TÉCNICO LISBOA

Flexible, Multi-platform Middleware for Wireless Sensor and Actuator Networks

Rui Miguel Guerreiro Francisco

Thesis to obtain the Master of Science Degree in

Telecommunications and Informatics Engineering

Supervisors: Prof. Artur Miguel do Amaral Arsénio
Eng. Fernando Nabais

Examination Committee

Chairperson: Prof. Paulo Jorge Pires Ferreira
Supervisor: Prof. Artur Miguel do Amaral Arsénio
Members of the Committee: Prof. Rui António dos Santos Cruz

May 2014

To my parents, brothers and grandparents

Acknowledgments

First of all, I would like to thank to my supervisor, Professor Artur Arsenio for all the support and guidance during my master thesis. His advices and fruitful discussions were very enlightening and made me learn a lot. A special word to João Carias, David Gonçalves, Rui Silva, José Teixeira, Joaquim Guerra, Felipe Tavares, Nuno Menezes, Nuno Costa and Helena Santos for all their support and friendship during the stage in the Ydream Robotic. A word of gratitude to Ricardo Gomes and Carlos Azevedo from the Plux for all their support during the realization of my thesis. To my friends Hugo Serra, Pedro Torres, João Ambrosio, Nelson Sales for all their friendship, fun hard work and for being there in the most stressful moments. To Catarina Sá Borges and Mariana Silva for being such good friends, for their inspiring company, support, motivation and for being always presents when I needed during this last year. Last but not least, to my family. To my parents for all the love, support and comprehension. To my brothers for always being by my side. To my grandparents for all the patience, love and courage.

Resumo

A presença da tecnologia pode trazer algumas vantagens importantes para a nossa vida quotidiana, pois alguns dos dispositivos estão a ficar mais inteligentes e podem começar a fazer algumas tarefas que no passado, apenas os seres humanos poderiam fazer. Os dispositivos obtêm estas novas características quando estes se ligam à Internet. Estes acontecimentos levaram ao nascimento de um novo paradigma chamado Internet das Coisas. Desta forma os dispositivos têm acesso a mais informação tornando possível tomar melhores decisões. Com os progressos realizados ao nível da comunicação entre os objetos da Internet, surgiu outro paradigma que se chama redes de sensores sem fios. Estas redes são compostas por pequenos sensores que realizam a aquisição, recolha e análise de dados. No entanto, as redes de sensores sem fios têm alguns problemas ao nível do consumo energético e processamento. A capacidade quase infinita de armazenamento, as grandes velocidades de processamento e a rápida elasticidade faz da computação em nuvem uma boa solução para estes problemas. Para gerir com eficiência os recursos dos dispositivos e conseguir uma comunicação eficiente com várias plataformas móveis, este trabalho propõe um *middleware* que integra sensores e atuadores em múltiplas plataformas. Dando assim a possibilidade de transferir fluxos de execução entre dispositivos e a plataforma *cloud*, sob certas condições.

Palavras-chave: Internet das Coisas, Máquina a Máquina, Redes de sensores sem fios, Computação em nuvem, *Middleware*

Abstract

In these days technology is present everywhere, such as in our homes or in our jobs. This presence of technology in our lives can bring some important advantages because the devices can start doing tasks that in the past only humans could do. The devices gain these new characteristics when they become connected to the Internet. This leads to the birth of a new paradigm named Internet of Things. With Internet of Things the devices have access to more information and with it they can make better decisions. With the progress made in the communication between the Internet objects, another paradigm called Wireless Sensor Network emerged. The Wireless Sensor Network is composed by small sensing nodes which perform the acquisition, collection and analysis of data. However, the Wireless Sensor Network has some problems such as energetic consumption and CPU load. The almost infinite capability of storage, the large processing speeds and the rapid elasticity makes Cloud Computing a very good solution to these problems. This work proposes a middleware that seamlessly integrates sensors and actuators on multiple platforms, to efficiently manage the resources of the devices and achieve efficient communication with various platforms (cloud, mobile). The goal is to allow the flow of execution to be transferred between the device and the platform under certain conditions.

Keywords: Internet of Things, Machine-to-Machine, Wireless Sensor Networks, Cloud Computing, Middleware

Contents

List of Tables

List of Figures

List of Acronyms

ACE Adaptive Communication Environment

CPU Central processing unit

$DaaS$ Data storage as a Service

$H2H$ Human-to-Human

$HTTP$ Hypertext Transfer Protocol

$IaaS$ Infrastructure as a Service

$IoIT$ Internet of Intelligent Things

IOT Internet of Things

$JSON$ JavaScript Object Notation

JVM Java Virtual Machine

$M2M$ Machine to machine

$OSGi$ Open Service Gateway initiative

$PaaS$ Platform as a Service

QOS Quality of service

$RaaS$ Robotics as a Service

RAM Random-access memory

$REST$ Representational State Transfer

ROS Robot Operating System

$SaaS$ Software as a Service

SOA Service Oriented Architecture

$SOAP$ Simple Object Access Protocol

SSH Secure Shell

TCP Transmission Control Protocol

UDP User Datagram Protocol

$WSAN$ Wireless Sensor and Actor Network

WSN Wireless sensor network

$YARP$ Yet Another Robot Platform

Chapter 1

Introduction

The revolution that has occurred in the past years in sensor and actuator technology is making much easier to build Wireless Sensor and Actuator Networks (WSAN). This WSAN consists in a set of nodes (sensors and actuators) that cooperate among them to achieve the goal of collecting data and make some decisions based on collected data. Autonomous network nodes either have a short range, or their computation power is weak. But when used collectively they are effective over large areas offering higher computational power.

Nowadays WSAN have a more pronounced growth, and it is not unreasonable to expect that in a few years our lives become dependent of WSAN in certain areas such as environmental, medical, transportation entertainment and city management. Although there has been an evolution of the nodes in WSAN these continue to have limited battery, limited computational power, etc. And with these problems the network node can crash due to lack of sufficient resources, and this may lead to jeopardize the smooth operation of the infrastructure. So, in such cases sensor and actuator networks cannot operate as stand-alone networks.

But there must be an efficient way for the captured data to be stored and manipulated. Cloud computing may offer attractive solutions for these issues. Indeed, it allows the reduction of the initial costs associated with the computational infrastructure. Another relevant aspect is that the cloud computing resources are easily and automatically adjustable according to the real infrastructure needs. This way, the computational resources are easily scalable following the growth of the infrastructure. Another important point is related with the fact that the customer only pays for the cloud resources that he actually used, therefore he does not have the problem of paying for resources that were not used. But the most important aspect to this work is that the cloud computing ensures unlimited battery, unlimited storage, unlimited computational power due to its immense of resources.

Now that a possible solution to the problem has been found, is necessary to find a way to know when the node of WSAN does not have the capability to perform some operation, and how to communicate in a transparent manner using the cloud infrastructure. To solve these issues efficiently it is necessary a mediator that manages the resources of the nodes of WSAN efficiently and communicates transparently to users with various platforms. We can call this mediator a middleware.

1.1 Goal

The main purpose of this work is to design, implement and test a middleware for collecting, sharing and transmitting data between systems of sensors and actuators. More specifically the middleware will dynamically transfer execution flows between mobile devices and the cloud, according to the battery status of the devices, the network performance and the device itself. To demonstrate the potential of the solution a prototype will be implemented.

The expected goals of this work are the following:

- The design of a solution and architecture that solve the proposed problem;
- The development of a functional prototype for the proposed solution that allows to demonstrate the potentialities of the middleware
- The deployment and evaluation of the developed prototype

1.2 Contributions

The contributions of this work are the following:

- A complete State of the Art document on the background of M2M communication, WSAN and its integration with Cloud Computing and middleware resources management of WSAN ;
- A comparative evaluation of currently related architectures;
- The development of a middleware for collecting, sharing and transmitting data.

1.3 Document Organization

This work is organized as follows. Chapter 2 presents a review on the State of the Art for the current work and the description of some related architectures. The architecture for the solution is presented in Chapter 3, referring the different parts of the system while abstracting contextual details. Chapter 4 clarifies the architectural implementation process in the rationale behind its methodology criteria and requirement-oriented decisions. Chapter 5 provides a basis for the assessment methods, while presenting the experimental results of this solution. In the last chapter, final remarks are made, along with a contextualised summary of the contributions. Finally, future work is proposed.

Chapter 2

State Of The Art

This section starts with the explanation of a promising paradigm, named Internet of Things (IoT). This paradigm is presented in section 2.1 discussing several potentialities and some application scenarios. Afterwards section 2.2 presents an emerging paradigm called Internet of Intelligent Things (IoIT). In section 2.3 it is described a new communication paradigm, called Machine-to-Machine (M2M), as well as its main challenges. Then, in section 2.4 it is presented an emerging technology named Wireless Sensor and Actuator Networks. The Cloud Computing area is presented in section 2.5 as well as its concepts and incredible potentialities. Next, in section 2.6 it is described the impact that the cloud computing concepts and characteristics could bring to the M2M and Wireless Sensor Network's area. Being the robots constituted by sensors and actuators, one robot could also be considered like a Wireless Sensor Network (WSN). This way in section 2.7 it is presented Cloud Robotics that is a new paradigm aiming to solve some current robotics issues. Finally, some interesting cloud-based platforms will be presented and analysed.

2.1 Internet of Things

Internet changed the way we communicate. And as time passed by a large number of objects were connected to the Internet, giving birth to the Internet of Things , which refers to a multitude of uniquely identifiable objects (things) that are connected through the Internet [1]. For instance, fridges might be connected to the Internet and present people with information about current fresh food stock, clothes might read relevant biomedical information and alert people if any action is needed, or umbrellas might receive information from the weather service and glow when it is about to rain so people do not forget to take them. The concept behind IoT is "*the pervasive presence around us of a variety of things or objects – such as Radio-Frequency Identification (RFID) tags, sensors, actuators, mobile phones, etc. – which through unique addressing schemes are able to interact with each other and cooperate with their neighbours*" [2]. The IoT paradigm is therefore changing, and will continue to change, the lives of people worldwide, whether its effects are obvious to the user or not. Application areas are various, spanning from home automation, assisted living, e-health, smart energy management, logistics, automation, etc.

According to a joint study by GSMA and Machine Research[4], there are 9 billion connected devices in the world today. By 2020, there will be 24 billion and over half of them will be non-mobile devices such as household appliances. The importance of the IoT paradigm [3] [4] has already been recognized worldwide. For instance, the U.S. National Intelligence Council [4] lists the IoT among the six technologies that may impact U.S. national power by 2025.

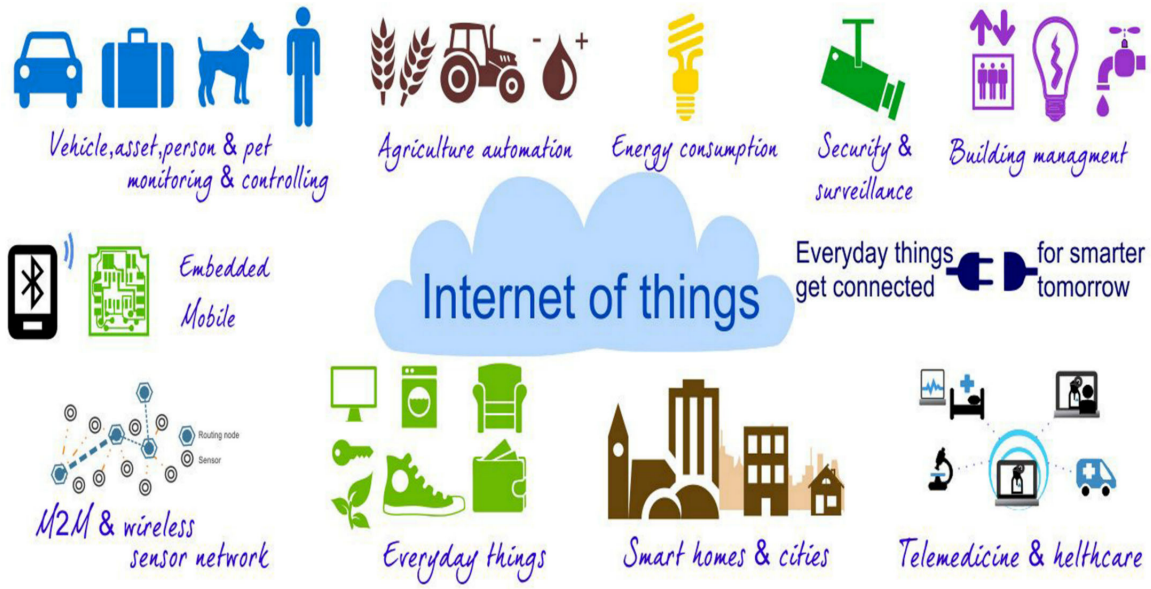

Figure 2.1: Internet Of Things (some of possible connected applications)

Some studies made by GSM and Machine Research have concluded that in 2020 the top ten connected applications are (see figure 2.1):

Connected Car Vehicular platforms behind today's satellite navigation or stolen vehicle recovery devices.

Clinical Remote Monitoring Reducing healthcare costs by remote monitoring and immediate, local corrective actions, for instance.

Security Remote Monitoring Using wide-area connected security systems for providing higher quality and cheaper solutions.

Assisted Living Enabling better quality of life and independence for senior and infirm people, as well as improved health tracker systems.

Pay-As-You-Drive Car Insurance Insurance policy that charges users according to driving behaviour.

New Business Models for Car Usage Exploiting car connectivity for providing value-added services such as vehicle tracking and remote access.

Smart Meters Number of smart meters deployed globally is expected to increase by an order of magnitude. Benefits include remote meter reading and a reduction in fraud and theft.

Traffic Management Cities traffic management includes connected road signs and tolling for improving the efficiency of traffic flow.

Electric Vehicle Charging (EVC) EVC as a **connected** activity, for locating, booking and paying for vehicle charging.

Building Automation Energy savings, better quality of life by intelligent usage of domestic appliances to ensure optimum use of energy.

IoT enables a state of ubiquitous networking in which every object in our daily lives could be a potential connected thing on the Internet, with its own set of sensors, actuators and, possibly, these could even have a sense of purpose and intelligence to react to their surroundings, which leads us to the IoIT. But before diving into the IoIT, lets describe some of the challenges that the IoT still has yet to overcome [3], such as:

Connectivity the still on going deployment of IPv6, which has not reached most end users, and the fact that the range of available IPv4 addresses is clearly insufficient. In addition, the possibility of having billions of things uniquely addressable over the Internet, presents a challenge to the goal of ubiquitous computing, which IPv6 should solve with the advantages of allowing easier network management and improved security features.

Energy management of sensors and actuators although there has been large improvements in this aspect with the appearance of technologies that allow to recharge batteries without the need of an external power source – such as producing energy through bodily movements and through the usage of environmental energies. Improvements in this area are even more evident in the new paradigm since these objects need to have computing capabilities to enable them to take decisions and act accordingly.

Creation and usage of standards for the various systems that are connected by this paradigm. This will enable IoT to evolve so that networks and various sensors become integrated and standardized.

Turning connected objects into robots requires small, low-weight and cheap sensors, actuators and processing hardware as components for robots that exhibit complex behaviours, integrating mechanisms for social interactions, autonomous navigation and object analysis, among others.

2.2 Internet of Intelligent Things

New paradigms for the Internet of Things are therefore crucial for migrating from nowadays sensor networks into networks of intelligent sensors enabled with actuation mechanisms. Such future networks will consist of the ”Internet of Intelligent Things”. This paradigm is the next step in the evolution of networking, for creating the experience of an ubiquitous [5], and intelligent, living, internet. It derives from the need to give commonplace objects the ability to comprehend their surroundings and to make decisions autonomously. Hence, decisions do not need to be sent to core decision making nodes, by giving intelligence to sensors and giving them the ability to act according to the stimulus perceived by sensors, enabling the IoIT to respond better to time-critical situations because the decisions are made in a non-centralized way [7].

The best way to be clear what is IoIT and how IoIT can help the improve of people lives is an example of scenario [28].Imagine a coffee mug mat that is connected to the IoT and has connected to it a few sensors (heat sensor and a pressure sensor) with them we imagine this mat doing things such as remembering the user's preferred coffee temperature or knowing if the coffee mug is empty or not. But why would the coffee mug need to know these things? Imagine, if the user dislikes really hot coffee the mat – being a smart object – can warn the user (through an acoustic signal or other) that the coffee is too hot for them, or if the mug has been empty for too long the mat can remember the user to put it in the washing machine.

