DEPARTMENT OF COMPUTER SCIENCE AND ELECTRICAL ENGINEERING

UNIVERSITY OF STAVANGER

FlockWatch
Control system for sheep tracking network

Responsible Professor:
Erlend Tøssebro
University of Stavanger

Master's thesis of:
Laurenţiu Cocanu

Supervisor:
Ragnar Stølsmark
University of Stavanger

June 2011

Abstract

FlockWatch is part of an ongoing sheep tracking research project. The goal of this project is to design a system for locating sheep or other animals grazing in the mountains by using sensor nodes (Waspmote), equipped with GPS, GPRS and long range radio transmitters.

The specific aim of this thesis is to provide a control and monitoring system in the form of a desktop application capable of showing at any time the position of the nodes (sheep) on a map, able to facilitate the configuration of the network and play historical data in a chronological order by interacting with a server where the database is stored.

Additionally, FlockWatch is intended for use in the field when collecting the animals in an offline mode, together with a Waspmote node connected to the host portable computer and communicating with the other nodes on the sheep. A supplementary effort was invested into coding a sensor node that acts as a listener, intercepts the radio traffic around the flock and serves FlockWatch with data necessary for a real-time visualization.

I express my sincere gratitude to Erlend Tøssebro and Ragnar Stølsmark for their continuous support and guidance. Our periodical meetings were very helpful in succeeding to deliver successfully this master's thesis.

Contents

Glossary

- AES - Advanced Encryption Standard
- API - Application Programming Interface
- ASP - Active Server Pages
- CDT - C/C++ Development Toolkit
- DBMS - DataBase Management System
- DLL - Dynamic-Link Library
- DTR - Data Terminal Ready
- FTP - File Transfer Protocol
- GPL - General Public License
- GPRS - General Packet Radio Service
- GPS - Global Positioning System
- GSM - Global System for Mobile Communications
- GUI - Graphical User Interface
- HART - Highway Addressable Remote Transducer Protocol
- ID - Identification
- IDE - Integrated Development Environment
- IEEE - Institute of Electrical and Electronics Engineers
- ISM - Industrial, Scientific and Medical
- JTAG - Joint Test Action Group

- LED - Light-Emitting Diode
- LINQ - Language INtegrated Query
- LR-WPAN - Low-Rate Wireless Personal Area Network
- MAC - Media Access Control
- MIT - Massachusetts Institute of Technology
- OOP - Object-Oriented Programming
- ORM - Object-Relational Mapping
- OS - Operating System
- P2P - Peer-to-Peer
- PC - Personal Computer
- PCB - Printed Circuit Board
- PDF - Portable Document Format
- PHY - Physical Layer
- POP3 - Post Office Protocol 3
- RDBMS - Relational DataBase Management System
- ROM - Read-Only Memory
- RSSI - Received Signal Strength Indication
- SMS - Short Message Service
- SMTP - Simple Mail Transfer Protocol
- SQL - Structured Query Language
- SVG - Scalable Vector Graphics

- TDMA - Time Division Multiple Access
- UART - Universal Asynchronous Receiver/Transmitter
- UI - User Interface
- USB - Universal Serial Bus
- VB6 - Visual Basic 6
- WCF - Windows Communication Foundation
- WF - Windows Workflow Foundation
- WPF - Windows Presentation Foundation
- XAML - Extensible Application Markup Language
- XML - Extensible Markup Language

List of Figures

List of Tables

Chapter 1

Introduction

1.1 General Considerations

Even from ancient times, humans felt the need to know the location of themselves, other people, domestic animals or perhaps other points of interest. Spies were sent to inform about the enemy's moves and positions, while a compass was used to determine the north cardinal point in order to help sailors to head in the right direction. Other techniques to find out the geographical coordinates involved the current location of the sun or during night the position of stars and planets on the sky and the usage of precise clocks to determine the latitude and the longitude. All these strategies combined most of the time with the existence of a map of the surrounding territory gave a quite accurate image on the positioning and navigation.

