FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Environmental Monitoring Services based on Wireless Sensor Networks

Carlos Eduardo Azevedo Oliveira

Thesis submitted under the course of:

Integrated Master In Electrical and Computer Engineering

Major in Automation

Tutor: Gil Gonçalves (Eng.)

February 2009

Resumo

Actualmente é evidente a dependência da sociedade moderna nas tecnologias de informação baseadas em sistemas computacionais, capazes de agregar e manipular enormes quantidades de informação. Associado ao aparecimento das tecnologias sem fios, à miniaturização dos circuitos integrados e a ao aumento da sua eficiência energética, surgiu uma nova área de estudos denominada comummente por: *Wireless Sensor Networks*. As "WSNs" resultam da junção de conhecimentos das áreas de Ciências de Computação e da Electrónica, que possibilitam o desenvolvimento de plataformas capazes de recolher dados de determinadas variáveis, de forma distribuída.

São vastos os campos de aplicação das tecnologias baseadas em redes de sensores sem fios, dadas as vantagens destas soluções quando comparadas com plataformas que recorrem a redes ligadas por cabos. Estas redes são normalmente estabelecidas utilizando pequenos dispositivos, alimentados por baterias, que recolhem informação no espaço que os rodeia. Dado o interesse nestas tecnologias, surgiram especificações e standards, tais como o *ZigBee* e o IEEE802.15.4 que definem os canais de comunicação e permitem interoperabilidade entre diferentes plataformas.

Estas novas plataformas vieram juntar-se a tecnologias *wireless* já bem estabelecidas, que estão presentes no quotidiano. O *Bluetooth* é um exemplo de uma dessas tecnologias. O seu uso tornou-se vulgar, tanto para fins pessoais como nos locais de trabalho. A expansão do número de dispositivos capazes de estabelecer redes sem fios é um acontecimento inevitável dado o aumento das taxas de transmissão, da fiabilidade deste tipo de comunicação de dados e da aceitação geral por parte do consumidor.

Esta dissertação documenta o desenvolvimento de uma plataforma baseada numa rede de sensores sem fios, capaz de recolher e publicar online dados ambientais. Um dos objectivos principais deste trabalho é o desenvolvimento de uma aplicação, capaz de integrar os dados recolhidos com ferramentas de geo-referenciação disponíveis publicamente. Tal é feito respeitando os standards abertos existentes. É utilizada uma abordagem incremental para demonstrar os vários passos utilizados na construção do sistema, e como foram resolvidas as várias dificuldades encontradas.

Este sistema é de seguida utilizado num caso de estudo. Nesta experiência as aplicações desenvolvidas são utilizadas para fins concretos no estudo da influência de diferentes materiais de construção no conforto térmico em espaços públicos.

Esta dissertação mostra que as redes de sensores sem fios se adaptam particularmente bem em aplicações que requerem uma recolha de dados espacialmente distribuída. Variações micro climáticas podem ser parametrizadas utilizando as ferramentas desenvolvidas. A integração dos dados recolhidos com sua localização geográfica possibilita uma percepção aprofundada da possível correlação entre a evolução das grandezas medidas e as características espaciais locais. Este estudo demonstra também que existem ainda limitações relacionadas com a qualidade das conexões entre nós e nos algoritmos responsáveis pelo estabelecimento rotas. A performance das aplicações desenvolvidas é também analisada por forma a demonstrar esta afirmação.

É possível estender o sistema desenvolvido para outras aplicações de monitorização. Um exemplo é a possível integração de sensores para monitorização dos níveis de poluição sonora, ou

mesmo da qualidade do ar, adicionando sensores de qualidade do ar e microfones. A monitorização do consumo energético de edifícios, bem como uma estimação da sua eficiência energética pode também ser conseguida expandindo algumas das funcionalidades do sistema desenvolvido.

Abstract

It is nowadays notorious the dependence of modern society in computer systems, able to aggregate and manipulate tremendous amounts of information. In the last decades, associated to the surge of the new wireless technologies, to the increased miniaturization and energy efficiency of integrated circuits, a new study subject emerged: Wireless Sensor Networks. WSNs join knowledge from several fields of science, like Computer Science and Electronics, to enable platforms that can be used to distributively sense specific variables.

The fields of application of WSN technologies are immense due to several advantages over common wired sensing networks. Standards and specifications like the IEEE802.15.4 and ZigBee emerged, that specify the communication channels and allow interoperability between different hardware. These networks are usually established with small battery powered devices , that collect data in their surrounding environment.

These new platforms join already well established applications in the field of wireless personal area networks, that are today present in people's lives. Bluetooth serves as an example; this technology is today seamlessly used in working and home environments. The penetration of devices capable of establishing wireless networks is bound to increase due to its massive acceptance, increasing data transmission rates and reliability.

The result of this work is a complete platform, based on a wireless sensor network, able to gather environmental data and providing it online through a set of services. One of the main purposes of the research subject is to develop a system, complying with applicable open standards that allows easy integration of data gathered at the wireless network level with current publicly available geo-location tools. A step by step documentation of the system is used, in order to demonstrate its build steps and the methodologies used in order to circumvent the difficulties found. This research is supported by a case study, where the system is used for real life purposes in the study of construction materials and their influence on thermal comfort zones.

This study shows that wireless sensor networks are particularly well suited for gathering environmental data in a spatial distributed context. Small micro-climatic changes can be parametrized using the developed tools. Integrating the evolution of different measurable variables with its geographic location enables a deeper understanding of how the location characteristics influence such variation. This research will also show that there are still issues pertaining link quality, and routing algorithms when using WSNs. The developed application performance is analysed in order to backup such statements.

It is possible to extend this system for other monitoring appliances. Further research could be made to extend this application for pollution monitoring, by adding air quality sensors and also noise sensors. Monitoring the buildings energy consumption, and estimating their thermal efficiency could also be achieved by extending the developed framework.

Acknowledgments

I would like to thank all the valuable insights, recommendations and support given to this study by a number of people and research groups. Professor Gil Gonçalves provided valuable feedback during all stages of this research, which greatly improved the overall research outcomes.

Several recommendations provided by members of the USTL group made it possible to develop this work taking into consideration interoperability questions and issues with existent solutions. Also some very useful and valuable advices were given towards the use of specific solutions. I would like to thank in particular José Pinto for his "crash course" on some of the tools and software libraries used, and also for his advices on the viability of some of the employed algorithms. I should also mention Eng. Alexandre Sousa for his practical advices on the build and assembly of the used hardware platforms.

During this study, the support and encouragement of friends and family, made it possible to withstand the occasional changes in ones social routine. Their presence and care often reminded me of life's true priorities.

The results from this research were positively benefited from the aforementioned supports. Any error or omission present in the resultant work is of the author's responsibility.

Carlos Eduardo A. Oliveira

"Good ideas are not adopted automatically.
They must be driven into practice with courageous patience."

Hyman George Rickover

Contents

List of Figures

List of Tables

Abbreviations

GIS	Geographic Information Systems
GPS	Global Positioning System
USART	Universal Synchronous/Asynchronous Receiver Transmitter
USB	Universal Serial Bus
LED	Light Emitor Diode
SPI	Serial Port Interface
ADC	Analog to Digital Converter
SMA	SubMiniature Version A
SQL	Structure Query Language
HTML	Hypertext Markup Language
PDF	Portable Document Format
Db	Decibel
KB	KiloByte
ADT	Abstract Data Type
ANDF	Architecture-Neutral Distribution Format
API	Application Programming Interface
CAD	Computer-Aided Design

Chapter 1

Introduction

There as been, traditionally, in many areas, the need for data collection or data sensing, on a broad array of variables. Most of the times, until recently, wired sensors have been used in order to analyse variables of interest. In industrial environments, protocols like MODBUS [2], CAN [3] and similar ones, over mediums like RS232 [4], RS485 [5] or even Ethernet [6], have been normally used to connect sensors to central stations.This poses, however, several inconvenients from deployment and management point of views. Sometimes the areas that needs to be sensed are un-easy to access and may be so wide or complex, that makes cable installation a daunting and expensive task. The situation aggravates itself whenever the deployment of sensors is temporary. Installing and un-installing arrays of sensors on industrial environments for short amounts of time can impact plant production, and the benefits from the data measurements may vanish when compared with the profit loss of conducting the relevant studies.

A typical example of this paradox is machine calibration conducted with vibration analysis. Prior to the setup of such vibration sensors it is often necessary to shut off the machines in question, to proceed with cable installation in order to wire the required sensors. Each machine status is normally analysed, in order to fix any deviations and increase performance. The time that the "machine" is stopped means, in many industries, loss of money. This loss itself may be enough to postpone or eliminate the study from happening. The following paragraphs show some examples of how wired sensors are being used to retrieve data in different contexts. The underlying disadvantages of using these type of solutions are emphasized.

Nowadays, energy consumption studies are still, most times, conducted using wired devices capable of measuring power consumption. To characterize a vast industrial area, looking for the source of potential energy loss, often needs allot of wiring. The preparation time, due to the nature and usual placement of the energy conducting cables, is laborious and expensive. This as made companies to pre-install energy counters in strategical areas during the construction of their plants in order to minimize the need of such studies. However to locate energy loss sources, it is often necessary to analyse in detail on the plant floor each device.

The environment and climate change are a topic of discussion that have motivated several studies. When we think about climate, climate change, and environmental issues, we often think in a macro-scale, instead of what we usually feel everyday. To analyse the impact made by mankind in the global climate we often recur to data that is recorded by weather stations scattered around the earth, that give a general perspective about climate and climate behaviour. There is often only average readings of certain variables' evolution in specific places, and those readings are considered representative of a wider area. Some fields of science acknowledge that there is a need to have a deeper analysis on subjects like micro-climate, micro-climate changes and its impact on human life and animal species. This can be done by taking less scattered readings on several measurable quantities. Several studies have traditionally used wired solutions, based on portable weather stations, some times expensive, in order to characterize volcanic areas, rain forests and iced continents among others. This places a lot of issues related with the power supply of that kind of hardware and how wide the area can those measures represent.

Human networks, and urban life are often parametrized by adhoc studies using "in-situ" observations by researchers and observers. These interactions between people, also with the surrounding environment and urban spaces is of extreme interest for both marketing companies and social behaviour scientific studies. How to reliably collect data about peoples lives and people's interactions, without interfering with their privacy has opened its own field of public discussion. City planners can use computerized city models to run and test new construction developments as well reorganizing existing ones. It would be possible to reduce power consumption, traffic jams and increasing the city's quality of life. All of these statements depend on a consistent and distributed data gathering platform. It is hard to imagine or calculate the amount of wiring, and installation time, needed to collect signals from sensing platforms, even if using pre-existent platforms like the telephone lines,cable or electrical power lines.

Home automation platforms have started to rely on previous conception and preparation of buildings, for the installation of proper cabling. Part of the home automation devices require communication with sensors distributed through the house. Some automation systems communicate using the powerlines. Still installation of small devices that measure light intensity or temperature, may need relocation, or can, due to the nature of its placement, hamper the use of the power source for communications, normally due to aesthetic reasons.

Many artificial intelligence algorithms work by analysing patterns and comparing it with current readings in order to take a "decision". Some artificial intelligent (software) agents are used in simulated environments through which data is fed. In real life, many systems use sensors, cameras gps (among others) as their data acquiring mechanisms, in order to sense and possibly act upon the world. Distributed computing systems can be used as a way to sense the world. This systems can work as a colony, silently registering and analysing data looking for determined or unknown patterns.

The previous paragraphs show the need for a distributed data gathering system, that can be used in a broad array of existing situations. These devices need to have short installation times, and similar reliability to already existent wired solutions. The mentioned facts resulted in the

development of wireless sensor solutions.

Even if one can gather all the information required for a certain application, there is still the need for a easy and flexible data storage and analysis tool. Normally software is build in a "case by case" analysis. Sometimes problems exist with software portability between different operative systems. Some require specific instructions or previous knowledge prior to its usage. A new approach is now appearing emerging. The aggregation of publicly available data on the world wide web, with data collected by sensing platforms is starting to be used. This delivers a more compelling and easier way to analyse data, by enabling users to use common and known tools. It also enables public data to be readily available everywhere.

This thesis was developed as part of the current activities from the Network Centric Control Systems Group (NCCS), of the Electrical and Computer Engineering Department at FEUP. The outcome of this research was added to the several solutions already developed by one of the NCCS research groups: the FEUP WSN Group [7].

The FEUP WSN Group has been developing solutions on the field of Wireless Sensor Networks: Monsense [8], Early Warning Fire Detection System, Aquatic Monitoring Solution, Distributed Noise Monitoring System among others.

1.1 Research Objectives

The main goals of this research is the development of a wireless sensor network application that can be integrated with current GIS or earth browsing software. Network management,data storage and data analysis are also accounted for.

The system was developed taking in mind a broad range of applications. This work takes into account questions like portability, software modularity and system integration. A clear, and scalable approach is also seen as a paralel objective.

1.2 Scope and Limitation of the Research

The sensing platforms described throughout this work are introduced up to a sufficient level of detaill, whenever needed. A similar approach is taken when describing technologies used during research. For instance it is out of the scope of this research a deep characterization of the underlying technologies used by sensors. Instead whenever needed mere equations describing the functionality of such devices are introduced.

The literature used during research is deeply connected with the pratical aspects of this work. It is not feasible to introduce all of the required theory behind the networking protocols used during development. Whenever possible a small description of the features used is made leaving external references for further analysis.

A great part of this research is in fact dedicated to the development of a system capable of reaching the proposed objectives. There are several software tools that were used to develop such system, and there are some external applications used by the system itself to produce the expected

results or objectives. Although knowledge about such applications had to be gathered in order to interact and use its functionalities, it is considered that it is not a research objective to explain in a detailed manner how this interaction with external applications is implemented. External references that were consulted in order to fulfil the required implementation tasks are displayed whenever necessary. The development process is instead documented in a more detailed manner, showing the research and implementation steps taken.

Time was a constraint present during the research and development stage. There is a trade-off between the research schedule and the number of aspects and functionalities researched or implemented throughout the work. Decisions were often made taking into account the devised research or project deadlines, although some of them might not be optimal, they fulfilled the requirements and objectives devised at each stage. Time constraints have also limited the case study, it was still possible to retrieve enough valuable data for analysis.

1.3 Document Overview

This work describes the development of a platform based on a wireless sensor network, which is able to publish gathered data on-line. The building of this framework will be discussed thoroughly through the entire document. The system test and actual deployment is later on discussed, together with findings on the test subject. This dissertation is composed by five chapters.

In Chapter 2, related works and current technologies are reviewed and discussed. Chapter 3, shows the methodology employed to develop the *Environmental Monitoring Services* system, by describing each of the system's layers. A case study in Chapter 4 shows the deployment and usage of the constructed system, together with the analysis of the data generated. In Chapter 5, a review is made about the overall performance and behaviour of each system layer. Findings and suggestions for further research conclude the last Chapter.

Chapter 2

Analysis of Relevant Literature and Related Projects

This literature review confines itself to specific subjects pertaining to the development of wireless sensor network applications, and the storage, treatment and display of data. It does a brief introduction to the current wireless sensor, state of the art applications. It is out of the scope of this review to introduce on a detailed level how the physical interfaces which compose the network nodes interact to establish communication.

2.1 Overview Of Wireless Sensor Networks

The recent development and miniaturization of micro-controllers has enabled new applications and systems to emerge at an increasing rate. These silicon devices allow perceiving the world as well as acting upon it in ways that were impossible a decade ago [9]. Nowadays it is possible to establish a network of small devices, designed to retrieve data from their surrounding environment, or acting on it, leaving only a small footprint of their presence. These nodes normally forward their readings to a sink node, usually called base-station. Many issues still exist pertaining network scalability, hardware reliability, ease of use and development tools. Real-life applications normally recur to specially engineered products designed to fulfill a determined function. Platforms can normally be adapted to a different array of sensors. The usual sensors that can be found to work out of the box with these devices are light, temperature, humidity, vibration and barometric pressure sensors.

> "computers are noticeably more widespread, smaller, and mobile. What we often overlook are the billions of computers around us that we never see. Over 10 billion microcontrollers ship each year, and they exist in unexpected objects. Even my skis have a small microcontroller in them." [10]

In [10], new applications in the field of pervasive computing are discussed. The author particularly emphasizes the fall backs of "sense and send" methodologies and encourages the wide-spread use of small remote devices with collaborative abilities. These collaborative abilities could be used to detect environment variations and act upon it. Another fact that is mentioned is that currently, to develop applications using these platforms, one must be capable of programming specific code to work with the various micro-controllers' platforms and usage scenarios. This normally requires some prior knowledge of circuitry and low level programming which is something that greatly limits the number of people capable to perform such task and hinders potential applications for the already existing technologies. A different approach is incited: the adoption of a well known programming language, to develop applications that run on these small devices.

There is an intrinsic, and apparently unsolvable problem when working with remote deployed devices: their power source. More often than not, the wireless nodes can not draw their power from a power grid. This is normally due to the remote nature, or difficult to reach locations in which these sensing solutions are used. Studies show that it is possible to scavenge energy from the environment in order to keep the sensor networks live.

> "Energy harvesting techniques can deliver energy densities of 7.5 mW/cm2 from outdoor solar, 100 uW/cm2 from indoor lighting, 100 uW/cm3 from vibrational energy and 60 uW/cm2 from thermal energy typically found in a building environment. A truly autonomous, "deploy and forget" battery-less system can be achieved by scaling the energy harvesting system to provide all the system energy needs." [11]

A detailed overview on energy harvesting techniques can be found in [11]. Several studies [12, 13, 14] show that it is also possible to use human movement to feed the nodes. The Seiko Kinetic [15] wrist watch is one of the examples in which energy is derived from body movement. Normally wireless nodes have different power states, that correlate to the functionalities which are enabled. Increasing production energy levels can be achieved by adding more harvesting sources. Rechargeable batteries should be used in order to avoid energy waste when energy production superseeds the node's energy consumption. When the energy derived from the surrounding environment is not enough to power the node, then the node can use the battery as its energy source.

In [16] a system capable of deriving power from solar energy is demonstrated. This system is able to power wireless network nodes using a small solar panel connected, through a charging circuit, to an array of capacitors and a backup battery. The charging circuit is controlled by the wireless nodes' software. It is demonstrated that if the node's (the telos platform [17]) full power duty cycle need is of 10%, then its operation lifetime is of 4 years.

During night time, there is still the need to recur to the rechargeable batteries.The limited battery recharge cycles are the only hindering factor of such systems. With the inclusion of large capacitors the need to recharge the batteries is reduced. Capacitors are usually employed in energy harvesting techniques greatly due to their cost, (almost) infinite recharge cycles and higher power efficiency. In most cases a combination of capacitors and rechargeable batteries is used. To characterize node power consumption, it is necessary to define each platform's components'

power state and energy consumption levels. Although power consumption levels vary from platform to platform, there is generally a pattern that can be found. When active, the transceiver is responsible for the most part (around 80%) of the nodes power consumption, followed by the micro-controller and sensors. Such fact needs to be taken into account when developing a new platform, or developing an application for the existent ones.

> "With current hardware technology, the most of the energy is spent by the radio transceiver. Besides the energy spent during transmission and reception of messages, also the energy consumption due to listening on the channel waiting for incoming packets is relevant (idle listening)." [18]

In [18] a radio duty cycle approach, together with a low level communication protocol, is devised in order to reduce the transceiver's power consumption. This work shows how to characterize the power profile of the transceivers in order to optimize their power consumption through the implementation of an alternative MAC protocol. This protocol makes the use of preamble messages, already used for radio sleep modes, to schedule communication between nodes. This approach works by sending a countdown time message in the preamble, allowing the receiving node to calculate when actual data will be sent through the communication channel; therefore the receiving node can remain sleeping until that calculated time. This strategy shows a maximum of 30% increase in battery lifetime.

2.1.1 Routing Schemes

The several nodes that build a network often need to establish routes for information to be delivered at base. This occurs due to the limited transmission range of each node. Several techniques have been developed in order to build such routes.

Routing protocols have a major role in wireless sensor networks. There is a large volume of literature and publications dealing with this subject. The routing algorithm can directly impact the WSN energy consumption. As seen in [19], different routing schemes may lead to different power loss, and different message loss.

A (non) routing scheme is data "flooding". Simply put, each node simply broadcasts each received message. This wastes energy, in a variety of ways but eventually, the message reaches a base station. Other variations are "gossiping" in which nodes calculate a random node within their proximity to send its messages.

Simple routing algorithms calculate the number of nodes between a giving node and the base station without accounting for link quality within each pair of nodes. Nodes periodically send their data through the routes to the base-station. This in theory works, but the variable behaviour of the link quality traduces itself in sporadic link loss, possibly leading to message loss.

In [20] the "expected transmission count metric" is used to find the maximum throughput rates in multi-hop wireless networks. This metric works by estimating the necessary message retransmissions from a given node to the base, taking link quality into account. This can then be

used to find the optimal route for message delivery, by assigning a network hierarchy to the nodes. This approach is used in [21] to establish the collection routes.

Other routing protocols use a data-centric approach. Sometimes a group of nodes in proximity may have similar reading from ones of its sensors, within a certain interval. Also, different nodes can have different power (or other resources) available. The several nodes negotiate with each other to share the data in which they are interested on. The nodes take into account their available resources. This negotiation eliminates redundant data. The base-station then negotiates with the network for a particular set or range of values from its network. With the redundant data eliminated, redundant transmissions are eliminated as well, and the valuable data evolves through the network until it reaches the base-station. This kind of approach can be seen in the SPIN (Sensor Protocols for Information via Negotiation) [22] algorithm and in the "Directed Diffusion" Protocol [23].

Ideally each node's power consumption should be the same. However in practice, some routing algorithms fail to take that into consideration. Some of the nodes end up being used more often for retransmission than others. This fact leaves the network power consumption unbalanced. If a critical node in an established path "dies", due to the fact it consumed all of its provided energy, then most protocols try to find out another route. However, nothing changes the fact that no future readings will be retrieved from the dead node. This as lead some researchers to develop algorithms that try to rearrange the routes in order to maximize battery lifetime evenly through the network [24].

Another approach is hierarchical clustering. Nodes are grouped onto clusters. Inside a cluster, nodes send their data to a cluster head or "parent". Nodes form a cluster by choosing the cluster parent that provides the best link quality. This parent node is similar to the other nodes, but needs to cope with the task of receiving and aggregating data from the nodes within that cluster. Cluster parents are the ones that establish routes to reach the base-station. This kind of routing as one apparent disadvantage: power consumption at each cluster parent node. In [25] this fact is overcome by periodically assigning the cluster parent role to a different node. Another study [25] has a similar approach but the nodes decide randomly when to become cluster parents. These techniques make energy consumption evolve homogeneously through the various nodes.

Some routing schemes take into account geographical information. By using the expected radio transmission radius, and each mote position it is feasible to define the possible transmission paths. Sometimes several nodes can establish redundant paths. Those path redundant nodes can shut off their radios, and the critical path ones listen periodically to the channel. Negotiation between the nodes establishes the listening time from the critical path nodes. The path redundant nodes communicate with the critical path ones in the devised times. Periodically the critical nodes are replaced with one of the redundant nodes, keeping load balancing. This kind of methodology can be seen in GAF (geographic adaptive fidelity) [26].

One different communication scheme is multi-path based routing. In multi-path based algorithms the nodes try to establish, not one, but many links to the base-station. This works well if the base-station is on the center of the WSN. There is an energetic cost by employing this methodol-

ogy, but the gains in redundancy may justify it, depending on the application. Lets imagine that the base-station can only communicate with a small portion of nodes from the WSN and that multi-path routing is being used. The messages from the remaining nodes will need to pass through the nodes the base-station communicates with, making them consume more power. This leads to load unbalancing, making those nodes "life expectancy" shorter. Many solutions have been proposed to deal with this subject, one passes by using different kind of power sources for the nodes that in theory have to stand a higher number of communications. A strategy employed in [27], uses the concept of multi-path routing employing multiple sinks (or base-stations). In this methodology the nodes try to establish multiple routes to the available sinks. These paths topology resembles a star-burst, due to the fact that the sink nodes are placed on the perimeter of the WSN. These sinks are then connected using other kinds of networking mediums. An application controls the data received at the multiple sinks and eliminates redundant messages from the nodes.

The previous routing schemes try to maximize reliability and optimize energy consumption. The issue of response time, combined with packet loss needs also to be addressed. For instance, some WSNs can be used to monitor a specific event. If a node detects such event, and the event data is critical, then it is expected that not only that information is delivered reliably, but it is also delivered in the shortest time possible. To address these issues, recent studies try to develop and apply "quality of service" (QoS) metrics, by creating new routing paradigms or modifying existent ones. One study [28] uses a multi-path approach together with QoS measures, that enables critical packets to follow the best overall routes and relays standard (non-critical) information to the remaining paths.

2.1.2 Hardware Platforms

Wireless sensor nodes (typically called motes) are available from several different manufacturers, mainly US-based. These include Sentilla [10],Crossbow [29], Dust Networks [30] and Meshnetics [31], among others. Some of these devices have been traditionally used as testbeds in wireless sensor network applications. These sensor nodes usually employ micro-controllers, radio transceivers and sensors. Each platform normally comes with an operating system.

2.1.3 Operating Systems

Although each of the platforms introduced in the previous subsection usually comes with pre-built software, it is possible in some cases through the use of open source development tools to build specific applications. These development tools often contain an operative system, or kernel, over which the developer can run its code.

On the proprietary side, Sentilla has created a Java Virtual Machine for their motes. It allows JAVA code to be run on the motes, by using their special developed packages. Meshnetics provides BitCloud to work with their motes. BitCloud is mentioned here because it is based on a software

stack implementing the ZigBee [32] feature set. It uses C language. They also provide "MeshNetics OpenMAC" which is an open source implementation of the IEEE802.15.4 [33] Media Access Control (MAC) layer.

TinyOs [34] is an operating system written specifically to work with wireless embedded sensor networks. It is open-source and has an active development group. This OS has a component based, application construction methodology. TinyOs provides a library with components that execute specific functions. There are abstract components for general functionalities, and specific components that are tied to work with specific hardware. The execution model is event-driven, and allows task-scheduling in a FIFO order. Thread support is not incorporated by default in the OS. Programs are developed in a language programming language called nesC. This language syntax is similar to C, but adds other functionalities and operators to specifically deal with hardware abstractions and event handling. In the end the application developed is statically linked with the TinyOs code, building a binary that can be used to flash the motes. TinyOs is distributed as a set of tools and libraries. These tools include compilers, applications for flashing different platforms and utilities for communicating with the platforms, for example through USB.

Contiki is also an open source operative system specially developed for memory constrained networked devices with small processing power. It offers a multi-tasking execution model. Contiki provides an IP communication stack for IPv4 and IPv6. The development tools provide simulators in order to improve and accelerate both the development and debugging stage. Contiki programs are written in the C programming language. Like TinyOs, it uses an event-driven kernel. One of its most attractive functionalities is the ability, by default, to load and unload programs at run-time on top of its kernel. This OS also provides lightweight thread support on top of the event driven kernel.

2.1.4 Applicable Standards

The radio transceivers used on wireless sensor nodes are usually compliant with the IEEE 802.15.4 standard. This standard defines the PHY and MAC requirements and definitions for personal are networks. The medium access mechanism used is carrier sense multiple access with collision avoidance (CSMA-CA). The standard includes also definitions for star and peer to peer topologies.

A set of standards [35] from the OpenGeospatial Consortium, is available for the display of sensor related data on web enabled platforms. This set includes, among others, the "Sensor Model Language" standard (SensorML) [36], the "Sensor Observation Service" standard (or SOS) [37] and the "Sensor Planning Service Implementation" standard (or SPS).

The "SensorML" standardizes the way to describe sensors and sensor systems, as well as its readings. The "SOS" standard describes the methods that should be present in a web service in order to retrieve information about sensor systems and sensors. Sensor Planning Service (SPS) provides a standard interface to collection assets (i.e., sensors, and other information gathering assets) and to the support systems that surround them.

The "Geographic Markup Language" (or GML) [38] is an OpenGeospatial standard that provides a set of XML encodings to represent and transport geographic data. GML provides coordinate reference systems, temporal reference systems and units of measure. GML tries to conforms its representations with other well established ISO standards.

KML (formerly Keyhole Markup Language) is an XML grammar used to encode and transport representations of geographic data for display in an earth browser, usually Google Earth [39] or Google Maps [40]. KML complements GML, and uses some of its geometric representations.

2.2 Related Projects

In [41], a long term outdoor experiment over a wide area is conducted to analyse the performance of a wireless sensor network. In the experiment several parameters were analysed, like the radio performance, and the stability quality of their provided links. Also the multi-hop environment is thoroughly tested.

In [42], wireless sensor networks are used in oceanography. The mechanical design, circuit design and networking software design are described together with the issues found.