2.3 Machine-to-Machine

One of the main concepts behind IoT is Machine-to-Machine communication. M2M communication is a definition of rules and relationships between computers, embedded processors, smart sensors actuators and mobile device without human interaction [10]. The main idea is to connect different devices that communicate through different technologies (for example Bluetooth to transfer data).

There are four types of communication between devices. First we have the internal type that occurs when two M2M devices are in the same private address. When we have a communication between two M2M devices without an intermediary is called direct, if there is a intermediary is called indirect. Then when one of the two devices use public address and the other has private address the communication between them is indirect and external. Finally when both M2M devices use public addresses their communication is external and direct. Current wireless networks are optimally designed for human-to-human (H2H) communications, so M2M to be reliable, scalable, secure and manageable like H2H it is necessary to solve some challenges [8].

Some of those challenges are unique to M2M Communication. For example M2M network needs to support a larger number of devices and these devices access the network frequently to transmit small bursts of data. At the moment some standard groups, such as IEEE, 3GPP and ETSI, have become more active in seeking a solution to this problem due to market demands.

The most notable application areas of M2M are security, transportation, e-health and manufacturing.

M2M communication has expanded beyond one-to-one communication and changed into a system of networks that transmits data to personal appliances [9].

2.4 Wireless Sensor and Actuator Networks

The advances in computing, communication and sensing technologies are prompting to a new generation of sensor network the wireless sensor and actuator networks [11] [12].

WSAN is a distributed system of sensor nodes and actuator nodes that are interconnected over wireless link. The three essential components in a WSAN are: sensors, actuators and base stations. The base station is responsible for monitoring and managing the network through communication with sensor and actuator. Sensor gets information over physical world, next the sensor transmits the collect information to the actuator. These nodes are low-cost, low-power small devices and equipped with limited sensing data processing and wireless communication capabilities. The actuator receives the information through single-hop or multi-hop and performs some actions to change the behaviour of the physical system. Comparable to sensors actuators nodes these have stronger computation and communication power and longer battery life because of more energy consumption. But resource constraints apply to both [13]. The nodes of WSAN can be stationary or mobile.

WSN and WSAN share some common consideration such as reliability connectivity scalability and energy efficiently but the coexistence of sensors and actuators cause difference between WSAN and WSN. The primary concern in WSN it is power consumption however in some WSAN is more important reliable communication [14]. There are characteristics of WSAN that are unique like real time requirement and coordination.

Real time requirement The response to the sensor input may need to be fast depending on the application. For example, in a fire application, actions should be initiated on the event area as soon as possible. The collected and delivered information must be valid in the moment of the acting. For example, if the sensors detect a problem in one area and transmit this information to the actuator this problem must be in the same area when the actuator start acting. From what was said earlier the issue of real time is very import.

Coordination In WSN the central entity executes functions of data collection and coordinator. While in WSAN what happens is a coordination. Sensor-actor coordination provides the transmission of event features. In order to make the right decisions to perform an action, the actuators need to coordinate with each other when receiving the event information.

There are some challenges for this emerging sensor network. Wireless channels have adverse properties such as path loss, multi-path fading, adjacent channel interference, Doppler shifts and half duplex operations. These are known to be unpredictable and inherently unreliable. In the case of lower power

communication and in the presence of node mobility it is much noticed [15].With this characteristic previously mentioned it is very difficult to ensure the quality of service (QOS) of the network. The difficulty to ensure QOS results in time-varying delay and packets loss as regards the application control. All of this could significantly degrade the control performance or even cause system instability.

The applications controls are being revolutionized with the advent of WSAN. WSAN can become the backbone of many control applications enabling an unprecedented degree of distributed control. The use of WSAN in control application brings many advantages such as more flexible installation and maintenance, fully mobile operation and monitoring and control of equipments in hazardous and previously difficulty to access environments and the most important low costs [16]. There are two types of architectures the automated architecture and the semi-automated architecture [11].In an automated architecture there is no controlling entity. The controllers are inside the actuators and control algorithms that decide what actions to make will be executed on the actuator nodes. In the other architecture (semi-automated architecture) one or more controller entities exist. The controller entities are embedded in the base stations or separated nodes equipped with computation and communication capacities. In these architectures sensor collected information and the controller execute certain control algorithms to decide what action to make and send to the actuator.

2.5 Cloud Computing

Cloud Computing is fundamentally an evolution of distributed systems technologies such as Cluster Computing and Grid Computing, relying heavily on the virtualization of resources. Before mentioning the characteristics, elements, concepts and other important things of cloud it is important to define cloud computing. According to the National Institute of Standards and Technology the definition of Cloud computing [34] is as follow:

> *Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction*
>
> NIST (National Institute of Standards and Technology)

The main goal of cloud computing is to make a better use of distributed resources, combine them to achieve higher throughput and to be able to solve large scale computation problems [33]. Cloud computing has 7 major components: application, client, infrastructure, platform, service, storage, and processing power.

In the definition of cloud computing previously mentioned there are some important characteristics such as Broad Network Access, Resource Pooling, On-Demand Self Service, Rapid Elasticity and Measured Service, as described by previous research work[34] [35].

Cloud computing services are categorized into three service models [36] (see figure 2.2):

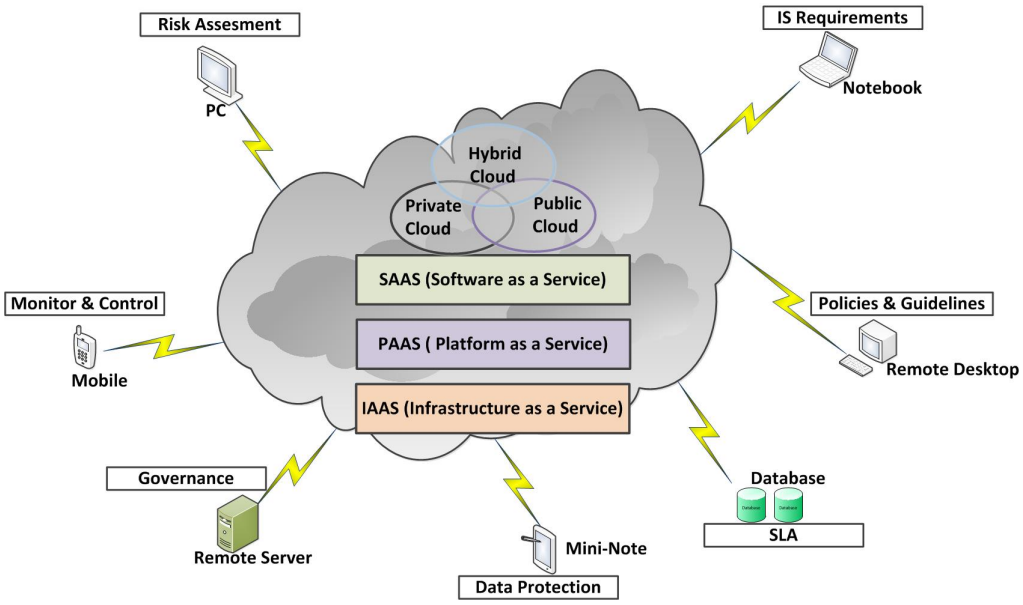

Figure 2.2: Cloud computing map

Infrastructure as a Service (IaaS) IaaS offers computing resources such as processing and storage which can be obtained as a service.

Platform as a Service (PaaS) Independently of lower layers issues, software developers write their applications according to a given set of platform specifications, independently of lower layer infrastructure issues.

Software as a Service (SaaS) Cloud consumers send their application to a hosting environment in the cloud.

Data storage as a Service(DaaS) DaaS is a special type of IaaS in which storage is provided as a separate cloud service using virtualized resources.

Cloud computing can also be classify in four ways according to its structure model: Public Cloud, Private Cloud, Hybrid Cloud and Community Cloud [36] (see figure 2.2).

The utilization of cloud computing can bring some advantages to its clients (organization, simple clients, etc.) and to devices that use the cloud such as easy management, cost reduction, uninterrupted service, disaster management, green computing as detailed in [33].

However, cloud computing still offers some challenges, namely concerning security, costing model, charging model or service level agreement [35].

2.6 M2M communications in cloud

M2M refers to technologies that empower devices with the ability to communicate with each other [17] [18]. In M2M, sensors are often employed to capture events which are relayed through a network to an application that translates them into meaningful information. With the IoT, M2M has expanded beyond one-to-one communication and changed into a system of networks that transmits data to personal appliances.

Sensor Commons is a future state whereby data will be available in real-time, from a multitude of sensors that are relatively similar in design and method of data acquisition. This allows comparing data across multiple sensors (important for cooperative sensing). Devices do not need to be all calibrated together however, one of the devices should ensure that all are more or less recording the same with similar levels of precision. In order to be able to store, process and display the information gathered by the multitude of sensors that build a M2M system, significant computing power is required. With the emergence of Cloud Computing this problem can be more easily addressed as described hereafter.

Indeed, Cloud Computing allows for data collection, processing, interface and control to be separated and distributed to the most appropriate resources. Current M2M implementations combine the above mentioned aspects into one solution, either chips in sensor bodies or an on-site laptop or desktop PC tied within a local network perform the data processing and determine the control logic.

However, once data moves to the Cloud most current limitations disappear [19][20] [21]. Storing data in the cloud means that, from a storage point of view, data buffers within devices no longer matter. Cloud storage is near limitless and so historic data can be saved for as long as it is considered valuable. Applications are also not restricted by small processors, low-power consumption, and special purpose programming languages. In addition, there is not only access to super-fast processors [22] [23] [24], but if there are server or storage bottlenecks, these can be quickly addressed by launching on-demand more servers or horizontally scaling storage. Lastly, interfaces can move to Web browsers, mobile phones, wallboards, and tablets, eliminating the need for having screens as a part of every device. Separating the processing data input from display not only means lower costs (less components to the devices) but also easier upgrading of diagnostics and far better visualization capabilities. Hence, separating the components and putting the data in the Cloud allows devices to get smarter while not necessarily becoming more complex.

2.7 Cloud Robotics

The revolution that has occurred in the past years in sensor and actuator technology is making much easier to built WSAN. This WSAN consists in a set of nodes (sensors and actuators) that cooperate among them to achieve the goal of collecting data and make some decisions based on collected data. A node of the network autonomously has a short range or their computation power is weak but when used collectively they are effective over large areas and with a biggest computing power. Nowadays WSAN growth is more pronounced and it is not unreasonable to expect that in a few years our lives

become to be a little dependent of WSAN in certain areas such as environmental, medical, transportation entertainment and city management. Although there has been an evolution of the nodes in WSAN these continue to have limited battery, limited computation power, etc. And with these problems the network node can crash due to lack of sufficient resources to perform, and therefore jeopardizes the smooth operation of the infrastructure. So sensors and actuators network cannot operate as stand-alone networks.

But there must be an efficient way for the captured data to be stored and manipulated. Cloud computing may offer attractive solutions for these issues. Indeed, it allows the reduction of the initial costs associated with the computation infrastructure. Another relevant aspect is that the cloud computing resources are easily and automatically adjustable according to the real necessities of infrastructure. This way, the computational resources are easily scalable following the growth of the infrastructure. Another important point is related with the fact that the customer only pays for the cloud resources that he actually used, therefore he does not have the problem of paying for resources that were not used.

Robotics has faced strong barriers in its evolution because there are inherent challenges to it that until recently technologies were unable to overcome. Researchers have been able to apply robotics to controlled environments because robots in factories have fixed behaviors and simple sensory systems for pre-defined stimulus. But for robotics to reach a level of pervasiveness in which robots are present in our daily lives, robots need to adapt themselves to environments that can change very frequently and need to be able to respond to unexpected events. Robots to reach this level of adaptation need to analyze their surroundings and to store information about it and to process this information [25]. Robots however usually do not have enough computational and battery power to process that information while still being mobile and with a small form factor size.

Cloud Robotics is a new paradigm aiming to solve some of these current robotics issues. Cloud robotics is a combination of cloud computing and robotics. Robots operating from a cloud can be more portable, less expensive and have access to more intelligence than an ordinary robot [25]; these robots could also offload CPU-heavy or energy demanding tasks to remote servers, relying on smaller and less power demeaning onboard computers [26].

Robots in a cloud configuration can perform autonomous learning, obtain knowledge and share knowledge, and even evolve. Through the robotics cloud, robots can execute collaborative tasks and provide efficient services. Robots can upload and share their acquired knowledge on the servers, which are responsible for knowledge, storage and scheduling [25].

The cloud is promising for the evolution of robotics, but it also has its short-comings. Robots rely on sensors and feedback to accomplish a task, this might not be accomplished on the cloud because cloud based applications can be slow or difficult to run because of Internet limitations [25], and robots usually have several closed feedback control loops between sensors and actuators demanding real-time action, for which network latency is an important concern.

Besides the introduction of the cloud for computing in robotics, there is a new paradigm in robotics, which is Robotics as a Service (RaaS). By using a Service Oriented Architecture (SOA), robots can publish their functionalities as services available to the clients and can extend those functionalities

by searching for services available in the cloud [27]. RaaS provides some advantages like having a layer of common services with standard interfaces, and making the invocations of the services device-independent.

RoboEarth [1] is an existing solution for Cloud Robotics. This platform lets robots collect, store and share information independently of specific robot hardware. Such capabilities enable robots to provide functions and behaviours that were not explicitly programmed, such as the ability to recognize certain objects. RoboEarth platform provides a repository in the cloud for robots and high-level components for interaction with the robots, all packaged in a Representational State Transfer (REST)-full API. It implements components for Robot Operating System (ROS) compatible operating systems, as well as components for robot-specific controllers accessible via a Hardware Abstraction Layer platform.

2.8 M2M and cloud platforms

We will now briefly present and analyze some M2M cloud platforms, such as Axeda, RoboEarth and SmartM2MPT.

Axeda Axeda[2] platform is a complete M2M data integration and application development with infrastructure delivered as a cloud-based service. This platform guarantees scalability and security needs. In addition offers a powerful development environment with flexible API.

The main characteristics of Axeda is the powerful open source API that has built-in functions to access the core of the platform that provides to the user ability to manage assets, query and historical data.

Axeda also has authentication, authorization and transportation security services. The web service communication of this platform is based on Simple Object Access Protocol (SOAP) and REST. Another important aspect is that an agent in a device can include more intelligent functions. Related with these previously mentioned aspects is the Axeda AnyDevice Service. With these services it is possible for a device to communicate with Axeda (even whenever the device protocol is unknown by Axeda) because the protocol of the device is converted to one that Axeda understands. This enables customer's partners and system integrators to easily install, configure and use M2M devices that communicate with a proprietary protocol and/or are incapable of running an Axeda Agent or the Axeda Wireless Protocol (AWP). So there exists a more vast set of M2M devices that can communicate with this platform [40].

SmartM2MPT SmartM2MPT[3] is a M2M platform that was launched by Portugal Telecom.

This platform has the purpose to improve the communications between devices to make decisions automatically, increasing this way the global efficiency of the systems and decreasing enterprises operating costs.

[1] `http://roboearth.org/` (accessed last time on 14 May 2014)
[2] `http://www.axeda.com` (accessed last time on 1 June 2013)
[3] `http://www.smartm2mpt.pt` (accessed last time on 1 June 2013)

SmartM2MPT has four basic services. The connectivity service has the functionality to manage and supervise the communications between the deployed devices in real time. The monitoring service allows the interaction and the achievement of the status of the assets in a continuously and remotely way. Metering services are used to eliminate the on-site meter reading of electricity and water. Finally the localization service is used to locate geographically the devices.

The SmatM2MPT platform can be applied in various branches of business such as industry, public administration, agriculture, construction, distribution and healthcare.

RoboEarth RoboEarth[4] [42] is a World Wide Web for robots. It is a giant network and database repository where robots can share information and learn from each other about their behaviour and their environment.

The goal of RoboEarth is to allow robotic systems to benefit from the experience of other robots. This platform lets robots collect, store and share information independently of specific robot hardware. Such capabilities, enables robots to provide functions and behaviors that were not explicitly programmed, such as the ability to recognize certain objects. The creators describe it as platform that provides a repository in the cloud for robots and high-level components for interaction with the robots, all packaged in a friendly REST-full API.