In an attempt to overcome the difficulties risen by the existing procedures to compute the exact location especially for navigation and guidance, United States funded several programs which eventually led in February 1978 to the first experimental Block-I GPS satellite being launched into space and so the development of modern-day GPS systems began[5]. An incident resulting in a commercial plane drifting away from the main course being shot down by a soviet aircraft determined US' decision to make the GPS system accessible for civilian purposes as soon as it would have been inaugurated. This goal was achieved in late 1993 and the system could provide the sufficient data for a receiver to calculate its location with an acceptable error margin

making use of 24 fully operational satellites.

Once GPS proved itself as a reliable and efficient method to obtain the position world wide and favorized by the release of Google-like maps, a large increase in the number of GPS based applications was noticed. More and more devices started to incorporate a GPS receiver, transforming operations like navigation, tracking, monitoring or guidance easy tasks compared to the previous solutions. Nowadays, one can check her location by a touch of screen on a smart phone, a driver can reach the final destination with ease without relying on the road signs but by only using a navigator, a stolen car can be monitored and a simple command can be sent via SMS to stop the engine, wild animals can be observed by attaching to them tiny receptors that give precious information about migration habits or the remaining specimens from an endangered species.

The area of examples of the GPS usage becomes wider and wider everyday with new ideas implemented and the declared goal of making easier tasks that required a higher effort previously.

1.2 The Idea

Being a country with only 5% farmland of the entire surface, Norway had to adapt and provide other means of food production. The predominant mountainous topography and the wet, decently warm climate due to the Gulf Stream make this country a suitable place for grass and similar herbs. This green abundance was a good reason for humans to grow herbivorous domestic animals in this part of the planet. The husbandry animals, sheep, goats, cattle etc. were introduced into Norway more than 5000 years ago as settlement became more permanent[1].

Nowadays, cows are mainly used for milk production, so it is important to have them in the vicinity of the processing facility, their nutrition being made on area-limited and fertilized pastures. On the other hand, the agricultural practice of growing sheep is focused more on the meat production and this particular characteristic gives the farmers more freedom on the techniques used to feed and take care of the animals. Therefore, the farmers together with their sheep had to follow an annual cycle, adapted

to the local Norwegian restrictions. A common scenario would include an initial phase with new lambs being born in the late winter or beginning of the spring, the same time the mature sheep are kept and fed inside a sheepfold. As the spring progresses, the animals might be allowed to graze on a fenced land for some time, but eventually they will be left free on open space, unsupervised for 90 to 100 days until the fall comes. During this time, the sheep eat almost half of the necessary food for one year and the lambs go through their main growth. The open range period finishes with the sheep being collected by the farmers and a short time of feeding them again behind the fences. As the winter approaches, the animals are moved indoors and the cycle repeats.

A delicate part in this cycle is the period when the sheep are moving freely and are exposed to several threats. The sheep holders can not know or control a situation when one of the animals is attacked by a predator or fells into a ravine. Also, gathering the sheep may be a difficult task as they are usually moving away from the original point where they were left in the spring and the farmers must base their search on intuition or experiences from previous years.

This project is part of an ongoing sheep tracking research project. The goal of the sheep tracking project is to design a system for locating sheep or other animals grazing in the mountains by using sensor nodes (Waspmote), equipped with GPS, GPRS and long range radio transmitters. The specific aim of this thesis is to provide a control and monitoring system in the form of a desktop application (FlockWatch) capable of showing at any time the position of the nodes (sheep) on a map, facilitating the configuration of the network, playing historical data in a chronological order by interacting with a server where the database is stored. The same application is also intended for use in the field when collecting the animals in an offline mode, together with a Waspmote node connected to a host portable computer and communicating with the other nodes on the sheep.

1.3 Overview of the Thesis

The rest of the thesis is organised as follows:

The first two chapters are dedicated to the technologies involved in the development process. Chapter 2 describes a stratified structure of FlockWatch on three levels. The first tier, Presentation, is built with the WPF and the Microsoft's Ribbon bricks and is sustained by the second layer, Logic that coordinates the application based on the .NET framework (C# & LINQ) and a couple of external libraries. At the bottom of the pyramid lies the Data level with the two relational database systems (MySQL & SQLite) and the ORM solution coupling with the upper tier. The technology is also the center of the next chapter (3), but this time focusing on the sensor node platform, Waspmote. The topics of this section are referring to the Waspmote modules (ZigBee, GPRS, GPS) and the ordinary programming routine.