In [43], a study on human interaction behaviour is conducted. Although not directly related with WSN technologies this study shows the potential enclosed in sensing technologies when applied in pervasive ways. According to this study, city planning needs to take into account the flow of urban life, and its interaction with new technologies.

Microsoft has developed a project [44] called "SenseWeb" to display sensor readings in its GIS mapping tool [45]. This project includes software libraries, to transform the data gathering by sensor platforms (which can include wireless sensor networks), and publish it on the "sensormap" webpage [46].

"OOstethys" [47] is a community of software developers and marine scientists that develop tools, using the latest applicable open standards, to display oceanographic information in the Google Maps Api [40]. Their project delivers an integrated platform to publish on their website information retrieved by oceanographic sensor platforms.

Several projects have been developed in [7] using wireless sensor networks for a variety of purposes. These include an "Early Warning Fire Detection System", the "MonSense Application - Planning, Deployment, Monitoring and Control of WSNs", a "Distributed Noise Monitoring System" and an "Aquatic monitoring solution".

The University of Melbourne as been developing a project called "The SensorWeb project" [48]. This project aims at developing a set of applications and software layers that interconnect sensing platforms with distributed computing grids. By integrating sensor networks with grid computing, it is possible to offload some of the typical tasks that usually run in centralized systems. The "Sensor Web" concept embraces a larger set of systems, in which wireless sensor networks can be part of.

In the Washington State University there is also a "Sensorweb Research Laboratory" which is developing Sensor Web systems and applying this technology to scientific and social applications.

The technologies being developed share the visions employed by the NASA in their own "SensorWeb" researches. NASA has been developing applications enabling the Sensor Web concept [49].

Chapter 3

System Analysis and Development

This chapter begins by providing the methodology used to reach the research objectives. Throughout this chapter the constructive parts of each level of the system are explained.

3.1 Methodology and Development Model

To reach the proposed objectives an incremental development model [50] was followed. During the planning stage, it was acknowledged that to allow further enhancements and system reuse it would be interesting to layer the system into subsystems, in its logical boundaries [51]. Thereby the system was divided onto different layers, each one with its own time-line, requirements and characteristics. All layers in the end work together towards the common objectives. These layers became effectively the development goals.

By using an incremental/iterative model throughout implementation, it was possible to refine some of the specifications, and add functionalities that were previously not defined. This allowed some degree of freedom as the system evolved, and also enabled early testing of the functionalities.

3.1.1 System Scope and Initial Assumptions

In order to develop a reliable and coherent wireless network, together with data aggregation services to serve as a mean to gather environmental data, several parameters need to be defined before actually starting the development. The immediate question that may arise is *"'where will the system be used?"'*. This work has in its essence the quest to build a versatile environmental data gathering platform that can be used in various scenarios, which were already mentioned in previous chapters. There were, however, several assumptions made beforehand in order to develop or find concrete hardware and software solutions. These assumptions will be presented on the following paragraph.

The WSN is targeted to observe environmental phenomena. The variables to be measured are light, temperature and humidity. Variation of the measured quantities is not expected to occur

in a timescale smaller than one minute. Data retrieved is collected at one node of the WSN, the base-station. Current hardware only allows a few hundred meters of wireless transmission range. To cover wider areas, it is expected that the nodes have the ability to establish a route so that their messages reach the base-station.

An important question is time synchronization, and message time-stamping between the nodes of the network. *"Where and When"* is crucial in data analysis, but so is sensor precision and transmission errors. Reliable storage and aggregation mechanisms of the gathered data must be designed to cope with information flowing from the network into the base-station.

Making data easily accessible through easy-to-use tools can make the user experience more compelling and productive. Furthermore, the ability to reconfigure network parameters can allow for multiple usage scenarios, where homogeneity or heterogeneity between the nodes, their assigned tasks and configurations may exist. Geo-spatial information about sensor orientation and node location allows precise data characterization; methodologies to define both are also explained in this chapter.

Three layers emerge from the previous statements: a data gathering layer (WSN), a data storage and network management layer, and a data/web service providing layer 3.1.

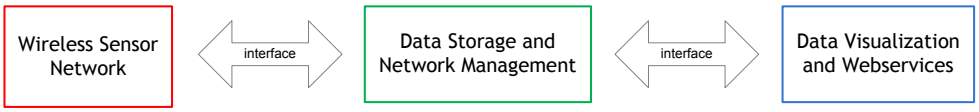

Figure 3.1: System Layers and Interfaces

Each layer is based on a combination of hardware and software designed to satisfy defined tasks. To apply a scalable methodology, comprehensive layer boundaries must be defined.

The following sections will detail constructive software and hardware parameters, when applied, to each of the layers mentioned.

3.1.2 Possible System Architectures

It is not premature to discuss the considered system layouts. In fact, although each layer can be developed separately, there is one common starting point: the flow of data from the wireless sensor network until the end-user. The definition of the data to be exchanged is derived from the expected system usage. In this case, to accurately characterize the measures and the measure procedure, the the information contained in diagram 3.2 needs to be exchanged.

Each node needs to be identified in order to distinguish between the several nodes' sampled data. The main parameter, expected to be exchanged by the nodes, is the data sampled by the platform's available sensors. Each node is placed in a specific location and has a configuration. This can be used to characterize the data to be stored.

Due to the limited storage ability of the current available motes, storing sampled data during long term deployments is not feasible. The data storage layer needs to be built on a different

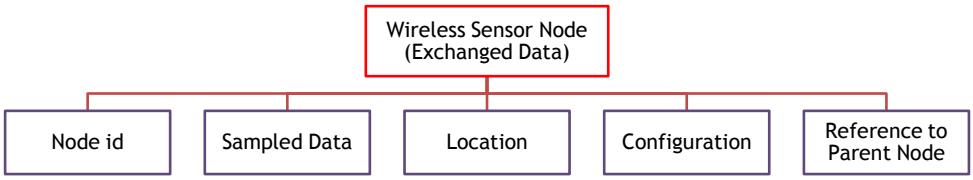

Figure 3.2: Information exchanged by nodes

platform. An application running on a common personal computer could do such a task while connected to the base-station node. The same application layer could allow for network reconfiguring (e.g. enabled nodes, time between samples), since it is already in direct contact with the base-station. On top of the central layer, the data visualization layer can be built making use of an available web-programming technology. A possible system layout can be seen here:

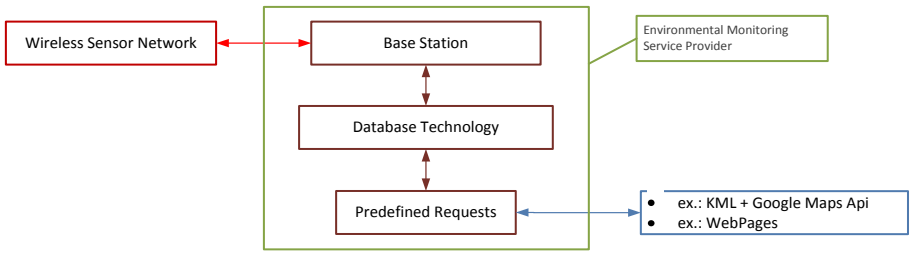

Figure 3.3: Possible System Layout

Although the system diagram 3.3 in could be used as a starting point for system development, it does not allow for an interactive user experience, or for convenient data analysis. The user experience is diminished by using predefined requests over the sampled data.

An alternative approach, compliant with the OGC set of standards [35] would need a dedicated server application with XML parsing capabilities. Also extra work would have to be done in order to implement the methods described in the Sensor Observation Service.

It should be noted that during research only a few applications were found that support OGC's Sensor Observation Service, being that the majority that currently exist only support a subset of the software methods described in the standard. Also SensorML [36] can be used to describe the sensor to account for "sensor describing requests".

As an example, in both diagrams KML [52] and the Google Maps API [40] were considered as constructive elements of the final layer.

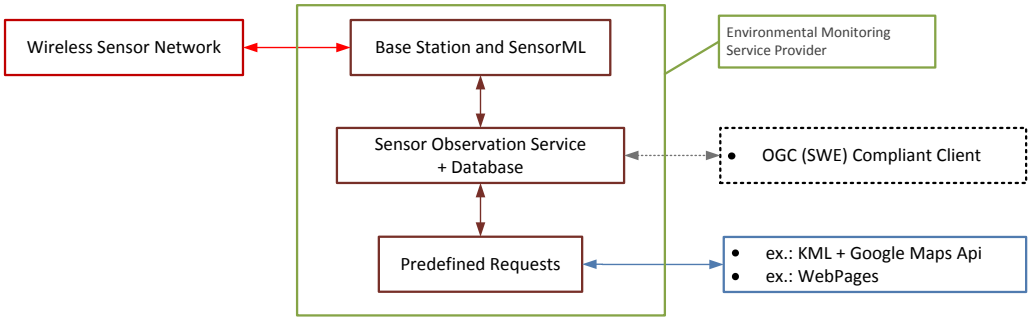

Figure 3.4: System Diagram with OGC SWE support

The architecture shown in figure 3.5 shows another possible layout, that takes into account personalized data requests.

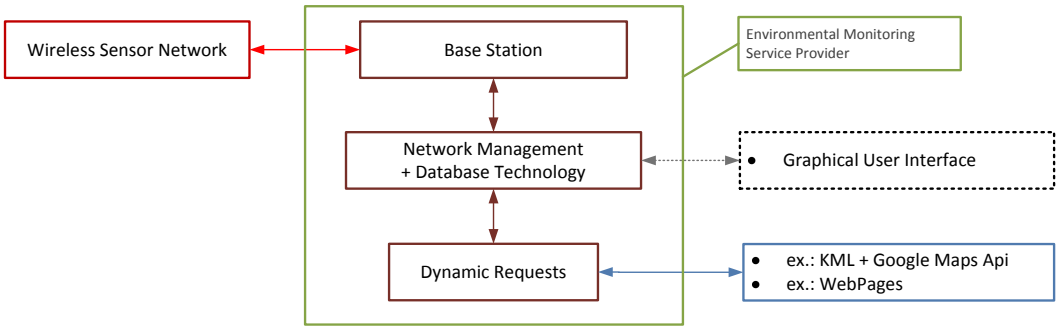

Figure 3.5: Adopted System Layout

3.1.2.1 System Concept

The diagram in figure 3.5 shows the guideline followed through the implementation of the system. With this arrangement, it is possible for the user to specify which particular data he/she wants, and how it should be aggregated before display. For instance, the user may request to read the average temperature at each hour from a specific set of nodes, in a specific time frame. It also includes a GUI to allow easy reconfiguration of the WSN subsystem.

This choice takes into consideration that each software layer can adapt well to a variety of operating scenarios (e.g. when the WSN is online or offline). Also, this kind of approach allows specific functionalities to be implemented, when needed, in a specific layer, without breaking previously existing features. Lets imagine that a new type of WSN is added to a system that was developed with the considerated guideline. The "Data storage and Network Management Layer" would need to be changed in order to connect to the new WSN but the data visualization should suffer no changes. And the same can be said to any new feature added to the data visualization

layer, when considering the remaining layers. This separation also helps the developer to better manage the development cycle.The following subsections contain the employed development algorithms for each subsystem.

3.2 System Requirements Analysis

An overview of the established requirements for the system can be seen in figure 3.6. This guideline was used during the planning stage to better understand each of the subsystems tasks. A refinement to these requirements is made during each subsystem development subsection.

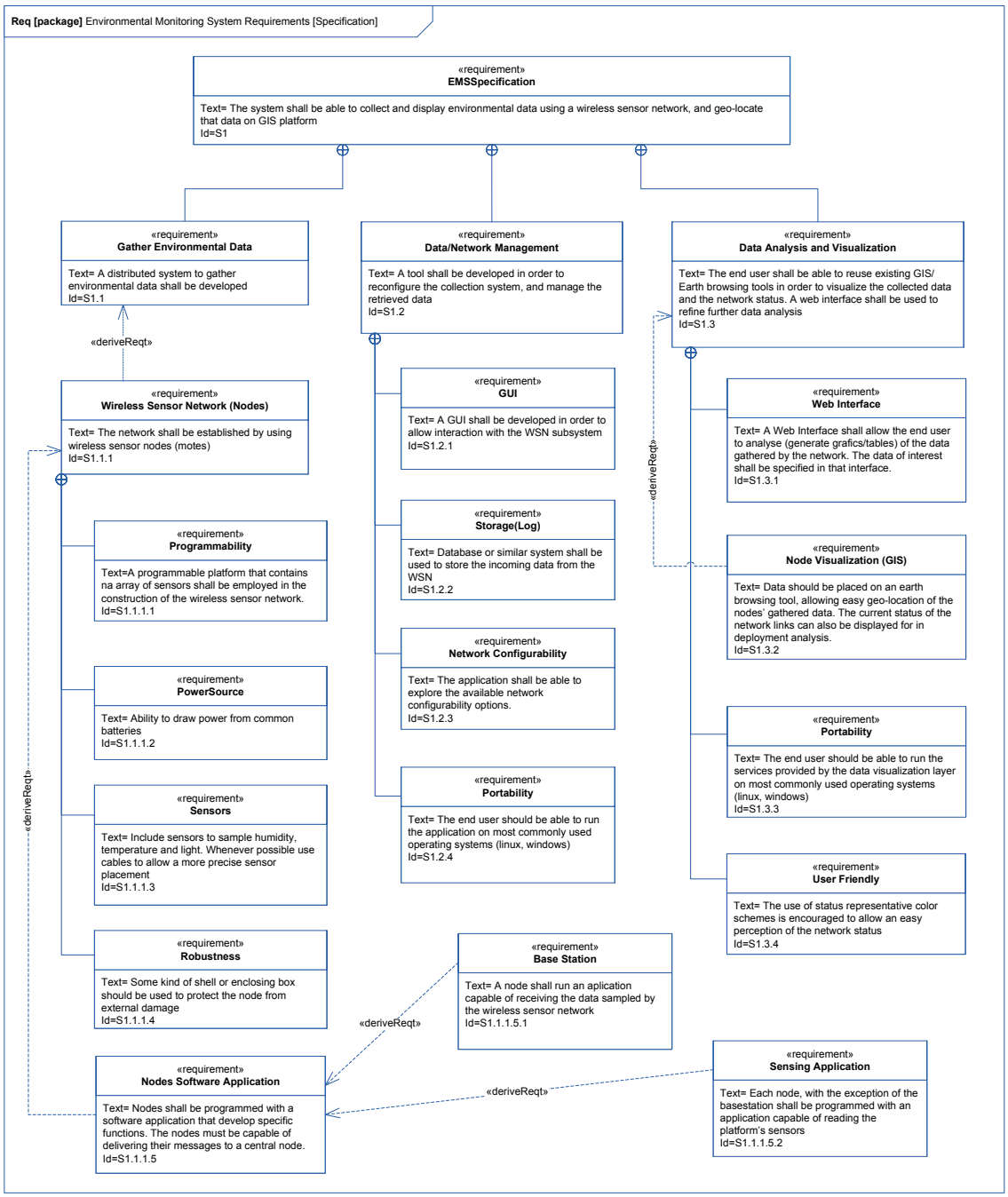

Figure 3.6: Considered Initial Requirements

3.3 Wireless Sensor Network

The wireless sensor network layer is composed by network nodes. These nodes contain a variety of sensors. The final sub-system is a symbiosis between hardware and developed software functionalities. This sub-section details the development of the WSN layer.

3.3.1 Requirements Analysis

A list of requirements was built in order to choose the appropriate hardware (wireless nodes) and software development platform, as well as proceeding with the design and implementation of the WSN. This list is composed by hardware and software requirements. Starting with the hardware requirements:

- Programmable platform.
- Stable and mature platform, focused on stability instead of state-of-the-art hardware.
- Ability to draw power from typical batteries.
- Include sensors to sample temperature, humidity and light.
- If possible, attach sensors to the node using cables (allows accurate sensor placement).
- If needed, build or use an existent box to surround each node, in order to prevent damage from possible accidents or misuse during deployments.
- Some kind of on board debugging mechanism or error monitoring mechanism should be present (for instance LEDs).

One of the main concerns to be considered when programming the motes is that they should be as configurable as possible, maintaining of course stability. This allows the end user to be able to fully explore the system without the need to recompile code. The software was developed bearing in mind the following requirements:

- Programmable state (active/inactive mote)
- Programmable sensors – which sensors sample data
- Programmable sampling time
- Programmable collect time
- Positional data permanent storage algorithm
- Sample time-stamping
- Sample data storing mechanism in case of system reset.

- Configuration data storing algorithm
- Programmable radio listening time
- Multiple wireless networks
- Scalable coding (allow easy integration of new messages and functionality)

3.3.2 Chosen Development Hardware

The TmoteSky [17] hardware platform will be used to construct the wireless network. This hardware is suitable when considering the established requirements, and it was readily available for usage. These motes have inbuilt sensors for light measurements, and optional temperature and humidity sensors. Easy integration of batteries for standalone operation can be made by soldering the appropriate connectors. Low power consumption modes can be activated to extend battery lifetime. Communication will be established between the motes using the 2.4 GHz radio frequency conforming with the IEEE802.15.4 specifications [33]. The maximum transmission rate expected to be achieved by each node's radio is of 250Kbit/s. Quite a few software development packages are compatible with this device.

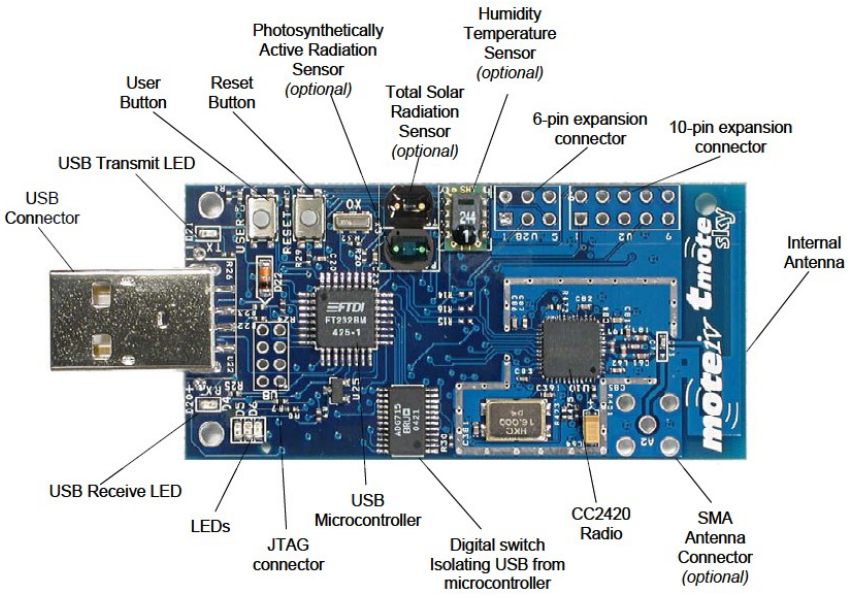

Figure 3.7: TmoteSky Platform - Picture from the TmoteSky data-sheet [17]

This platform is built with several different semiconductors instead of one single chip or package. It's a versatile design which allows for further enhancement through the use of the expansion connectors or even the repair of broken components if needed. The several components listed on the TmoteSky's data-sheet play different roles in the node.

At the heart of the platform there is the Texas Instruments MSP430 micro-controller. The MSP430F1611 is a 16-bit RISC CPU which has 48KB of ROM, 256 Bytes of internal flash memory and 10KB of RAM. The micro-controller data-sheet mentions that it has been specifically developed for applications were low power consumption is essential. It allows five low power modes (disabling specific functions) and an active mode. The return to the active mode from any sleep mode can be made through an interrupt event. With supply voltage of 3 Volts and clocked at 1 MHz, the MSP430 consumes a maximum of 600 μA. This micro-controller can be clocked up to 8 MHz, where it normally draws 4 mA; it supports two serial communication interfaces that can function as asynchronous UART or as SPI/I^{2}C interfaces. Two built-in 16-bit timers, 8 external ADC ports, 8 internal ADC ports, and dual 12-bit D/A converter are also available. All the other sub-components are connected to the micro-controller. The MSP430 is placed on the backside of the PCB, as it can be seen here:

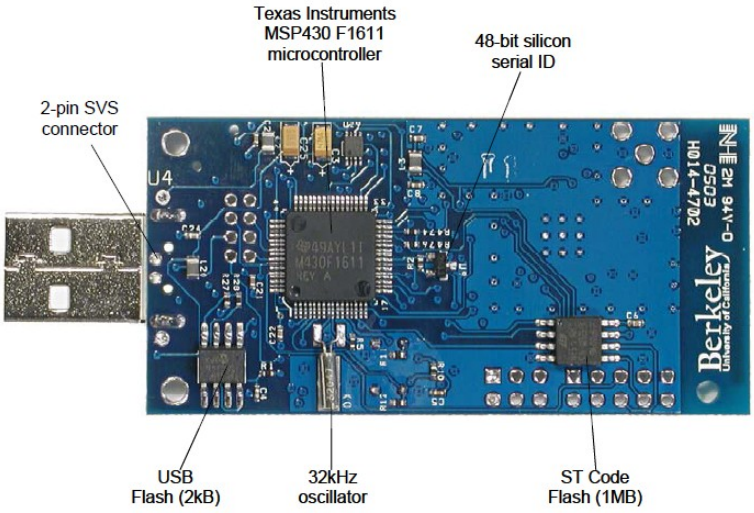

Figure 3.8: TmoteSky Platform's Microcontroller - Picture from the TmoteSky data-sheet [17]

The micro-controller could sample the sensors and send the sampled data immediately; there are however several drawbacks with this approach. In the case of a transmission failure or a system reset, the sampled data would be forever lost. The micro-controller flash memory capacity is sufficient for a limited number of samples. By using the external flash (ST M25P80 40MH serial code flash) it is possible for the nodes of the network to store a predefined number of samples for further collection. This allows the network to be sleeping or sampling most of the time and waking up at predefined time intervals in order to send the sampled data to the base-station. Such methodology greatly improves battery lifetime, and limits the loss of sampled data by only allowing deletion upon successful collection. The external flash has 1024KB of storage capacity; it can be also used for external code storage.

Wireless functionality is provided by an 802.15.4 compliant radio: the Chipcon CC2420. The radio is connected with the MSP 430 through the SPI bus. It provides eight selectable power states. The transmission output power varies with the selected power state. The CC2420 also provides a digital received signal strength indicator and a link quality estimator. These functionalities are the founding base for establishing routes for the collection of data through the network. The Tmote has an onboard antenna, and an optional SMA antenna connector can be soldered to the board. Expected ranges that can be achieved with the onboard antenna vary from 50 meters indoors to 125 meters outdoors. To improve the transmission range the Tmotes have been soldered with SMA connectors, and generic 5 dBi antennas have been used. The transmission ranges extended from 125 meters to 200 meters outdoors(open field) when using maximum radio power.

The Hammamatsu photodiodes allow light measurement. The two photodiodes have different tasks. The conversion of light intensity onto electric current varies with wavelength; the *S1087-01* has a spectral response range from 320nm to 730nm, which corresponds to the photosynthetically active radiation spectrum. The *S1087* responds to the full visible spectrum and infrared (320 to 1100 nm). Both photodiodes' package is made from ceramics, which does not allow light from the backside or the sides to reach the sensing area of the photodiode. Conversion to Lux from the output current can be achieved by analyzing the following graphic, which contains the response characteristics of the mentioned diodes and similar photodiodes:

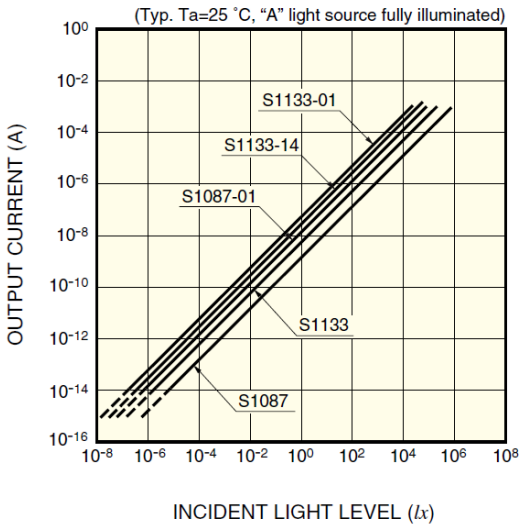

Figure 3.9: Output Current vs Incident Light Level - From the Hammatsu S1087 data-sheet [53]

Please note however that the graphic 3.9 corresponds to the short circuit current. In the Tmote the photodiodes are in series with resistors, resulting in a variable voltage (figure 3.10). This voltage is then read by the 12 bit ADC ports.

$$Vsensor = (ADCvalue \div 4096) \times Vref \tag{3.1}$$

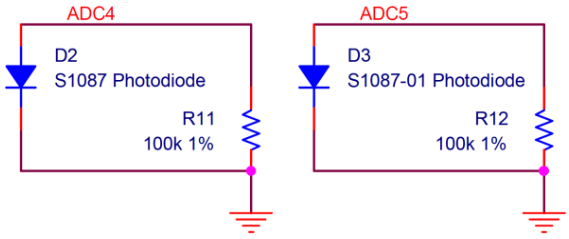

Figure 3.10: PhotoDiodes connected to MSP430 ADC Ports

$$Vsensor = (ADCvalue \div 4096) \times 1.5 \quad (3.2)$$

$$Isensor = Vsensor/(100 \times 10^3) \quad (3.3)$$

$$Isensor = ADCvalue \div (100 \times 10^3 \times 4096) \times 1.5 \quad (3.4)$$

$$LuxFullSpectrum = 0.625 \times ((10^4 \times ADCvalue) \div 4096) \times 1.5 \quad (3.5)$$

$$LuxPhotoActive = 0.625 \times ((10^4 \times ADCvalue) \div 4096) \times 1.5 \quad (3.6)$$

Conversion from the ADC count to the actual voltage is made using equation 3.1. The TmoteSky's ADC reference voltage is 1.5 Volts [17], which gives equation 3.2. Finally observing figure 3.10, the photodiode's current goes through a 100kΩ resistor thereby producing equation 3.4.

The S1087 linear response seen in figure 3.9 is easily written into equation 3.5 for the full spectrum and equation 3.6 for the photo active radiation. These formulas can be used directly on the microprocessor or in upper system layers.

This mote can be programmed through USB. The USB connector is normally used to exchange data with a personal computer. This feature allows any mote with the appropriate program to become a base-station by simply connecting it to a computer. The TmoteSky has expansion connectors that allow direct access to the micro-controllers ADC ports, I2C port or the USART. An array of 3 LEDs, a reset and a user interrupt button are also available.

The Sensirion SHT11 is a small size, relative humidity and temperature digital sensor. It comes fully calibrated from factory and it interfaces with the micro-controller through a bidirectional two wire serial interface. The data-sheet also mentions that this sensor has been specifically designed for platforms that require low power consumption.

At 25°C this device has a temperature accuracy of $\pm$ 0.4° C and a (relative) humidity accuracy of $\pm$ 3%. The temperature sensor has a typical resolution of 0.01°C, the response range starts at -40°C and ends at 123.8°C, whereas for the relative humidity one has a typical resolution of 0.03% RH and a response range starting at 0% and ending at 100% RH. Accuracy varies within each sensor's response range.

The SHT11 has a default measurement resolution of 14 bits for the temperature and 12 bits for the relative humidity. The SHT11 data-sheet shows the conversion formula between the raw values and values represented in SI units. The formula's coefficients vary with supply voltage and

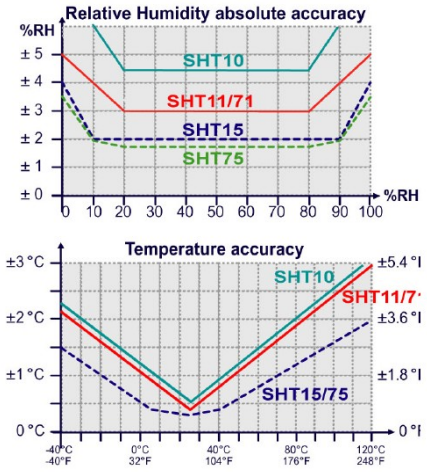

Figure 3.11: Accuracy Charts - From the SHT11 data-sheet [1]

resolution.

$$T_{Celsius} = -39.60 + 0.01 \times RawT_{Value} \quad (3.7)$$

$$H_{Rel} = -4 + 0.0405 \times RawH_{Value} + (-2.8 \times 10^{-6}) \times RawH^2_{Value} \quad (3.8)$$

$$H_{Comp} = ((T_{Celsius} - 25) \times (0.01 + 8 \times 10^{-5}) \times RawH_{Value}) + H_{Rel} \quad (3.9)$$

By considering the default resolution for both sensors, and that on the TmoteSky this sensor is supplied by 3 volts, the conversion formula for the temperature is given by equation 3.7. Similarly to this, the conversion formula for the relative humidity is defined by equation 3.8. The sensor's humidity reading needs to be compensated for temperatures significantly different from 25°C, which can be done by considering formula 3.9. Two internal sensors are available on the MSP430, an internal voltage monitor sensor and an uncalibrated thermistor. Both are sampled using the CPU's internal 12 bit ADC channel. The TmoteSky data-sheet formula provides the following conversion formulas:

$$Voltage_{Sensor} = (V_{ADCvalue} \div 4096) \times 1.5 \quad (3.10)$$

$$Internal_{Temperature} = ((T_{ADCvalue} \div 4096) \times 1.5 - 0.986) \div 0.00355 \quad (3.11)$$

The optional sensors come, by default, soldered directly on the TmoteSky circuit board. This poses a certain inconvenient, for instance when trying to measure surface temperature. To solve this problem, the SHT11 devices were purchased separately and soldered to small printed circuit boards.