RoboEarth provides a boost in the robots machine cognition, robot-to-robot interaction, their storage and an ability to share knowledge, this also leads to a more sophisticated human-machine interaction.

2.9 Related Work

This section presents and discusses some interesting related work to the Middleware for Wireless Sensor and Actuator Networks. It is necessary to have in mind that robots can be considered a WSAN with extended functionalities. The related work that is going to be described hereafter does not satisfy completely our requirements.

2.9.1 ROS

Robot operating System [37] is an open source operating system for robots that was designed to achieve a specific set of challenges taking into account the goal for developing large-scale service robots. The philosophical goals of ROS are: Peer to Peer, Tools-based, Multi-lingual, Thin and Free and open source. These philosophical goals influence the design and implementation of ROS, as described hereafter.

Peer to Peer ROS system consists in a number of different hosts connected at runtime in a peer to peer topology (figure 2.3). Peer to Peer connectivity combined with buffering or “fanout” software modules is used to avoid unnecessary traffic flowing across the wireless link that occurs in central server.

[4] `http://http://www.roboearth.org/` (accessed last time on 1 June 2013)

Multi lingual because many people have their preferred programming language. For these reasons ROS supports four languages C++, Python, Octave and LISP.

Tools-based ROS has a microkernel instead of a monolithic development and runtime environment. In this microkernel a large number of small tools are used to build and run ROS components.

Thin Most drivers and algorithms could be used in other projects, but some code has become so entangled with the middleware that it is difficult to extract. To solve this problem ROS encourage all drivers and algorithm developers to write standalone libraries without dependencies on ROS. This is achieved by placing virtually all complexity in libraries and only creating small executables.

The fundamental concepts of ROS implementation are: node, messages, topics, and services. Nodes are processes that perform computation. ROS is typically comprised of many nodes. The nodes communicate with each other by passing messages. A message is a typed data structure and can be composed of other messages and array of other messages. A node sends a message by publishing it to a given topic. A node that is interested in a specific date will subscribe. In general publishers and subscribers are not aware of each other. Publish-subscribe mode is a flexible communication paradigm but broadcast routing scheme is not appropriate for synchronous transactions. To treat this issue ROS has services. A service is composed by name and a pair of messages: one for the request and the other for the response.

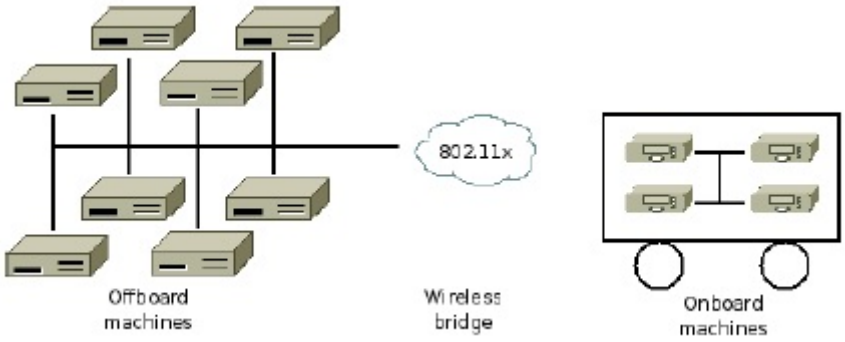

Figure 2.3: ROS Network Configuration

2.9.2 YARP

Yet Another Robot Platform [38] is an open-source project to reduce the effort in development of a infrastructure-level software. It facilitates code reuse, modularity and so maximizes research-level development and collaboration. The heart of YARP is a communication mechanism to make writing and running such processes as easy as possible. YARP enables the write of processes that are location independent and that can run on different machines without any changes in the code. This way it is possible to move the process between machines that are in a cluster to redistribute the computational load

and to recover from hardware failure. Automatically allocating processes are ensured by the developer.

The addition of new components can interfere with the existing one in YARP infrastructures. But this problem is alleviated allowing the inclusion of more processors to the network. The human cost of stopping and restarting a process can be very high in robotics. These high levels are related with the dependency that exists between the processes. YARP minimizes the dependencies between processes. When a process is killed or dies this does not require processes to which it connects to be restarted. And the communication channels between existing processes can continue without process restart.

For reducing dependencies with the operating system, YARP uses Adaptive Communication Environment (ACE), an open source library providing a framework for concurrent programming across many operating systems. Communication in YARP uses the Observer pattern. The state of special Port object is delivered to any number of observers (in any processes). To manage these connections YARP insulates the observed from the observer and the observees from each other. In YARP a port is an active object managing multiple connections. Each connection has state that is changed by external commands. YARP uses many different communication protocols such as Transmission Control Protocol (TCP), User Datagram Protocol (UDP), multicast, etc. The ports can be connected programmatically or at runtime. The communication is asynchronous and as such messages are not guaranteed to be delivered unless a special provision occurs.

2.9.3 PEIS Kernal

Physically Embedded Intelligent Systems Kernel [31] is a middleware built employing the concept of Ecology of Physically Embedded Intelligent Systems. The PEIS Kernel provides a shared memory model, a simple dynamic model for self-configuration and introspection, and supports heterogeneous devices.

The goal of this middleware is to provide a common communication and cooperation model that can be shared among multiple robotic devices. Any robot device that has a software control in the environment is considered a PEIS.

A PEIS is a set of inter-connected software components developed to control sensors or actuators. All of these devices are connected by a uniform communication layer. This type of communication allows the exchange of information between the PEIS devices, and it also allows dynamic joining or leaving. Using a uniform cooperated model allows comparison between all of PEIS devices. Devices that participate in the cooperative model can use functionalities of other PEIS devices to complete their own functionalities.

The layer structure is divided into three layers: the communication layer, peer to peer network layer and Tuple layer (figure 2.4). The communication layer at the lowermost abstraction is used to provide potential communication links and device detection for shared medias. This layer also provides services for initializations, calling functionalities, etc. This communication layer also serves as a bridge translating message to a more compact protocol suitable on the low-bandwidth network.

On top of the communication layer is the peer to peer network layer that uses P2P algorithms for

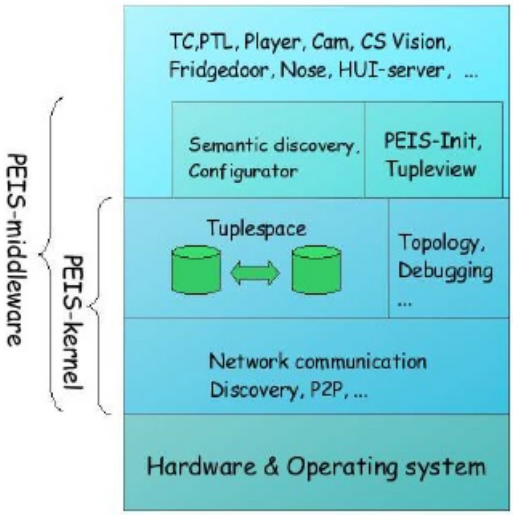

Figure 2.4: The applications tack used for individual PEIS components

optimizing connectivity and performing routing.

Finally the Tuple layer is on top of the peer to peer network layer. In this layer functionalities are implemented for maintaining the distributed tuplespace and the API functionalities used by component programs. Tuplespace is a decentralized version of the shared memory where a number of tuples containing keys and other data can be stored and retrieved by any participating process using an abstract tuple. In PEIS version of this space a tuple has been specified to consist of a namespace, key, data as well as a number of meta attributes such as timestamps and expiration date.

The database implementation to store tuples assumes that each PEIS component can be used to store all relevant tuples. This database gives some special attention to an abstract tuple, a tuple in which one or more fields have been initialized to a wild-card value while the remaining fields have been given concrete value. This abstract tuple is used for three reasons. The first one is they are used whenever an application is accessing the database to query the current value of a tuple. Other reason is before a PEIS can access tuples at a remote location, a subscription to the corresponding tuples must be made. And third reason is abstract tuples are used by the event mechanism to setup callbacks when tuples changes value.

2.9.4 Player/Stage

Player/Stage system is a middleware platform for mobile robotics applications that guarantees an infrastructure, drives and algorithms [28] [29]. The main features of this middleware are the platform-programming language, transport protocol-independence, open source, and modularity.

Main components of this middleware are the player and the stage. The component player is a device repository server where we can find robots sensors and actuators. Each one of these devices have an interface and a driver. The interface is used by the client of this middleware to obtain information collect by the sensor to control the actuator.

The algorithms implemented by the drivers can receive data from other devices, process the received

data and then send it back. Other thing the drivers can do is to create arbitrary data when needed. The other component (stage) is a graphical simulator that models devices in a user defined environment.

This system has an architecture with three tiers. In the first tier the clients are software developers for specific robot application. The player which provides common interfaces for different robots and device are the second tier. The third tier is the robots, sensors, and actuators.

Different programming languages like C,C++, Java, and Python are used to access services. Client side libraries are in form of proxy objects. Clients can connect to the Player platform to access data, send commands, or request configuration changes to an existing device in the repository.

2.9.5 MARIE (Mobile and Autonomous Robotics Integration Environment)

Mobile and Autonomous Robotics Integration Environment is a middleware that was made for developing and integrating new and legacy robotic software [32]. MARIE is a flexible middleware, which allows sharing among developers, reuse of code and integration of different robotic software.

The main characteristics of MARIE are interoperability and reusability of robotic application components.

The architecture of the MARIE middleware is divided in three layers which are the following: Core, Component and Application. The core layer is where we can find the services for communication, low-level operating functions and finally the distributed computing functions.

The second layer (component) is the layer that is used to add components that are going to be constantly used by services and to support domain specific concepts.

Finally the application layer has some services and tools that are going to be very useful to build and manage the integrated application. One of the most important aspects of this middleware is its flexibility. This is visible for the middleware to provide some services that allow the adaptation of different communication protocols and applications.

MARIE uses the Adaptive Communication Environment (ACE) communication framework. This framework allows a variety of software components to connect to MARIE using a centralized component. Apart from the centralized component, there exists four functional components that are: application adapters, communication adapters, communication managers, and application managers. The application adapter behaves as proxy between the central component and the application. The goal of communication adapters is to translate the data exchange between application adapters. The connections are created and managed by the communication managers. Finally, application managers instantiate and manage components locally or across distributed processing nodes. MARIE also provides mediator interoperability layers among adapters and managers.

2.9.6 RoboEarth Cloud Engine (Rapyuta)

Rapyuta is cloud robotic platform for robots that implements a platform as a Service (PaaS) framework [30]. This framework is open source and is built upon a clone based model.

This clone based model provides secured customizable computing environment (clone) in the cloud. This way the robots can receive help in heavy computation. The robots connect to the Rapyuta and can start the computing environment by their own initiative, launch any computational node uploaded by the developer, and communicate with the launched nodes using the WebSockets protocol. The use of WebSockets protocol provides a full duplex communication channel between the robot and the cloud with predefined messages. The computing environments that are started by the robots have high bandwidth connection to the RoboEarth repository. Thus, the robots are allowed to process their data inside the computational environment in the cloud without the downloading and local processing. Another aspect of this platform is that the computing environments are interconnected with each other.

The architecture of Rapyuta consists mainly of four elements: the computing environment, the communication protocols, the core tasks and the command data structure. The computing environments are built with Linux Containers. These containers provide isolation of processes and system resources within a single host, and they allow the applications to run at native speed because they do not emulate hardware. Linux containers allow easy configuration of disk, memory limits, I/O rate limits and Central processing unit (CPU) quotas. Thus it is possible to enable one environment to be scaled up to fit the biggest machine instance of the IaaS provider or scaled down to just relay data to the Hadoop backend.

The computing environment has to run any process that is a ROS node. All processes within a single environment communicate between them using ROS interprocess communication. The communication protocols of Rapyuta are divided in three parts: internal communication protocol external communication protocol and the communication between the Rapyuta and the applications that are running inside the Linux container. The internal communication protocol is the protocol that covers all the communication between the processes of Rapyuta. The external module has the goal to define the data sent between the physical robot and the cloud. The core task has four task sets: master, robot, environment and container. The master is the task controller that monitors and maintains the data command structure. The robot is defined by the capabilities necessary to communicate with the computing environment. Finally the container is defined by the capabilities necessary to start/stop computing environment.

Finally Rapyuta is organized in a centralized command data structure with four components. The network is the most complex of the four. These components are used to organize the communication protocols and to provide abstraction to all platform. The next component is the user that is the group of humans that has one or more robots that are going to be connected to the cloud. The authentication of these robots is made by API key that is unique to each user. To manage the robots that are running in the computing environment, there exists the loadBalancer. Finally the distributor has the functionality to distribute incoming connections from robots over the available robots.

2.10 Home automation and Sensing Platforms

The usage of IoT technology in Home automation and Ambient Intelligence environments is a powerful means to interconnect objects and materials, supported by wireless network solutions enabling the automatic acquisition of such data. We will first give a brief overview of IoT Platforms and equipment

that is already available that can be employed as a starting point in IoIT implementation, presenting their comparative evaluation.

There are platforms that enable the creation of IoIT projects and extensions of these, such as Social Internet of Things and Robot as a Service projects

Cosm [48] (formerly Pachube) that lets sensors and other equipment post and read data to feeds – much like twitter works for people – which allows them to trade messages and take action. Other Home Automation platforms include openRemote [49] and openHAB [50]. These solutions allow users to connect virtually any device and have the platform take action when a command is given or when a condition is met. Ninja Blocks [30] provides a set of open source hardware parts for people to create their own custom devices that connect to the platform, whereas openHAB and openRemote use more standard hardware platforms like Arduino.

Platforms like IFTTT [52] and SmartThings [53] are platforms that are more oriented towards defining intelligent behaviour from devices, like informing the thermostat that the users are close to their home, through the GPS in their smartphone, and thus turning on the Air Conditioning.

Funf [54], developed at MIT Media Lab, is another open source sensing and data processing framework, now a commercial product. Funf consists on software modules (Probes) that act as controllers and data collectors for each sensor's data. It also allows intelligent processing of the personal captured data (e.g. monitoring a person's physical activity by an activity monitor probe that already incorporates motion logic over the accelerometer sensor, and sharing such data on the network). With respect to backend storage, it allows saving data on a remote backend (e.g. on a cloud) or via Wireless Sensor Networks.

SenseWeb [55] is a large-scale ubiquitous sensing platform, aimed at indexing globally sensor readings. Its open-source layered modular architecture allows to register sensors or sensor data repositories, using web-based sensing middleware.

The following table 2.1 presents a comparative overview for these platforms.

2.11 Discussion

All the mentioned middleware solves problems that this work also tries to solve such as: flexibility (MARIE), code reuse, modularity (YARP and ROS) and even share information with other devices (player/stage and PEIS Kernal). However, some of these middleware are constrained by the limited capacity of the device (actuators sensors robots) where they are executed. As is the case of the player/Stage middleware that solves some problems that our proposed solution tries to solve, but brings other problems, like when one device sends some data to other device and the device that receives the information does not have the capabilities to run at the moment or even never. This way the device will perform the action even if he has no capacity for such.

Contrary to the other middleware, Rapyuta middleware can solve the problem inherent to the limitations of the hardware by running some algorithms on the cloud platform. But in this the middleware does not exists an algorithm that decides when is necessary to run some code in the cloud or in device

Platforms	Supported OS	Programming Languages	Communication Protocols	Hardware	License
Cosm	Multiplatform	javascript, ruby	HTTP (RESTfull API)	Any Hardware	Proprietary
OpenHAB	Multiplatform (JVM)	Java-OSGI	Any protocol (depends on bindings	Any Hardware	Open Source (GPLv3)
Ninja Blocks	Multiplatform (the blocks run Ubuntu)	Ruby, Node.js, PHP, Python	HTTP (RESTfull API), Any protocol	NinjaBlocks (OpenSource)	Proprietary, OpenSource hardware
openRemote	Multiplatform	Java	HTTP (RESTfull API)	Any Hardware	Open Source
SmartThings	iOS, Android	SMART	HTTP (RESTfull API)	Only supported hardware	Proprietary
IFTTT	Web App	Non - programable	HTTP	No hardware required	Proprietary

Table 2.1: Home Automation platforms comparison

(robot, actuator sensor).

Chapter 3

Proposed Solution

Based on the concepts presented in section 2, this section presents the proposed architecture as well as its detailed description. The described solution is to be applied in a flexible, multi-platform middleware for wireless sensor and actuator networks. This solution intends to distribute execution flows between mobile devices and the cloud, according to the battery status of the devices, the network performance and the device itself (security aspects are left outside this thesis scope).