In a similar approach, the next two chapters exhibit the implementation details. During the 4th chapter the reader steps into the implementation of FlockWatch, the desktop application developed here. First, there are some explanations about some operations that had to be performed prior to the start of the actual development. Then, the chapter continues listing the most important aspects considered along the project evolution with sections analysing each main part of the user interface together with the logic behind. The following chapter, 5, jumps into the C code of the programs written for the Waspmotes. The initial configuration, RSSI locator and sniffer are described here, aiming to give an overview of the key parts of the implementation.

Naturally, the implementation sections are followed by the tests and results, chapter number 6. Each of the sections here are divided in two separated fragments, one for FlockWatch and one for the Waspmote solutions. The last chapter, counted as 7, concludes the study and forecasts the future release by giving some possible ideas. The user manual for the main application is also inserted at the end of the current report as an independent and distinct component.

Chapter 2

Technologies I - FlockWatch

FlockWatch is a desktop application targeting the newer operating systems supported by Microsoft, namely Windows XP, Windows Vista or Windows 7. The standard prototype of the final user of this application (the farmer) is using a computer running most likely a Windows OS. Also, the decision of adopting the Microsoft technologies for this project was justified by the enhanced expertise of the developer in this field, making them to prevail over other alternatives as Java or Web-like solutions.

A 3-tier architecture will be used as support for presenting the technologies involved in the development of this desktop application, named FlockWatch. A possible definition of this software paradigm might be: ”the typical 3-tier approach dictates that you have a user interface, or display, service, a business rule service, and a database service”[12]. Even though this model is probably too limited for most of the large enterprise implementations, it seems quite sufficient for this special case.

Indeed, the user interface is built mainly with the bricks provided by the WPF (Windows Presentation Foundation), together with the specific functionality of a Ribbon component; then, behind the presentation layer stands the business logic which helped by the C# language in general and by the GMap.NET and Timeline libraries in particular, makes the things happen. The last but not the least level in this hierarchy is the storage where the data reside, ready to be employed by the logic service.

The next figure adapted from Wikipedia (Figure 2.1)[1] shows in a more expressive manner the architecture.