Each PCB was then soldered to a cable. The other cable end is terminated by a female connector and the male connector pins were soldered on the mote. Better placement of the sensor is achieved with this methodology. The mote can be placed in a safe place (for instance protected from the rain) and the sensor (adequately protected) can still be near or in contact with the point to be measured.

All the components mentioned before are connected to the MSP430. By analyzing the diagram layout one can notice that the microcontroller SPI lines are shared between the CC2420 radio and the external flash. This must be accounted for in the software.

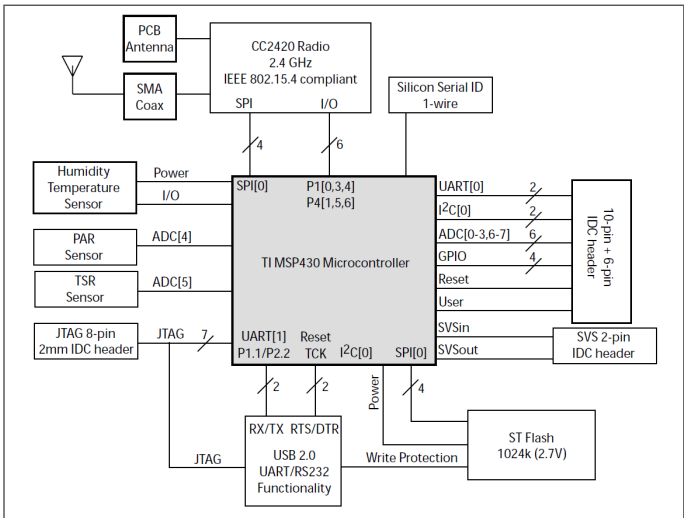

Figure 3.12: MSP430 Central Role

3.3.3 Chosen Development Software

Several operating systems are reported to work with the TmoteSky. These are not the common operating systems that usually drive standard personal computers. They are lightweight, highly power optimized, (usually) event driven operating systems. Although the term operating system is used, normally the application running on top of this layer must be written to work with a specific platform and perform the desired function. TinyOs, Contiki, Sentilla JVM, SOS and MantiOs all report to be compatible with the TmoteSky platform.

TinyOs was chosen to develop the required network functions. This choice was based on the developer's previous knowledge and familiarity with the C programming language model, with which nesC has some resemblance. Also this Open-Source OS has been used successfully in other FEUP WSN projects [7]. Many applications and examples come pre-built with the package, which allows a fast learning curve. The version used was TinyOs 2.1.0.

3.3.4 Developed Application and Algorithms

Before starting to write the application code, it is necessary to define the data types that are going to be exchanged between the nodes of the network and the base-station. To reach the desired goals, the samples, the information on the position and on the configuration of each node needs to be exchanged. The choice to include the position and configuration information as an exchangeable message enables the upper system layers to quickly characterise the network by *"asking"* the network its current status.

Knowing the sensor's number of bits allows one to define a data type to hold the sensed data. The sensors conversion formula could either be applied directly on each variable on the Tmote or the sensed value could be exchanged has a raw value. We chose to do the math outside the mote. Upper system layers can apply the conversion formulas on the sensed data or they can store both the raw sensed value and the converted value. By using this methodology the MSP430 is free to execute other tasks and it helps to lower power consumption.

As previously stated, both light sensors are connected to 12 bit ADC ports and the values retrieved from the Temperature and Humidity sensors have a 14 bit resolution. Each network message containing the samples also identifies the mote and the wireless network to which it belongs. TinyOs's Programming manual recommends that a network message structure should be defined as a new *"C structure"* type in order to be used in the application. This structure is then used to create the actual variable where data is stored. Another value should be indicated: the *AmType*. This value is used by the radio stack to differentiate incoming messages (similarly to ports in the TCP/IP protocol). This methodology also allows the application to easily construct the packets to be sent based on the defined structures. To exchange the sensed data by each node the *samplesMsg* structure was defined, containing the following fields:

Table 3.1: The samplesMsg Structure

netsamplesMsg [AmType_6]	
Type	Field Name
nx_uint16_t	Nodeid
nx_uint8_t	Wsnid
nx_uint32_t	Timestamp
nx_uint8_t	Activesensors
nx_uint16_t	Photopkt
nx_uint16_t	Lightpkt
nx_uint16_t	Temperaturepkt
nx_uint16_t	Humiditypkt
nx_uint16_t	Voltagepkt
nx_uint16_t	Internal_Temperature

The prefix *"nx_"* denotes a platform independent variable with big endian format. This feature allows this code to be reused in other platforms, without the danger of having specific hard-

ware storing these numbers in different formats; hence TinyOs is aware of the necessary conversions. The sensed values *photopkt*, *lightpkt*, *temperaturepkt*, *humiditypkt*, *voltagepkt* and *internal_temperature* are stored on unsigned integers with a 16 bit length.

The field *active_sensors* is an 8 bit unsigned integer which indicates which sample fields contain valid data (as the sensor was active when sampling took place). Each bit corresponds to a determined sensor. Using this later field allows having the same message structure for motes configured in different ways. When flashing the motes with a new program, TinyOs identifies each node with a user defined node ID. This node ID makes it possible to identify the origin of the messages. The *nodeid* field is an integer to be filled with the value corresponding to the node's ID and the *wsnid* is the node's network ID.

Table 3.2: The positionMsg Structure

positionMsg [AmType_14]	
Type	field name
nx_uint16_t	Nodeid
nx_uint8_t	Wsnid
nx_uint8_t	Version
nx_int32_t	Latitude
nx_int32_t	Longitude
nx_int16_t	Altitude
nx_int16_t	Rotational_x
nx_int16_t	Rotational_y
nx_int16_t	Rotational_z

A position message structure was defined to contain the geo-location of the mote. The proposed structure assumes that the motes will be stationary during their use. If the mote is considered to be moving then a timestamp parameter should be added to this message structure. Note that the TmoteSky has no built-in GPS module. To consider that the motes could be moving while sampling would increase the complexity of the application's algorithms without adding any functionality.

Most GIS systems accept coordinates in a variety of coordinate reference systems. Coordinates in decimal degrees with 6 digits after the decimal are sufficient for the geo-location of the nodes (less than 11 centimeters of precision). For instance, Google Maps [40] zooming abilities currently do not benefit from more than 6 digits after the decimal. Altitude encoding can be made in meters.

The tuple (latitude, longitude, altitude) expressed in the format already mentioned complies with the KML standard position attribute and with the available GML coordinate reference system's encodings. There are however some caveats in choosing the decimal degree format. Not every platform supports double value encoding and if they do support it, its internal representation may vary between different platforms.

TinyOs does not allow easy conversion from the Double type to pre-established types, like with the Integer case. Ultimately this would lead to the inability of exchanging messages between different platforms. Using 32 bit signed integers for storing the latitude and longitude solves such problem. The signed type *int32_t* has a 10 digit precision, which is able to accommodate the latitude and longitude values. Upper system layers must then be aware that there are 6 digits after the decimal and apply the appropriate conversion.

The *version* field is a parameter used for position version control. When a deployment ends, resetting the device can assign a default location to the mote's position variable. By (arbitrarily) assigning version values greater than 100, the mote's assigned position is made persistent, surviving a system reset. By assigning version values below 100, on the node's reset, this position defaults to a predefined position. The algorithm responsible for this functionality is explained later on in this subsection.

Three more fields were added to specify the orientation of the mote: yaw or *rotational_z*, pitch or *rotational_y* and roll or *rotational_x*. These fields can be used to better characterize the motes sensed data. For instance, the light sensors are sensitive to the mote's orientation. In the case of an outdoors deployment, if the mote's enclosure casts a shadow over the light sensor, its light sensor readings will be completely different from a mote placed in a similar location but with the sensor directed towards sun light. Consider the following reference frame *x'y'z'*, where *x'* is an axis pointing in the direction of true north, the *y'* axis points east and the *z'* axis points down towards the earth center of gravity.

Considering the lines defined by easily seen elements on the motes, the mote's *xyz* reference frame was defined. The *x* axis corresponds to the line perpendicular to the plane defined by the TmoteSky PCB or the box cover; it points outwards from the box through the box cover and it intercepts the line defined by the antenna. The *z* axis corresponds to the line coincident with the antenna. The *x* axis is perpendicular to both this lines. Orientation can be seen in figure 3.13.

Consider that the mote's reference frame shares the same origin with the earth's reference frame. Any rotation of the mote's frame in relation to the earth's frame can be made by using a sequence of yaw, pitch and roll rotations (transformation also known as Trait-Byron rotations). Each angle's transformation is applied on the obtained reference frame, being that the yaw is applied on the *z* axis, the pitch on the new *y* axis and roll on the latter obtained *x* axis. A way to define the yaw, pitch and roll angles is, with the mote placed on site, to measure the necessary rotations for the mote's reference frame to return to the earth's reference frame.

However during actual deployment this is surely not applicable. One simple way to speed up this measurement process is to try to align the mote's axis with the earth's defined axis. For instance the mote's *x* axis could be positively aligned with the earth's *y'* axis, *z* with *z'*, and y negatively aligned with *x'*. The yaw angle would then be 90^{o}, pitch would be 0^{o} and roll 0^{o}.

The alignments between the earth's axis and the mote's axis can be achieved by using a magnetic compass. Using gravity, a simple rock and a wire can do the trick for the *z* axis. The angles are stored in 16 bit signed integers. On bootup each mote creates a variable from this message type

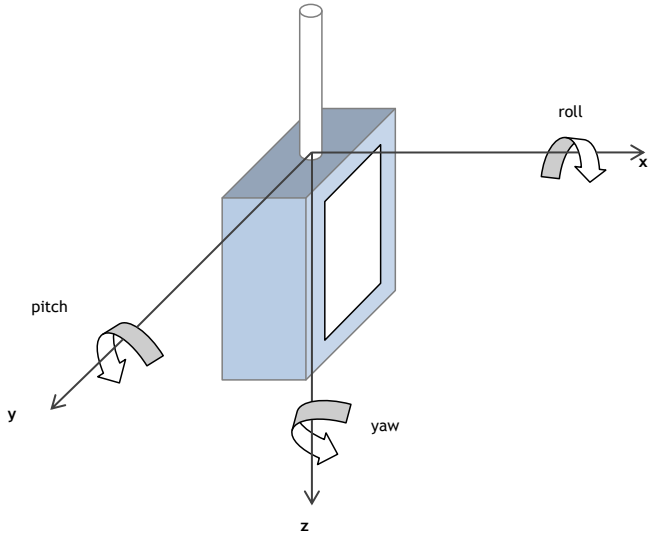

Figure 3.13: Definition of Rotational Axis

with a predefined content; specific routines are responsible for the reception and configuration of each mote's positional variable.

The ability to reconfigure the motes is made possible by adding a configuration message to the network. This configuration message holds the data that (yet to be discussed) algorithms use to decide which actions to take, enabling or disabling functionalities, and configuring temporal parameters during run-time. Each mote has its own configuration. Similarly to the previous messages, a *C* structure was used to store and share each node's configuration. By default, predefined values are assigned to this variable on boot-up. Specific code routines deal with the task of receiving a new configuration and updating this variable on each mote.

Table 3.3: The setupMsg Structure

setupMsg [AmType_10]	
Type	Field name
nx_uint16_t	Nodeid
nx_uint8_t	Wsnid
nx_uint16_t	LPLvalue
nx_int8_t	Version
nx_int8_t	Sensors
nx_int16_t	Collect_time
nx_int16_t	Sampling_time

The version parameter has the same functionality as the one described for *positionMsg*. The sensor's field variable is used to configure the active sensors. The *collect_time* and *sampling_time* are variables that store the time between collections and the time between samples. The *LPLvalue*

is a field created to store the radio low power listening time interval. Additionally defined message structures will be opportunely explained during this section.

A TinyOs application works with the motes hardware through software abstractions called "components". These components usually provide interfaces so that the software can execute specific functions on the hardware. This scheme allows the code to be reused in different platforms. For a deeper view on this subject please refer to the TinyOs documentation [54].

The application called *MoteSensingC* was developed. This application is composed by several source code files. The file *datatypes.h* contains the message structures already mentioned. The file *MoteSensingC.nc* contains the code developed to interact with available components in order to perform the required functionalities.

Listing 3.1: Configuration Code Example

n	Code
1	#include "OptionalSomething.h"
2	configuration MoteSensingCAppC { }
3	Implementation {
4	Components example as renamedexample;
5	//...
6	MoteSensingC.some_interface -> renamedexample;
7	//...
6	}

The components used from the TinyOs library are linked to the *MoteSensingC.nc* file through a descriptive application configuration file, arbitrarily named *MoteSensingCAppC.nc*. That file follows the logic seen in Listing 3.1. The first line optionally includes a header file, for instance containing some user defined types. The second line indicates that this is a configuration source file.

By using the "components" keyword the compiler will use the given component, in this case *example*, and optionally rename it. The renaming functionality enables the same component to be created and used multiple times. In the sixth line, an interface is used to provide the component's functionality to the source code being developed. The special operator -> executes the "wiring" between the interface available on the *MoteSensingC* and the *example* component. Interfaces are the gateways between components. They provide standardized ways for the components to interact and take advantage of each others functionalities.

The nesC language actually interprets the *MoteSensingC.nc* file as a new component. For instance, the reading and writing to the flash is achievable by "wiring" the STM25 flash TinyOs component to the application source code but the reading and writing to the flash by itself is not sufficient to produce the required storage functionalities; there is the need to write specific code to differentiate the storage of samples from the storage of configuration data. The components

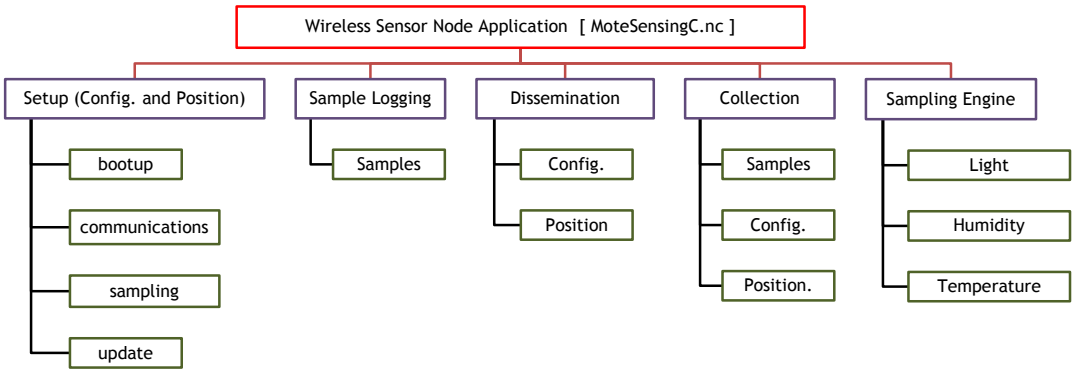

Figure 3.14: Division of Source Code Sections

used, and their respective "wiring" to the *MoteSensingC.nc* source code file, will be introduced whenever necessary in the pages that follow.

Similarly to the TmoteSky hardware subcategories, the node's application can be divided into sections of code that deal with specific functionalities. These code sections use certain components plus user developed code to achieve a determined goal. The diagram in figure 3.14 identifies each code's developed subsection. This division in subsections was made in order to clarify the implemented algorithms. In the end, the subsections will actually communicate with each other by calling routines in order to perform a specific action. The setup code section is built from a series of routines that deal with configuring the mote. TinyOs has a "special" component called *MainC*. This component provides an interface which triggers an event code routine, signaling that the mote has booted up.

Listing 3.2: Configuration - Code added to MoteSensingAppC

n	Code
1	#include "datatypes.h"
2	configuration MoteSensingCAppC { }
3	Implementation {
4	...
5	components MoteSensingC;
6	components MainC, LedsC;
7	components new ConfigStorageC(VOLUME_CONFIGMOTE);
8	...
9	MoteSensingC.Leds -> LedsC;
11	MoteSensingC.Boot -> MainC;
12	MoteSensingC.Config -> ConfigStorageC.ConfigStorage;
13	MoteSensingC.Mount -> ConfigStorageC.Mount;
14	}

The MainC component is wired to the Boot interface (line 11 on Listing 3.2). The *boot.booted*

event is launched once the mote is ready to run the application. The LedsC provides routines to trigger de leds, which is useful to signal the status of the boot up process. During the setup phase the leds are used to identify configuration errors. The ConfigStorageC enables reading and writing new configurations from/to the flash. The code developed to start the motes peripherals is purely sequential.

The nesC language is not object oriented, however the components already mentioned provide methods. Also the component wiring seen through out the configuration file resembles object instancing. Therefore UML diagrams will be used to demonstrate some of the developed functionalities. For instance the boot sequence behaves according to the UML sequence diagram in figure 3.15.

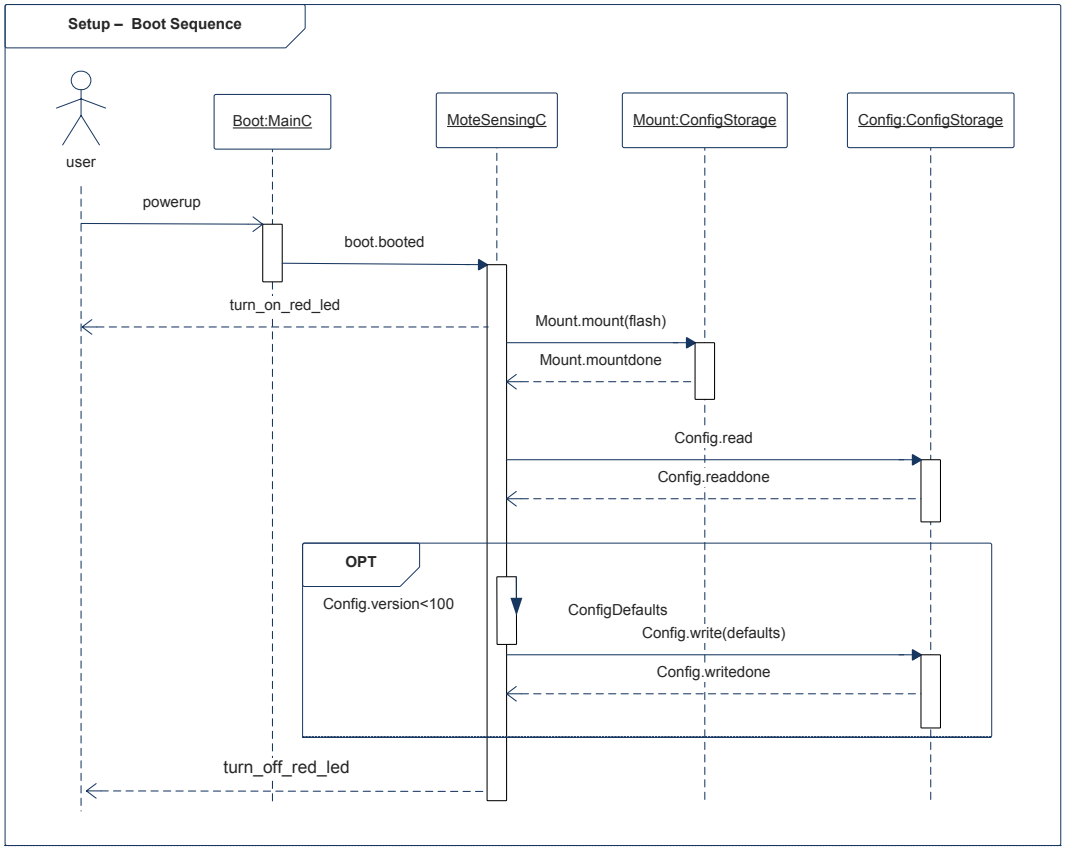

Figure 3.15: Powerup UML Sequence Diagram

When the *boot.booted* event is signaled there is a call to the component *ConfigStorage* in order to prepare the flash for usage. Once the flash is ready, the configuration is read to a global variable and so is the stored position. The global variables, containing the latest stored *configMsg* and *"positionMsg"*, have their version field compared to *100*. If it is less than *100*, the global variables are reset to defaults and so is the flash. Note that on the UML diagram in figure 3.15 the MoteSensingC is considered an entity.

The flash is divided into volumes, much like partitions in common operating systems. In the developed application the flash was split into two volumes, a *config_volume* and a *log_volume*. The *config_volume* stores the latest received configuration and position. The *log_volume* stores the samples until collection.

Upon completion of the boot up sequence, communications are set up to establish a link with the neighbor motes. The *ActiveMessageC* component, which is wired to the radio control interface, is used to init the radio stack. Two TinyOs components are then used to establish communication with the base station: the *Collection* component and the *Dissemination* component. Listing 3.3 shows the components used to establish communications.

Listing 3.3: Communications - Code added to MoteSensingAppC

n	Source Code
1	#include “datatypes.h”
2	configuration MoteSensingCAppC { }
3	Implementation {
4	...
5	components ActiveMessageC, CollectionC, DisseminationC;
6	...
7	MoteSensingC.RadioControl -> ActiveMessageC;
8	MoteSensingC.DissControl -> DisseminationC;
9	MoteSensingC.RootControl -> CollectionC;
10	...
11	}

The ActiveMessageC component provides interfaces for dealing with messages; it also provides routines to create packets with a low level of abstraction and to send them through the radio to a specific node, if the node is in range.

However, the Dissemination and Collection components also provide methods for packet creation based on the established message structures, but with the advantage that the communication route is established by underlying algorithms. The *Dissemination* is a TinyOs component that enables the nodes to share values among the network, thus the term *dissemination*. Its goal is simple: to maintain coherence between the disseminated value among the network. One node can play the role of the root disseminator. There is no route establishment. This component is based on the "Tricle" [55] algorithm. In "Trickle" the nodes exchange information about their current data (the value meant to be consistent among the network nodes). This is done by "gossiping" that information periodically. If a mote hears someone "gossiping" about old data, it sends the new data to the "gossiper". Gossiping times are automatically when new values appear on the network.

The dissemination component was primarily used to disseminate configuration and position messages. To add to the application the notion of several disseminated values, the components in the listing 3.4 need to be added to the configuration file.

Note that the component renaming is arbitrary. The component’s arguments indicate the type

Listing 3.4: Communications - Disseminated Values - Code on MoteSensingAppC

n	Code
1	...
2	components new DisseminatorC(configCTRL,0x1239) as Diss16C;
3	components new DisseminatorC(setupMsg,0x1234) as Diss17C;
4	components new DisseminatorC(positionMsg,0x1235) as Diss18C;
5	...

of message being disseminated by that *"instance"* of the component Disseminator and an exchange ID. This ID allows for example the creation of several *disseminators* based on the same value type. Here it was used to distinguish each dissemination group. To allow heterogeneity of configurations and obviously positions between the motes, the *node_id* field shown in the *setupMsg* and *postionMsg*, was used by each mote to check if the new disseminated value should become its new internal configuration.

The unmentioned *configCTRL* message type, seen has a component creation argument in source listing 3.5, is a secondary message that contains a nodeid parameter. This message is disseminated in order to ask the motes their current configuration. The inclusion of a broadcast ID enables the easy reconfiguration of the entire network, if needed. The broadcast ID was arbitrarily defined as *254*. Whenever a new *setupMsg* arrives, the *"Setup.valueChanged()"* routine is run. This routine checks if the mote is not locked on sending. If this the case, it then verifies if the *setupMsg* node ID field matches its own ID or the broadcast ID. At this stage, the update routine stops the sampling process, loads the new configuration, stores it on the flash and restarts the sampling process. This process can be modelled by the activity diagram in figure 3.16. The *positionMsg* update process is identical to the configuration update.

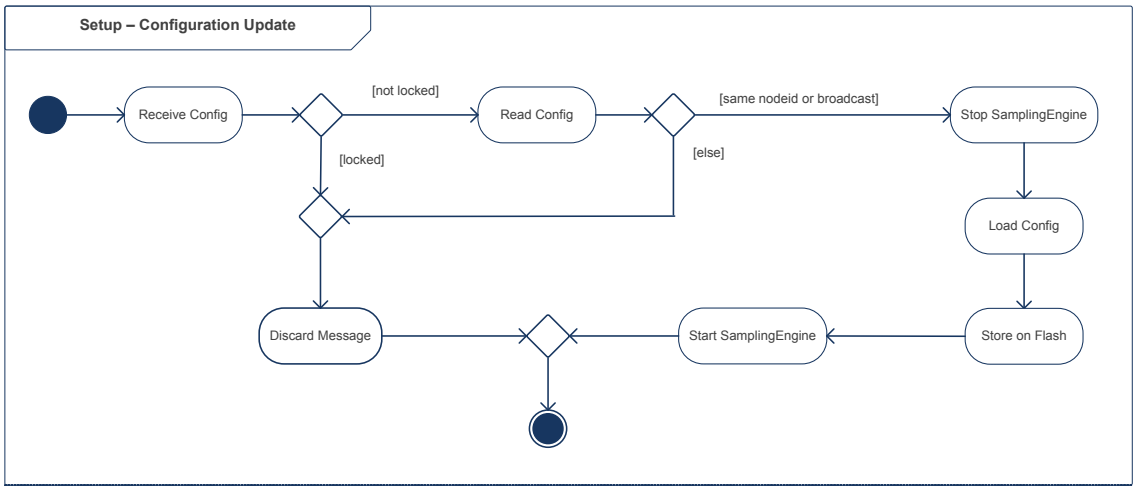

Figure 3.16: Configuration Activity

All transmissions from the motes to the base-station use the *CollectionC* TinyOs component.

This component establishes routes to the base-station according to each node's *expected message retransmission parameter* or ETX [20]. That parameter takes into account the number of motes the message needs to go through before reaching the base station and the overall link quality between the mote's radios. For instance, if the message needs to pass through 3 motes to reach the base station then the ETX will probably be 3, if link quality is optimal between each pair of motes. The nodes compute all routes and choose the one with the smaller ETX. The three described message types were added to the collection tree. The wiring to the MoteSensingC.nc file will be from now on omitted.

Listing 3.5: Collection Trees - Code on MoteSensingAppC

n	Code
1	#include "datatypes.h"
2	configuration MoteSensingCAppC { }
3	Implementation {
4	...
6	components new CollectionSenderC(0xea) as SampleSend;
7	components new CollectionSenderC(0xeb) as PositionSend;
8	components new CollectionSenderC(0xec) as SetupSend;
9	...
14	}

The *ColectionSenderC* component is used to provide *Send* interfaces for each message. The argument on the *ColectionSenderC* component creation is an identifier that allows the base-station to filter the message received from the *CollectionC* receive interface and to process it. The dissemination interface can be seen has the base-station sending mechanism. It tries to send the disseminated info to all the nodes. The sequence diagram in figure 3.17 shows a typical configuration and position request from the base station.

The Base station sends to the network a *configCTRL* message. Once one mote on the network finds that the *confiCTRL.nodeid* field is equal to its own nodeid, it sends its position and configuration info, through the collection component, to the base station. The collection component acts has the base station's receiving channel.

To sample the sensors, based on the configuration established on each mote, and to send the sampled data, a variety of algorithms were implemented. This code section was called *Sampling Engine*.

The *Sampling Engine* is composed by several TinyOs components: Timers, Sensor components and Storage components. The two instances of the timer component were called *Sampler* and *CollectionTimer*. The *Sampler* timer is initiated right after the boot up sequence. It is configured to launch an event periodically. The period is defined by the *setupMsg.sampling_time* field. Similarly the *CollectionTimer* is launched with the period contained in the *setupMsg.collect_time*. One second corresponds to 1024 units in each field. Upper system layers must be aware of this conversion.