The architecture can be divided into four relevant layers (figure 3.2). The first one is the hardware layer that is related with WSAN and robots. The second is the middleware layer. This layer aims to mediate between the hardware and the application. The third layer is the application layer that allows the user interaction with the system. Finally the last layer of this architecture is the cloud computing system. This platform will be responsible to store some information and process the data collected by devices that do not have sufficient resources to perform the operation.

These four parts will be described in detail in the following subsections. Hereafter some important considerations about the target environment of this solution will be stated.

3.1 Environment

This architecture does not have a specific real deployment scenario, so it was designed to be a generic deployment. Therefore, the specific technical details of the solution may be adjustable according to the characteristics of each particular application case. Despite the existence of these variations, some environmental requirements were defined:

- The device has to perform all the operations by itself, even in situations of critical point, because it does not have a good network connection to send data to cloud infrastructure or it is out of range for finding a device, or the discovered device also does not have enough resources to do the requested operation.
- The device does not have enough resources to perform the operation, like lack of CPU or lack of battery. **But the network connection is also not good enough to send data to cloud infrastructure.** When this occurs the device searches for others devices that have sufficient resources.

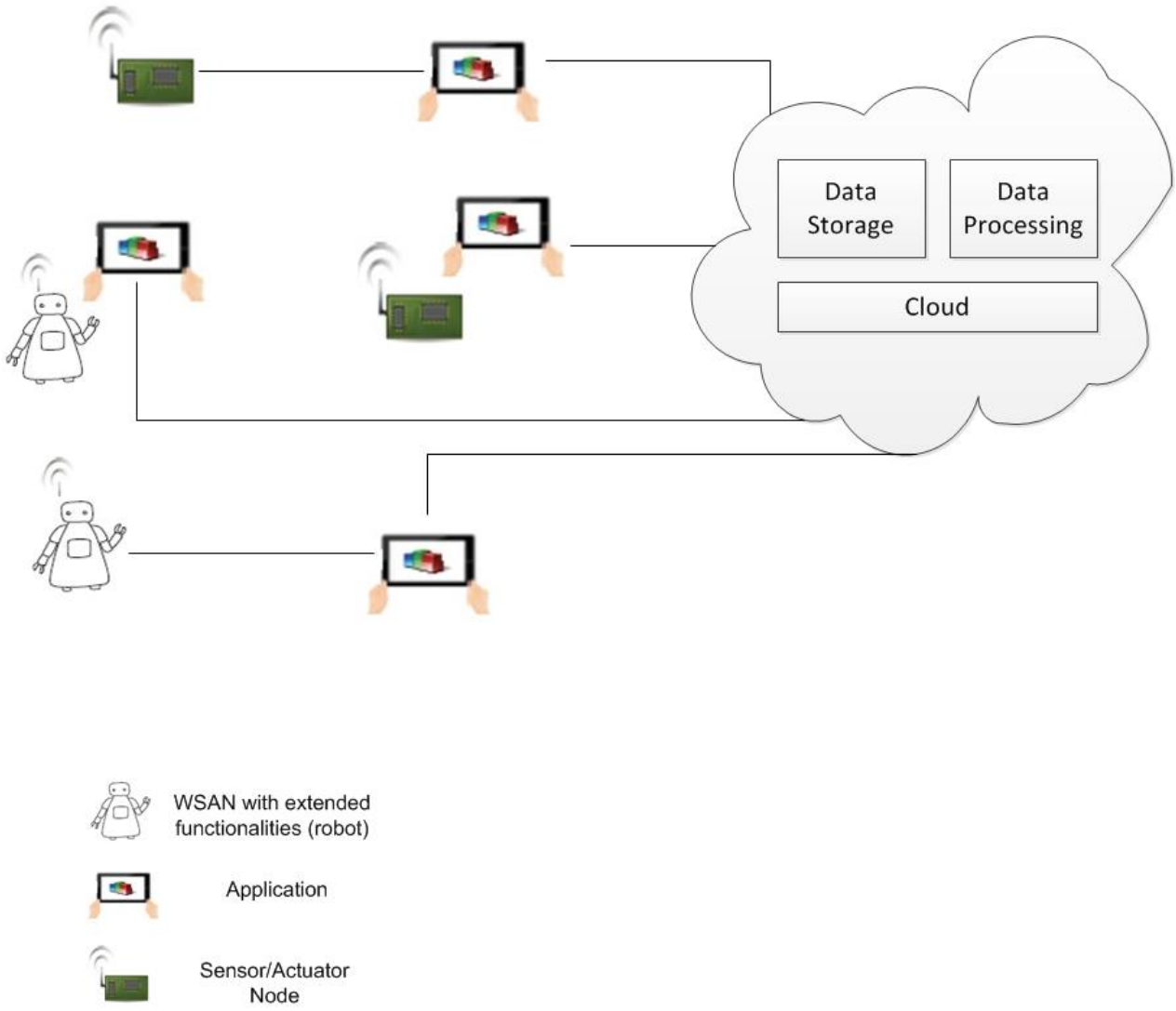

Figure 3.1: Proposed Architecture

- Whenever the device does not have sufficient resources to do an operation and the network connection is good, the data will be sent to the cloud infrastructure. This way the operation that the device cannot do is execute at the cloud. After the requested operation finishes the result is sent back to the device that has requested the cloud to do this job.

3.2 Hardware

The hardware layer of the architecture (see Figures 3.2 and 3.1) is the WSAN infrastructure or the robot hardware (such as the robots produced by Ydreams Robotics [1] have produced) . This infrastructure is composed by three types of nodes: actuator, processing, and sensor. The sensor collects information from the physical world. Actuators receive the information from the sensor or from the middleware, and perform some actions. The processing unit is the device that runs the application that controls the

[1] `http://http://http://www.ydreamsrobotics.com//` (accessed last time on 1 June 2013)

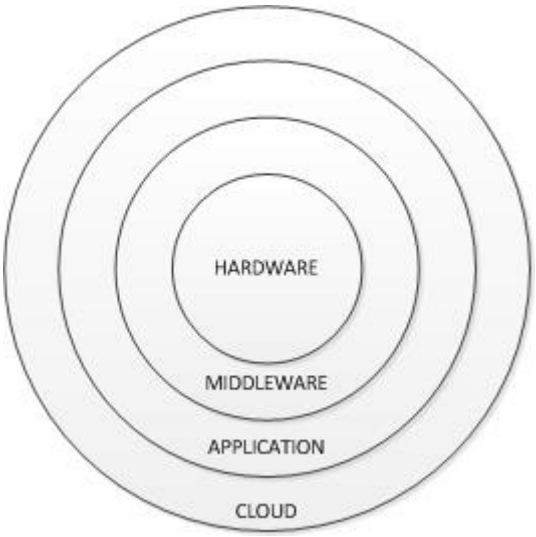

Figure 3.2: Architecture

hardware, collects information from the hardware and was created by the programmer.

Although processing units have stronger computation and communication power, and hence need longer battery life than actuators and sensors, in many situations they make the energy bottleneck. Because of the battery, the computation power is limited at the nodes, and these may start to fail due to lack of energy resources.

3.3 Middleware

The Middleware runs inside the application (is integrated with the programmer application) and can be considered as an extension of the operating system which provides a transparent communication layer between the hardware and the applications. The objective of this middleware is to solve some problems that a modular design has, such as interoperability and communication configuration. To achieve the objectives mentioned above the proposed middleware needs the following resources:

Simplifying the development process application development is not easy for the robotic environment because each robot manufacturer has its own API. Middleware should simplify the development process by providing higher-level abstractions with simplified interfaces that can be used by developers.

Support communications and interoperability The robotic and WSAN modules are designed and implemented by different manufacturers. The middleware must provide functions that help to have an efficient communication and simple interoperability mechanisms between these modules.

Provide efficient utilization of available resources The device (single sensor, single actuator, robot) may have single or multiple microprocessors, one or more interconnection network and it needs to execute intensive task in real time. Therefore efficient utilization of the resources is needed. Middleware helps in efficiently utilizing these resources [39] [41].

Providing heterogeneity abstraction The communication and cooperation between hardware and software is very important. To hide the complexity of the communication level and the heterogeneity of the modules the middleware is used as a collaboration software layer

Supporting integration with other systems A device (single sensor, single actuator, robot) needs to interact with other devices to help each other to achieve their goals in real time. Therefore the middleware should provide real time interaction services with other systems.

Supporting low resources devices devices (single sensor, single actuator, robot) may have several limitations such as limited power, small memory, limited connectivity, and so the middleware needs to have adequate functionalities to manage the resources.

Providing automatic recourse discovery The devices (single sensor, single actuator, robot) are dynamic systems due to their modularity and mobility. Therefore automatic and dynamic resource discovery and configuration are needed in the middleware.

To provide the above mentioned functionalities the middleware is divided in three layers (see Figure 3.3): the communication layer, the peer to peer network layer and manage device layer. The communication layer provides potential communication links with multi platform such as cloud or mobile device, and also provides device detection for shared information. The goals of this layer consists on providing heterogeneity abstraction and supporting communications and interoperability. The layer also provides services for initializations, calling functionalities, etc (see figure 3.3).

On top of the communication layer is the peer to peer network layer that uses algorithms that optimizes the connectivity or algorithms that performs routing. With the peer to peer network layer is possible to achieve integration with other systems and automatic discovery.

The manage device layer runs an algorithm of problems detection to detect if the device has lack of battery and if the load of CPU is too high. This it way it is possible to support low resource devices and provide an efficient utilization of available resources.

3.4 Application

These applications will be running on smartphones, tablet, WSAN nodes or in robots with different operating systems and are not written by the programmer that deploys the middleware.

The application can have these types of actions:

- obtains a textual or graphical representation of the information acquired by the device which can be stored in the device or in the cloud platform depending on certain conditions.
- has the capability to process information received from the device
- performs operations according to the results of information processing.

- performs some types of operation in a device

The middleware will be integrated with the application thereby ensuring that the application receives information about the status of the hardware components that the middleware is monitoring. Thus the integration between application and the middleware can help to avoid critical situations (examples excess CPU usage) and prevent the death of the device (for example lack of battery)

3.5 Cloud Platform

The cloud-based platform that will be used in this work can be one of the platforms referred in section 2.5, acting as a support system. It will be in charge of two relevant functions: data processing, data storage (Figure 3.3). As this solution is cloud-based, the cloud computing resources can be easily and automatically adjusted according to new application demands or as the application's requirements grow.

Data Processing The Data Processing component is in charge of processing the data that is sent by the devices. This component is going to do the heavy work that the device can not do because it does not have enough battery or simply does not have computing resources to process the collected data. And after this component obtains the result it is sent back to device. This component allows the device not to crash by executing the work that the device could not do.

Data Storage This component is in charge of storing in a persistent way the collected data from each node of the WSAN. This way, if some of the devices do not have space to store the collected data because the device memory is full, this can be transmitted to the cloud platforms, so that the data is not lost. This date is not dropped because with the help of the middleware, the application sends the data to be directly store in the cloud when the memory of device is almost full.

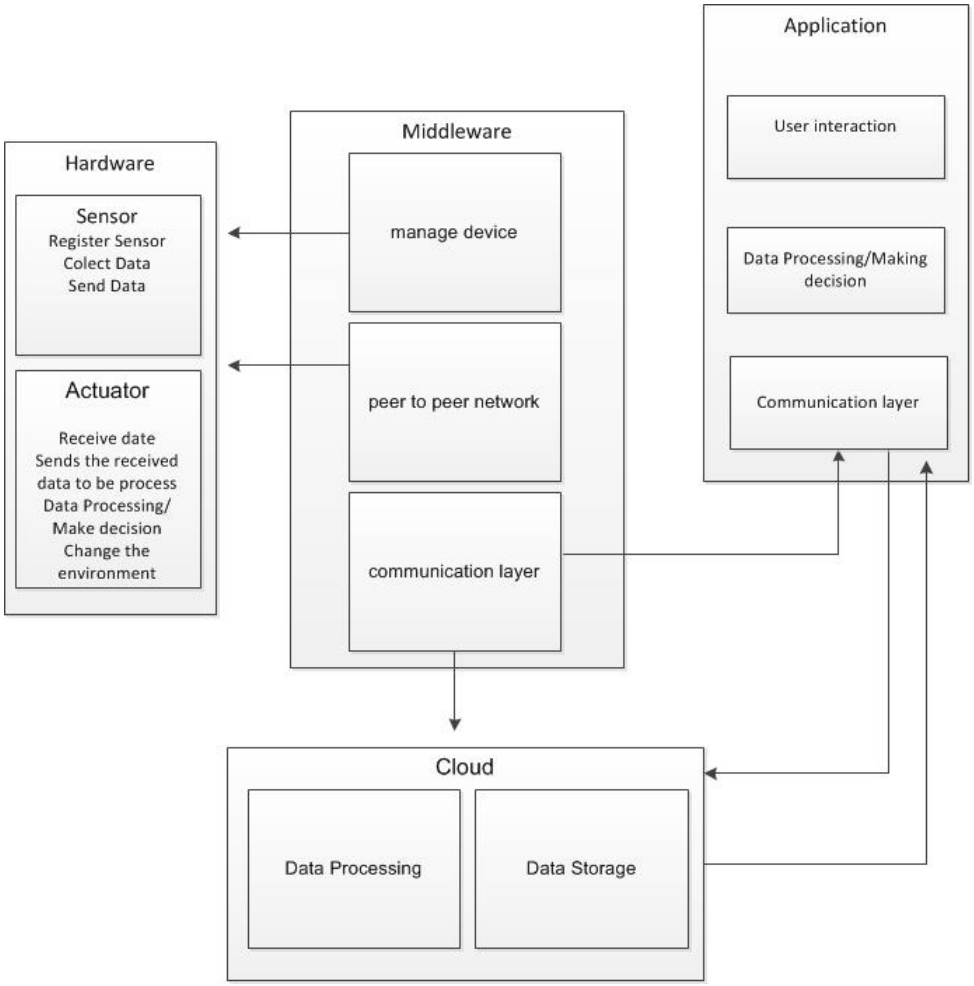

Figure 3.3: Detailed Architecture

Chapter 4

Implementation

The developed system is to resolve and minimize certain problems that the hardware components (CPU, memory, battery of devices) can bring to applications. During the system implementation, decisions aimed at overcoming the challenges that are inherent to the development of the architecture had to be made. The rationale behind these decisions will be explained, together with the different options. A key approach in the development methodology was the exploitation of existing frameworks and libraries in the proposed solution implementation process. By taking advantage of existing validated work (for example Windows Azure or IKVM), the development focus is shifted, thus effectively tackling the problems at hand. Examples of this approach are materialized by harnessing the potential of works, such as:

- Windows Azure
- ini4j.jar [1]
- IKVM [2]
- jython [46]
- openssh
- OpenHab [3]

Moreover, when facing equivalent solutions that would cover the systems requirements, preference was given to open source solutions. In addition, technological choices that would benefit system configuration flexibility and modularity were preferred, as these properties were deemed relevant. Lastly, system-level platform portability was taken as global criteria in the choice of tools to approach each of the system implementation challenges.

The objective of this work consists in building a middleware that enables applications that run on devices with limited resources to transfer their flow of execution to the cloud in order to keep working, and therefore that a device's life expectancy increases. To make this goal possible we resorted to the use of a cloud platform. This choice will be explained in the following sections.

[1] `http://ini4j.sourceforge.net/` (accessed last time on 14 May 2014)
[2] `http://www.ikvm.net/` (accessed last time on 14 May 2014)
[3] `http://www.openhab.org/`(accessed last time on 14 May 2014)

Another important point in this work is the design of a flexible and modular middleware that is agnostic to the application's programming language. Otherwise, application programmer users would not feel receptive to the use this middleware, for not supporting the language in which the application was developed.

Nowadays the "Internet of Things" is a theme in focus on the media and academia, the integration of the middleware with a platform which is regarded as the glue of "Internet Of Things" has very interesting. For that reason OpenHab, will also be described in detail.