[1] http://en.wikipedia.org/wiki/Multitier_architecture

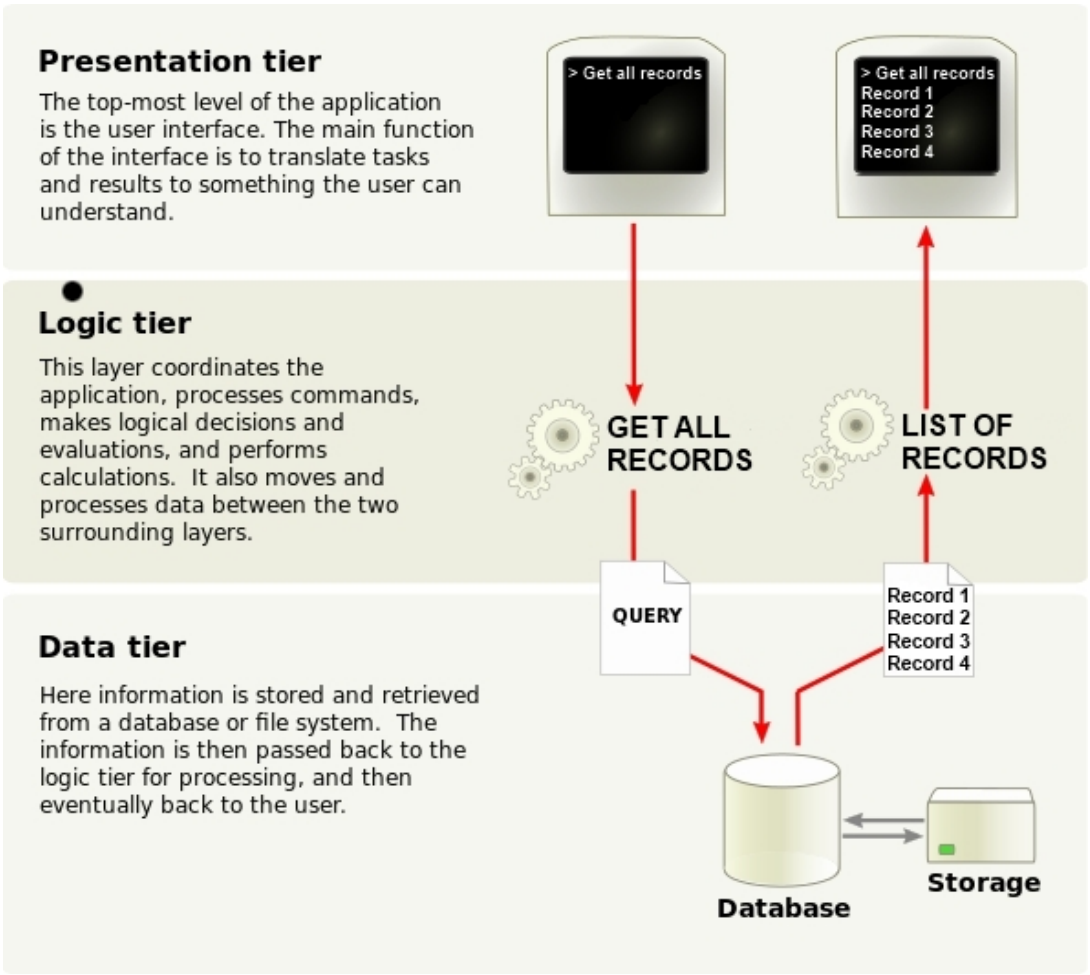

Figure 2.1: A 3-tier architecture [Source Wikipedia]

2.1 Presentation Tier

The presentation tier is the welcoming layer of the application the user is in direct contact with. It must be intuitive and easy to be handled by all categories of users, experienced or not.

2.1.1 WPF

Windows Presentation Foundation or in short WPF is a relatively new technology by Microsoft created to facilitate building graphical interfaces. This fresh approach to construct the first layer of the interaction with users was released in 2006, as part of .NET 3.0, a version of the framework which represented more a package of new technologies as WPF, WCF or WF, than an improvement of the 2.0 version.

Perhaps for the final user the difference between an application developed with WPF and one implemented using Windows Form might not be so obvious as both can consist of plain forms, document-centric windows, animated cartoons, videos, immersive 3D environments, or all of the above[10], but the programmers coming into contact with WPF for the first time may need a longer time to adapt to the new environment and fully make use of the WPF tricks, as the general impression is of something not seen before and the learning curve considered to be steeped.

The way a user interface is defined in WPF is totally different than in the past by using a subset of XML called XAML (Extensible Application Markup Language). This approach means a distinct programming technique, declarative programming, where the logic of the program is depicted without showing the control flow. This is in contrast with the imperative programming most developers are familiar with. The new form of building GUI is very expressive, and even if in the beginning it might not look straight-forward, eventually it will save a good amount of time which would have been necessary for having the same result using a programming language like C#. It must be said that the same interface can be constructed fully in C#, but most of the times, a combination between XAML (for arranging the elements of the interface) and a .NET programming language (for adding logic to the interface) is preferred.

In FlockWatch, the presence of WPF can be felt in all the graphical components (main window, pop-up windows, ribbon menu, labels, text fields, buttons, etc) of the application, even in those originating from the used third party libraries.

2.1.2 Microsoft Ribbon

Microsoft surprised in 2007 with the release of a new version of the Microsoft Office suite by replacing the previous conventional menu with a completely changed and innovative layout.(Figure 2.2)[2]

Figure 2.2: Ribbon in Microsoft Office Word 2007 [Source MSDN Library]

The definition of the Ribbon given by Microsoft is "the modern way to help users find, understand, and use commands efficiently and directly - with a minimum number of clicks, with less need to resort to trial-and-error, and without having to refer to Help."[3]

The Ribbon is meant to be a substitution for the existing traditional menu and toolbars, bringing the user in a closer contact with the application once the basic commands and functions are learnt. The new user interface element organizes the numerous commands commonly found in a complex application in logical connected groups which corresponds to tabs in the visual formation.

The same interface entity has been made available to the Windows application developers in form of a component easily integrable in any .NET application by only reading and accepting an Office UI License. Soon enough the general opinion was that what Microsoft provided is insufficient for many applications, lacking many important features. This situation has resulted in the development and release of several commercial products capable of filling the gaps observed in the Microsoft

[2] http://msdn.microsoft.com/en-us/library/aa338198.aspx
[3] http://msdn.microsoft.com/en-us/library/cc872782.aspx

component. This triggered Microsoft's reaction and in October 2010 a new version solving most of the issues was made available for download.