The routine that listens to the *Sampler* firing event will sample the sensors and store the sampled content in the appropriate variables defined in the *samplesMsg*. This routine uses the value

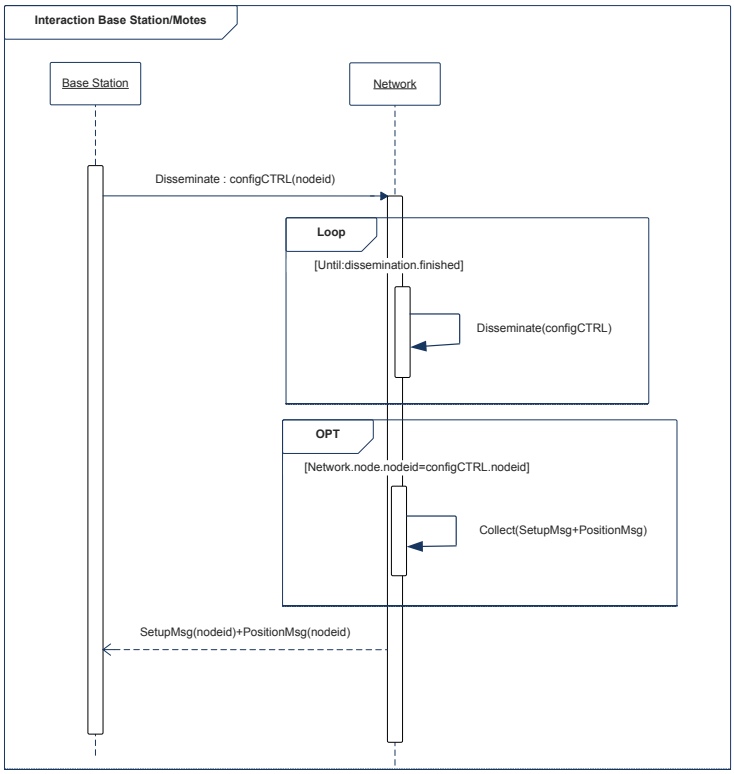

Figure 3.17: Interaction between Base Station and Motes

stored on the *"setupMsg.sensors"* field to decide which sensors to sample.

Each sensor has a TinyOs component that deals with the proper hardware abstractions in order to read the value. Once a read is issued on a sensor component, an event is later on triggered, informing the application that the read is complete and providing the sampled value for the given sensor. Each sensor sample is stored in its corresponding *samplesMsg* field.

Listing 3.6: Sensor Components - Code on MoteSensingAppC

n	Code
1	#include "datatypes.h"
2	configuration MoteSensingCAppC { }
3	Implementation {
4	...
5	components new TimerMilliC() as Sampler;
6	components new TimerMilliC() as CollectionTimer;
7	components new SensirionSht11C() as HumidityTempC;
8	components new HamamatsuS10871TsrC() as TotalSolarC;
9	components new VoltageC() as InternalVoltage;
10	components new HamamatsuS1087ParC() as PhotoActiveC;
11	components new Msp430InternalTemperatureC() as InternalTemperatureC;
12	components new LogStorageC(VOLUME_SYNCLOG, TRUE);
13	...
14	}

In the end the *samplesMsg* is stored on the flash, namely on the *SYNC_LOG* volume using

the functionalities provided by the *LogStorageC* component. Once the *CollectionTimer* fires, the samples are read from the flash and sent to the base station through the samples collection tree.

Storing samples before sending them enables the radio to be in a low power listening mode most of the time. The radio, when sending messages, is responsible for most of the node's power consumption. If the samples were sent immediately after sampling, then the neighbour motes, to ensure a successful message delivery to the base station, would themselves have to turn on their radios.

With a small sampling time, this could escalate on to a situation where the radios could stay on retransmitting neighbour sample messages and sending their own samples for a larger amount of time than what would actually be necessary.

The solution encountered to reduce the power consumed by message retransmission is a network sample collection duty cycle, based on the firing of the *CollectionTimer*. If the motes are all configured in the same way, then they start sending the stored messages at approximately the same time. If the mote is still busy retransmitting, queues on the CollectionC component store each received message for retransmission. This optimizes the usage of the radio.

To allow heterogeneity between collection times and to attend to the base station requests, the radios will still be in a low power listening (LPL) state. The component that provides the sleeping functionality is the CC2240ActiveMessageC. This component allows a radio listening duty cycle to be established.

In the implementation developed, the radios were configured to check the transmission channel for activity every 300 ms. The sending node must be aware that its neighbours only check for activity on the channel once every 300 ms and create a message preamble long enough for the receiver node to acknowledge transmission. The bigger the LPL value, the less responsive the network becomes when the base-station tries to reconfigure it.

Consider the following worst case scenario on a route with a depth of 5 nodes. To reach the 5^{th} node with a *configCTRL* message, with all radios checking the channel at exactly the same instant, it could theoretically take longer than 5 times 300ms for the message to reach the last node.The sample collection sending duty cycle feature working together with the LPL duty cycle improves power savings when compared with the regular listening approach.

There was, however, more than one problem found with applying the LPL technique. The LPL component is still not intertwined with all sub components. For instance, the TinyOs dissemination component is not aware if the developed application is employing the LPL component. The dissemination component tries to broadcast the message without the use of a preamble. This would end in the node trying to disseminate the requested value at a predefined retransmission rate, due to failure. This would eventually disseminate the requested value, but it was certainly a less than optimal solution and quite unreliable. Some "precise" changes were made on the implementation of the dissemination protocol in order to eliminate such problems. These changes involve adding the LPL component to the dissemination configuration file (in table 3.4), and also the appropriate preambles to the disseminated messages (table 3.5).

Table 3.4: Code added to the Dissemination Component Configuration

DisseminationEngineP.nc (TinyOs 2.1.X)	
Action	Code
Added	components CC2420ActiveMessageC as LPLProvider;
Added	DisseminationEngineImplP.LPL -> LPLProvider;

Table 3.5: Code added to the Dissemination Component Implementation

DisseminationEngineImplP.nc	
Action	Code
Added at Line 59	uses interface LowPowerListening as LPL;
Added at Line 126	call LPL.setRxSleepInterval(&m_buf, LPL_VALUE);
Added at Line 159	call LPL.setRxSleepInterval(&m_buf, LPL_VALUE);

One issue yet to be discussed was time syncing the motes with the base station in order to timestamp each *"sampleMsg"* stored on flash. TinyOs provides a *timesyncC* component intended to translate each mote's internal watch, a 32KHz counter running since power-up, to a global time. This component time syncs the motes, by taking into consideration the possible internal clock deviations. Initially, the motes would randomly crash with the *timesyncC* component enabled. This seemed to indicate that it suffered from the same problem as the dissemination component. However, with the same correction as the one applied to the dissemination component the motes would still crash.

Relaxing the precision and the retransmission of the component's time syncing messages did not solve the problem either, but it did help increase the time between crashes. There are no apparent explanations for this fact, yet one may speculate that the stress imposed on the microcontroller by that component in conjunction with the features already demonstrated would lead to the crash.

One simple approach that could be used, would be estimating the time of a given node's sample. This could be done by incrementing a counter field on each stored message and using the information about the time between samples. However this method considers that the transmission time is instantaneous. This can work but with an undetermined and unbounded error

To keep the project schedule on track, a custom algorithm was built. Project requirements specify that the application is intended to register variations, on measured values, in a timescale not greater than one minute. If several samples are taken during one minute, even if their time stamp value is not accurate, by averaging those samples the overall deviation from the one minute interval is minimized. To do so, the dissemination component was used, to periodically send a *timeMsg* with the base station time to the network. This *timeMsg*, on the motes, is incremented on every *Sampler* timer interrupt, with the value *"setupMsg.sampling_time"'* and the *sampleMsg* is stored on the flash with the new time value.

The figure 3.18 shows how the time sync message is disseminated to an arbitrary mote. Note that there is a delay represented by *T'*, once *Ti* is disseminated. *Ti**, which equals *Ti*, is received and becomes the new mote's time value, that will be incremented periodically and then used to timestamp the samples. The time of interest *"Tofi"* represents how further back in time, measured

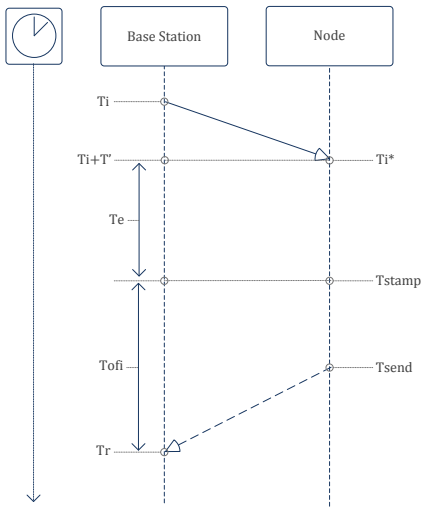

Figure 3.18: TymeSync Base Station and Motes

in milliseconds, the sample timestamp event happened.

$$T_{ofi} = T_r - (T_e + T' + T_i) \tag{3.12}$$

$$T^*_{ofi} = T_r - Tstamp = T_r - (T^*_i + T_e) \tag{3.13}$$

$$T_i = T^*_i \tag{3.14}$$

$$\Delta e = T_{ofi} - T^*_{ofi} = T' \tag{3.15}$$

There is an error introduced by the delay T'. Ideally, equation 3.12 would be equal to equation 3.13. If one can neglect transmission time, that error can be considered null. To calculate the sample time equation 3.16 can be applied by the data gathering framework.

$$T_{real} = T_{currenttime} - T_{ofi} \tag{3.16}$$

At a first glance this seems to work right out of the box, but what about the LPL feature? In fact this solution does work, but with an error, proportional to the route depth taken by the disseminated message.

The average error in a determined route depth can theoretically be modelled by an expression like the one seen in equation 3.17. Here T_{LPL} is the LPL value and *k(n,t)* an expression correlated to the number of motes in the route and their radio's LPL duty-cycle alignment.

$$\Delta e_{avg(n,t)} = k(n,t) \times T_{LPL} \tag{3.17}$$

Dissemination is a *"send to all"* method that does not guarantee that a certain path will be followed, so trying to correct the error introduced by the number of motes the message has gone

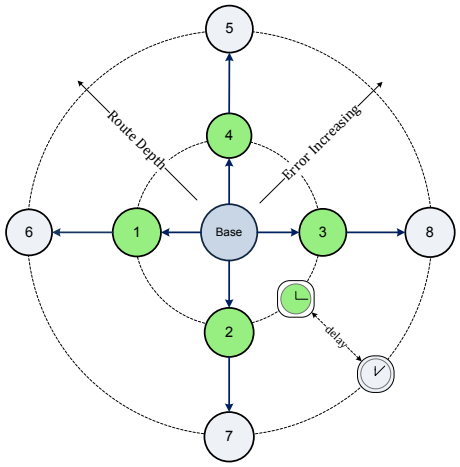

Figure 3.19: Error Increasing with TimeSync Dissemination

through is not feasible without a major rework on that component. An estimation of such error could however be made through the collection interface under specific circumstances. By using the ETX parameter it is possible to reduce the error introduced by the LPL feature and the time message retransmission. A mote's ETX parameter is proportional to the number of nodes in a route between that mote and the base station.

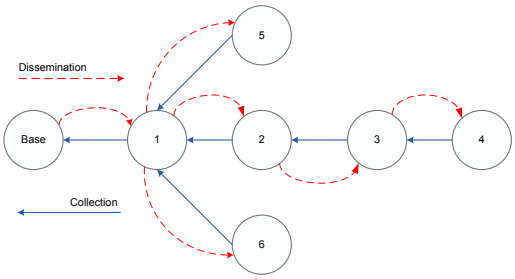

Figure 3.20: Collection and Dissemination Sharing the Same Routes

This would work assuming that the best route chosen by the collection component to a given mote is the one that first brings the disseminated time message to that same mote. This is true when the motes are spaced apart in such a way that the disseminated value can only follow through the same unique path also used by the collection component until reaching a given mote, as seen in figure 3.20.

It is, however, false that whenever any time message reaches a given mote it has passed through a smaller number of motes than the ones used to establish the collection route. At a given time, the path followed by the collection interface may differ from the path taken by a disseminated value (figure 3.21); the adaptive behavior of the collection interface helps reducing that fact, but it does not guarantee otherwise.

Some initial testing showed that the error is in fact proportional to the ETX introduced if

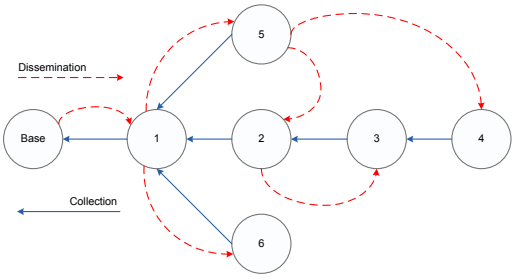

Figure 3.21: Collection and Dissemination not Sharing the Same Routes

conditions apply (path taken by the dissemination is approximately equal to the path used by the collection component). Without employing any correction, this error is on average approximately 1.5 seconds for the nodes with an ETX of 3. This is roughly a 3.5 % error from a minute-scale point of view.

For applications with a longer route depth or covering wider areas, the ETX correction may need to be employed. However, as mentioned there is always the option to disable LPL, which can be useful in short duration deployments or even in environments where the link quality can also interfere with time syncing.

A new message was added to the collection interface containing the ETX. This message, sent by each node, also contained that node's parent ID in the collection tree and two fields containing time information.

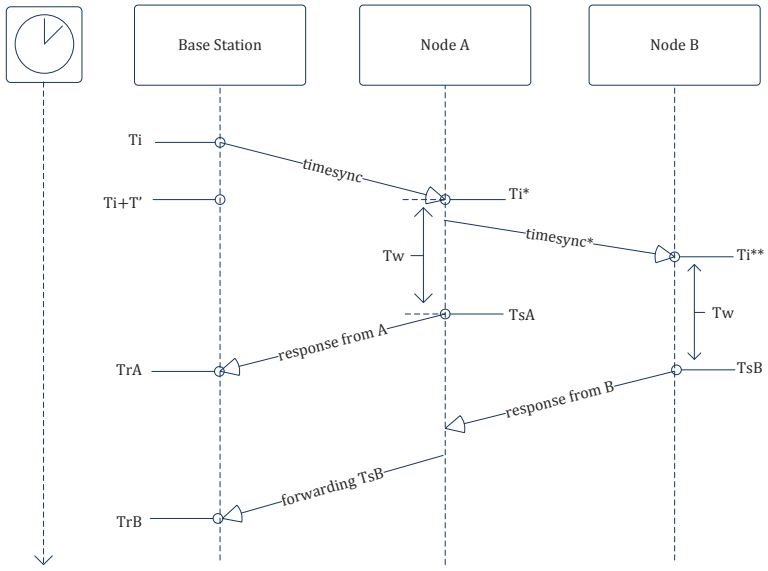

Figure 3.22: Time Dissemination and Lag

This new message called *lagMsg* is sent by the nodes every *Tw* seconds after a new time sync message is delivered on to a node. The first time field is filled with the received time *Ti* value

plus the *Tw* seconds that have elapsed. The second time field is filled by the base station when the message arrives. This situation corresponds to the diagram in figure 3.22. Here, node A and B both receive time sync messages, where node A is the parent of node B in both collection and dissemination routes. Once *Tw* elapses on node A, it responds with *TsA*. Similarly node B responds with *TsB*.

$$\Delta e_A = T_{rA} - T_{sA} \tag{3.18}$$

$$\Delta e_B = T_{rB} - T_{sB} \tag{3.19}$$

The total round-trip error for A and B motes can be calculated with equations 3.18 and 3.19. This parameter can then be compared to the ETX value multiplied by the low power listening duty cycle in order to analyse if the communication path being used is the same for both the dissemination and the collection interface. Upper system layers can save this information to correct any unacceptable timing deviations.

The base station application, arbitrarily named *networkbase* also needs to be mentioned here. It shares some of its features with the *moteSensingC* application, namely the collection and dissemination interfaces over the same message types. It does not sample its own sensors. It has an extra component, the *uartC* component that allows for the upper system layer to exchange messages with the base station and vice-versa through the USB port.

Its functionality is fairly simple, to transmit incoming messages from the network to the computer's application and to receive network management messages, like *"setupMsg"* and *"positionMsg"* to reconfigure the network. To sum up, the base station implements the interface between the network and the computer application. It can receive from the computer side *"configCTRL"* , *"setupMsg"* and *"positionMsg"* messages. These messages are forwarded to the network through the dissemination component. There is a TinyOs tool that allows the defined message types to serve as templates for classes, for a variety of object oriented languages. This enables easy integration of those messages in the user application. This tool called *Mig* was used to translate the message types into Java classes. Using the TinyOs *net.tinyos.listen* Java application, testing was performed on the developed algorithms.

3.3.5 Testing and Validation

During the development of the application, at the end of each code subsection, the features provided by each algorithm were tested. After successful verification of the developed features, a test network was assembled to gather data for stability testing purposes. It is not an easy task to perform ,without some sort of user developed computer application, to parse through the raw data gathered by the network and discover errors.

The TinyOs "message" listening tools were used to perform initial testing. A simple test methodology was employed at this stage, using 10 similarly configured motes. This methodology

consisted on, after analysing the validity of part of the sensed data on random incoming messages, to count the total messages sent to base by each mote.

This test showed no message loss when the motes were able to establish a route to the base station. Testing showed that the reconfiguring mechanisms worked as planned. The time synchronization error is a fraction of the round-trip message time. Without the LPL feature on, the round-trip messages would take times ranging from 5 ms to 40 ms also varying with the route depth. When LPL feature was on, the round trip message would take times in the order of seconds, being that this value would increase linearly with the number of nodes on the path. This in practice means that the time synchronization would be less accurate.

In the base station, there is one bug that seems to affect some TmoteSky devices when working with the UART. This bug ocasionally drops the UART reception, making the base station temporarily unresponsive to messages incoming from the computer side. There is no loss of functionality from messages incoming from the network. However, if there is the need to immediately reconfigure the network and the base station UART has stopped receiving the PC's messages, a manual reset is needed, either by clicking the reset button, unplugging the USB or, for instance, restarting the remote machine to which the mote is connected. Other TinyOs users report similar bugs with certain combinations of FTDI drivers and baud-rates. This bug did not however appear during the case study deployment.

3.4 Data Retrieval,Storage and Network Management

This section will show the software developed to store the data gathered by the WSN. It is detailed how the messages exchanged by the network are stored and managed.

3.4.1 Required functional aspects

Taking into account the developed functionality on the WSN system layer, the following list of application requirements was developed:

- Log incoming data to a database (database can be located on a server or a simple file)
- Allow sensor network reconfiguration.
- Allow individual mote configuration.

3.4.2 Chosen Development Software

The Java programming language was chosen to develop the network management software and the data storage layer. This choice was based on the portability provided by the Java Virtual Machine. There are TinyOs Java libraries that allow establishing communication with the base station. The IDE used to aid in program construction was the Eclipse Java IDE (Version 3.3 - Europa). Primarily, a linux OS was used to develop and establish communication with the base

station. TinyOs 2.x Java libraries, by default do not allow Windows OS Systems to communicate with the motes through the FTDI USB [56] virtual COMM port driver. A solution was found to this problem, and the developed application now runs seamlessly in both operating systems.

3.4.3 Used Hardware

The hardware used to develop code and run the application was a common PC platform. No special hardware requirements exist. It should however be taken in consideration that some processing power is needed to accomplish the data aggregation, chart creation and dynamic query creation tasks proposed by the requirements. Any current commercial available personal computer is able to cope with this task.

3.4.4 Developed Application and Algorithms

Data gathered by the motes is stored in a database. The base station is connected to a computer with the developed Java application. This Java application stores the retrieved data through the JDBC Api onto an Apache Derby SQL database. Any database technology would, in principle work. The Apache Derby was chosen by its embedded operation mode. This mode allows the application, without any external running services, to create on the local file system, the files needed to store the data. This application is comprised by several Java packages developed to fulfil the requirements. Each package name tries to resemble the functionality delivered by such package. Some packages are strongly intertwined to produce a certain function, where some others can work independently. As the developed Java code is quite verbose UML diagrams will be used to explain the implemented functionalities.

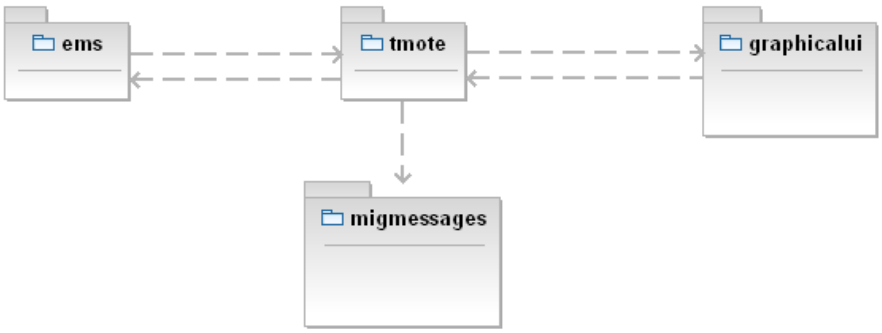

Figure 3.23: Main Developed Java Packages

The four packages seen in figure 3.23 have different purposes:

- *ems* – provides hardware and database connection abstractions
- *tmote* – provides connection to the TmoteSky base station, deals with storing data on the database and provides methods for reconfiguring the network.

- *migmessages* – stores the Java classes generated by the mig tool. These classes can be used with TinyOs Java libraries to generate messages to be sent to the network or extract the raw data from incoming messages.

The *ems* package, short for "environmental monitoring services", is composed by two Java classes. The *ems.DatabaseProvider* class is responsible for loading the database driver, and allowing other threads to access such database. The *ems.ServiceProvider* class is responsible for loading the TinyOs API that can be used to create a connection to the base-station. These classes were developed taking into consideration that other systems could be integrated onto the network management software. Linking to another wireless platform could be as easy as loading that platform's API to the *ServiceProvider*, without losing functionalities with the Tmote motes.

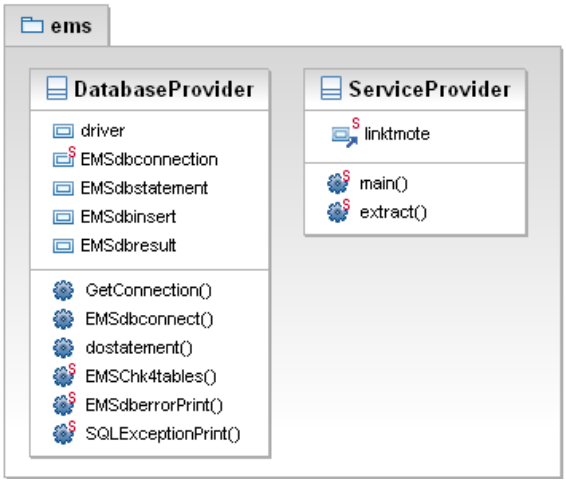

Figure 3.24: Developed Java Package : *ems*

If the data stored is of the same type, the same database could be shared with the one created by the classes inside the tmote package. The developed methods names attempt to explain each method functionality. Note that the *EMSChk4tables()* method was constructed to enable other classes to load or create the required tables for storage proposals. Communication with the *tmote* package is bidirectional.

The *tmote* package is comprised by classes that connect with the base station mote. They provide the necessary methods to convert data from the raw format into a human readable format, normally in SI units and store it on the database. Each developed class will be here briefly described. The class that provides connection with the base-station is the *tmote.Connector* class.

It provides simple methods that use the drivers loaded by the *ems.ServiceProvider* to connect to an address provided. This address can be a serial port or a network IP address from a machine running the SerialForwarder [57] application. Next the *BaseStationInterface* class is used to deal with base-station incoming messages or creating new ones to send to the network. This class has the UML class diagram seen in figure 3.25: The run method in the *BaseStationInterface* class initiates instances of other classes in the tmote package. Methods exist on the *tmote.BaseStationInterface*

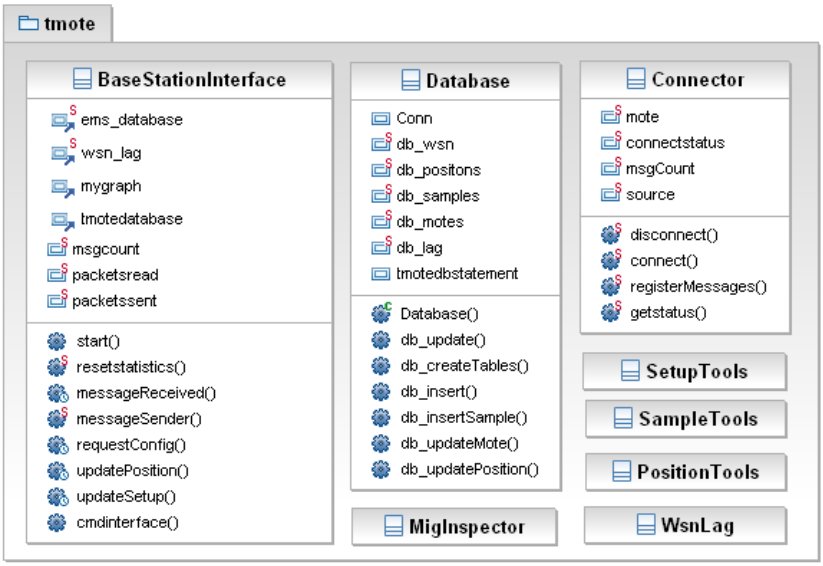

Figure 3.25: Classes contained in the *tmote* package

that track the number of received and sent packages. Some of the methods are used to poll the network for its configuration messages and positional messages. The *messageReceived* method is launched whenever a new message is transferred through the USB port by the base-station. This method ensures proper follow up of incoming messages. The *run* method is used to initiate or reuse a database.

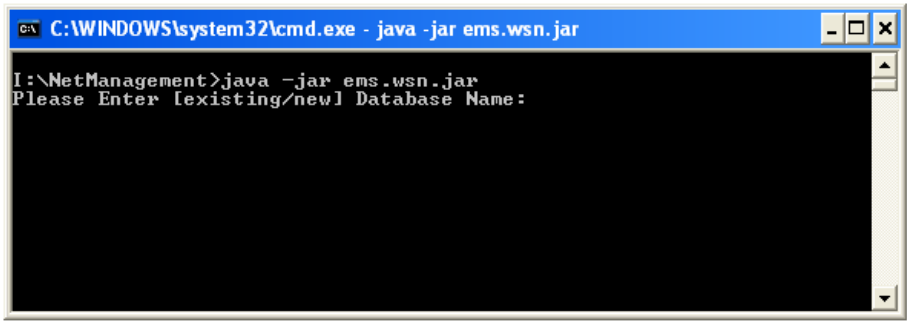

Figure 3.26: Apllication Initiated

The user is prompted for a database name, allowing the application to use an existent or create a new database. After insertion of the database name, the application uses the *connect* method on the *ems.DatabaseProvider* class to create or reuse a database. Next the *tmote.Database* class creates or reuses the necessary tables on the database. It allows the insertion and update of data on those tables.

The Database Entity-Relationship Model used, was derived by analysing the following relevant entities to the database:

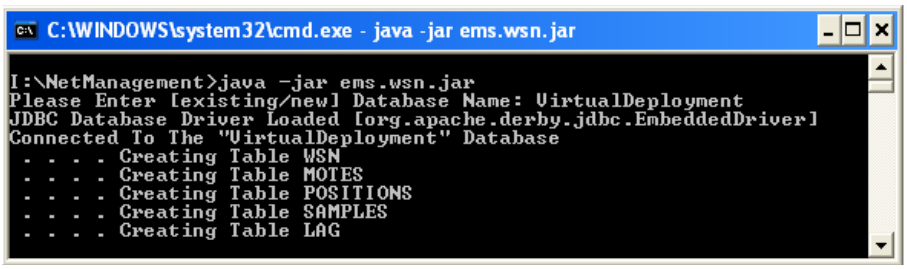

Figure 3.27: Command Line Interface

- Mote
- Wireless Network
- Sample

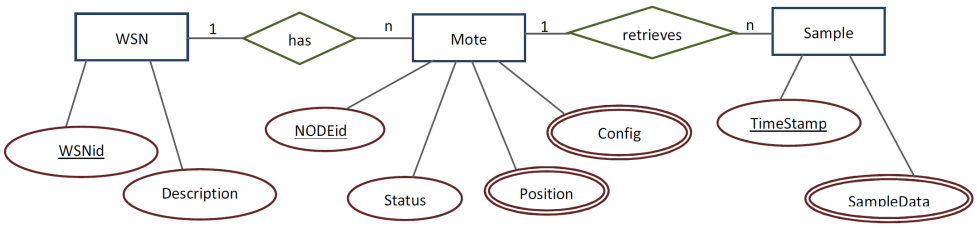

Figure 3.28: Database Entitiy-Relationship Diagram

The diagram in figure 3.28 can be obtained by studying the relationship between the entities. Note that this diagram bases itself on the following assumptions:

- The motes will be stationary during the retrieval of data;
- Each mote has the ability to store its own configuration.