4.1 Cloud

The cloud platform is one of the most important points of this middleware, it is with the help of cloud platform that will be solved some hardware problems, such as: revolving lack of a memory device. So, it is important to choose a cloud that meets certain requirements:

- The chosen platform fits into one of these two categories: Infrastructure as a Service or Platform as a Service. The main difference between PaaS and IaaS concerns the amount of control that users of the services exercise over the system. Total control is provided by IaaS. PaaS instead lets vendors manage everything, and thus typically provides little control to users. This translates into virtually zero administration costs for users of PaaS. IaaS, in contrast, has administration costs similar to those of a traditional computing infrastructure. Although there are these little differences between IaaS and PaaS, the choice of the model to use is not very important because it is only required that the cloud has the ability to run code created by the application programmer who uses the middleware.

- Another important aspect in choosing a good cloud solution is having a large and dynamic community support.

- To be an open and flexible platform, it also has to allow for quickly create, deploy and manage applications. In other words, it can satisfy any expected application needs.

- It also must allow the use of any language and tool.

Nowadays many of the cloud platforms already fulfill the above requirements. The best-known cloud platforms are Amazon Elastic Compute Cloud (AWS)[4], Google Cloud Platform [5] and Windows Azure [6]. Of the three platforms the chosen to be used during development and testing of the middleware was the Microsoft cloud (Windows Azure). As the three platforms fit the choice points mentioned above, the point that weighed more in choosing this cloud solution over the other two was the fact that there existed a *priori* knowledge of this cloud platform, allowing a faster learning of all application programming interface. The choice of Windows Azure does not preclude that the middleware in the future cannot contain more

[4] `http://aws.amazon.com/pt/ec2/` (accessed last time on 14 May 2014)
[5] `https://cloud.google.com/` (accessed last time on 14 May 2014)
[6] `http://azure.microsoft.com/` (accessed last time on 14 May 2014)

clouds, giving the possibility to choose which cloud the application programmer prefers and have more confidence.

4.2 Choice of the language

The programming language chosen for developing this middleware was Java. This choice was due to the fact the Java technology does not have any cost related to its use and has a satisfactory support from the Java community that in nowadays is very extensive. This fact is reflected on community events, available material, articles, journals and forum discussions. This way the communication between Java programmers is easier. Java is also a technology that enables parallel processing which for solution development, is quite important and has portability, as the same code can run on different platforms without the need for code changes and so the applications can easily be migrated across multiple machines. And there was already some experience with this programming language. Although the core of the middleware has been developed in Java that does not preclude that the application programmer (person that is going to use the middleware) cannot use the language he normally uses. To allow the middleware to be as flexible as possible, and simultaneously guaranteeing that it was not necessary to create a solution for every type of language, the following tools were used: IKVM and jython.

IKVM[7] is Open Source software that allows to directly run compiled Java code in C Sharp, transforming a jar in a dll. The IKVM has the following components: a Java Virtual Machine (JVM) implemented in .NET, implementation of Java libraries in .NET, a tool that translates Java byte code to IL NET and tools that enable interoperability between Java and .NET. Jython [46] is the successor of JPython and is an implementation of the Python programming language in Java. The Jython programs can import and use any Java class, with the exception of some standard modules. Furthermore Jython includes almost all modules of the standard Python distribution, with only some of the modules originally implemented in C.

Using these two tools, the middleware becomes more flexible because the core of the middleware can be executed on C Sharp and Python.

4.3 OpenHab

OpenHab[8] is an open source software that allows the integration of different systems and home automation technologies in one solution. This software is different because all the other systems of this area only cover certain types of equipment specified by the system manufacturer, being almost impossible to integrate new equipments. Being an open source solution that is maintained by a large community and that does not depend of a single company the problem mentioned above is solved, being this policy the driver for OpenHAB. This software is open source and is mould to the point of being able to be used outside the world of home automation and to be used in other space such as: smart gardens that responds to commands sent per visitor. So, using OpenHab is not a bad idea, considering this software

[7] `http://www.ikvm.net/` (accessed last time on 14 May 2014)
[8] `http://www.openhab.org/`(accessed last time on 14 May 2014)

as the glue of the ”Internet of things”.

The OpenHAB runtime is a set of Open Service Gateway initiative (OSGi) bundles deployed on an OSGi framework. It is a pure Java solution. Being based on OSGi, which allows adding and removing functionality during runtime without stopping the service. This open source software has two different internal communication channels: an asynchronous event bus and a stateful repository. The event bus is the base service of OpenHAB. The bundles that do not require stateful behaviour use the runtime service to inform other bundles about events and to be updated by other bundles. The main two types of events are: Commands which trigger an action or a state change of some item/device and status updates which inform about a status change of some item/device. Not all functionality can be covered purely by stateless services. Out of this reason openHAB also offers the Item Repository. This repository is connected to the Event Bus and keeps track of the current status of the items. The Item Repository can be used whenever it is necessary to be able to access the current state of items.

OpenHab software has a great potential, and the integration of a resource management middleware could make this software even more powerful. The aim of this integration is to show to show that the integration of the middleware developed is able to resolve and delay the onset of certain problems at the hardware level such as the lack of battery or memory.

4.4 Communicaton

This middleware is divided into two parts, the management of certain hardware components and the cloud component.

The communication between these two components and the application that uses middleware is guaranteed through: publish/subscribe channel, the secure shell protocol (OpenSSH) and sockets (these sockets use TCP and UDP protocols). In this client-server architecture, the communication occurs between the cloud (the server) and the programmer application in the client, being the socket the endpoints of this architecture. For this communication to be possible we must first create a Secure Shell (SSH) communication channel allowing to run commands remotely to boot the server in the cloud. As one of the goals is to satisfy most applications requirements such as speed of execution, the middleware provides two types of sockets: a TCP socket and an UDP socket.

To meet the requirements of real-time applications, the middleware provides a socket with UDP protocol. Using this protocol, the communication between the application and the cloud will be very fast although this protocol does not guarantee that packets reach their destination. Even with this restriction, the use of this protocol allows the main requirement of real-time applications (predictability) to continue to be fulfilled, even if part of the application code is running on another machine. Although UDP is a big help in situations of real-time for the reasons already given, its lossy nature may also not be very acceptable if there is necessity of the application to run in real-time, but the integrity must be maintained at all costs. For these situations, the middleware uses the TCP protocol with a mechanism that drops packets that are outdated and have not yet been processed. In the case of applications with more stringent security requirements, integrity of exchanged messages, ordering of messages, and guaranteeing the

delay time in the transmission has no inconvenience to the application, the middleware guarantees that these requirements are met by providing a socket that uses the TCP communication protocol [45].

To ensure greater efficiency in the delivery of the message between the client and the server, and vice versa, it was used JavaScript Object Notation (JSON) [47]. JSON is a text format for serializing structured data. This format supports text objects, being their dimensions reduced and their reading simplified. These characteristics are quite advantageous as they allow the speed of transmission of messages, as well as the processing speed, to be faster. To ensure that the application programmer always has access to the state of the CPU, the battery and the memory of the device, it is used the Publish/Subscribe model (see figure 4.1) in which the hardware monitoring software is the publisher and the application is the subscriber. To this end there is a Java Queue (FIFO) for each hardware component, which is shared between the hardware monitoring software and the application. These queues always keep an entry with the hardware component state, according to the middleware settings chosen by the application developer. This way it is always ensured that whenever a hardware component reaches the critical point defined by the application programmer itself, the application will be informed of this event.

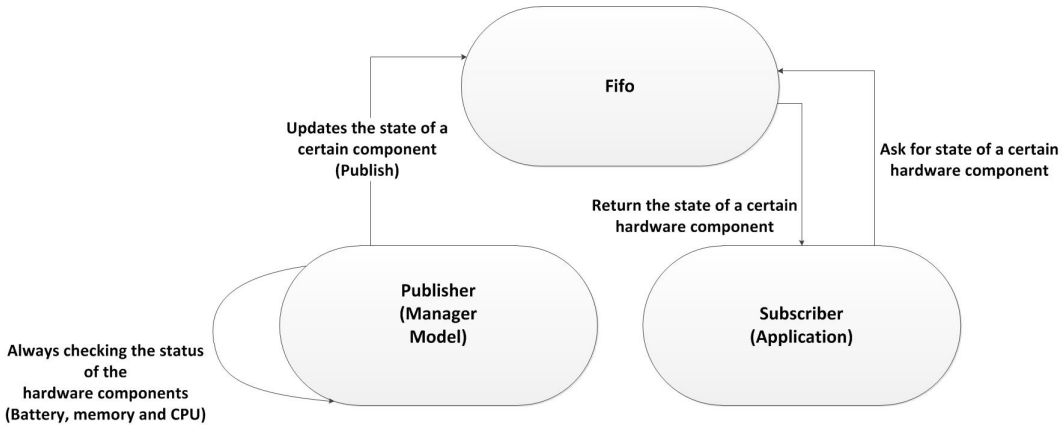

Figure 4.1: Publish-Subscribe example

The communication between application and the Microsoft cloud when it comes to data storage and downloading of information from the cloud is ensured by the Microsoft cloud API that uses the Hypertext Transfer Protocol (HTTP) REST protocol.

4.5 User configuration

To ensure that the configuration of certain points of the middleware is conforming to each application needs, and to set certain aspects of the cloud (such as the login to the machine that was obtained in the cloud), configuration files were used, employing the INI file format (ini4j.jar). This choice was due to the fact these files are simple text files, with a very basic structure that is structured in sections and properties. Furthermore this format is quite often used for drivers settings and and Linux and Unix systems also use this file format for system configuration.

Because the Android platform (can be used as the device that execute all the operations of sensors

and actuators) does not support the ”ini4j.jar”, it also was used the SharedPreferences class to do the necessary configurations. This class provides a framework that allows saving and retrieving data in a persistent manner (so that it is always accessible). This framework is used in the Android community for configuring applications because the data is stored in persistent form. This way the application programmer only needs to input the configuration data once, allowing some inconveniences to disappear during the execution of an application that uses the middleware, such as the login in the Azure virtual machine.

4.6 System

This section introduces the distribution of modules over system components and communication architecture.

It is followed by the discussion of system modules. This discussion elaborates on the techniques used to make the system objectives possible.

4.6.1 System Components

The system consists in two separate physical components: devices running applications (for example clients) and the cloud that makes data processing and saves data (for example server)(see figure 4.2) . The different logic is distributed among them. There are four types of communication between these two components through the protocols TCP, UDP, SSH and HTTP REST. There is also a publish subscribe model for internal communications in the device. Many of the messages exchanged by these protocols trigger events in system components.

Device Devices are equipments (example cell phones, tablets, and computers) that have a minimum capacity to be able to run applications. Compared to other hardware components, they have limitations such as limited memory and limited battery. Inside of these limited devices there are running applications that were developed by programmers. It is inside of these applications that the management model and cloud client side model are going to be run (see figure 4.2) if the application programmers do not want the hardware components to reach exhaustion because of exhaustive work, ensuring that their application has a longer life.

Cloud It is an open and flexible cloud platform that enables to rapidly create, deploy and manage applications. And uses the Platform as a Service model. The main goal of this component is to give to the applications that are running in devices under limiting constraints, more processing power and memory, allowing the device components not to be pushed to the limit.

The cloud server side model runs in this component (see figure 4.2).

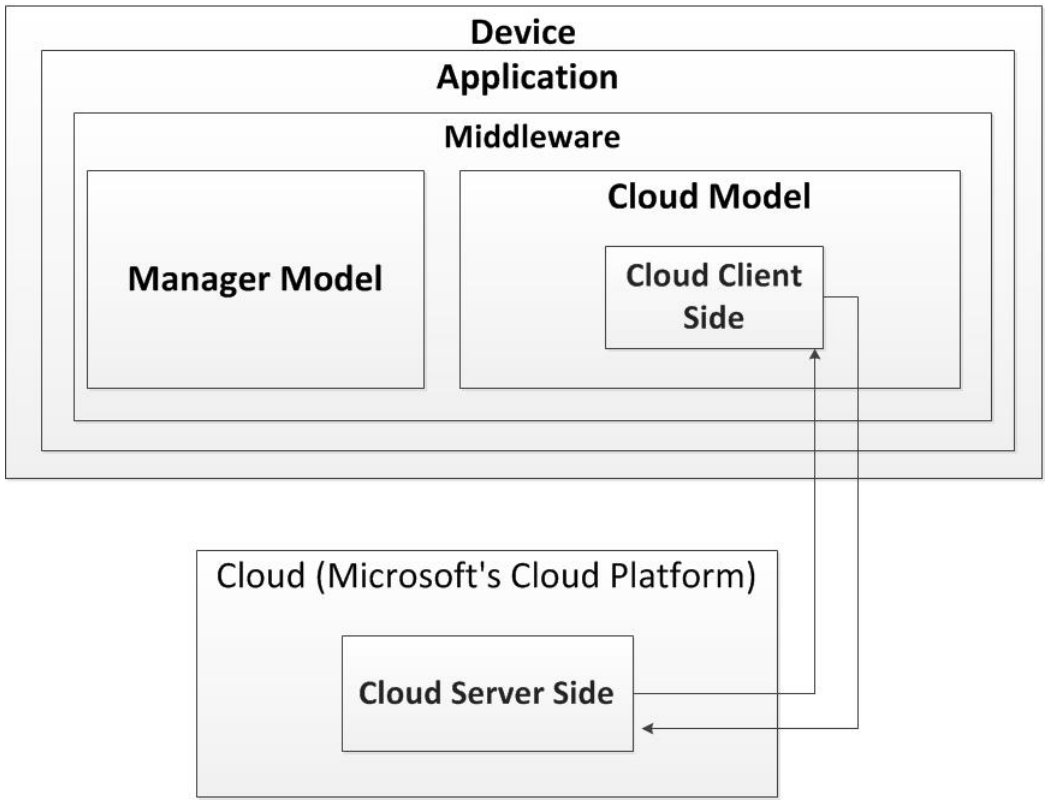

Figure 4.2: System Components

4.6.2 Manager Module

The management model aims to determine the state of certain hardware components, as well as the state of the Internet connection (depends on how the application programmer defines a good connection or bad connection). The hardware components that will be monitored are the battery, the CPU and the memory. The application programmer defines each component's critical state on a configuration file, before the middleware starts to be used. When one of these components is equal or superior to the critical value and Internet connection is good (the definition of a good Internet connection state is also defined by the application programmer) a certain execution will no longer be run on the device and starts to be executed in the cloud. The transaction to the cloud is going to be explaining in the section cloud.

Figure 4.3 shows the state machine of the management model.

The management model is initialized in the ”Middleware” state when the ”Application” state sends a command to ”start the management middleware” (initialization class monitor). The “Middleware” state contains the conditions for the components to be monitored. These conditions are updated by the state ”Monitoring” through a shared queue between the two states.

The ”Monitoring” status is initialized when the ”Middleware” state receives the command to initialize the management middleware. This state is always checking the conditions of the Wi-Fi signal quality and the conditions of the battery, CPU and memory, through calls to device hardware that runs the application (example robots and actuators) and is integrated with the middleware. The values obtained are compared with the critical values stipulated by the application programmer. The CPU test load is a

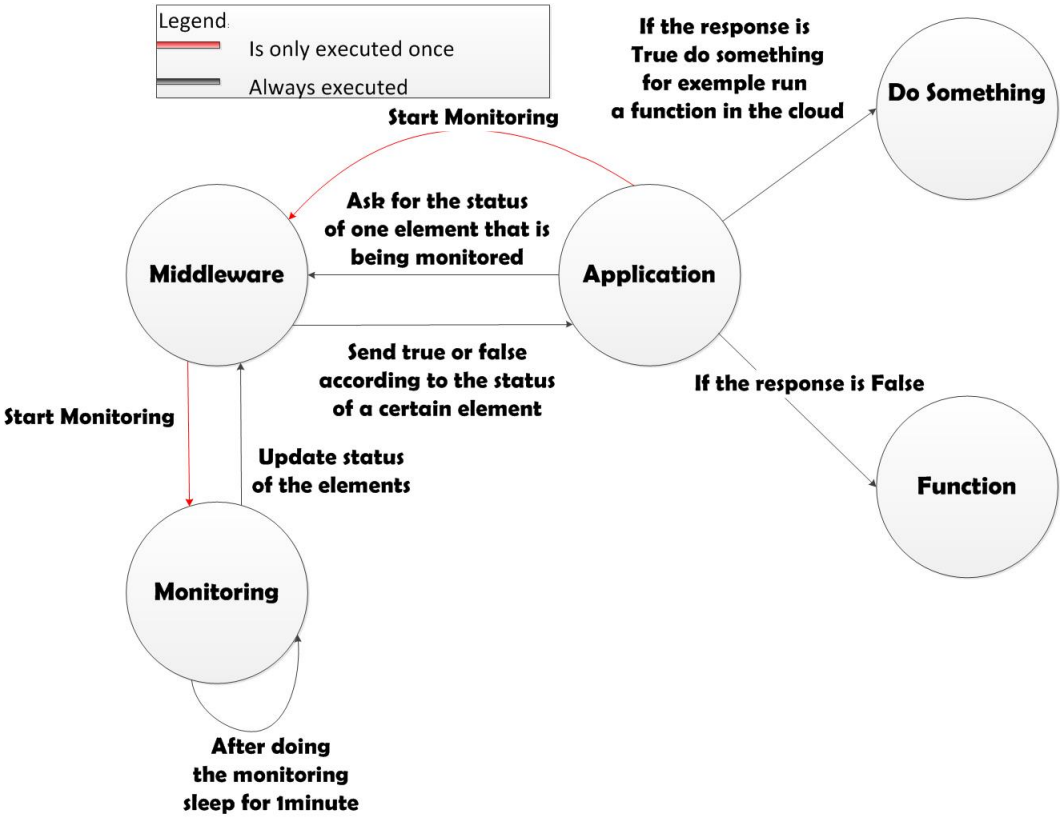

Figure 4.3: state machine of the management model

little bit different from the other tests because it only sends a notification if the read values were three times superior to the critical value. This option avoids reactions to sporadic peaks. If the signal quality of the wireless network is below the critical value stipulated by the application programmer the remaining monitoring tests will not be performed. After each monitoring cycle of the hardware components the ”Monitoring” state goes into sleep mode (for example one minute). The sleep mode duration is also defined by the application programmer.

