud computing towards pervasive healthcare,” *Proc. - 6th Int. Conf. Innov. Mob. Internet Serv. Ubiquitous Comput. IMIS 2012*, pp. 922–926, 2012.

[19] L. Costantino, N. Buonaccorsi, C. Cicconetti, and R. Mambrini, “Performance analysis of an LTE gateway for the IoT,” *2012 IEEE Int. Symp. a World Wireless, Mob. Multimed. Networks, WoWMoM 2012 - Digit. Proc.*, 2012.

[20] S. Tilkov, “The modern cloud-based platform,” *IEEE Softw.*, vol. 32, no. 2, 2015.

[21] Q. Zhu, R. Wang, Q. Chen, Y. Liu, and W. Qin, “IOT Gateway: BridgingWireless Sensor Networks into Internet of Things,” *2010 IEEE/IFIP Int. Conf. Embed. Ubiquitous Comput.*, pp. 347–352, 2010.

[22] S. R. Sinha and Y. Park, *Building an Effective IoT Ecosystem for your Business*. Springer, 2017.

[23] M. Rouse, “TechTarget - IoT gateway,” 2017. [Online]. Available: http://whatis.techtarget.com/definition/IoT-gateway. [Accessed: 22-Aug-2017].

[24] J. Treadway, “TechTarget - Enter the IoT gateway,” 2016. [Online]. Available: http://internetofthingsagenda.techtarget.com/feature/Using-an-IoT-gateway-to-connect-the-Things-to-the-cloud. [Accessed: 22-Aug-2017].

[25] R. Buyya and A. V. Dastjerdi, *Internet of Things : Principles and Paradigms*. Elsevier, 2016.

[26] J. Mesnil, *Mobile and Web Messaging : Messaging Protocols for Web and Mobile Devices*. O’Reilly Media, Inc., 2014.

[27] J. Kreps, N. Narkhede, and J. Rao, “Kafka: a Distributed Messaging System for Log Processing,” *ACM SIGMOD Work. Netw. Meets Databases*, p. 6, 2011.

[28] G. C. Hillar, *Mqtt Essentials : A Lightweight IoT Protocol*. Packt Publishing Ltd, 2017.

[29] P. Zaitsev, “Exploring Message Brokers: RabbitMQ, Kafka, ActiveMQ, and Kestrel,” 2014. .

[30] E. Ayanoglu, Y. Aytas, and D. Nahum, *Mastering RabbitMQ*. 2015.

[31] B. Moyer, “EEJournal - All About Messaging Protocols,” 2015. [Online]. Available: http://www.eejournal.com/article/20150420-protocols/. [Accessed: 06-Sep-2017].

[32] A. Piper, “VMware - Choosing Your Messaging Protocol: AMQP, MQTT, or STOMP,” 2013. [Online]. Available: https://blogs.vmware.com/vfabric/2013/02/choosing-your-messaging-protocol-amqp-mqtt-or-stomp.html. [Accessed: 06-Sep-2017].

[33] Z. Shelby, K. Hartke, and C. Bormann, “The Constrained Application Protocol (CoAP),” pp. 1–112, 2014.

[34] B. Moyer, “EEJournal - Distributing Data, Machine to Machine,” 2014. [Online]. Available: http://www.eejournal.com/article/20140324-rti/. [Accessed: 06-Sep-2017].

[35] D. G. Reina, M. Askalani, S. L. Toral, F. Barrero, E. Asimakopoulou, and N. Bessis, “A Survey on Multihop Ad Hoc Networks for Disaster Response Scenarios,” *Int. J. Distrib. Sens. Networks*, 2015.

[36] A. Gunther, “WHAT ARE 802.11 TOPOLOGIES?,” 2016. [Online]. Available:

https://boundless.aerohive.com/technology/What-Are-80211-Topologies-Going-Back-toBasics.html. [Accessed: 11-Sep-2017].

[37] P. Sareen, “Available Online at www.ijarcs.info A Study of Ad-Hoc Networks,” vol. 6, no. 7, p. 2015, 2015.

[38] A. Saini and A. Malik, “Routing in internet of things: A survey,” in *Communication and Computing Systems*, 2017.

[39] J. Loo, J. L. Mauri, and J. H. Ortiz, *Mobile ad hoc networks : current status and future trends*. CRC Press, 2016.

[40] T. Magalhães, I. C. Oliveira, and J. M. Fernandes, “Message based integration in Cyber-Physical System: firefighters in the field,” *12th EAI Int. Conf. Mob. Ubiquitous Syst. Comput. Netw. Serv.*, pp. 0–1, 2015.

[41] Oracle, “Java™ EE at a Glance,” 2017. [Online]. Available: http://www.oracle.com/technetwork/java/javaee/overview/index.html. [Accessed: 30-Oct-2017].

[42] R. Hat, “What is WildFly?,” 2017. [Online]. Available: http://wildfly.org/about/. [Accessed: 30-Oct-2017].

[43] A. Al-Akkad, *Working Around Disruptions of Network Infrastructures: Mobile Ad-Hoc Systems for Resilient Communication in Disasters*. Springer, 2016.

[44] H. Fairhead, *Exploring Intel Edison*. I/O Press, 2016.

[45] A. B. B. Torres, A. R. Rocha, and J. N. De Souza, “Análise de Desempenho de Brokers MQTT em Sistema de Baixo Custo,” *XXXVI Congr. da Soc. Bras. Comput.*, pp. 2804–2815, 2016.

[46] H. P. Silva, J. Guerreiro, A. Lourenço, A. Fred, and R. Martins, “BITalino: A novel hardware framework for physiological computing,” *Int. Conf. Physiol. Comput. Syst. PhyCS 2014*, no. January, pp. 246–253, 2014.

[47] R. Pujar, “EMBEDDED WORLD - Learn how a heart beat sensor works,” 2017. [Online]. Available: http://www.raviyp.com/embedded/140-learn-how-a-heart-beat-sensor-works. [Accessed: 19-Oct-2017].

[48] B. Stern and T. Cooper, *Getting Started with Adafruit FLORA: Making Wearables with an Arduino-Compatible Electronics Platform*. Maker Media, 2015.

Appendices

A.1 Public Presentations

The project developed in this dissertation was presented in a public event, during 2017, called students@deti in the DETI department of UA. The following poster was created to present the project in the event.

Figure 45 – Poster of the VR-Banway system. Presented at students@deti 2017.