If the motes are not intended to be stationary then the position attribute should be assigned to the *Sample* entity instead of the *Mote* entity. The composed attributes vary according to the definition of the network exchanged messages. In practice the composed attribute *Config* is stored in the table *Motes* and the *Position* one in a table called *Positions*. It should be stated that an additional table *Lag* was created. It stores the ETX and routes chosen by the motes during deployment. This feature provides means to replay the deployment and analyse network link quality as well as routes. The class that implements this model is called *tmote.Database* and its attributes and methods can be seen in figure 3.25.

The *db_wsn* variable is a two column matrix of strings that stores the field name in the first column of the array, and the field type in the second column, in SQL language. The other *db_* variables follow the same approach. The *db_create(table,name)* method is used to generate a *"create*

table" SQL statement using the stored string array variables. By separating the types and names of the columns, insert and update methods can dynamically create insert and update statements based on those table array variables. Further more, using regular expressions *(Java.util.regex)*, these fields can be compared to the message fields in the classes generated by the "mig" tool.

This is an important step. In the event of adding a message field to the structures discussed in the previous section, the developed code is still valid to insert or update values of the previously established fields. If the new field is added to the table specific to that message, as long as the table field name is somehow included in the message field name, the value of that field will be automatically inserted into the table, with no code rewrite. It will be demonstrated how the insert statements are generated inspecting the messages contents. Once the database is created the *cmdInterface* method from the *tmote.BaseStationInterface* class is launched. The command line interface provided allows this application to run on a machine providing a simple SSH connection. This is useful for remote deployments (for instance in wild areas) where a computer, with a GSM connection or wireless link, can still drive the application in command line interface, without the need of a graphics card.

It is now time to introduce the messages generated by the "mig" tool and how these ones are transformed into readable fields. The messages generated by the mig class are all stored in the mig messages package. This package can be described by the UML package diagram in figure 3.29 Their names match the *nesC* structures already devised in the previous subsection. Their provided methods will be introduced briefly.

Figure 3.29: Messages Generated by the Mig Tool

Each message provides *getters* and *setters* methods for the retrieval or creation of message field parameters. These "getters" and "setters" are a small part of the methods provided by the mig messages. The unused methods are not displayed in the class diagrams that follow.

The *tmote.BaseStationInterface.requestcfg()* method creates a *configCTRL* message, described by the UML diagram in figure 3.30, and sends it to the base station. The algorithm employed by the bases station to disseminate this message was already displayed in the previous subsection. Please notice the method *set_lpl_setter*. This method is used to deactivate or activate the low power listening feature on the motes when this message is disseminated. If employed to deactivate LPL, it accelerates the response by the network, when sending back the responses of their *"positionMsg"* and *"setupMsg"* messages. The *"setupMsg"* class then provides the methods to retrieve the motes configuration.

Note that the *tmote.SetupTools* class extends the *migmessages.setupMsg* class, providing methods to convert the raw data from the *"setupMsg"* into SI units or some other unit system, and

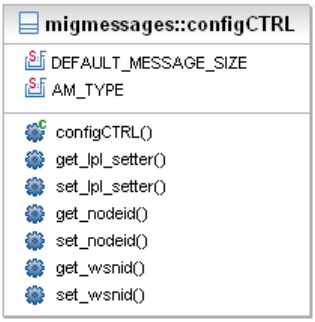

Figure 3.30: Control Message

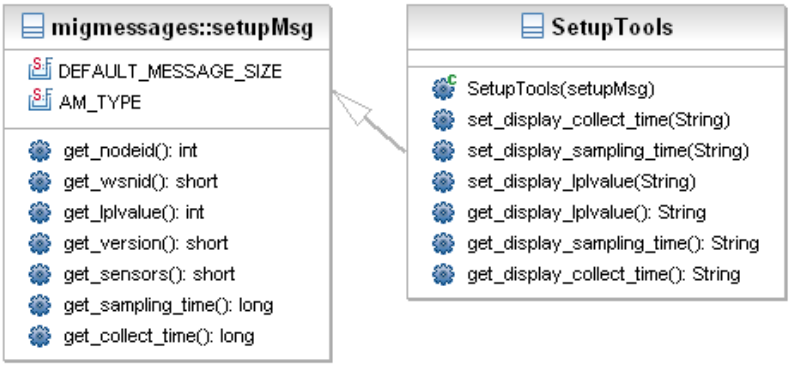

Figure 3.31: Setup Message Extended

vice-versa. This relationship is described by the UML class diagram in figure 3.31. The option to not override the methods from the *"setupMsg"* was taken in order to still provide access to the raw fields, thereby only the *SetupTools* class needs to be instantiated in other classes. The *SetupTools* class is used by the program to make new or receive incoming setup messages. The *"display_"* methods are primarily used by the GUI, being that the command line interface is built to access the raw values of messages. *"But how are these message fields and methods automatically found by the databas insertion and update methods?"* The basestation interface console redirects each message type to the appropriate database insertion methods. But these insertion methods are not by themselves able to parse through the message and automatically find the message fields names and contents for immediate insertion on the database. The *tmote.MigInspector* class provides methods to inspect the internals of the received messages.

This class, represented in figure 3.32 uses the *Java.lang.reflection* package to retrieve each message "getters" and "setters" fields, as well as their types. The database routines, using the *MigInspector* class methods, are able to check if each database column name is part of one of the message field names. For instance the *getReadingFields(message)* method returns all fields names from any message passed to that method. The *getHumanReadableFields(message)* method returns all of the fields that exist that can convert readings into SI units or other units, plus the

Figure 3.32: Mig Message Inspector Class

ones that do not need any type of conversion from the raw data format (for instance the *nodeid* integer field). The *getDisplayReadableFields(message)* is primarily used to parse through specific message methods that may exist on the message that enable the translation of such values directly onto the GUI.

The *getHumanReading(Message,Field)* gives the field value of the message argument. All messages registered on the database follow this automatic approach. Similarly to the *setupMsg* class, the *positionMsg* is extended by the *positionTools* class. The samples are the most frequently received message. The *SampleTools* class extends the *samplesMsg* and implements the methods that convert the raw values into SI units. The *tmote.Database* class inserts this message onto the database easily using the previously explained classes.

The *readable_temperature()* method executes the equation 3.7 seen in the previous subsection on the raw temperature value and returns a string representation of the calculated value. The *readable_light* and *readable_photo*, execute equations 3.5 and 3.6 respectively on the raw values. The *WsnLag* class is used to store instantaneous *lagMsg* messages sent by the nodes. Also the node's parents are kept. Its main purpose is to provide an interface that shows the last *lagMsg* activity received from each node, its theoretical synchronization error, the collection tree established as well as each mote ETX parameter. Although the command line interface is sufficient for configuring and managing the network, a Java GUI was built to create a more compelling and easy to work interface. This GUI is useful for rapid configuration of the developed wireless network application and allows consulting the values stored in the database.

It is out of the scope of this work to explain the internals of the GUI and how they interact with the underlying layers. There are, as previously mentioned, special methods that allow the MigInspector tools to search for specific conversions to be made in order to display values on the GUI. For instance in the GUI the LPL value can be read as 'on' or 'off', but in practice it is stored

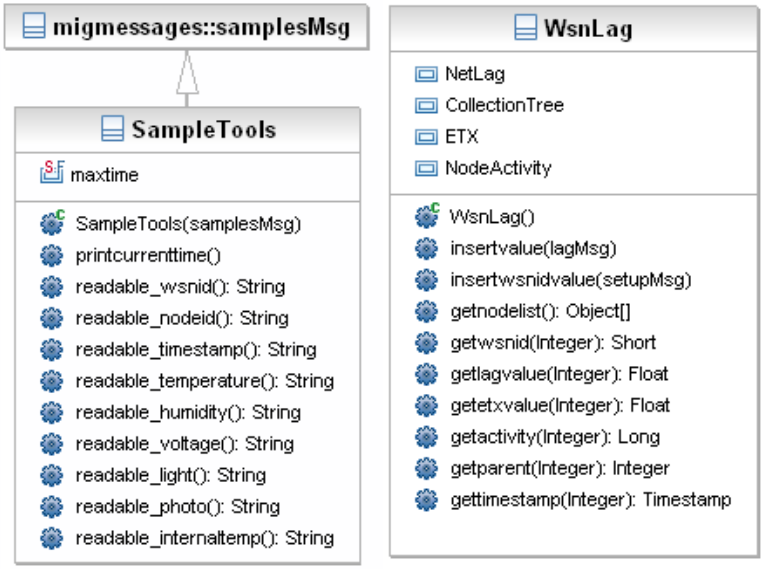

Figure 3.33: *Samples* and *WsnLag* messages

on the message as '300' or '0' milliseconds. Using these methods the GUI can search for the fields in the incoming messages, defaulting to the regular *get* methods whenever a *display* method is not provided for that field.

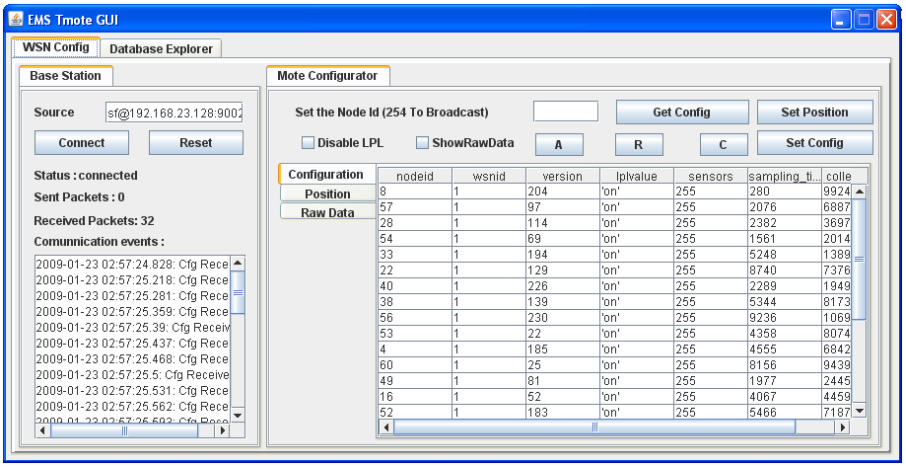

Figure 3.34: GUI Network Management Tab

Whenever a new message arrives the GUI is updated with the contents of that message. This interface can also be used to request the configuration of the network. The received configuration and position messages can be changed on the GUI in SI units when applicable and sent back to the network, thereby updating the motes with a new configuration. The raw data from the incoming samples can also be seen for debugging purposes. This GUI provides views of the database tables,

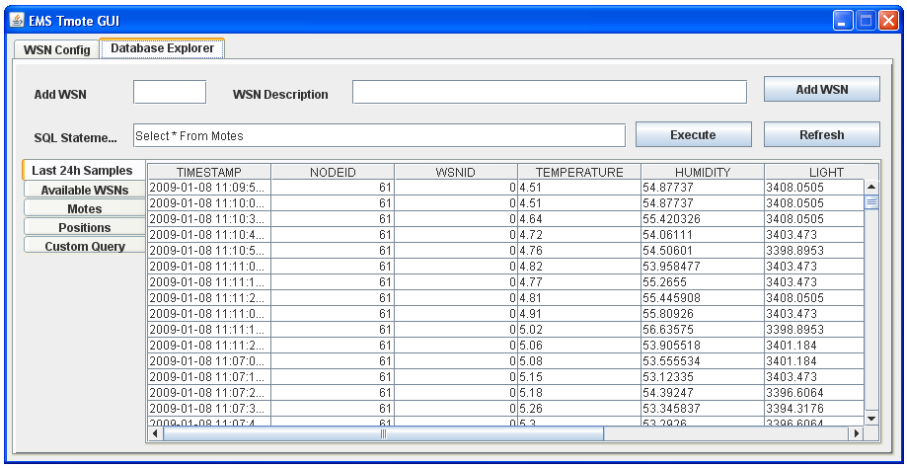

Figure 3.35: GUI Database Management Tab

and also a query field to do specific searches or database manipulation.

3.4.5 Testing and Validation

This layer was tested together with the WSN layer showing that the requirements were fulfilled. During construction, each new functionality was tested, in a bottom-up approach, building the basic foundations first, and ending with the construction of the GUI. No known “bugs” exist in this application layer.

3.5 Data and Network Analysis Services.

In this subsection the data aggregation and network analysis system is documented.

3.5.1 Required functional aspects

The data and network analysis layer needs to be able to provide the user with efective and easy to use tools to catalogue and analyse the data retrieved by the network:

- Permit queries on the data gathered by all the motes.
- Enable queries on single motes.
- Allow averaging by an array of motes.
- Find max readings of an arbitrary field.
- Create charts within arbitrary time intervals.
- Allow aggregation in common time units.

- Enable easy geo-location of gathered data.
- Allow visualization of network status.
- Permit visualization of the network spatial distribution on freely available GIS tools.
- Show routes established by the network, and allow a periodic update of that information.
- Operating System Independent.

3.5.2 Chosen Development Software

To develop the top layer the Java programming language was used. Using Java servlet technology, the application developed in the previous subsection is easily extended to provide the necessary services.

3.5.3 Developed Application and Algorithms

The last layer of the system focus on making the data accessible to the end user by a set of *"mashup"* services running on top of a GIS tool. Google Earth was used as a way to display information regarding the network status. This application layer is able to place the WSN gathered data and also runtime parameters in the virtual landscape provided by this tool. Google Earth was primarily chosen due to the KML open standard. It allows the creation of rich and useful environments for this kind of application. It is compatible with Google Maps as well, something that is useful, by eliminating the need to install Google Earth. A web interface was developed to permit refined queries to the data contained on the database, and also observe the network status.

By using Java Servlets [58], this layer is able to generate, and update data instantaneously. This subsection will detail the software developed that accomplishes the already mentioned tasks.

This layer bases itself on web interface,that is composed by several Java packages, each package relates to a specific task or function. In figure 3.36 the developed Java packages and their inner classes are shown.

Like in previous subsections, a brief overview over the developed code, in the form of UML diagrams, will be introduced whenever necessary.

The *sqlqueryengine* package, on diagram 3.37, contains a single Java class file. The *QueryTool.java* is able to generate *SQL* query statements, that are used to query the database. There are different methods that return specific types of queries. The *netQuery* method takes as arguments the nodes, the reading field, the aggregating function (average, maximum or minimum), the aggregation time pace and the time interval to query the samples. It returns a query statement that can be used to retrieve the data from the database. Other classes can use this method to retrieve the data from a given array of nodes, averaged (for instance) in each minute. Alternately this method can be used to give a query to return the results from a single node's reading field, with its samples averaged by minute, within a given time frame. Multiple calls would allow comparison of data between the nodes.

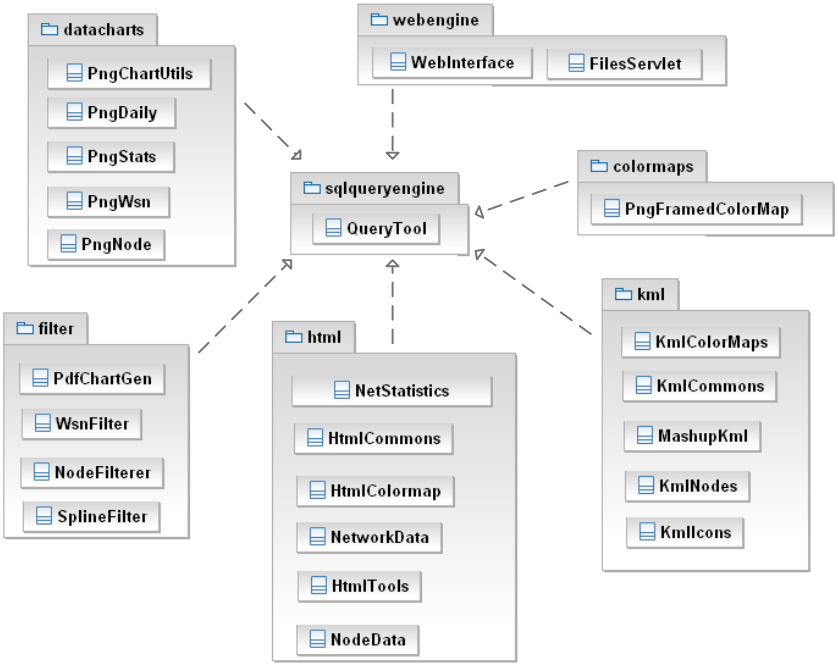

Figure 3.36: Web Interface Packages

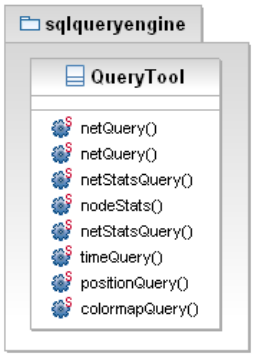

Figure 3.37: SqlQueryEngine Package

The *colorMap* method, uses the *netQuery* and the *positionQuery* methods. The obtained query is used to correlate the samples with the positional data. The result can be used to generate an image with the data localized at each point. The *netstats* method is used to generate a query statement that returns the number of samples gathered by each node so far. This is useful to inspect if a given node is not sending the data to the network base station.

The *"webengine"* package starts a web server on the machine hosting the data gathering layer, it creates the *URL* directory structure that can be accessed by common web browsers. This directory structure contents, seen in figure 3.38, is generated upon request by the client. For instance the *http://exampledomain/netstatspage* would retrieve dynamically information on the network status

and generate a web page for user view.

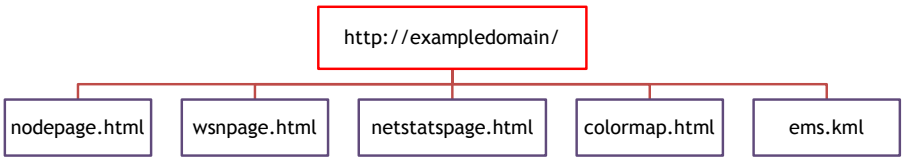

Figure 3.38: Url Directories

The *html* package is composed by a set of classes that generate the webpages. These are produced when specific web server directories are accessed. These html webpages can be used as an interface for the end user to analyse the data retrieved by the network. There are four webpages produced by this package. Each webpage HTML code is made from smaller HTML building blocks "handwritten" in the *HtmlCommons* webpage. It was built into this class methods that generate html code, providing forms, tables to contain data from the database and other generic elements like image place holders. This approach enables easy expansion from the already built interface into other interfaces, maintaining a cohesive set of variables and parameters passed on the html *get* methods. Each of the four webpages has a different purpose. The *html.nodedata* class generates the *"nodepage.html"* that can be seen in figure 3.39.

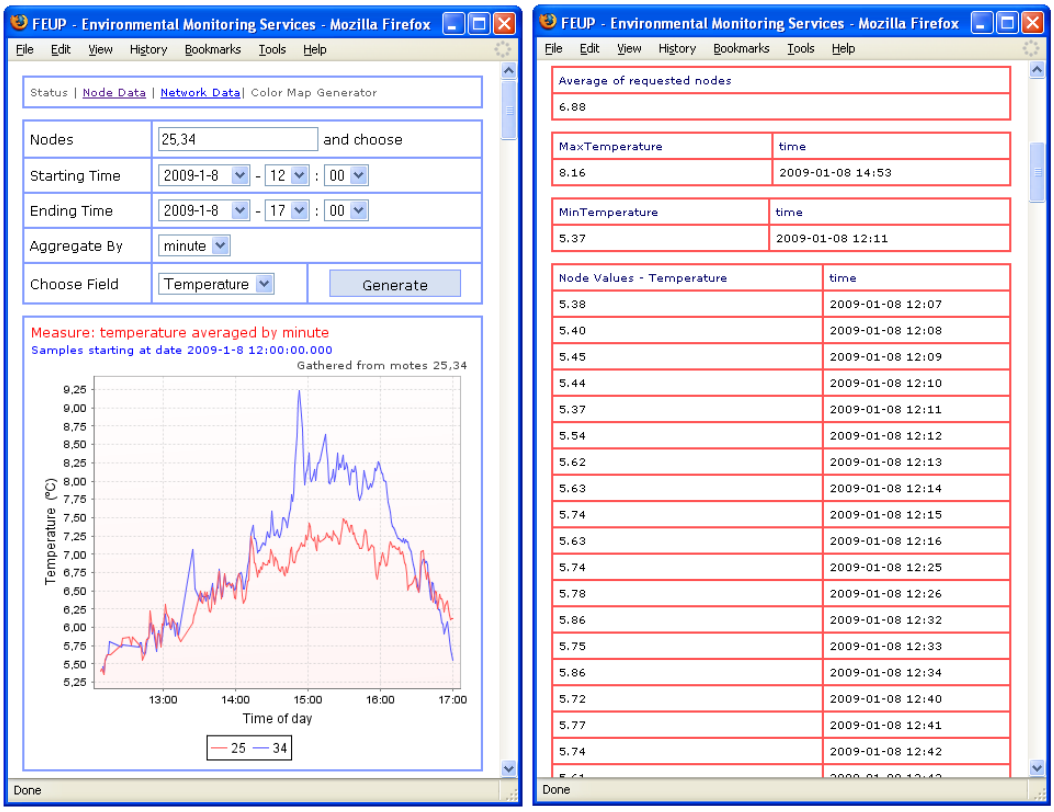

Figure 3.39: Nodepage Html

Once the user inserts the requested node's data it may hit enter so that the time frame fields correspond to the starting date of the samples, and also the date they end. With all fields defined, including the aggregation time pace and the measured quantity, clicking the generate button will produce a webpage containing the data results (like in figure 3.39).

Please note the chart included in the webpage. This chart is actually generated by another developed Java class contained in the datacharts package, the *PngNode* Java class. The user may analyse the chart and redefine the time interval once again in the form and requesting a new webpage with data within that time-frame. This webpage also contains the data from the database tables, and the maximum and minimum values from the requested node's field. Querying one node can be useful, but comparing the nodes readings may be a often required feature. This webpage can be used to compare readings from several motes by inserting an interval in the html node field, in the manners depicted by the intervals 3.20, 3.21 and 3.22.

$$NodeEnumeration \implies 1,2,3,4,\ldots,n \tag{3.20}$$

$$NodeInterval \implies 1-n \tag{3.21}$$

$$MixedForm \implies 1-10,11-n,\ldots \tag{3.22}$$

The graphic chart now shows the readings from all the nodes specified. However, for display purposes the values shown in the tables are the resulting averages in time from all the nodes.

The *wsnpage.html* has a similar functionality to the one of the node page, but this time around the graphic always displays the average of the readings in time, from the nodes inserted on the node's id field. The Java class responsible for producing this webpage is the *html.networkdata* The network status can be accessed by loading the *netstatspage.html* (figure 3.40). This webpage contains a bar chart with the number of received sample messages by each node. If the WSN is running, the loaded webpage will show the time elapsed between that mote's last report, and current time. The last retrieved ETX parameters, the communication delay estimation and each node's route parent id is shown at the bottom of that webpage. The Java class developed to build this webpage is the *NetStatistics.java*, where as the graphic is built by the *PngStats.java*

Analyzing the net status webpage, in an homogeneously configured network, it becomes easy to identify if any mote has communications problems, either by comparing the number of received sample messages or by checking its last activity report.

The final webpage, the *colormap.html* webpage, generated by the *htmlcolormap.java* class, serves as an interface to characterize the gathered data, within a given space region. It displays the requested data by the use of html tables, and also a colormap image. The nodes can be specified and the bounding coordinates are automatically calculated to generate the colormap. If the user wishes to refine such coordinates it can re-submit the form with different ones.

The colormap image is generated by the *colormaps.PngFramedColormap.java* class, that adapts the data from the database to a format compliant with an API developed by the *FEUP Underwater Systems and Technology Laboratory* [59], and later produces the output image. The

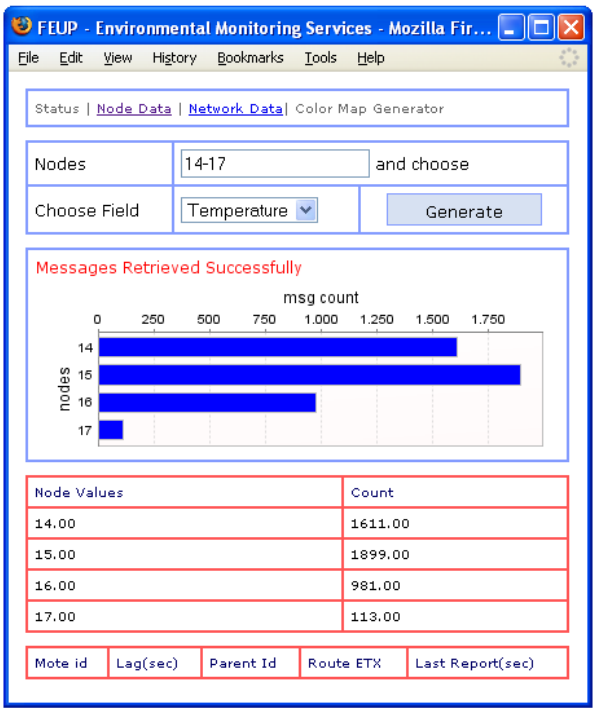

Figure 3.40: NetStatus Html

ColorMaps class is able to place the colormap image, generated by the *PngFramedColormap*, from a given measured field and an array of nodes, directly onto Google Earth. This enables an indepth analysis of the data and its correlation with the nodes location. This feature can be used by loading the *colormap.html*, generating a colormap for a given set of nodes, and clicking the *"Mashup on Google Earth"* link. This situation corresponds to figure 3.41. All of the KML files are generated on request. To automatically update this data a *kml network link* file is used, with an (arbitrary) update rate of 5 seconds.

This final system layer is able to generate KML files, that contain the network status, including information on the routes established, as well as the data gathered by the network during deployment. These KML files can be imported by Google Earth, or Google Maps, and show icon representations of the nodes. The KML package is composed by the classes seen in diagram 3.42.

The URL *http://exampledomain/ems.kml* links to a dynamically generated kml file. That file displays the current status of the network and each node, in the form of interactive icons (provided by the developed class *"KmlIcons"*). If the network is live, then the routes between nodes will also be displayed like seen on image 3.43. The links and nodes can be turned on or off using the sidebar by clicking each entity. This is specially useful when certain motes are placed in nearby locations.

Each node icon's color relates to the current delay status by using a color gradient. Absolute green is used to display nodes with 0 seconds delay, absolute yellow starts at delay of 1 second, absolute red represents nodes with a delay superior to 4 seconds. In between delays fade to in

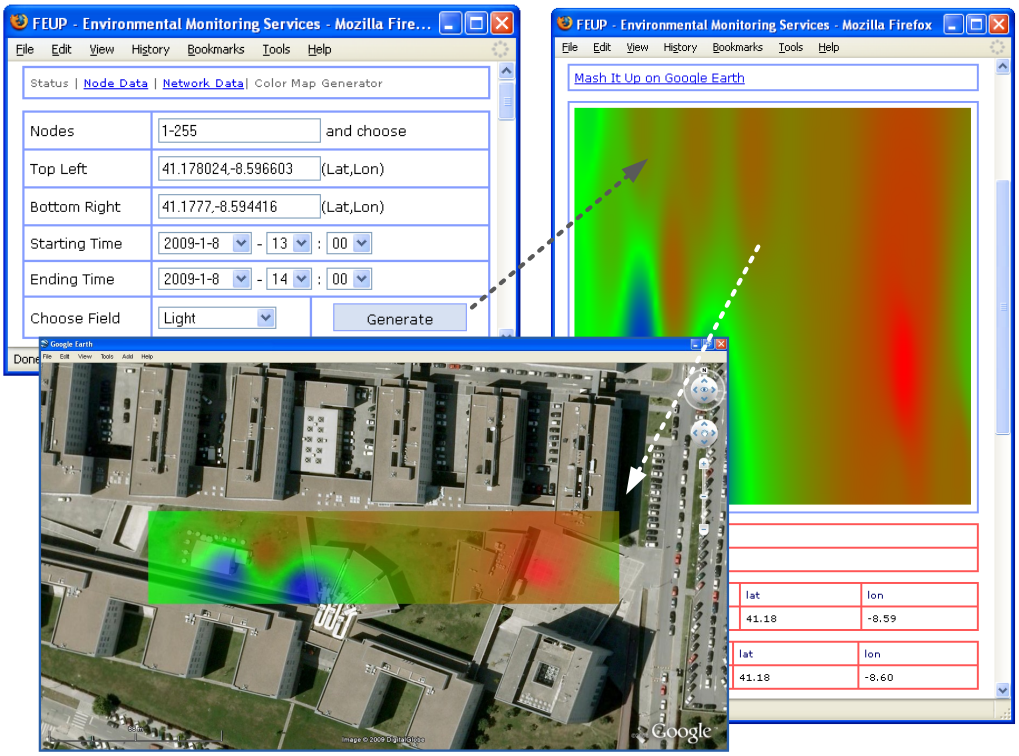

Figure 3.41: Colormap Html

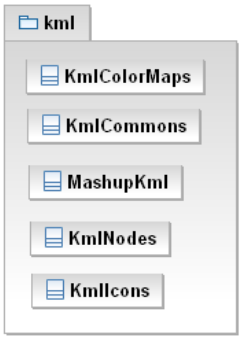

Figure 3.42: Developed KML Package

between colors. The icon also contains the nodes current theoretical delay in seconds. By clicking an icon, a balloon is generated with the current readings of that node which can also be used while offline as seen in figure 3.44.