The ”Application” state sends requests to the state ”Middleware” on the conditions of a certain component (example battery) awaiting for a reply of ”True” or ”False”. If the answer is ”False” it means that the state of the component is below the critical status (and hence running with enough resources on the device, so that no action is needed) stipulated earlier by the application programmer. This way the programmer's application can continue to run without any changes, and no event is initiated. This moment is depicted in the state machine through the transition between the ”Application” state and ”Function” state. The transition between the ”Application” state and the ”Do Something” state represents the situation where the response sent by the ”Application” state is equal to ”True”. When the answer is equal to ”True” it means that the conditions of the component exceed the critical value. In these cases the application programmer has the freedom to choose which action to take after receiving the message. One possible option can be to use the services that a cloud platform provides. For example performing a certain action on a machine provided by the cloud platform. This way some load is removed on the application and on the device that is running the application, and some resources are released (for

instance, making available more memory).

4.6.3 Cloud Module

The cloud module has the goal of reducing the load on the hardware components. This cloud module is divided into two sub modules: the cloud client and the cloud server. In the following paragraphs both modules will be described. The cloud module state machine is shown in figure 4.4.

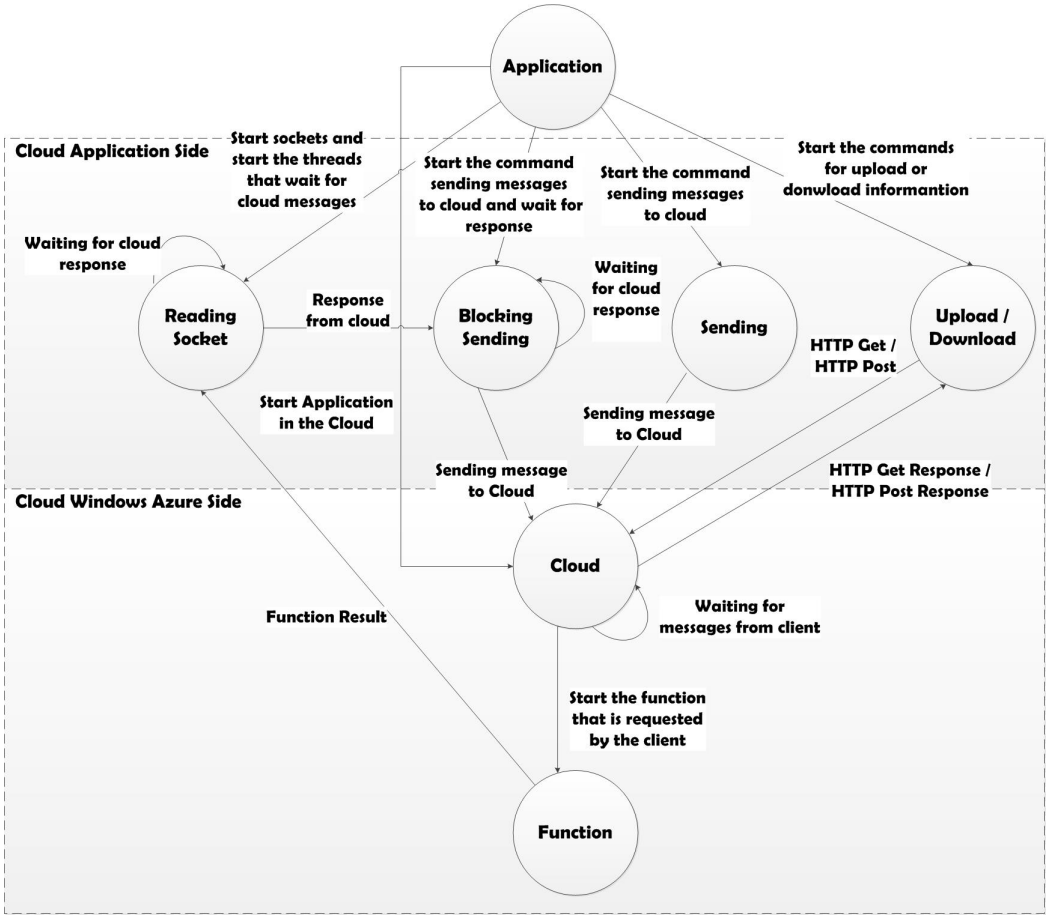

Figure 4.4: Cloud module state machine

4.6.4 Cloud Client Side

The Cloud Client Side module depicts the communication between the application and the cloud. This communication is initialized when the programmer application makes an initialization call to the middleware. This boot is portrayed in the connections between the ”Application” state and ”Reading Socket” state and between the ”Application” state and the ”Cloud” state. The first step is to boot the server in the cloud via an SSH command, this way it is possible to execute a command remotely. Secondly the TCP socket and the UDP socket from the client side are created. Lastly are also configured the access settings of POST and GET commands of HTTP REST protocol. When the application programmer intends to perform an upload of information in the cloud or download information a POST or GET command to cloud i sent.

The response of the cloud can be one of these four options: either it is a confirmation that the information was successfully saved, or there was an error while performing the operation of storage of information, or the information requested by the GET command, or message for some reason was not been possible to obtain the information requested. This communication is represented in the transitions between the states ”Application”, ”Upload / Download” and ”Cloud”.

In the case of UDP / TCP connections between application and cloud there are two types of connection: the blocking call and the non-blocking call. A blocking call is portrayed through the linking between the states ”Application”, ”Blocking Sending”, ”Cloud”, ”Function” and ”Reading Socket”. In this connection a message is sent to the cloud with the following information: the function name and the arguments that the function needs executed. After the message is sent the state ”Blocking Socket” enters a blocking state and waits for the result of the function that is going to be executed in the cloud. This result will be delivered by the state ”Reading Socket.”

In the non-blocking connection a message equal to the blocking connection is sent. But in this case the state does not stays blocked after the message is sent, the program continues to run and when the response is returned by the cloud it is saved in the device memory until the program needs the response. The states involved in this connection are ”Application”, ”Sending”, ”Cloud”, ”Function” and ”Reading Socket.” The state ”Reading Socket” after being initialized enters in block mode waiting for new entries in the socket that arrive from the state ”Cloud”. When there is a new message on the socket this will be processed in two ways depending on the information of one of the fields of the message. If this field is ”True” the message is from a blocking function and the result of the function is sent to the state ”Sending Blocking”. In the case of field is ”False” the function name and the function result will be stored in the device memory until the program needs the result of this function.

4.6.5 Cloud Server Side

This module depicts the communication between the cloud and the application. The component is initialized when the machine where the application server is running receives a SSH connection with the start command. The state ”Cloud” after his startup gets locked waiting to receive messages from the client. To ensure greater efficiency in real-time situation that uses a TCP connection this state contains a mechanism that drops packets. This mechanism aims to solve a possible problem that can occur when the speed of transmission of packets is much higher than the packet reading speed. This can bring an undesirable delay to the application programmer that is using the middleware. Because of that the mechanism is based on the idea that if a new packet arrives and a previous packet was not yet read, the oldest packet is discarded and replaced by the new packet, as this new packet is much more recent than the last. Upon receipt of the message and its decoding it is possible to identify the name of the function intended to be performed and which are its arguments. Knowing what is the name of the function to be performed it is possible to go from state ”Cloud” to ”Function” state. The ”Function” state consists in the execution of the functions that were chosen by the application programmer to be in the cloud. At the end of the execution of a function a message is sent to the client (state ”Reading Socket) with the function

name, with the function result and a small information for the state ”Reading Socket” can identify if the response is to the blocking function or to the non-blocking function. The sending of the message is portrayed by the link between the ”Cloud” state and ”Reading Socket” state.

In Figure 4.5 it is possible to see in more detail the messages exchanged between the following components: the application, middleware and the cloud.

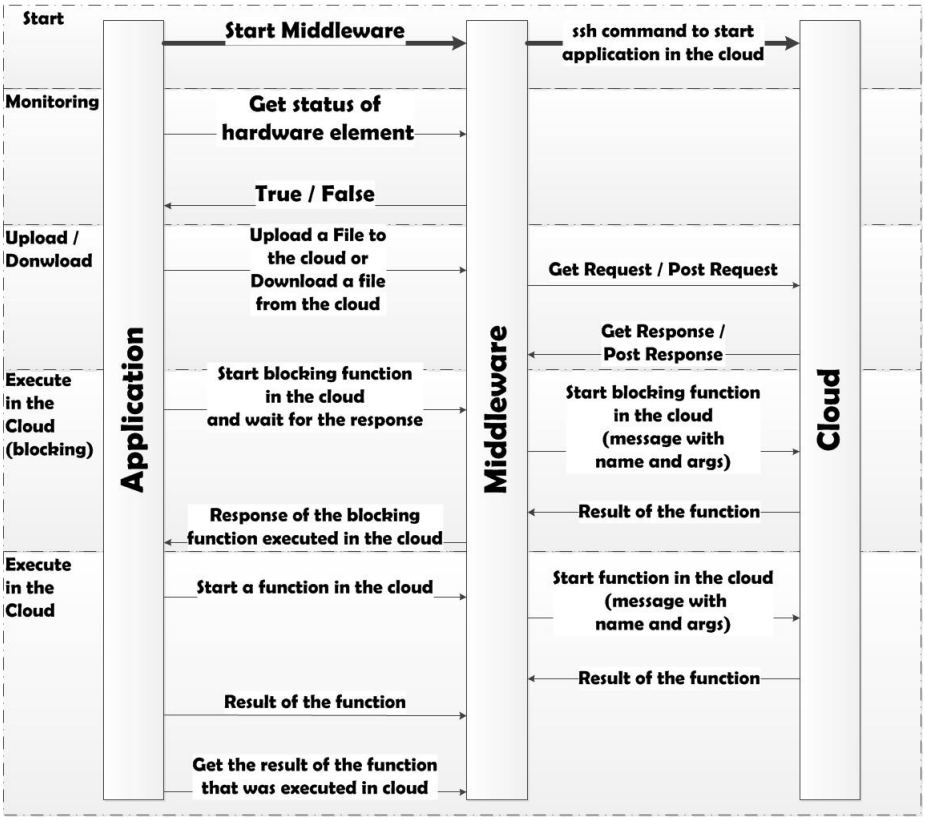

Figure 4.5: Messages exchanged between the following components: the application, middleware and the cloud

In Figure 4.5 the ”start” section represents the commands sent to initialize the middleware and cloud. The section ”Monitoring” represents the messages sent between the application and the middleware to verify the states of the components: battery, CPU load, memory and Wi-Fi signal quality. The following section represents the messages exchanged to do an upload or download of information between application and the cloud and the remaining represent the messages that are sent to achieve the distribution of execution flows, differing on the existence of blocking, waiting for a reply and a non-blocking state.

Chapter 5

Experimental Results Assessment

The objectives of the result assessment is the validation of the implementation, and the determination of adequate system improvements. For achieving these goals, several metrics were firstly defined. This chapter is divided into areas of analysis consisting of sets of testing experiments that cover different analysis aspects. Results are gathered through the execution of a number of experiments that were defined for each area. These results are stored, taking into account the expected value and standard deviation for the set of samples of the targeted metrics. In the end, results analysis are presented in a chart format, so that their interpretation is clearer to the reader. The general assessment methodology focuses on testing, and if possible validating, the different parts of the system individually, and then progressively integrating more complexity. The detailed experimental methodology is described under each test area subsection.

5.1 Metrics

The following metrics were used to evaluate the potential gains that may be brought by the solution:

- Energy consumed when running the application alone
- Energy consumed when the middleware is integrated with the application
- Time that takes to perform a function in the device
- Time that takes to perform a function when the middleware is integrated
- Delay times when exists migration of execution flow to the cloud platform
- CPU load when application runs stand-alone
- CPU load when running with the help of the cloud.

5.2 Simulation scenario

5.2.1 First Scenario

To test the middleware in the worst conditions that an application may be subject an application that makes video processing was chosen. This choice was due to the fact that this kind of applications need a huge CPU load, spending a lot of battery, and requiring rapid responses. The first test scenario consists on the integration of the developed middleware with an Android application (which was being developed in YDreams Robotics). This Android application uses the Computer Vision Library (OpenCV) to do face detection, as well as face tracking after the detection. As this is an application that makes a lot of image processing, it turns out to be a fairly heavy application in terms of CPU processing power, battery consumption and generated traffic. Thus, the device performance gets worse over time. This test intends to

- Check what is the percentage of spent battery
- What are the CPU loads during the application execution
- If the inclusion of middleware brings some significant delays to the application that uses it

In this scenario one BQ tablet with the Android operating system was used, and one virtual machine with one core and 1,75GB of Random-access memory (RAM) for the Microsoft's Cloud Platform.

5.2.2 Second Scenario

The second test scenario is the integration of the middleware with the Android application of BIOPLUX [1]. This Android application connects via Bluetooth to a bracelet that contains a motion sensor. This sensor is always sending coordinates to the Android application that saves the coordinates in a file. As the Android device has reduced memory resources it is very likely that the Android memory will eventually be fully allocated after some time because the file will grow. The integration of the middleware with the Android application will solve this issue because when the occupied memory reaches a certain value the data will be sent to the cloud. This way the application will not block, and data will not be lost. With this test, whenever the memory arrives to values equal or superior to 50% occupancy and the Internet connection is good (where what is a good Internet connection is defined by the application programmer), a group of files is sent to the cloud. This test scenario will evaluate if after the uploading of the files, the state of the memory will drop below 50% occupancy. And finally it will be checked if the files that were sent to the cloud were saved successfully. This test used a Plux motion bracelet, a BQ tablet with an Android operating system, and as a Cloud resource the Windows Azure Blob Storage.

[1] `http://www.physioplux.com/index.php/pt/` (accessed last time on 14 May 2014)

Figure 5.1: Motion Bracelet

5.3 Analysis

5.3.1 Energy Consumed

This application underwent two experimental test setups to check percentage values for the battery energy spent during the execution of the application described above. These experiments lasted 40 minutes and were repeated 5 times. The first experimental setup set the baseline, where the application that detects and tracks the human face was the only one running on the Android operating system and without any interference from the middleware. The second experimental setup consisted of the same application but integrated with the middleware, being part of the algorithm implemented in the cloud. In the graphs of Figures 5.2 and 5.3, it can be observed that there were no gains with the transferring of certain application execution flows to the cloud. In both figures and in table 5.1 it is possible to see that the results are better when the middleware is not integrated. Although these are almost irrelevant because the difference between the obtained values with the presence of middleware and the values without the presence of the middleware never exceed the 8%. The fact that does not exists any gains in relation to the percentage of battery energy spent with the integration of middleware may be due to excessive use of the camera as it is a hardware component that consumes a lot of battery. But this fact does not explain everything because the test that is conducted without the presence of the middleware also used this component extensively. Another aspect that may have influenced the results to be different from what was expected (the battery values being lower with the presence of the middleware) is the constant access to the wireless network because after the connection is made with the cloud, this connection will only be closed when the application is turned off or the application programmer specifies an order of termination of the connection. And this constant access consumes some battery [44].