All of the features that relate with the Google Earth can work together to deliver a full network, and data analysis user interface.

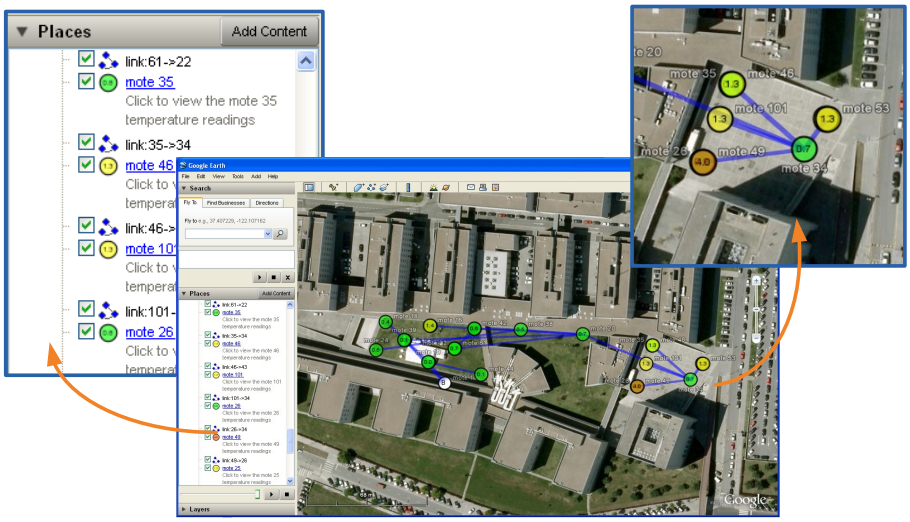

Figure 3.43: Google Earth Network Interface

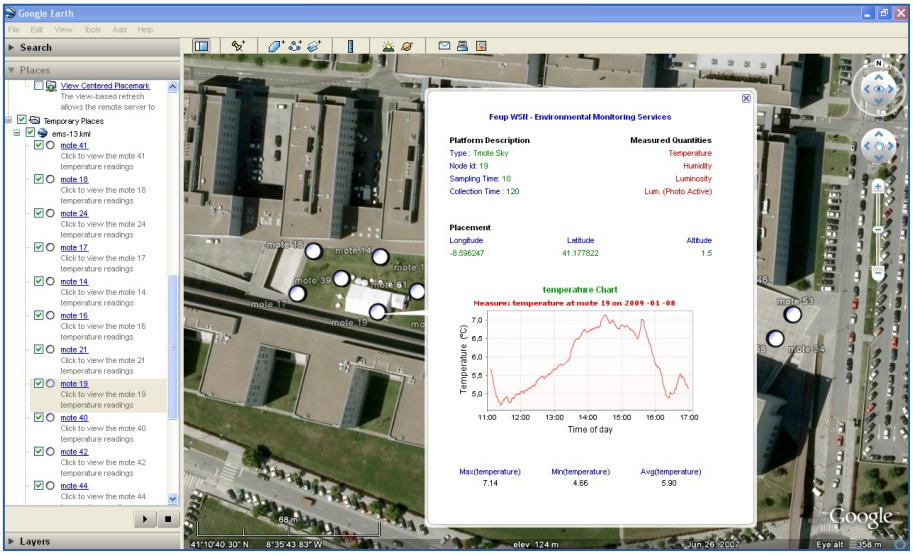

Figure 3.44: Google Earth Network Interface

3.5.4 Testing and Validation

As progress was made, during each Java package and class development , each new individual functionality was tested. One class was extensively tested after its development, the *QueryTool.java* class. All other classes rely on the data provided by this class. Once all packages were finalized, they were tested together, in an array of situations, using live data from a running test network. No known “bugs” exist in this system layer.

3.6 System Integration

This subsection shows an overview of the developed system layers working together. The messages retrieved by the motes are sent to the base-station through the explained routing algorithm. Once the messages reach the base station, they are converted from their raw values into SI units when applicable, and stored onto the database.

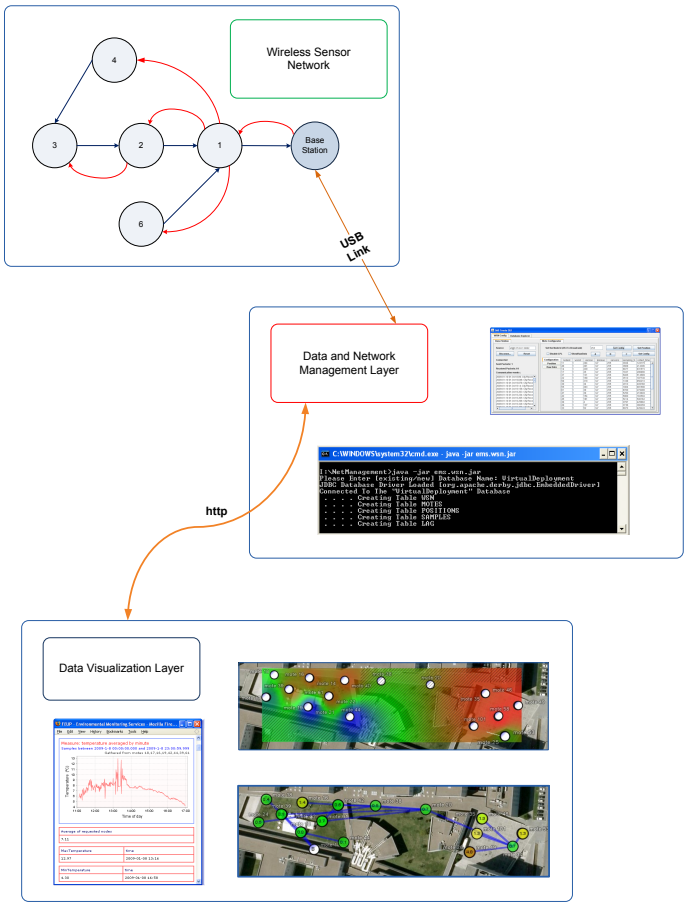

Figure 3.45: System Integration Diagram

This is done with the developed data retrieval and network management tool. Once the data is on the database, it is readily available for online view, through the use of either the webpages interface or the Google Earth interface. On the HTML interface it is possible to request specific variables, time-frames and aggregation methods. An overall status of the network performance can also be seen. The Google Earth interface allows a visual perception of the network links and of the average round-trip message time. It should be focused that, for security reasons, only the network management tool can reconfigure the network.

3.6.1 Full System Testing

As explained each layer was tested on its completion. On completion of the last layer, the system was tested as whole.A broad array of situations were used to check conformance with the requirements. Special focus was given to stability in each system layer. For instance, the test network comprised of 10 nodes was configured to read all sensors every 1 seconds, and send that information every 60 seconds. This situation is way beyond specification, that did not require such aggressive timings. This lead to a high load on the base-station during reception, and also placed strain on the data management layer. Nevertheless the system behaved stably in all the tested situations, and no significant message loss was registered when using the test network within the base-station range or in multi-hop situations.

The data analysis layer was used in conjunction with the lower layers, and no performance/reliability impact was noticed has expected. It should be noted however, that although no special care was taken in order to enforce multithreading onto the application, the application does benefit from a multi-core cpu environment. This though is mainly due to the way how current operating systems take advantage of such hardware, the way how Java Virtual Machine [60] handles multiple applications and also the instantiation of the developed classes for each system layer.

Chapter 4

Case Study and Platform Testing

In the test case, the developed system is used to retrieve environmental data in order to show possible micro-climatic changes due to the use of different construction materials. This test was conducted at the FEUP campus. The deployment is not only used to test the developed system, but it serves also as a valuable experiment to characterize the campus area, in terms of temperature, humidity and light levels at a scale that we feel.

The idea to use this system in this kind of context was planned in advance with a doctorate student in civil engineering (João Granadeiro Cortesão), studying the field of micro-climatic changes on urban spaces due to construction materials. This field has an urging need for distributed measurement systems. There is also another purpose in this experiment: to relate the gathered readings with the people's behavior in those public spaces, namely through observation, carried out during the measurement moment.

4.1 Deployment Preparation

The experiment was designed to happen in two different campus spaces, characterized by different spatial arrangements and construction materials. Each space had several motes. These spaces where divided into a grid and several points were chosen to contain 2 motes, one on the ground and one 1.5 meters immediately above. The total number of nodes needed were 30 to monitor the 15 spots. Measures were taken to mark the points and included in this reference map.

In figure 4.1 the layout of points to monitor is plotted. A reference table was constructed to correlate each stand to its pair of motes. The full table can be seen in Appendix A. The nodes were assigned with their previously labeled ids to avoid confusion. The option to double monitor each point relates to the fact that ground temperature may be different from the temperature 1.5 meters above, and such difference may be related to the surface material. The same can happen with humidity. Using several nodes at each location also allows redundancy in case of node failure.

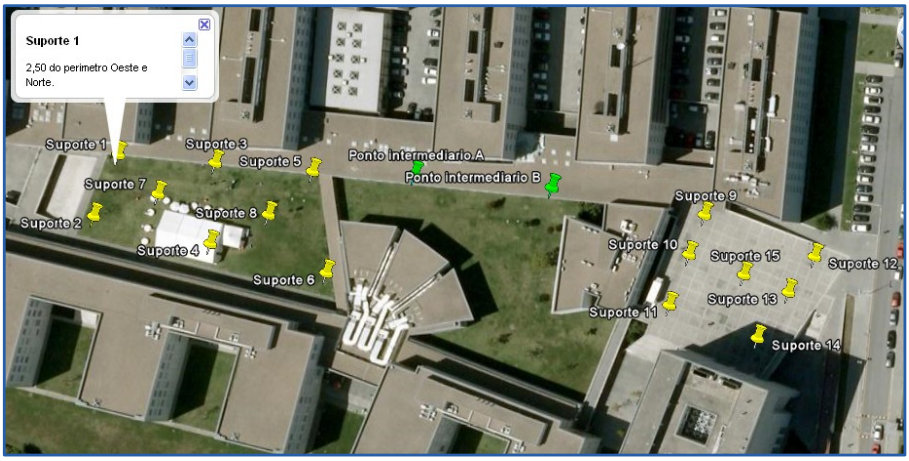

Figure 4.1: Points to Monitor

Some kind of support mechanism needed to be built in order to place the nodes on the designated areas. To allow further deployments in different areas a stand model was developed. This model can be seen in figure 4.2, and its cad design is in annex A. The design is justified by the easy assembly process and easy to find materials. It also allows two degrees of rotation that enable this stand to be used on inclined surfaces. Fifteen supports where built from wood and screws.

From the 10 motes used at the development phase, only 3 had temperature and humidity sensors. It was not enough to fulfill the deployment requirements. Material from previous FEUP WSN [7] projects could be used, but some appeared to be slightly damaged and needed a serious review. After parsing through the existing material 8 working sensors and cables where found. 6 more where gathered by combining working parts of damaged cable/sensor arrangements. Still, this was not enough. To overcome this fact extra sensors where bought. These sensors where soldered onto cables, and a connector was soldered onto extra motes available on the FEUP LSTS lab. It should be mentioned that this soldering process had to be done quite meticulously. The space between sensor pins equals 0.67 mm. A diagram can be seen in figure 4.3 with the assembly methodology.

Each node needed to be powered up by easy to find batteries. Two AA batteries are sufficient to deal with one node. Battery connectors where bought. Cables and connectors where soldered between the motes and the battery connectors. The nodes were then enclosed into preexisting boxes. These boxes enabled the mote to be protected from rain. However the boxes that existed were not enough. A basic solution was encountered: plastic cups. With the mote glued with duck tape, inside the plastic cup, and with the sensor placed outside in its own perforated (to permit air flow) plastic cup, the nodes would still be protected. Preparing for deployment took approximately two weeks of work. After mounting all nodes on the stands, a consistency between sensor readings was done using the data aggregation layer. Some easy to find problems were immediately solved. The batteries used consisted of rechargeable battery packs. These packs were available on the WSN lab. However, some of these packs were damaged or performing poorly. Something that

profoundly deemed the first day of experiments.

4.2 Case Study Deployment

The experiment started at 10 am of January 7. First, all stands were placed in place using the distance measured from building as reference points, while checking the map defined in the preparation stage. The motes on the top of the stand were oriented to north in order to provide comparable light readings. A notebook PC was placed in a nearby building, on the top floor, running the base station node, and the data gathering and network management application.

Downstairs, another notebook was used to remotely connect to the first machine and analyse the retrieved data through the data analysis layer. After checking for response by all nodes, the network was configured, using the network management GUI, in a homogeneous way:

- Sampling all sensors every 10 seconds.
- Collecting samples every 3 minutes.

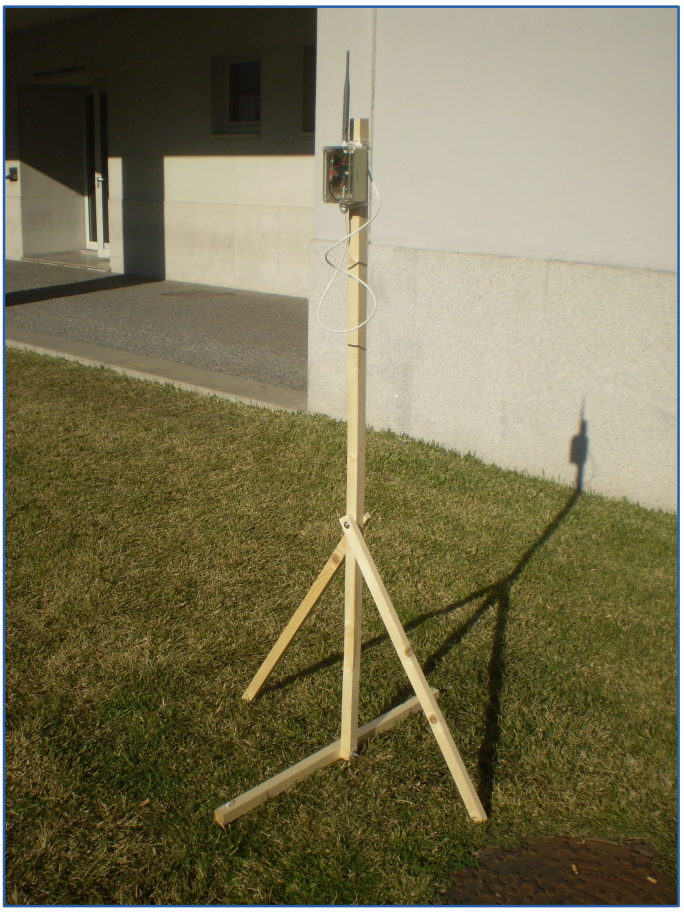

Figure 4.2: Stand Model

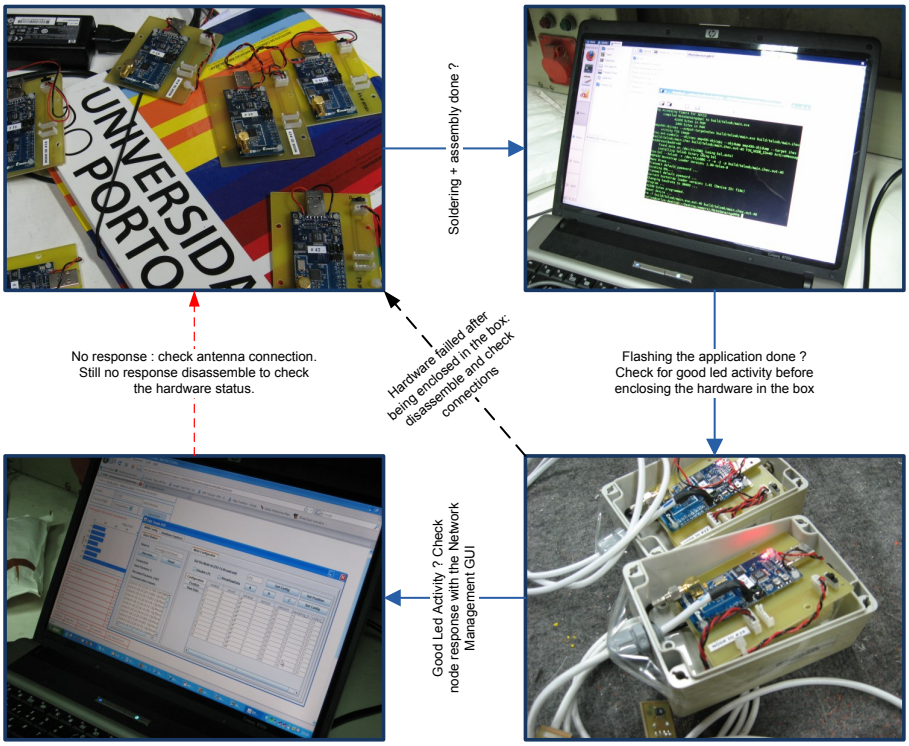

Figure 4.3: Assembly Line Stages

Some problems where immediately found. Some of the node's sample messages were randomly being lost. By inspecting the nodes that where failing it was visible by the developed LED debug activity that their batteries were dying. In some cases this was easily solved by replacing the batteries immediately. In the motes using the rechargeable packs this was not possible due to the previously constructed connection interface. Three motes where by 11h30 am not reporting their samples back to base in a consistent way. To solve this problem three motes were soldered to connectors and replaced the failing ones. Later that day the problem reappeared on other nodes connected to malfunctioning rechargeable battery packs. Also the base-station had to be relocated to a closer location.

The first day of experiments was finished by 4 pm due to this fact. It should be stated however, that the redundancy provided by the number of motes on each location minimizes the problems found and allows the remaining readings to be analysed. More connectors were soldered directly onto the motes allowing regular AA batteries to be used. In the second day of experiences no major problems were encountered. The data analysis will focus on the samples retrieved in the second day of experiments.

4.3 Environmental Data Analysis

All of each mote's sensors were active during deployment. This allowed comparing light levels with temperature and humidity. This analysis will focus first on the averages of the readings of space A (the garden) versus space B (the library), both ground motes and upper layer motes.

One problem that occurred in both days was random message loss and its impact on the average calculated from all readings. These problems particularly affect the calculus of each minute average, from all the motes. When some motes happen to have failed to deliver their messages to base on a given minute and the remaining motes delivered higher or lower readings of that time, then the average of the readings in that moment is strongly affected, resulting in a "spike" on the readings.

To filter out the abnormal spikes, and eliminate the reading gaps in a consistent way, the gaps were filled with the previous valid reading, followed by a bi-cubical spline interpolation over all values. It should be noted that sensor error band-gap at the temperatures recorded allows one to do just that when comparing readings between all motes. However if the reading gaps are wider than a few minutes, than this fix should be used with care, taking that fact into account when analysing the resulting data. To automate this procedure another JAVA class was built, although it did not make part of the initial project requirements.

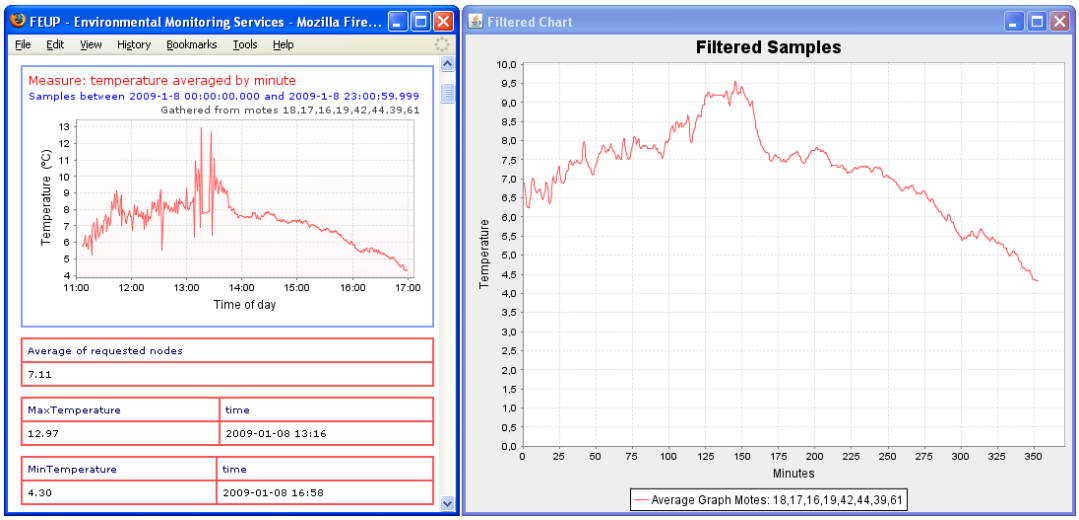

Figure 4.4: Example - Temperature Evolution from Space A, Ground Motes

This class eliminates samples from motes that report values above or below a certain threshold due to the fact that each sensor defaults to a particular value when not working. It then tries to find either the remaining motes have consistently delivered sample messages or have determined amount of "null periods". By looking to the database it was seen that no long periods had only null values. This way, only a total count was made on the null values, and the threshold was set to half the total expected messages. This was done due to the fact that multiple motes exist at the study sites. Although one mote may not be reporting its samples on a given minute, its neighbouring

motes, as it will be shown, do report. By averaging all of the values the error introduced by the spline interpolation is reduced. In the end this little application exports the resulting data charts on to PDF format, which can be a valuable feature to generate reports on deployments.

Either way the average evolution of each variable is the key parameter to be analysed in both spaces, both ground and upper motes, and as long as there is a sufficient number of motes reporting on that minute per site, conclusions can still be made. The next subsection will show that there are in fact enough readings per location to enable the analysis made on the following paragraphs.

Figure 4.5: On the left the garden versus the library on the right

Normalization using the procedure mentioned provided good results. The following graphics show the measured fields evolution in both spaces during day two of the experiment. As an example, in figure 4.4 the graphic obtained with the web interface is also included. To maintain performance when answering remote requests, the graphic on the web-page is generated without resorting to interpolation. It should be noted that the apparent flickering in that graphic, occurs due to the fact that the considered timescale is extremely small considering the graphic width. The high spikes on the web-page's graphic occur due to the facts already discussed. These spikes are filtered employing the methods already mentioned. When sample consistency is high then both the web interface graphic and the "filtered" graphic will roughly look the same. On the web-page the total average,the maximum and minimum (and their respective occurrence times) are displayed. All units on the graphics that follow are SI units.

It's now time to compare temperature evolution in all locations. The mote's registered temperatures averaged every 10 minutes are displayed in figure 4.6 for the garden and 4.7 for the library square. Please note that the possible error is represented at each reading by a vertical line in the graphic. This error was considered to increase or decreases with temperature according to the temperature sensor's error curve. This error curve can be seen in figure 3.11 in Chapter 3. All sensors showed to be bounded by that error curve in previous testing, at the experienced temperatures.

Both ground and upper motes in the garden location (Space A), share the same variation profile. Which happens identically between the motes in Space B. This can be said to be related to the fact that space A has much more solar access than space B thus having more radiated surfaces,

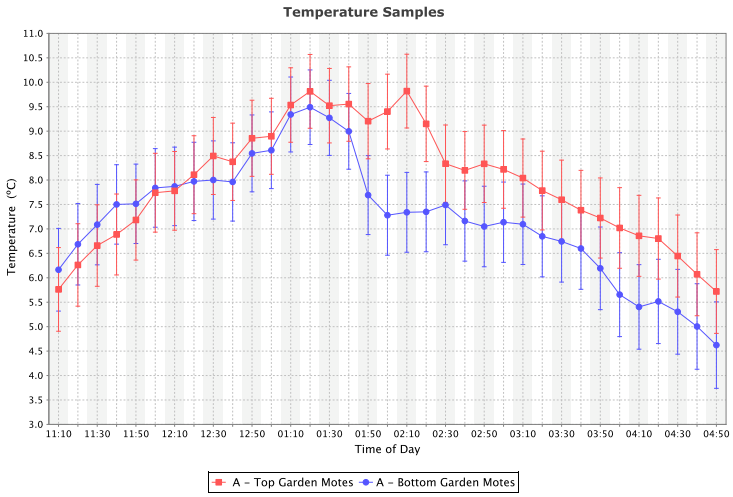

Figure 4.6: Temperature evolution at the Garden

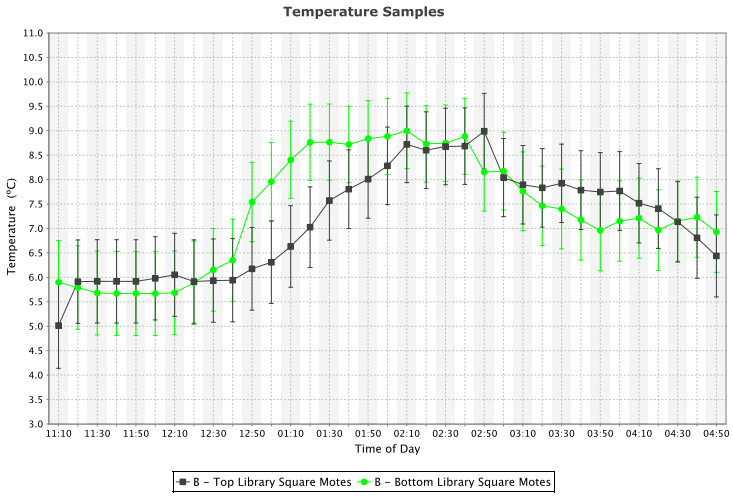

Figure 4.7: Temperature evolution at the Library Square

and on the other hand, is more sheltered from wind flows than space B due to its physical configuration. These two characteristics enhance space A's temperatures to be higher than space B's ones. This fact can be confirmed by averaging the temperature readings at each location. This is shown in figure 4.8. In this graphic it is possible to see that there are readings spaced appart by a gap that is superior to the maximum error.

These differences cannot be said to be induced by differences in the spaces' facing materials because both of them have exactly the same kind of facing materials in what concerns to vertical surfaces, and though the horizontal facing materials are quite different (green grass in space A and

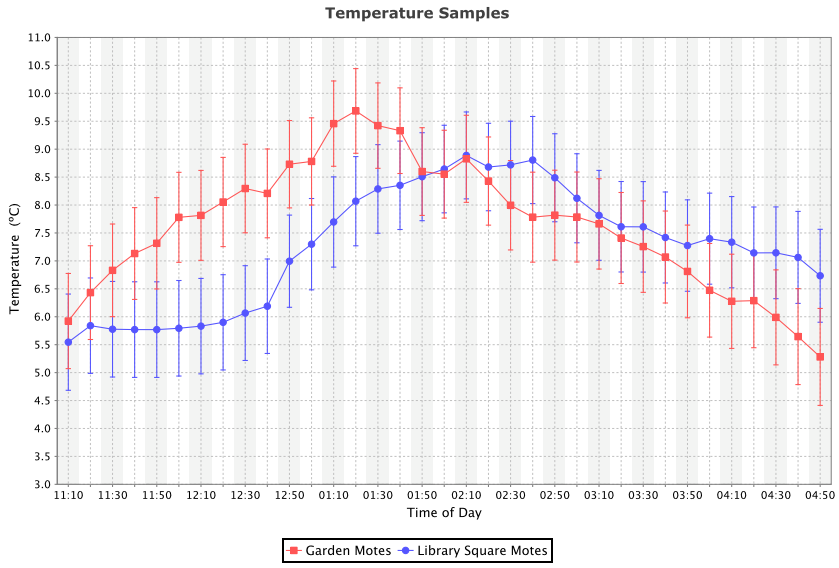

Figure 4.8: Temperature evolution on both Locations

granite stone cubes in space B), these were found not to be expressive enough to induce to great air temperature differences. In Space A the gap between the temperature readings from the ground and upper placements have close trajectories until approximately 13H30M of that day. From that moment on, both paths in the graphic follow (slightly) parallel trajectories, with a gap of roughly 1° Celsius.So it can be said that there are no significant differences between upper and ground temperatures in this space.

On its turn, the readings from space B show a different behaviour, where the temperature evolution paths from the bottom and upper sensors cross themselves in more than one spot, still with very close readings. To analyse the spatial distribution of the readings the web interface layer was used to generate a color-map to show the expected temperature gradient. Only the upper motes were used since they are representative of each location behaviour.

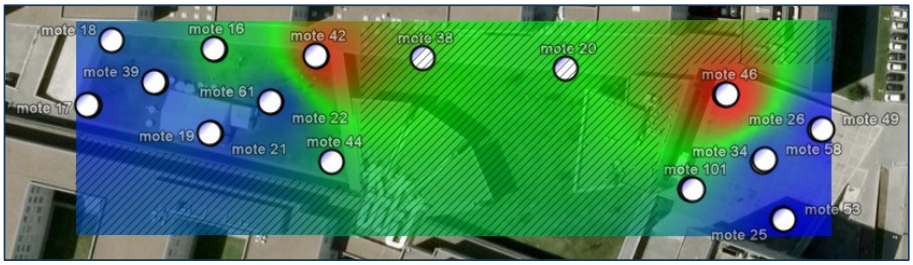

Figure 4.9: Temperature Colormap of the Upper Layers

In the graphic 4.9, absolute red corresponds to 13,5°C. Absolute blue corresponds to 6.8°C,

and in between temperatures fade to green. The dashed areas correspond to locations in which it is not safe to assume color-map accuracy due to the fact that no readings where recorded in those areas. This graphic was generated, considering each motes average temperature between 2PM and 3PM. At that moment, both regions averages are close, with the garden top layer average being of 8.79°C and the library one 8.67°C. But, one hour earlier, between 1PM and 2PM the averages are quite different as one can notice by inspecting the average temperature evolution in graphic 4.8. Generating another color-map for the average temperature between 1PM and 2PM results in figure 4.10

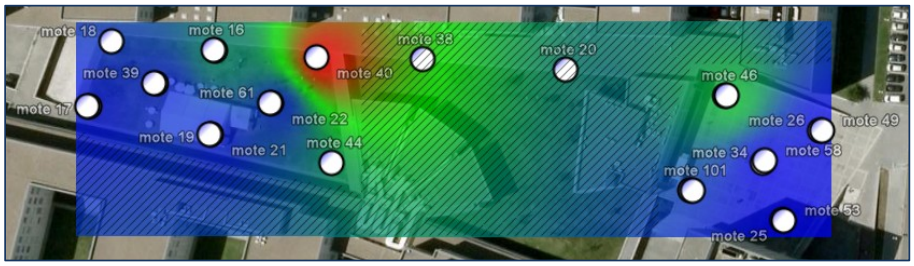

Figure 4.10: Temperature Colormap of the Upper Layers 1PM to 2PM

In the color-map graphic of image 4.10 absolute red corresponds to 18.36 degrees Celsius (node 40), and blue corresponds 6.37 (node 25) degrees Celsius. A tremendous difference between the hottest node and the coldest node that one might suspect that it could be related to an hardware malfunction. Using the“nodepage” to generate the graphic for each node average minute readings one can inspect the consistency of such results. In image 4.12 it is possible to see that in fact node 40 does have higher readings than its neighbouring motes until 12H50.

To confirm such results, the node’s CPU recorded temperature, plotted on figure 4.12 was compared to the other motes. Although these measures can be influenced by the node’s current tasks they do show that the node in question had a higher CPU temperature as well, and with a similar time variation. Light readings show that this node is one of the nodes with the highest light readings at that moment. That alone does not justify for a 12° Celsius difference with the coldest location. These sensors do have a higher error margin at these temperatures. This gap could be reduced to approximately 10 ° Celsius if one considers the maximum errors at these temperatures.

The one missing reading that could explain it all is wind-speed. From a practical perspective, this node is the one closest to a wall; a fact that can possibly reduce wind speed. Also this light-coloured concrete wall, as it reflects most part of the incoming direct solar radiation, might have had radiated an additional amount of temperature upon the sensor. During deployment, all members of the staff felt that the library square was windier than the garden. By coincidence a picture was taken of that node: the one that shows one of the constructed stands 4.2. That picture shows how the bright concrete wall may have contributed to the higher temperature readings.

The humidity graphic 4.13 shows higher differences between both locations. In Space A both layer’s readings are quite spaced apart, defining two very distinctive trajectories on the graphic.

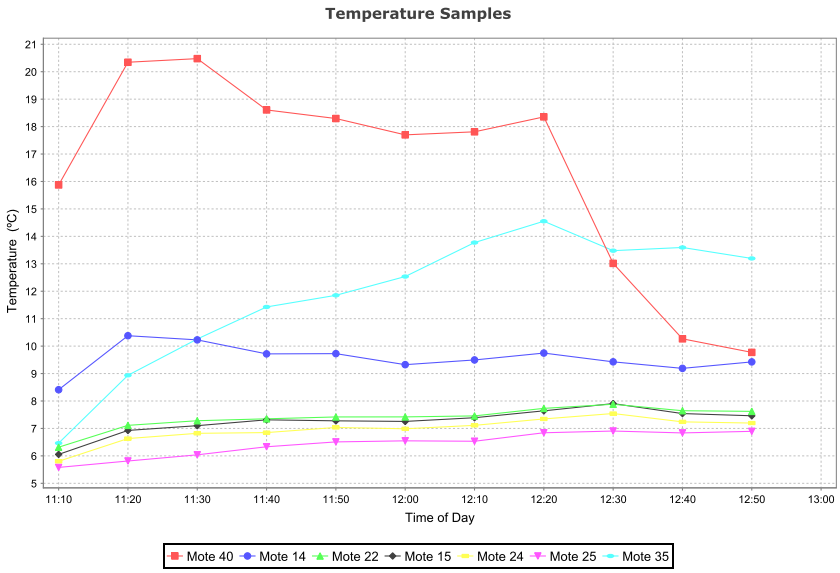

Figure 4.11: Temperature Evolution Of Upper Layer Motes

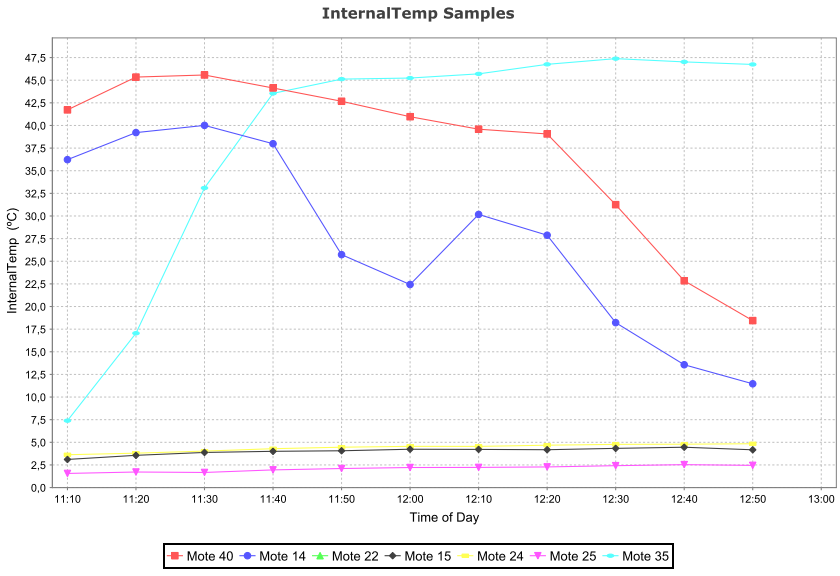

Figure 4.12: CPU Temperature Evolution Of Upper Layer Motes

Comparing location A readings with the B ones yields a more interesting fact. The Space B readings seem to follow the same path and are very close, as it would be expected for the paving material is made of granite stone, thus, does not present a transpiration rate. The space A top layer readings show lower relative humidity than space B, whereas the ground readings show a much

higher relative humidity than the one registered in space B.

Humidity out-coming from vegetation (in this case grass) is better observed in later hours towards the sunset, while it remains in a low rate when the sun strokes it directly. Whereas space B as it is constituted of granite stone, as aforementioned, presents more stable values. In this context, one might suppose that the mere existence of a green grassed area is not enough to produce expressive humidity on the top layer of space A. Consequently, it can be supposed that if there was more vegetation in number and species (namely trees in a more mature age from those in space A), the upper measurement layer would probably present completely different values where humidity rate would appear much higher than the space B's one.

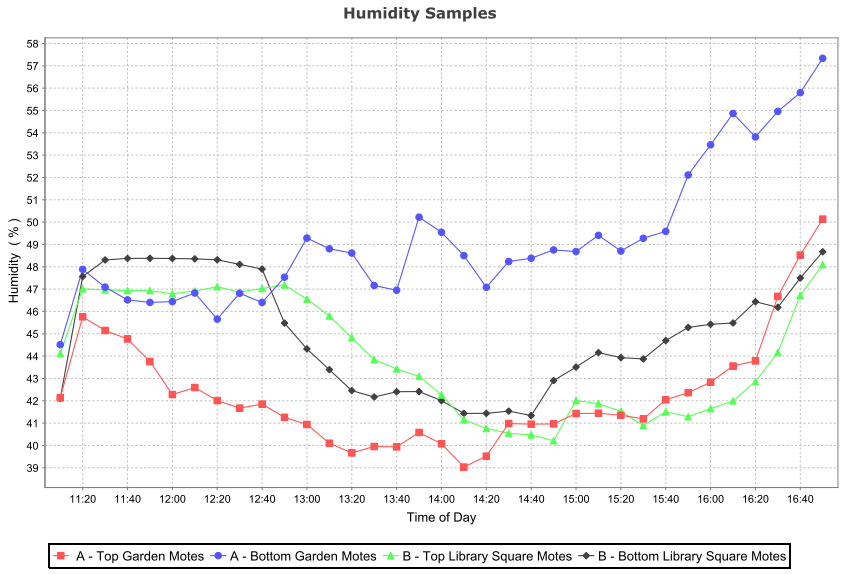

Figure 4.13: Relative Humidity on both Spaces

Knowing that each graphic series results from the average of several motes, makes this result even more interesting. By the end of the deployment, with the exception of the garden ground motes, the humidity samples converge to the same values. Further analysis on each individual mote reading shows similar results between spaces.

The geometry of the buildings has a significant impact on light levels. The sun does not shine homogeneously on all of the chosen spots. So the averages must be interpreted with caution. Averaging the light levels at each minute in each layer and location results in figure 4.14.

Public lighting becomes a fact to take into account by approximately 5PM. The space A top motes, and the space B top motes where both facing north, where has the ground motes light sensors where all (most of the time) pointing up towards the sky. Most of the day, a considerable part of the garden space is cast by a shadow from one of the buildings. The Library square has on average slightly higher light readings than space A. A fact that can help increase this difference is the light refraction that may occur on the buildings and grounds concrete surface. Some strange

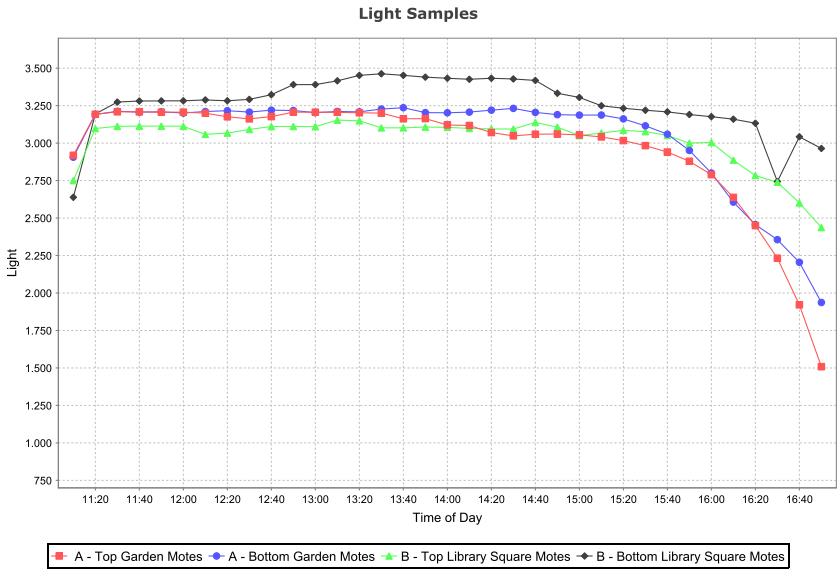

Figure 4.14: Light Averaged every 10 Minutes

phenomena happened with the readings from library square, bottom layer motes at around 16:20 PM. Probably someone or something interfered with one of the motes. Turning one of the light sensors towards the ground is sufficient to cause such an impact.

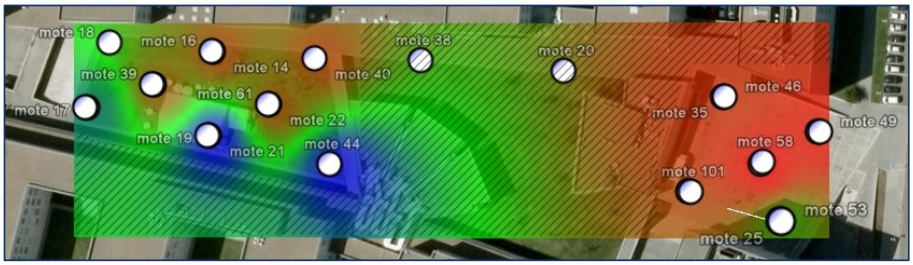

Figure 4.15: Light Colormap

The library square pavement and buildings surface is quite homogeneously light-grayed, with no apparent light absorbing elements. The garden is in theory less prone to reflections, due to the fact that a greater area is kept shadowed during most of the time. To further analyse this effect, a color-map figure 4.15 is used to show the light readings between 15PM and 16PM. In this color-map (figure 4.15) absolute blue is equivalent to 1251 *(*lx) and absolute red equals to 3300 *(*lx). In this figure it is possible to see the region that was kept most of the time shadowed.

This experiment allowed some initial insights on the behaviour of different, but near spaces. To achieve better results and perhaps observe different behaviours under different weather conditions this kind of deployments should be employed through a sufficient amount of time to detect a

behavioural pattern at each location. Therefore, it is not possible to conclude, without further studies the repeatability of the exposed facts. In this context the system proved to be a usefull tool for Architects and Civil Engineers, allowing them to characterize the environmenal behaviour of a given area. This information can be used to estimate developments or enhancements on locations of the characterized area.

4.4 Network Behaviour Analysis

Both days of experiments showed that the algorithms employed, and the tools developed worked as expected. There was random message loss, probably due to traffic overload at congested nodes. A random collecting time could be set at each node to prevent such events from happening at larger deployments. Lets start by analysing the consistency between the node's delivered sample messages.

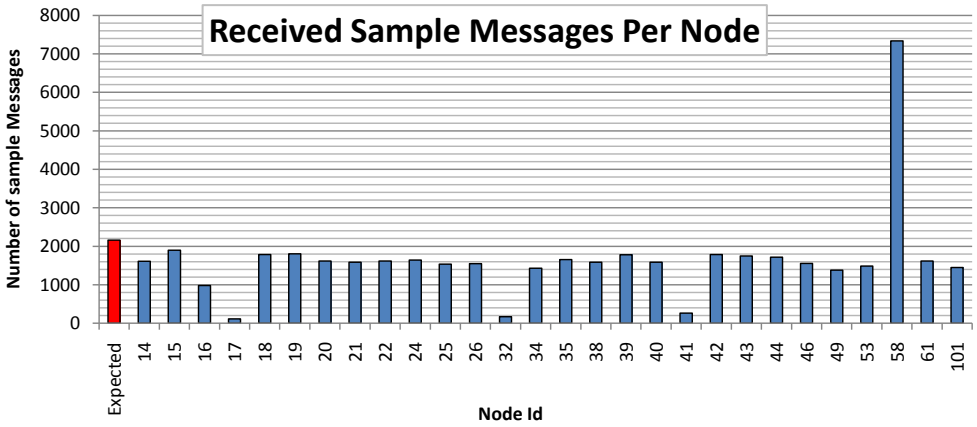

Figure 4.16: Number of Received Samples per Node

In figure 4.16, each nodes message count is plotted. Knowing that the deployment time was exactly 6 hours then the number of expected sample messages equals 2160 per node. None of the message counts reach that value. One of the nodes (node 58) actually sent more messages than what it was supposed to. Checking the database, it seems that the network reconfiguration message containing the defined sample and collection time did not reach the node in question. Motes 17,32 and 41 on the other hand failed to deliver most of their messages. In figure 4.17, the message delivery faillure rate is shown in percentage, disregarding mote 58. This analysis is made by taking the ratio between the actual message count and the expected message count.

Considering all nodes the average message failure rate equals 33%. Disregarding the motes that (probably) due to malfunction, show failure rates superior to 50% than, this failure rate decreases to 25% which is a good result considering the high number of links a message needs to travel to reach the base station and that also no active method was enforced by the base-station to request retransmissions. Taking into account that 6 samples were taken at each minute, then, if

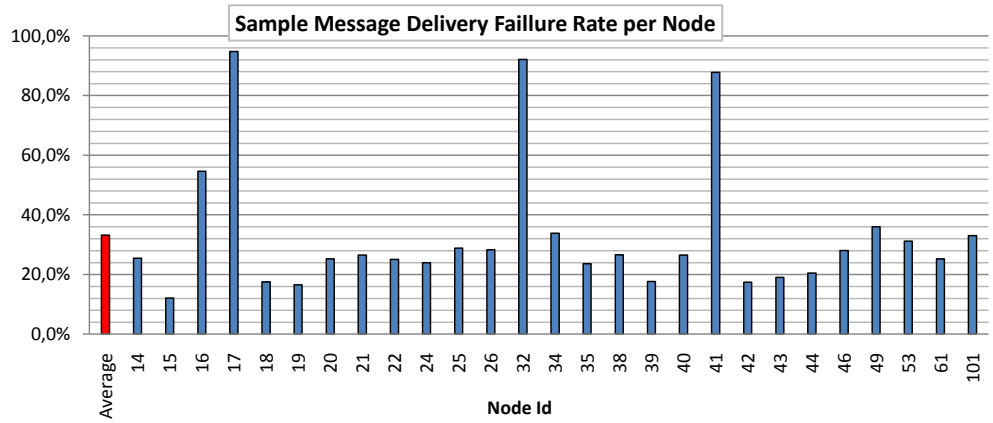

Figure 4.17: Difference Between Expected and Received Sample messages (Percentage)

message loss follows a random pattern, the effect on the daily variation readings is minimal. Consulting the database one can count the number of messages per minute to find out if the message loss is random, or if it occurs during periods of time wider than one minute. As described in the previous subsections there are in fact minutes without apparent readings from some of the motes. But are these blank minutes responsible for the total message loss?

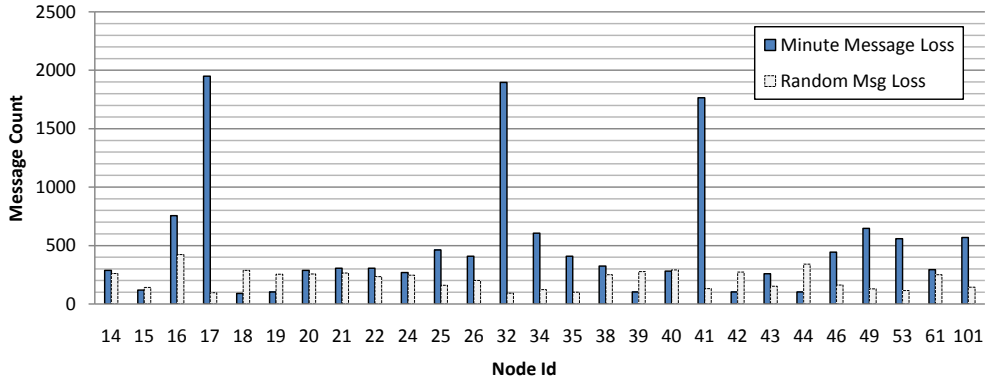

Figure 4.18: Comparison Between Null Minutes with Random Message Loss

Looking to figure 4.18, in most of the nodes, the number of "blank" minutes is responsible for most of the message loss. The blank minutes represent only part of the total message loss per node. A substantial part is still random. To calculate the total random message loss a simple subtraction between total message loss, and blank minute message loss is done. Observing the random message loss series, it seems that it does not correlate itself with the messages lost in blank minutes in any specific pattern.

Redundancy provided by the extra nodes at each location can only benefit the average readings if the blank minutes don't coincide between nodes. Blank minutes suggest that some congestion,

at a certain node, or interference happened during transmission time. This lead to a total message loss, or partial message loss during collection (samples were collected every 2 minutes). To find out the cause of the blank minutes, each mote's average minute samples was compared in order to generate a graphic of "null" sample minutes concurrency between motes.

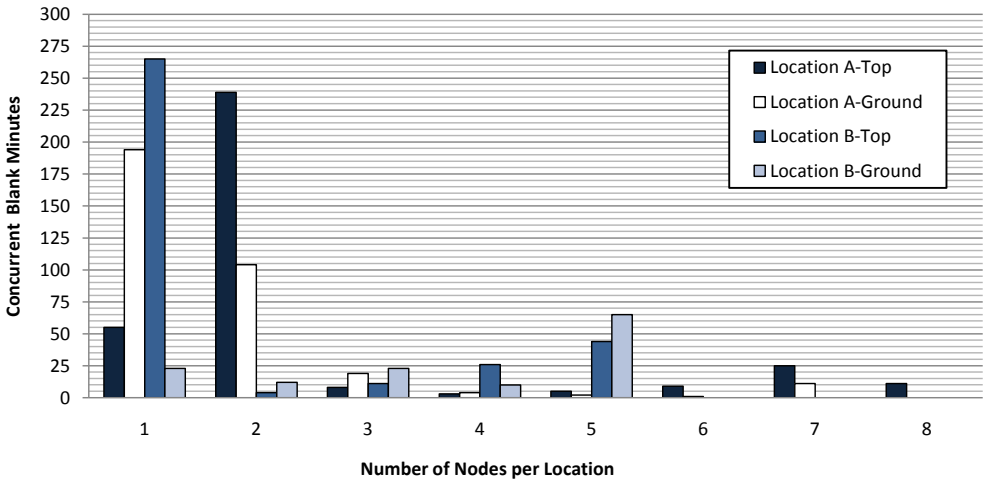

Figure 4.19: Null Minutes Concurrency

Figure 4.19 shows that there are few minute gaps with no readings from location A. Using the interpolation methodology devised in the previous subsection this effect can be minimized. The majority of concurrent blank minutes happens with three or less motes at each location. This proves that the redundancy employed by using several nodes at each layer works in minimizing the loss of readings. However, by adding more nodes more strain is placed on the network to support message flow. Nodes with direct contact with the base-station should, in theory, present minor message loss. However, if they are routing their neighbour mote's messages, than that particular node can become clogged up with the "forwarding" task . Also external interference can disturb link quality. To inspect the possible correlation between message loss and the position of the mote in the route hierarchy the ETX parameter needs to be analysed.

Before proceeding with such analysis lets first find out whether the employed time synchronization algorithm provided good results under these circumstances. Due to the short duration of this deployment the LPL feature during the experiment day was deactivated.

The graphic in figure 4.20 shows that the time sync algorithm worked on all nodes, and that it performed as expected. Also, it should be noted that the time-sync error is only an undetermined fraction of this time, due to the facts explained in Chapter 3. What is possible to say by analysing graphic 4.20, is that the time sync error is inferior to the total time elapsed in each motes round-trip message.

In figure 4.21 and 4.22, it is possible to compare various network performance values, between the motes at each location. In both graphics, all values were divided by the appropriate maximum registered value in order to produce each series (normalization). There is no apparent

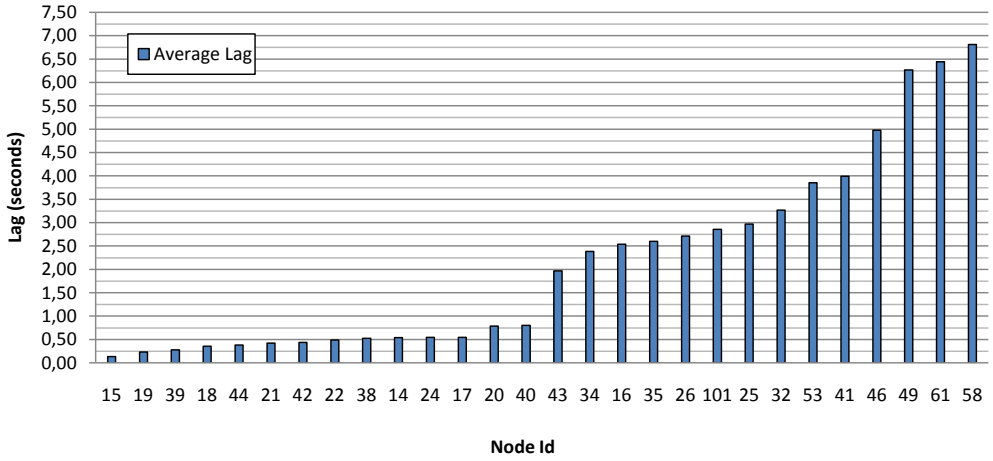

Figure 4.20: Average Lag

correlation between the average ETX and message loss or the round-trip time. Further more, the "Pearson product-moment correlation coefficient" between the average ETX and the average message loss per node is of 0.06; between the ETX and the round-trip time is of 0.03; and finally between message loss and the round-trip time is of 0.07. Values in the vicinity of 1 or -1 would indicate a linear dependence between the variables. These values clearly show that there is no linear relationship between any of these parameters.

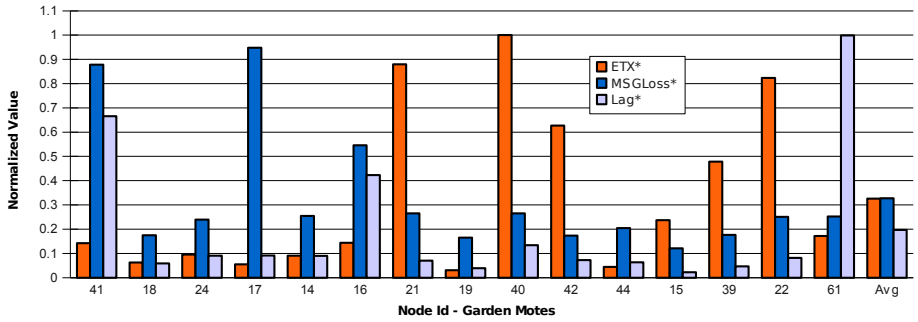

Figure 4.21: Garden Motes - Comparison of Performance Values

Correlation between the average ETX and the round-trip message time was not expected to happen due to the fact that the LPL was disabled. It also indicates that the collection paths established were not the same used by the dissemination interface. The library as on average higher ETX and round-trip times, although by a negligible margin. Another interesting fact is the very high difference for both parameters between nodes placed in similar locations. The high ETX values can lead to the conclusion that those motes were struggling to find a path for message delivery due to low link quality. There are two facts that can explain both message loss and the high ETX

by some motes. The first fact relates itself with the communication protocol used by the motes' radios, where as the second is related with external interferences.

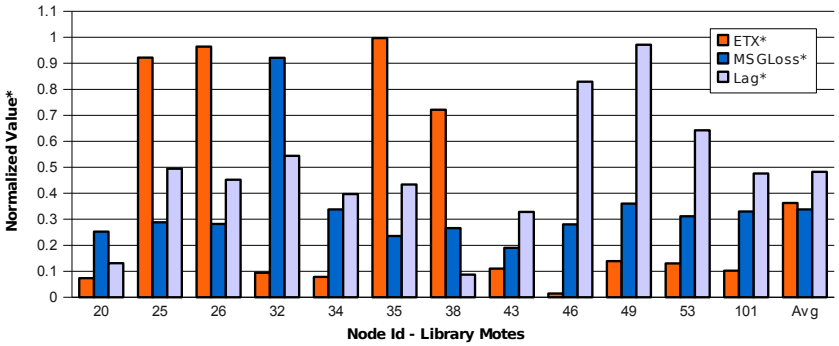

Figure 4.22: Library Motes - Comparison of Performance Values

Looking through the TinyOs documentation [61] it is possible to find that the CC2240 radio enforces the use of the IEEE 802.15.4 MAC. This means that the radios are using carrier sense medium access with collision avoidance. Hence, a mote before starting to communicate listens the channel inspecting for channel activity. If no activity is found then it proceeds with its communication.

Consider two motes, Y and X, not aware of each other that try to communicate with a third node K. The third node can be in reach of both, which results in a collision from the Y and X motes messages. This collision may be undetected by Y and X if no acknowledgement is made by the K node. Imagine that the K node informs Y and X that a collision has happened (which is what in deed happens when using the IEEE 802.15.4 standard). Y and X will calculate a random time to resend their messages. However, lets consider another node, node Z, which is parent of node K and a new node K'. Node Z sees both K and K' messages, however, both K and K' are unaware of each other. We already know that K is forwarding messages from Y and X, and that collisions have happened. Z may see the same effect from K (actually the K plus X and Y messages) and K' messages. There is a recursive collision problem that will unavoidably end in message loss, or even message corruption.