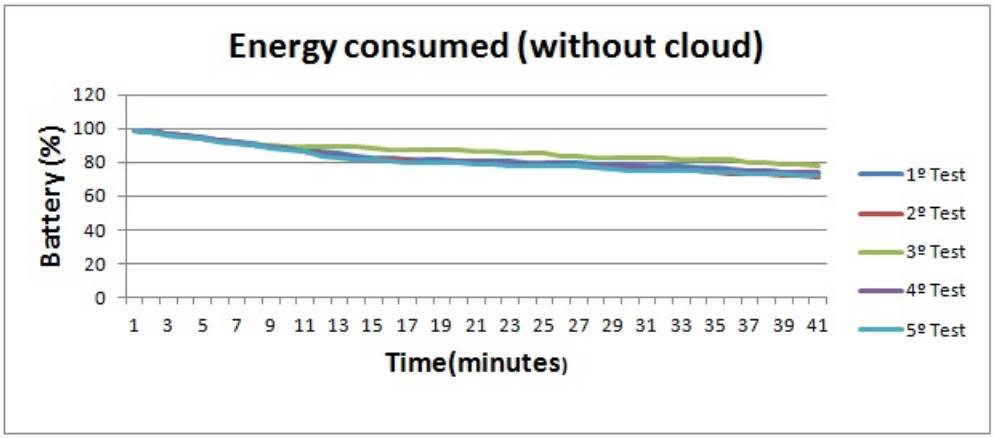

Figure 5.2: Energy Consumed without using the cloud

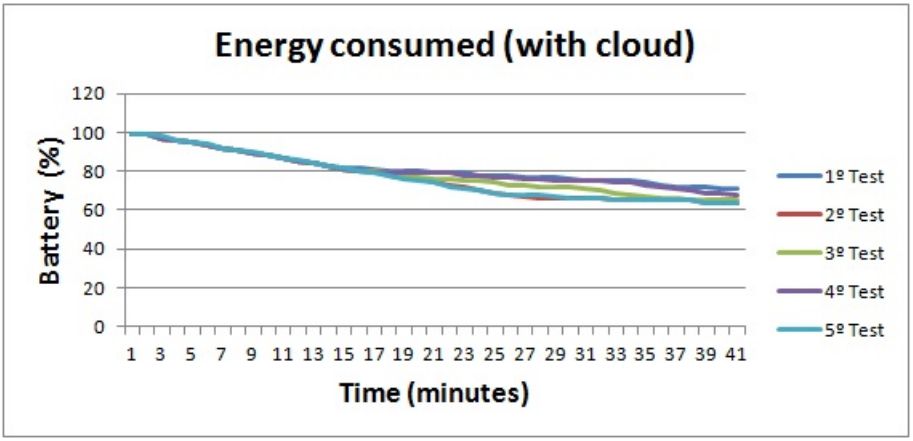

Figure 5.3: Energy Consumed using the cloud

5.3.2 CPU Load

To check whether there are advantages in using this middleware in relation to the CPU load, the ”Face Detect/Tracking” application was subjected to the following testing experiments (these tests lasted 20 minutes and were repeated five times):

- In the first experimental setup the application that detects and tracks the human face was the only one running on the Android operating system and without any interference from the middleware.
- The second experimental setup consisted of the same application but integrated with the middleware. The application only sends messages composed by pictures frames from the camera and sends these to the cloud with the analysis of the picture frames made in the cloud.

From figures 5.2, 5.4 and 5.5, it is possible to notice that there was a significant gain with the migration of the detection and tracking algorithms to the cloud. This increase is explained by the fact that the heavier work is being done in the cloud, which alleviates the processing in the device’s CPU that is running the application, as it only needs to get the frames and sending them. By reducing the CPU load, the lifetime of the battery should increase. As previously explained, this was not the case due to other factors, such as more battery energy required for wireless communications.

Experimental setup	Without Cloud	With Cloud
Average working battery charge (battery at the end of experiment)	73,6%	66,4%
Standard Deviation	2,70	3,04
Average of the battery charge spent by the five tests	25,2%	32,6%
Standard Deviation	3,11	3,04

Table 5.1: Energy Consumed (percentage of battery charge)

Experimental setup	without Cloud	With Cloud
Average Of the CPU Load	68,98%	45,84%
Standard Deviation	3,53	1,15

Table 5.2: CPU Load

The image processing algorithms requires a lot of CPU to be executed. This situation may preclude those others applications that should run properly (may enter a blocking state) while the image processing applications are also running. The same also happens with image processing application that ceases to have the CPU just for itself, competing for resources such as CPU processing time, and this competition may create difficulties to its execution, reducing the framerate, for example less frames per second for the analysis. In order to prove that the utilization of the developed middleware is a good solution to solve this competition problem for limited resources, the following test scenarios were performed: checking the CPU status when an exhaustively analysis of 100 frames is made, and checking the time consumed

- when the analysis is done on the BQ tablet
- when it is done in the cloud.

Through the observation of table 5.3 and Figure 5.6, one can verify that when the processing is made in the tablet, the CPU load reaches saturation values. On the other hand, the usage of cloud processing originates a significantly lower CPU load at the tablet.

	Without Cloud	With Cloud
CPU Load average	98,69%	25,53%
CPU Load standard deviation	1,20	2,28
Average of the execution time of the algorithms (ms)	1292	1100
Standard deviation of the execution time of the algorithms	5,60	4,80

Table 5.3: CPU Comparative Load in exhaustive case

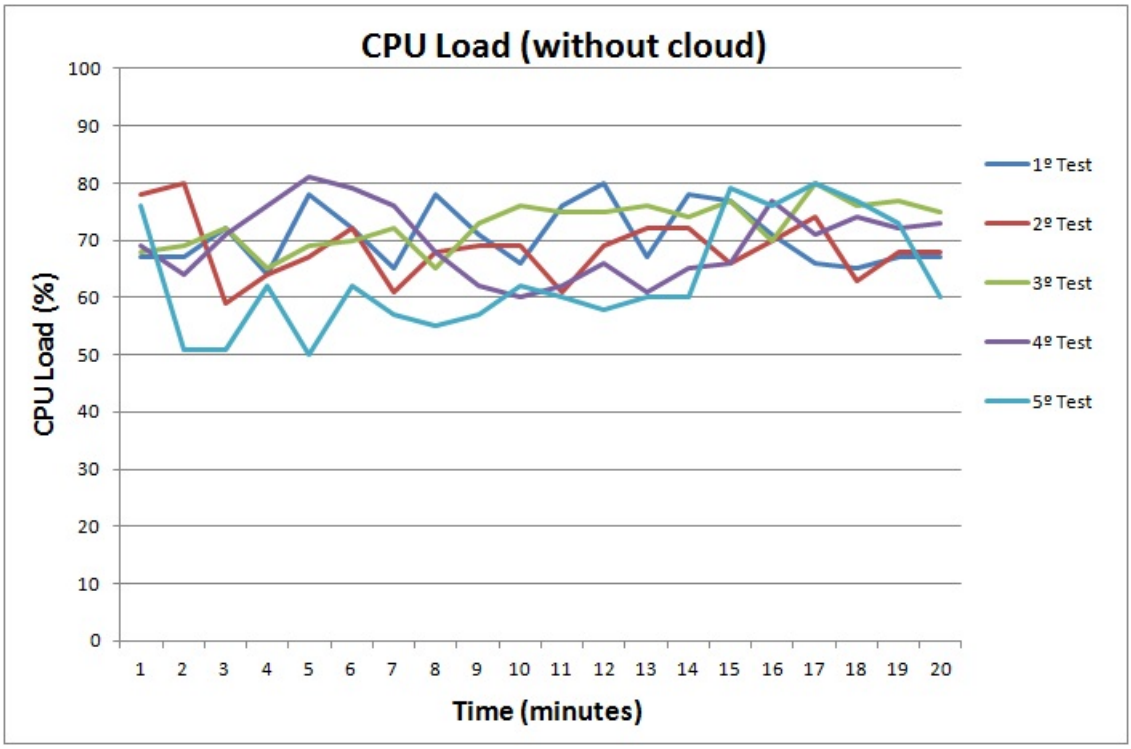

Figure 5.4: CPU Load without the presence of the middleware

It can be concluded that the integration of the middleware is beneficial when we want to run reliably more than one application on a device of limited resources, because with the transferring of execution flows to the cloud much of the processing is done outside the device thus freeing some of the CPU load, so that other device applications can also be executed.

5.3.3 Time spent

The time delay that can exist to perform certain execution flows in the cloud is addressed hereafter.

As expected the integration of the middleware brought some delay to the execution of the application and this delay may increase when the message size increases.

After the execution of the test (Tables 5.4, 5.6 and 5.5) it was observed that there is a significant delay when the algorithm is executed in the device. This longer delay is explained by the network conditions not being the best in terms of quality. But the factor that most impacts this delay is the utilization of the TCP communication between the application and cloud. Although this means of communication is quite reliable because none of the sent messages is lost, it can also bring much delay because when there is any error in the network the lost message segments will be sent again until message reaches its destination, thereby translating into an extra delay when sending the message. Even with the implementation of a mechanism to discard messages, these are only discarded when the message is fully received

In this test it can also be verify the occurrence of high standard deviation values. This is due to image variations in terms of complexity, since image complexity can cause the increase or decrease of the algorithm execution time.

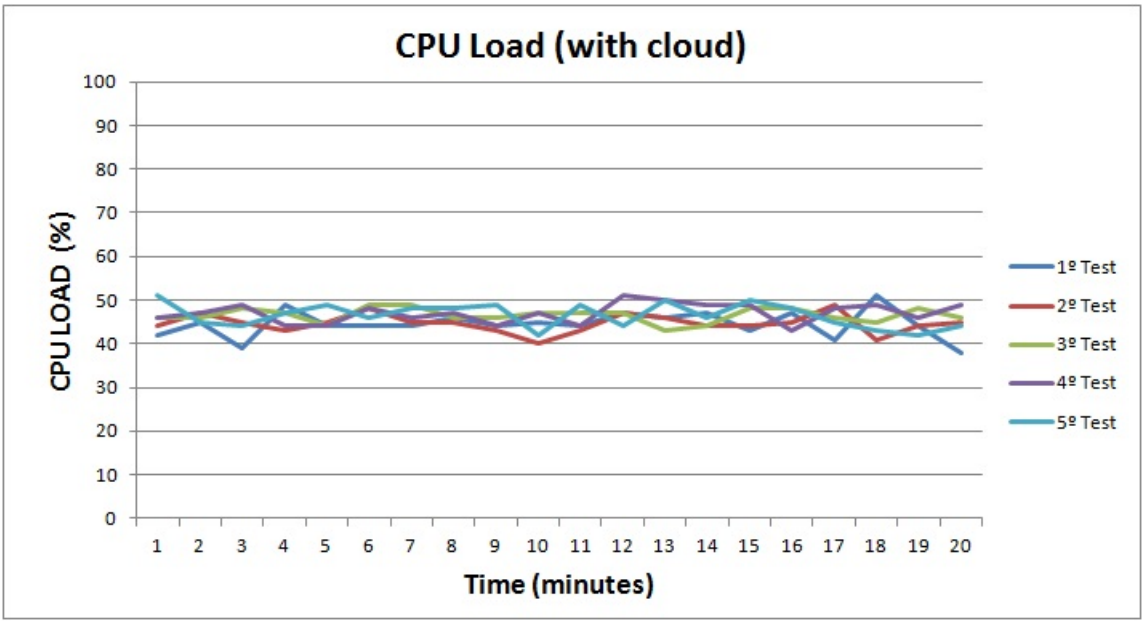

Figure 5.5: CPU load with the usage of the middleware

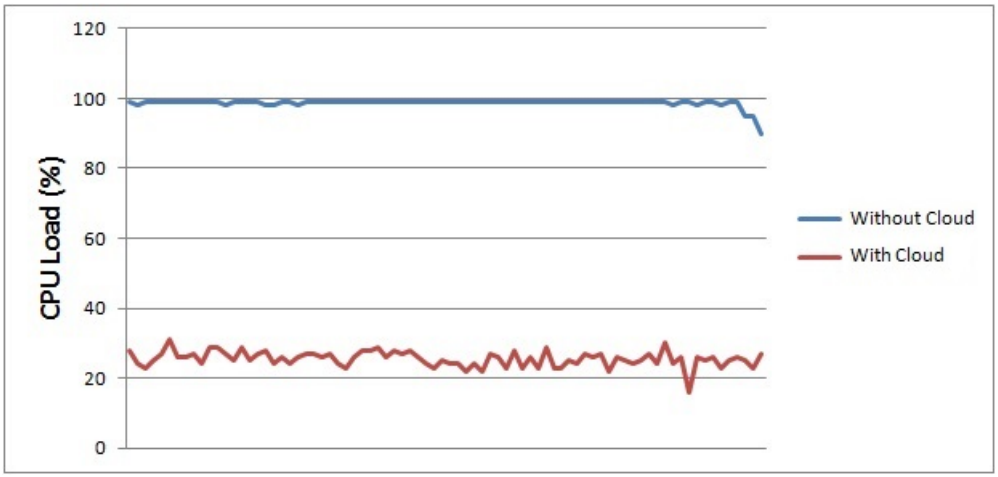

Figure 5.6: Load in exhaustive case of heavy consumption of CPU computational resources by applications

5.4 Functional test

To test whether the hardware management mechanism runs before an application enters a blocking state, or even crashes due to lack of resources, the following test scenario was performed: after the memory of the Android reaches half its capacity, another application sends all files at a specific folder to the cloud and erases them from Android device memory, freeing the memory so that the application can continue to run reliably. Through images 5.7 and 5.8 we can see that all the files that were in the folder were successfully stored in the cloud, proving that the management mechanism is reacting when the memory reaches is a critical point.

Experimental Setup		
Detect	Average (ms)	157,19
	Standard Deviation	4,79
Track	Average(ms)	9,51
	Standard Deviation	25,29

Table 5.4: Time (in ms) spent to detect and track face with the presence of the middleware (without transmission time)

Experimental Setup		
Detect	Average (ms)	575,56
	Standard Deviation	28,18
Track	Average(ms)	423,16
	Standard Deviation	0,934

Table 5.5: Time (in ms) spent to detect and track face with the presence of the middleware (with transmission time)

5.5 Discussion

It can be concluded that the usage of this middleware in applications that can have limited hardware resources (e.g. battery, CPU) brings some advantages for the lifetime of an application and avoids that the application enters in lock status because of lack of resources. However, in certain applications it is impossible to make gains on all hardware components as seen in aforementioned experimental tests. This is the case for applications that make excessive use of certain hardware components that require a lot of battery. Although the middleware also makes the device to lose some energy due to higher communication transmissions, this spending is often small when compared with the consumption by an application that has quite high CPU processing needs. But the downside of this middleware concerns the fact that the times of sending and receiving messages to the cloud are high. However this point can be improved with the implementation of UDP communications if the cloud platform allows. Even with these restrictions the developed middleware proves that it can solve some problems that exists in devices that are limited in terms of resources.