In small and close deployments, this behaviour should not be verified as it was not during testing. Looking at the average established collection tree it is possible to observe a very strange effect, where some of the motes choose not to follow the closest path to the base-station. Instead they choose a parent node, that is actually in the middle of both A and B locations. Lets call this node the "C" node. Obviously this parent node sees messages from both locations, where both locations are unaware of each other. This results in message collision and a high retry level from site A and B. This problem is commonly known has the "hidden terminal problem". Thee IEEE802.11 standard [62] widely used in wireless networks diminishes this problem by using a "rts" and "cts" handshaking approach, that works as a virtual carrier sense method to detect hidden nodes. The IEEE802.15.4 was designed for lower data transmission rates. To reduce network

overhead it makes no use of such feature and it relies solely on retransmission after a random time. To reduce latency, the maximum random time is however shorter than the one devised in the 802.11 standard, possibly increasing the probability of new collisions.

It is expected however that motes from site A are aware of each other due to their close proximity, and motes on the B site too. So the CSMA prevents message corruption there. Another strange effect is that the C node sends all messages back to another node in the A location, creating a cross-talk effect between the A motes. Some of the A motes are prohibited to communicate due to their own messages coming back from the C node. This would in theory explain the lost "blank" minutes if we add the messages from site B. However, there is the second mentioned fact: external interferences.

The frequency used by wireless sensor nodes varies from 2.405 MHz to 2480 MHz, in bands with a width of 3 MHZ and their centers spaced apart by 5 MHz. These 26 frequency channels overlap with the ones used by a broad array of appliances. For instance, Wi-Fi [62], Bluetooth enabled devices, wireless keyboards/mousse-pads and several remote control devices, all communicate using portions of the same frequency spectrum. Also microwaves emit radiation at these frequencies.

This particular deployment was configured to operate on channel 26 of the IEEE802.15.4, trying to prevent or minimize conflicts with the Wi-Fi routers and access points that are present throughout the FEUP campus. This strategy would work if the routers and access points used on campus all complied with the USA legislation, where the Wi-Fi channels do not overlap channel 25 and 26 of the IEEE802.15.4.

Unfortunately, in Europe legislation allows such frequencies to be used by Wi-Fi, leaving in fact no guarantees that a free channel exists to be used by IEEE802.15.4 compliant radios.

An in-depth analysis of interference between both these technologies [63], shows that when the WSNs are used in environments where they have to compete with Wi-Fi networks, communications can incur in message loss up to 58% depending on both networks activity. The FEUP campus has currently 129 Wi-Fi Access Points (Aps). Looking through these access points datasheet [64] it is possible to see that the maximum transmission power available is of 100mW. However IEEE802.15.4 used by the motes allows a maximum transmission power of 10mW. In reality it is inferior values are normally used due to the power constraints of the devices.

The motes signal can be completely obliterated by a competing Wi-Fi router in the same channel. The access points used on the FEUP campus are designed to comply with the legislation used in either continent. They have an auto-channel detection which allows them to in practice jump between the less occupied channels in order to maintain quality of service. However, as mentioned the motes frequency channel is narrower and the signals output power is lower, which might not be enough to guarantee that the access points would jump to another frequency if is using that channel.

The most practical way to guarantee that no access points or Wi-Fi routers or access points are using the same channels is to simply scan for the available access points , this can be done easily by using common tools provided by standard computer operating systems. This showed that no access

points of the FEUP Campus are currently using channel 13 or 14 of the IEEE802.11.b/g standard that overlap with channel 26 of the IEEE802.15.4. However it did show another interesting fact : the presence of private WLans that exist in each department, and even more unexpectedly some sudden adhoc networks, probably used by some of the campus students.

These networks are not always on, and in practice there is no guarantee that one of such configurations cannot surge using channel 13 or 14 of the IEEE802.11b/g. In fact using such channels seems to be a common strategy to avoid throughput issues by not sharing channels that are commonly used by wireless access points or routers. It also enables wider free regions of the band gaps than the ones of the lower channels, being that there are no other upper Wi-Fi channels.

One might conclude that if present, these external interferences can aggravate the hidden node terminal problem by adding extra channel occupancy at given locations. This effect can escalate even further if considering that messages are forwarded in a multi-hop fashion until reaching the base-station. Any sudden interference in one of the links can actually disable an entire route and could in theory also explain the blank minutes.

Bluetooth is not expected to cause any interference due to its frequency hopping behaviour. WirelessUsb is still in its early infancy, but already addresses this issue by searching for the least used channel. Currently in Europe cordless phones are normally used in frequencies near the 1.9GHz frequency using the ETSI DECT standard, keeping an open channel only for these appliances. DECT can also be used on the 2.4GHz spectrum on the USA. It is not uncommon to find online vendors selling 2.4GHz cordless phones shipping world wide that use other communication schemes. These phones can cause extreme interference with Wi-Fi devices or ZigBee/IEEE802.15.4. Such is the case when frequency hopping spread spectrum (FHHS) is used to transmit the signal on cordless phones. Once someone answers a phone, the signal hops within the frequencies on the entire 2.4GHz.

Looking through online forums many consumers report the loss of Wi-Fi connectivity once they answer their cordless phone, being that sometimes their routers are configured to operate on frequencies that the phone uses. Another problem is the power of the transmitted signal from these devices which is typically on par with the Wi-Fi signal, so if the signal would jump on to the frequencies of the motes, channel 26, then an ongoing communication would be destroyed.

There are other sources that can cause interference. One would be devices that would normally not operate under such frequencies but that, for some reason, due to a malfunction or design flaw they did. Other, more obvious, would be other IEEE802.15.4 compliant radios operating on the same channel, like for instance ZigBee appliances. It is not an easy task to find out whether a source of interference appeared during the deployment. There are still ways to minimize this problem. One of the solutions is redundancy. In this deployment, redundancy allowed to fill the gaps when some of the motes have failed to deliver their messages. But to determine when redundancy starts to have a negative impact, due to the high number of sending attempts, would be in itself a motive for other experiments and deployments.

Other ways involve time synchronized frequency hopping techniques, similar to BlueTooth, but still doing it blindly throughout the available communication channels would lead to message

loss. One example is Smart Dust's proprietary communication protocol, that is used with their motes. Their time synchronized frequency hope meshing algorithm tries to minimize interference in the RF spectrum. Other schemes propose a voting mechanism where all the motes agree to change to a globally less used channel [63].

A more refined approach would be identifying the least used channels between neighbouring links. This could in theory be done between the motes, by scanning in predetermined time intervals all the IEEE802.15.4 channels searching for noise. Then they could fall back to a predetermined channel and commit themselves with their immediate neighbours to periodically shift to the least locally used channel. By assigning different periods for this shift, a chain of motes can establish a route that avoids multiple interference sources that are spatially distributed throughout that route. The messages travel through the motes that shift communication channels and circumvent different interference sources along the communication route. This is similar to the combined use of TDMA and FHHS.

Lets imagine that no external interference sources were present. A safer approach to minimize competition for the communication channel would have been the use of different collection times for each location or even each mote. This would in effect work much like the time slotting seen in some communication protocols. There is actually no need to alter any of the developed features. This could be done by simply using the developed network manager GUI and assigning different collection times to each mote, not allowing multiples and giving a wide enough interval of time for each mote to send its messages. This method was not employed from beginning due to two facts:

- To evaluate the possibility of embedding this kind of system onto control structures, such has the one s that control the buildings heaters and air conditioners, or even other systems which control environmental variables that evolve at a faster pace, it can be crucial that information about all the measurable points is delivered roughly at the same time. Therefore, relying on the CSMA to avoid collisions and deliver the information as soon as possible seemed like the approach to follow.

- Previous testing during development did not show considerable values of message loss, but no "clustering with hidden node terminal effect" was experienced either, probably due to the close proximity of the motes during testing and also due to the inferior number of nodes.

Another safe approach that could have been used is "token" passing. One "token" could pass around the motes allowing each mote to send its message. The mote would then need to send the token to another mote to allow it to send its messages. Token passing would minimize the gaps between sends. To use this method it would be necessary to alter the features already developed in the lowest layer: the motes software application.

One approach that could be used, if using a different kind of motes could be the use of a different MAC Layer that provides a scheduled behaviour for communication. Scheduled communication behaviour also minimizes energy consumption by decreasing the number of retransmissions due to collisions.

Chapter 5

Conclusions and Future Work

This last chapter concludes the thesis. It provides a brief review over the developed work, together with an analysis of the overall obtained results. Towards the end of this chapter some suggestions for further research are exposed and discussed.

5.1 Review,Discussion and Implications

This thesis has shown that wireless sensor networks are well suited for scenarios in which a distributed data gathering system is needed. During the case study one could verify that the system allowed an in depth micro-climate analysis. Further more, the flexibility provided by the cable free installation allows the easy deployment and recovery of the sensor nodes, increasing work efficiency. This is perhaps one of the main interests on this technology. The installation time, and the small repair/maintenance time.

Although there are many advantages towards the use of WSNs, there are still strong issues pertaining network stability, link quality and power management. Many devices operate under the same radio frequencies as the motes in order to establish links. In a noisy environment the nodes are forced to boost their radio signal in order to successfully deliver their messages, but in many cases this is insufficient to prevent packet loss. In the end, interference from external sources results in higher power consumption due to retransmissions and higher transmission power and therefore smaller battery lifetime.

This was confirmed by the developed application and with the data recovered during deployment. The available routing protocols on the Tiny Os software library had better performance when used in small clustered networks. The issues found with the “hidden terminal” effect, together with strange routes, resulted in message loss. Still this message loss was roughly a quarter of the total expected messages. One can argue that closed and proprietary solutions may have better performance, but still, this is done at a higher cost, and normally less flexibility.

5.2 Main Results and Conclusions of the Thesis

During this work it was shown that it is possible to develop and integrate environmental data retrieved by wireless sensor networks, as well as the network behaviour with earth mapping tools. This integration offers to the end users the opportunity to discover and relate the gathered data with its geo-location.

The ability to observe the network links evolution in near real time on top of a mapping tool opens the opportunity for developers and researchers to analyse their routing algorithm performance in a visual way, correlating the network performance with the nodes location. If the nodes are moving, or attached to a moving platform with GPS coordinates this tool can become even more interesting. Re-using this ability on other systems can be done by simply using a database server instead of the embedded database, and storing the network parameters on that database.

The data analysis made via web-pages allows easy access to the data retrieved by a wireless sensor network. By generating color maps for the gathered variables, a spatial perception of the variation of a given measure can be obtained.

Developing a multi-purpose application to run on the nodes is a task that involves careful planning. The application worked as planned and fulfilled the requirements. On the overall, the proposed architecture can be expanded to gather data from other sensors, and maintain the same reconfigurability.

The network management GUI proved to be a valuable tool during deployment and testing. Managing networks with more than a dozen nodes using command line tools is not an easy task, and is highly error prone. Adding the database interface to the GUI also allowed easy inspection of the nodes sensor readings.

5.3 Suggestions for Further Research

We are currently seeing an increase in wireless enabled devices, from laptops to cell phones, passing through digital cameras. The same applies to wireless sensor nodes and WSNs in general. One can notice a different trend when looking at the solutions developed for WSNs between companies and academic research centers. There is usually an overshoot of functionalities provided by the hardware and software routing algorithms, on the solutions developed by academic researchers. These solutions are flexible, but still some provide inconsistent reliability. Companies are employing a different approach: hardware targeted for specific functions, attempting to provide reliability albeit at the cost of flexibility.

The previous statements show the need for new metrics and well established, standardized routing paradigms on these highly constrained devices. One should be able to compare the reliability, stability and power consumption between hardware with similar or different functionalities, using meaningful measures. The same is valid for routing algorithms in a variety of usage scenarios. If these metrics are enabled, not only the end user is benefited but also the developers can target specific products to emerging market segments.

Regarding the outcome of this thesis, there are several possibilities that could be developed to extend the functionalities already developed. Perhaps one of the first functionalities that could be added would be the integration of GPS on the motes. This would enable the ability of deploying these motes and automatically track them on Google Earth, using the developed software. Moreover it would allow hybrid scenarios where motes could be deployed on static or moving objects, and the moving motes could also be tracked using Google Earth, as long as their messages could find a way to the base station. Only small changes would need to be made to the database and communication model.

Air quality could be measured by adding air quality sensors, like CO2 sensors to the existing motes. The application developed could easily create a color map indicating the areas of a city with the lowest and highest air quality in almost real time. This would allow studies to be conducted with finer grain, and also identifying previously unknown polluting sources.

Another interesting feature that could be added would be an acceleration sensor. This would allow to know the motes acceleration when placed on moving objects. It could also be used to characterize object parameters like vibration, and estimate as well the forces being applied to it by extending the developed application.

There is an increasing interest in energy monitoring solutions. In factories, high power efficiency can be achieved by using machinery that comes with pre-built energy monitoring solutions. However, there are still allot of appliances that could use a separate energy monitoring solution in order to track down defective electrical installations, for instance in the building's lighting installation. Energy consumption studies could be done using the platform developed by adding electric current sensors to the nodes, and extending or adding a new message type. This sensors can normally be placed around wires to evaluate current flow. These nodes could be deployed in strategic points at factories, or even at peoples' homes, enabling a superior perception of how energy is being spent. Power consumption information could be added to the geographical representation of the nodes on Google Earth in order to easily identify problematic spots. With a sufficient number of nodes, the wires and their electric current intensity could even be represented on Google Earth using lines, similar to the ones used to display the network links.

In theory, by conjugating power consumption of the buildings air-conditioners or heaters, with the information provided by the temperature, humidity and light sensors, it could be possible to conduct studies that estimate a buildings thermal efficiency. A group of nodes could be placed inside the building monitoring the power drawn by the air conditioners, and also the environmental variables. Another group of nodes placed outside the building would monitor the surrounding environment. By conjugating these readings, the thermal behaviour, together with the spent energy, could be correlated.

Taking this idea one step further, the system could be used in a control structure, in order to adjust the air-conditioners outputs. This control could take into account the energy being wasted to achieve a determined temperature at a specific point. Also, because it allows distributed sensing, it could try to ensure a more uniform temperature throughout an area, if several air-conditioners are present. In buildings that allow remote control of windows or entrance points, and windows

shutters, this control could be extended. It would allow enabling or disabling the surrounding environment effect on the building temperature and light levels.

The idea of embedding this system onto a control study would be in itself motive for further research. There are questions pertaining the reliability of the network links due to the nature of the wireless links, even without external radio frequency interferences. Also security issues may arise due to the easy access to the wireless transmission channels. Power consumption levels must be designed carefully in order to satisfy the systems required deployment lifetime. Guaranteeing a constant time-sync, or a certain maximum amount of time-sync error between several nodes is necessary to allow this kind of systems to be embedded in control loops. This kind of systems are still in their early infancy when it comes to their usages on control systems.

Another possible application for these systems is extending it to cooperate with autonomous networked vehicles. Networked vehicles can use a wireless sensor network to communicate with a base station. For instance remotely operated, or autonomous vehicles can periodically drop motes in order to establish a communication path to a base station. This for instance would allow the exploration of harsh environments like mines or caves, not only by allowing the retrieval of spatially distributed data, but also by allowing remote interaction with the deployed nodes, by commanding the vehicle. Another paradigm would be allowing the WSN and the vehicle to operate autonomously in order to characterize a given area. The motes could parametrize a given location, and ask the vehicle to transfer them to another location. One can even envision that unmanned air vehicles can drop entire WSNs to remote sense a location. The motes could either try to send back their data using a long range connection to a base station, or they could wait for some kind of vehicle to come by and collect the data.

The idea of not establishing a wide range connection to a base station would fit, particularly well, scenarios where communication security matters. Motes have already been used, by employing magnetometers, to detect and identify the passage of vehicles, like cars, trucks or even war tanks. The combination of networked vehicles with wireless sensor networks opens a broader area of research.

Appendix A

Section A

Figure A.1: Developed Stand Model

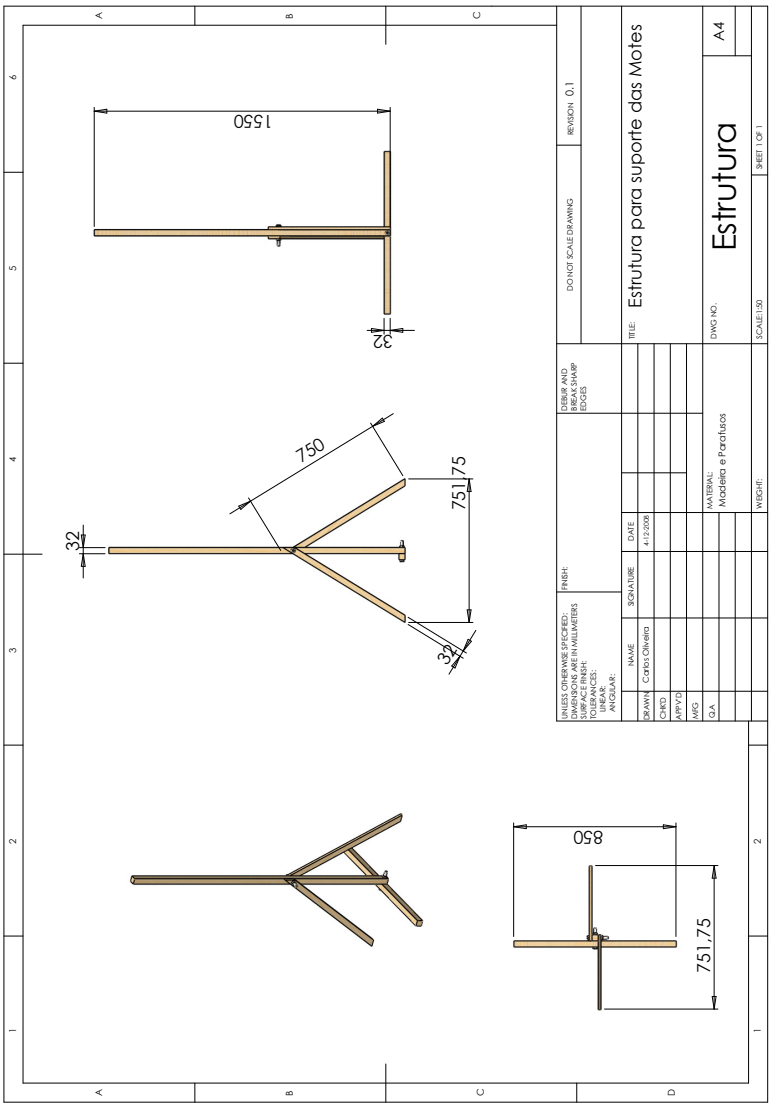

Table A.1: The Deployment Correlation Table

Correlation Table - Stands - Motes - Coordinates		
Garden		
Stands	**Motes**	**Latitude,Longitude,Altitude**
1	41	-8.596531398224045,41.17802296894783,1.5
	18	-8.596531398224045,41.17802296894783,0
2	24	-8.596603511393255,41.17788142785942,1.5
	17	-8.596603511393255,41.17788142785942,0
3	14	-8.59623554020142,41.17800685969063,1.5
	16	-8.59623554020142,41.17800685969063,0
4	21	-8.596246715969059,41.17782302958686,1.5
	19	-8.596246715969059,41.17782302958686,0
5	40	-8.595939043566105,41.17799067487688,1.5
	42	-8.595939043566105,41.17799067487688,0
6	100	-8.595893132676693,41.1777604699606,1.5
	44	-8.595893132676693,41.1777604699606,0
7	15	-8.596408183983186,41.17793397614538,1.5
	39	-8.596408183983186,41.17793397614538,0
8	22	-8.596070924972736,41.17789104270413,1.5
	61	-8.596070924972736,41.17789104270413,0
Library Square		
Stands	**Motes**	**Latitude,Longitude,Altitude**
9	35	-8.594750897371185,41.17790596909597,1.5
	46	-8.594750897371185,41.17790596909597,0
10	32	-8.594795667661487,41.17781500073819,1.5
	101	-8.594795667661487,41.17781500073819,0
11	26	-8.594853195883346,41.17769960606351,1.5
	49	-8.594853195883346,41.17769960606351,0
12	25	-8.594415965532591,41.17781493194507,1.5
	53	-8.594415965532591,41.17781493194507,0
13	34	-8.594493287414665,41.17773313010655,1.5
	58	-8.594493287414665,41.17773313010655,0
Extra C	38	-8.595629731885836,41.17798781809159,1.5
Extra D	20	-8.595214132416373,41.17796290561901,1.5

References

[1] http://www.sensirion.com/images/getFile?id=25 last visited on January 2009.

[2] Modbus-IDA. Modbus specifications and implementation guides, July 2006. http://www.modbus.org/specs.php last visited on January 2009.

[3] Iso 11898-3 controller area network (can), 2006.

[4] Eia standard rs-232-c interface between data terminal equipment and data communication equipment employing serial data interchange, August 1969.

[5] Eia standard rs-485 interface between data terminal equipment and data communication equipment employing serial data interchange, 1985.

[6] Ieee standard for information technology-telecommunications and information exchange between systems-local and metropolitan area networks-specific requirements part 3: Carrier sense multiple access with collision detection (csma/cd) access method and physical layer specifications, 2006.

[7] Feup wsn projects webpage, January 2009. http://whale.fe.up.pt/wsnwiki/index.php/Projects.

[8] J. Pinto, A. Sousa, P. Lebres, G.M. Gonçalves, and J. Sousa. Monsense–application for deployment, monitoring and control of wireless sensor networks. *ACM Real WSN*, 2006, 2006.

[9] IF Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[10] Sentilla. A new vision for pervasive computing - white paper. Technical report, Sentilla Corporation, 2007.

[11] C.Ó. Mathúna, T. ODonnell, R.V. Martinez-Catala, J. Rohan, and B. OFlynn. Energy scavenging for long-term deployable wireless sensor networks. *Talanta*, 75(3):613–623, 2008.

[12] D. Carroll and M. Duffy. Demonstration of wearable power generator. In *Proc. European Conference on Power Electronics and Applications*, pages 10pp.–P.10, 2005.

[13] T. Starner and J.A. Paradiso. Human generated power for mobile electronics. *Low Power Electronics Design*, pages 1–35, 2004.

[14] N. S. Shenck and J. A. Paradiso. Energy scavenging with shoe-mounted piezoelectrics. *IEEE M MICRO*, 21(3):30–42, May–June 2001.

[15] Seiko kinetic wrist watch. http://www.seikowatches.com/technology/kinetic/index.html last visited on January 2009.

[16] X. Jiang, J. Polastre, and D. Culler. Perpetual environmentally powered sensor networks. In *Proc. Fourth International Symposium on Information Processing in Sensor Networks IPSN 2005*, pages 463–468, 15 April 2005.

[17] MoteiV. Tmote sky datasheet, November 2006. http://www.sentilla.com/pdf/eol/tmote-sky-datasheet.pdf last visited on January 2009.

[18] M. Awenuti, P. Corsini, P. Masci, and A. Vecchio. Increasing the efficiency of preamble sampling protocols for wireless sensor networks. *Mobile Computing and Wireless Communication International Conference, 2006. MCWC 2006. Proceedings of the First*, pages 117–122, Sept. 2006.

[19] N. N. Pham, J. Youn, and C. Won. A comparison of wireless sensor network routing protocols on an experimental testbed. In *Proc. IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, volume 2, pages 276–281, 5–7 June 2006.

[20] D.S.J.D. Couto, D. Aguayo, J. Bicket, and R. Morris. a high-throughput path metric for multi-hop wireless routing. *Wireless Networks*, 11(4):419–434, 2005.

[21] Rodrigo Fonseca, Omprakash Gnawali, Kyle Jamieson, and Philip Levis. Collection (tep 119). http://www.tinyos.net/tinyos-2.x/ last visited on January 2009.

[22] W.R. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 174–185. ACM New York, NY, USA, 1999.

[23] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 56–67. ACM New York, NY, USA, 2000.

[24] Curt Schurgers and Mani B. Srivastava. Energy efficient routing in wireless sensor networks. In *in the MILCOM Proceedings on Communications for Network-Centric Operations: Creating the Information Force*, pages 357–361, 2001.

[25] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *HICSS '00: Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8*, page 8020, Washington, DC, USA, 2000. IEEE Computer Society.

[26] Y. Xu, S. Bien, Y. Mori, J. Heidemann, and D. Estrin. Topology control protocols to conserve energy in wireless ad hoc networks. *Center for Embedded Networked Computing Technical Report*, 6, 2003.

[27] Winston K. G. Seah and Hwee Pink Tan. Multipath virtual sink architecture for wireless sensor networks in harsh environments. In *InterSense '06: Proceedings of the first international conference on Integrated internet ad hoc and sensor networks*, page 19, New York, NY, USA, 2006. ACM.

[28] Xiaoxia Huang and Yuguang Fang. Multiconstrained qos multipath routing in wireless sensor networks. *Wirel. Netw.*, 14(4):465–478, 2008.

[29] Crossbow technology. http://www.xbow.com last visited on January 2009.

[30] Dust Networks. Embedded wireless sensor networking for monitor and control. http://www.dustnetworks.com last visited ond January 2009.

[31] Meshnetics sensor nodes. http://www.meshnetics.com/ last visited on January 2009.

[32] http://www.zigbee.org/ last visited on January 2009.

[33] Ieee standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirement part 15.4: Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (wpans), 2007.

[34] http://www.tinyos.net/tinyos-2.x/ last visited on January 2009.

[35] Open Geospatial Consortium. Sensor web enablement, January 2009. http://www.opengeospatial.org/ogc/markets-technologies/swe.

[36] Opengis sensor model language (SensorML) implementation specification, August 2007.

[37] Sensor observation service, October 2007.

[38] Opengis geography markup language (gml) encoding standard, August 2007.

[39] Google. Google earth, January 2009. http://earth.google.com/.

[40] Google. Google maps, January 2009. http://maps.google.com/.

[41] V. Turau, C. Renner, M. Venzke, S. Waschik, C. Weyer, and M. Witt. The heathland experiment: Results and experiences. In *Proceedings of the Workshop on Real-World Wireless Sensor Networks (REALWSN)*, 2005.

[42] J Tateson, C Roadknight, A Gonzalez, T Khan, S Fitz, I Henning, N Boyd, C Vincent, and I Marshall. Real world issues in deploying a wireless sensor network. In *Workshop on Real-World Wireless Sensor Networks REALWSN'05*, Stockholm, Sweden, June 2005.

[43] Eric Paulos and Tom Jenkins. Urban probes: encountering our emerging urban atmospheres. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 341–350, New York, NY, USA, 2005. ACM.

[44] Microsoft. Senseweb, January 2009. http://research.microsoft.com/en-us/projects/senseweb/.

[45] Microsoft. Live maps, January 2009. http://maps.live.com/.

[46] Microsoft. Sensor map, January 2009. http://atom.research.microsoft.com/sensewebv3/sensormap/.

[47] http://www.oostethys.org/ last visited on January 2009.

[48] University of Melbourne. The sensorweb project: Unifying sensor networks and grid computing with the world-wide web, January 2009. http://www.gridbus.org/sensorweb/.

[49] Nasa. Volcano sensor web, January 2009. http://sensorwebs.jpl.nasa.gov/.

[50] I. Sommerville. *Software Engineering*. Addison-Wesley, 2006.

[51] Systems engineering - application and management of the systems engineering process, July 2007.

[52] Ogc kml, April 2008.

[53] Hammatsu photodiodes datasheet. http://sales.hamamatsu.com/assets/pdf/parts_S/S1087_etc.pdf last visited on January 2009.

[54] Philip Levis. *Tiny Os Programming Manual*, October 2006. http://www.tinyos.net/tinyos-2.x/doc/pdf/tinyos-programming.pdf.

[55] P.A. Levis, N. Patel, D. Culler, and S. Shenker. *Trickle: A Self Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks*. Computer Science Division, University of California, 2003.

[56] Future Technologies Devices International. Virtual com port drivers, January 2009. http://www.ftdichip.com/Drivers/VCP.htm.

[57] Tiny Os. Serial forwarder, January 2009. http://docs.tinyos.net/index.php/Mote-PC_serial_communication_and_SerialForwarder.

[58] Java servlet technology.

[59] Underwater systems and technology laboratory. http://paginas.fe.up.pt/~lsts/ last visited on January 2009.

[60] http://java.com/en/ last visited on January 2009.

[61] Cc2420 2.4 ghz ieee 802.15.4 / zigbee-ready rf transceiver datasheet. http://focus.ti.com/lit/ds/symlink/cc2420.pdf last visited on January 2009.

[62] Ieee standard for information technology- telecommunications and information exchange between systems- local and metropolitan area networks- specific requirements part ii: wireless lan medium access control (mac) and physical layer (phy) specifications, 2003.

[63] R. Musaloiu-E and A. Terzis. Minimising the effect of WiFi interference in 802.15. 4 wireless sensor networks. *International Journal of Sensor Networks*, 3(1):43–54, 2008.

[64] Cisco aironet 1100 series datasheet. http://www.cisco.com/en/US/ last visited on January 2009.