Experimental Setup		
Detect	Average (ms)	195,06
	Standard Deviation	5,56
Track	Average(ms)	56,95
	Standard Deviation	0,93

Table 5.6: Time (in ms) spent to detect and track face without the presence of the middleware

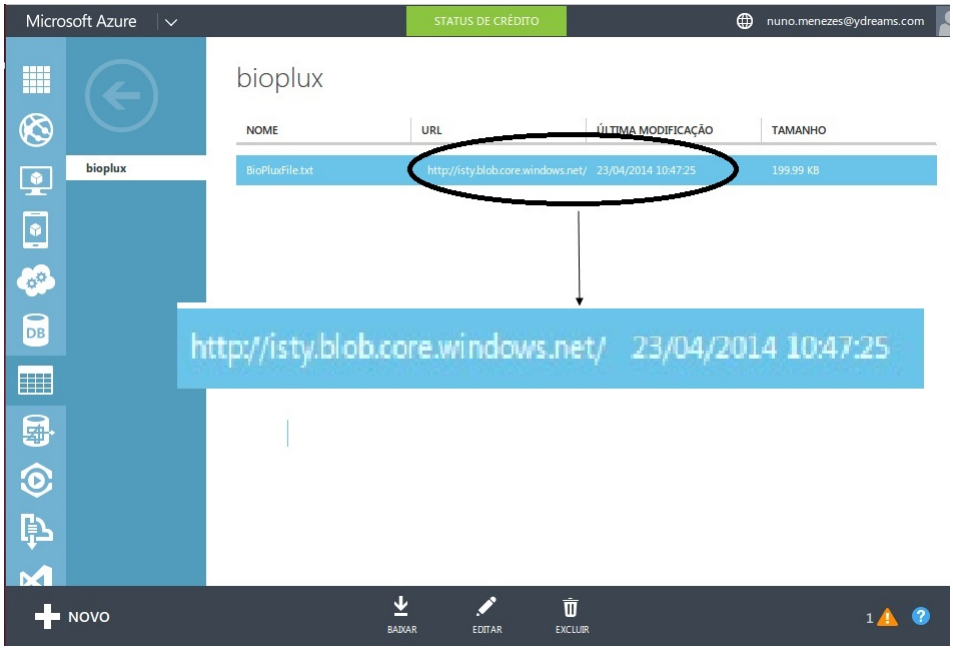

Figure 5.7: Azure Interface (showing that the file was uploaded to the cloud at 10h47)

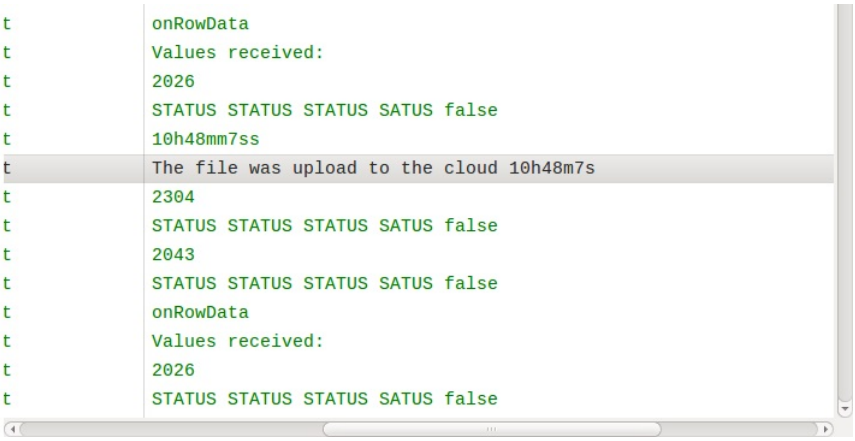

Figure 5.8: Android Log (showing that the file was sent to the cloud at 10h47)

Chapter 6

Discussion and Conclusions

Recent technological advances have presented new concepts applied to the various economic sectors of our society. WSN and M2M are emerging concepts in areas of growing interest. Indeed, the idea of monitoring several types of parameters in various environments has been motivating significant research works in these areas. Concerning some challenges presented by WSN's deployments, Cloud Computing platforms are a prominent element that can respond in a more efficient and powerful way to such issues. This thesis proposes a middleware whose major goal is to solve the problem of lack of resources in devices such as limited memory and battery, with the help of cloud platforms. The proposed architecture's most important models are the management and cloud models. The management model aims to monitor the most problematic hardware components and communicate whenever some of these components are in a critical condition. The cloud uses two sub modules. One is cloud client module and this module communicates with the management module and sends messages to the cloud server module, for instance whenever one of the components is in a critical state. The cloud server module is running in the cloud platform and is where the chosen algorithms are running. These algorithms start to be executed when the cloud server module receives a message from the cloud client module. The performed tests demonstrate that the use of the cloud to solve the lack of resources problem in devices is quite advantageous because it allows the CPU load to be reduced to lower values. This leads to battery with extended autonomy, thereby providing less inconvenience to device users. It also avoids applications entering in a blocking state due to lack of memory, and allows running more applications in simple device that otherwise would exceed the available resources. But most importantly, the decision wether to run an application locally or remotely is done dynamically, according to the status of the available resources as checked through active monitoring.

However, in certain situations it cannot be ensured that all issues are solved with the cloud introduction, because some of problems are typical to hardware components (like the utilization of camera that spends a lot of battery). The solution also presented a small flaw with regard to time lost when performing certain flows in the cloud. This delay is not related with the running time of the algorithm, but instead with the time lost in sending the message between the device (robot or actuator) and the cloud and vice versa. This problem relates to the fact that the middleware uses a TCP protocol for communications

which, though a reliable transport means, it introduces latencies due to the error-checking for packets.

This middleware will be beneficial for programmers who want to make the most of the hardware resources available on the devices.

6.1 Future Work

In order to improve the results of this work and to render the middleware even more flexible, robust and efficient, there are some points that must be referenced for possible improvements of the solution.

As already mentioned in this work, the solution was testes only with one cloud platform (Windows Azure). These days much of the cloud platforms already provide the requirements needed to support the middleware objectives and achieve to run it viably. So in the future the intention is to cover the greatest number of possible cloud platforms, and not be restricted to a specific cloud. Possible examples of cloud platforms that the solution may come to support are Amazon Elastic Compute Cloud or Google Cloud Platform. At this point the middleware only supports Android operating system when it comes to mobile operating system as such it is fundamental to open the range of mobile platform supported by the middleware such as operating systems from Apple and Microsoft. As important as the mobile operating system supported by the middleware is to run without any restriction over the biggest number of programming languages allowing the programmer that uses this middleware to make changes in his solution.

Although the middleware is focused in an environment where there is always a wireless network available and with good access, these requirements are not always fulfilled. Because Wi-Fi can be overused this can delay communication with the cloud. This issue can bring some inconvenient for applications that require fast answers. And if the middleware is used in places where network access is done via GPRS can also bring some inconvenience to the solution. To counter this problem one possible solution would be to let the middleware decide when it is viable to use or not the internet connection and if the internet connection is not in good state find a solution that fulfilled the objective of extending the lifetime of one device and not take this extreme case of use. Therefore it would be important to find other solutions besides the cloud to perform this flow distribution. A possible example would be to use Bluetooth technology to find equipment nearby that has the ability to perform the intended task and that at that very moment were available to do it. Thus it is possible to have more than one solution and do not stay limited to the cloud, although the cloud is the best solution for cases of transfer flows because it has always unlimited resources which in normal equipments such as smartphones is not possible to find.

As future work was also interesting to give more functionality to the cloud platform beyond allowing the transfer of flows between themselves and the application: for example allow connection between two applications being the cloud an intermediary between the two. Finally we cloud improve the speed of communication between the middleware and cloud and Vice Versa.

Bibliography

[1] H. S. Woelffle, P. Guillemin, P. Friess, and Sylvie. ”Vision and challenges for releasing the Internet of Things”. In Publications Office of the European Union. March 2010

[2] L. Atzoria, A. Ierab, and G. Morabito. ”The Internet of Things: A survey”. Computer Networks. 54, issue 15. 2010

[3] A. Iera, C. Floerkemeier, J. Mitsugi, and G. Morabito. ”The Internet of Things”. Guest Editorial. In IEEE Wireless Communications. 17, Issue 6. 2010

[4] D. Evans. ”Internet of Things How the Next Evolution of the Internet Is Changing Everything”. CISCO white paper. 2011

[5] K. Framling, and J. Nyman. ”Information architecture for intelligent products in the internet of things”. Beyond Business Logistics proceedings of NOFOMA. 2008

[6] M. Kranz, L. Roalter, and F. Michahelles, ”Things That Twitter: Social Networks and the Internet of Things”, What can the Internet of Things do for the Citizen (CIoT) Workshop at The Eighth International Conference on Pervasive Computing (Pervasive 2010). 2010

[7] Y. Chen, H. Hu. Internet of intelligent things and robot as a service. Elsevier Simulation Modelling Practice and Theory. 2012.

[8] K. Chang, A. Soong, M. Tseng, and Z. Xiang. ”Global Wireless Machine-to-Machine Standardization”. In Internet Computing, IEEE. March 2011.

[9] V. Galetic, I. Bojic, M. Kusek, G. Jezic, S. Desic, and D. Huljenic. ”Basic principles of Machine-to-Machine communication and its impact on telecommunications industry”. MIPRO, 2011 Proceedings of the 34th International Convention. IEEE, 2011.

[10] I. Bojic, D. Katusic, and M. Kusek. ”Communication in Machine to Machine Environments”.Proceedings of the Fifth Balkan Conference in Informatics.2012.

[11] I. Akyildizand, and I. Kasimoglu. ”Wireless sensor and actor networks: research challenges.” Ad hoc networks 2.4 (2004): 351-367.

[12] A. Rezgui, and M. Eltoweissy. ”Service-oriented sensor–actuator networks: Promises, challenges, and the road ahead”. Computer Communications 30.13 (2007): 2627-2648.

[13] T. Melodia, D. Pompili, V. Gungor, and I. Akyildiz ”Communication and coordination in wireless sensor and actor networks.” Mobile Computing, IEEE Transactions on 6.10 (2007): 1116-1129.

[14] E. Cayirci, T. Coplu, and O. Emiroglu. ”Power aware many to many routing in wireless sensor and actuator networks.” Wireless Sensor Networks, 2005. Proceeedings of the Second European Workshop on. IEEE, 2005.

[15] A. Willig, K. Matheus, and A. Wolisz. ”Wireless technology in industrial networks.” Proceedings of the IEEE 93.6 (2005): 1130-1151.

[16] J. Ploplys, P. Kawka, and A. Alleyne. ”Closed-loop control over wireless networks.” Control Systems, IEEE 24.3 (2004): 58-71.

[17] S. Kroc, and V. Delic. ”Personal wireless sensor network for mobile health care monitoring.” Telecommunications in Modern Satellite, Cable and Broadcasting Service, 2003. TELSIKS 2003. 6th International Conference on. Vol. 2. IEEE, 2003.

[18] D. Merico, A. Mileo, and R. Bisiani. ”Pervasive wireless-sensor-networks for home healthcare need automatic reasoning.” Pervasive Computing Technologies for Healthcare, 2009. PervasiveHealth 2009. 3rd International Conference on. IEEE, 2009.

[19] W. Shi, and M. Liu. ”Tactics of handling data in internet of things.” Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on. IEEE, 2011.

[20] J. Wu, L. Ping, X. Ge , Y. Wang , and J. Fu ”Cloud storage as the infrastructure of cloud computing.” Intelligent Computing and Cognitive Informatics (ICICCI), 2010 International Conference on. IEEE, 2010.

[21] R. Xue, Z Wu, and A. Bai. ”Application of cloud storage in traffic video detection.” Computational Intelligence and Security (CIS), 2011 Seventh International Conference on. IEEE, 2011.

[22] H. Wang , Y. Yu, P. Zhu, and Q. Yuan. ”Cloud computing based on internet of things.” 2011 Second International Conference on Mechanic Automation and Control Engineering. 2011.

[23] Q. Li, T. Zhang, and Y. Yu. ”Using cloud computing to process intensive floating car data for urban traffic surveillance.” International Journal of Geographical Information Science 25.8 (2011): 1303-1322.

[24] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. ”Above the Clouds : A Berkeley View of Cloud Computing”. In UC Berkeley Technical Report UCB/EES. 2009.

[25] F. Ren ”Robotics cloud and robotics school.” Natural Language Processing and Knowledge Engineering (NLP-KE), 2011 7th International Conference on. IEEE, 2011.

[26] E. Guizzo. ”Robots with their heads in the clouds”. Spectrum, IEEE 48.3. 2011

[27] Y. Chen, Z. Du, and M. Acosta. ”Robot as a service in cloud computing.” Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on. IEEE, 2010.

[28] M. Kranz, R. Rusu, A Maldonado, M. Beetz, and A. Schmidt. ”A player/stage system for context-aware intelligent environments.” Proceedings of UbiSys 6 (2006): 17-21.

[29] R. Rusu, A. Maldonado, M. Beetz, M. Kranz, L. Mösenlechner, P. Holleis, and A Schmidt. ”Player/stage as middleware for ubiquitous computing.” Proceedings of the 8th Annual Conference on Ubiquitous Computing (Ubicomp 2006)(Sept. 2006). 2006.

[30] D. Hunziker, M. Gajamohan, M. Waibel, and R. D’Andrea, . ”Rapyuta: The roboearth cloud engine.” Robotics and Automation (ICRA), 2013 IEEE International Conference on. IEEE, 2013.

[31] M. Broxvall, B. Seo, and W. Kwon. ”The peis kernel: A middleware for ubiquitous robotics.” In Proc. of the IROS-07 Workshop on Ubiquitous Robotic Space Design and Applications. 2007.

[32] C. Cote, Y. Brosseau, D. Letourneau , C. Raïevsky, and F. Michaud . ”Robotic Software Integration Using MARIE.” International Journal of Advanced Robotic Systems 3.1 (2006).

[33] Y. Jadeja, and K. Modi. ”Cloud computing-concepts, architecture and challenges.” Computing, Electronics and Electrical Technologies (ICCEET), 2012 International Conference on. IEEE, 2012.

[34] P. Mell, and T. Grance. ”The NIST definition of cloud computing.” National Institute of Standards and Technology 53.6 (2009): pp. 50.

[35] T. Dillon , C. Wu, and E. Chang. ”Cloud computing: issues and challenges.” Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on. Ieee, 2010.

[36] S. Patidar, D. Rane, and P. Jain. ”A survey paper on cloud computing.” Advanced Computing & Communication Technologies (ACCT)”. 2012 Second International Conference on. IEEE, 2012.

[37] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R Wheeler, and A. Ng. ”ROS: an open-source Robot Operating System.” ICRA workshop on open source software. Vol. 3. No. 3.2. 2009.

[38] G. Metta, P. Fitzpatrick, and L. Natale. ”YARP: Yet Another Robot Platform.” International Journal of Advanced Robotic Systems 3.1 (2006).

[39] N. Mohamed , J. Al-Jaroodi, and I. Jawhar. ”A review of middleware for networked robots.” International Journal of Computer Science and Network Security 9.5 (2009). pp. 139-148.

[40] M, Castro , A. Jara, and A. Skarmeta. ”An analysis of M2M platforms: challenges and opportunities for the Internet of Things.” Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on. IEEE, 2012.

[41] N. Mohamed , J. Al-Jaroodi, and I. Jawhar. ”Middleware for robotics: A survey.” Robotics, Automation and Mechatronics, 2008 IEEE Conference on. IEEE, 2008.

[42] O. Zweigle, R. Molengraft, R. d'Andrea, and K. Häussermann, . "RoboEarth: connecting robots worldwide." Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human. ACM, 2009.

[43] T. Bishop, , and R. Karne. "A survey of middleware." Computers and Their Applications. 2003.

[44] N. Balasubramanian , A. Balasubramanian, and A. Venkataramani. "Energy consumption in mobile phones: a measurement study and implications for network applications." Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference. ACM, 2009.

[45] C. Lee, S. Baset, and H. Schulzrinne. "TCP over UDP."

[46] S. Pedroni, and N. Rappin. Jython essentials. " O'Reilly Media, Inc.", 2002.

[47] C. Ihrig "JavaScript Object Notation." Pro Node. js for Developers. Apress, pp. 263-270. 2013.

[48] Cosm Ltd.; Cosm – Internet of Things Platform Connecting Devices and Apps for Real-Time Control and Data Storage; 15 November 2012; https://cosm.com

[49] OpenRemote; OpenRomete — Open Remote Automation — OpenRemote; 15 November 2012; http://www.openremote.com

[50] openhab; openhab – empowering the smart home – Google Project Hosting; 15 November 2012; http://code.google.com/p/openhab/

[51] Ninja Blocks; Ninja Blocks – The API for Atoms; 15 November 2012; http://ninjablocks.com

[52] IFTTT / About IFTTT; 15 November 2012; https://ifttt.com/wtf

[53] Physical Graph Corporation; SmartThings – Make Your World Smarter; 15 November 2012; http://smartthings.com

[54] N. Aharony, W. Pan, C. Ip, I. Khayal, and A. Pentland, Social fMRI: Investigating and shaping social mechanisms in the real world, Pervasive and Mobile Computing, (2011).

[55] W. Grosky, A. Kansal, S. Nath, L. Jie, and F. Zhao. SenseWeb: An Infrastructure for Shared Sensing, IEEE MultiMedia, vol.14, no.4, pp.8-13, Oct.-Dec. (2007).