The same release is employed in FlockWatch for a smoother and faster user experience. This component is placed on the top part of the main window and gives access to the most used and helpful commands for the operator during a regular usage of the application.

2.2 Logic Tier

The logic tier is the brain of the application, changing and updating the visual interface according to the user's actions and making sure that data is retrieved from and stored to the data tier correctly. Also this layer is in charge with processing all the inputs from the other tiers and output relevant answers. Being the middle tier, it is important to be efficient and assure that the round-trip between the presentation and the data storage happens as fast as possible, to have an insensible delay from the user action until the expected result is delivered.

2.2.1 .NET Framework

.NET Framework was designed to be an API for programming under Windows environment. It supports a high number of programming languages[4] and facilitates the interoperability among them.

C#: C# is a language written from concept to finished product, especially designed for the new .NET framework. Despite the fact that other languages existed the time .NET was launched, like C++ or VisualBasic, Microsoft opted to supply the new released API with a language capable of making the most of the present technology without concern of backward compatibility.

Officially, Microsoft describes C# as a "simple, modern, object-oriented, and type-safe programming language derived from C and C++". Most independent observers

[4] http://www.dotnetpowered.com/languages.aspx

would probably change that to “derived from C, C++, and Java“.[9] But even if at the time of its creation, C# was highly influenced by the revolutionary object-orientated language, Java, in the years to come C# evolved independently in a unique language, and in many aspects surpassing its competitors.

The C# language is disarmingly simple, with only about 80 keywords and a dozen built-in datatypes, but C# is highly expressive when it comes to implementing modern programming concepts. C# includes all the support for structured, component-based, object-oriented programming that one expects of a modern language.[6]

C# is employed everywhere in the code and is the programming language behind the visual interface of FlockWatch. Even the WPF controllers have a portion written entirely in C# where usually their logic is constructed. Through the language’s ease of expressing basic building blocks, tasks that would have required a significant amount of time and large volume of code with other programming languages were implemented faster and effortless with C#.

LINQ: LINQ (Language INtegrated Query) is an extension of the existing .NET programming languages (including C#), enabling programmers to write query operations in an SQL style on all sorts of data collections residing in memory. Few example of such data sources are arrays, lists, hash tables, enumerations, XML content, DataSets or relational databases. LINQ can be extended to support other third party data storages too.

LINQ gained popularity among .NET developers by providing a uniform way to interface data, regardless of its internal structure and organization. This translates in using the same query primitives for multiple data sources, avoiding the past situations when several languages, technologies or APIs had to be involved in basic operations related to data.

At the same time, LINQ is part of the .NET Framework and features like code completion or compile-time checking of the code make life easier for coders, saving hours of debugging if for instance, there is a misspelled SQL query placed in the source code as a string. Also, the code with LINQ is more condensed and more to the point, shrinking the program size in terms of lines of code and therefore narrowing the

chances of hidden bugs.

LINQ removes many of the barriers among objects, databases and XML. It helps the developers to work with each of these paradigms using the same language-integrated facilities. For example, it is possible to deal with XML data and data originating from a relational database within the same query.[7]

In FlockWatch, LINQ can be found in most of the places where operations with database (MySQL or SQLite) are performed, but also in normal situation when accessing data stored in collections in the program's memory.

2.2.2 Third Party Components

Developing a brand new application is not an easy task and in the particular case of starting from scratch, the mission can become quite difficult. Reinventing the wheel might sound reasonable in some cases, but in most of them it is a bad idea, triggering supplementary work, not always the best solution and probably delays in delivering the final product. Therefore it is advisable to include in the application commercial or open source components able to satisfy a specific requirement or to help the general development and allow programmers to focus on the really important parts of the project.

In the particular case of FlockWatch, only open source libraries were added. The external code integrated into the main application contains both WPF and C# elements, but they were considered parts of the logic tier as the functionality provided by them prevails over the graphical appearance.

GMap.NET: GMap.NET (Great Maps for Windows Forms & Presentation) is a powerful, free, cross platform, open source .NET control. Enables use routing, geocoding and maps from Google, Yahoo!, Bing, OpenStreetMap, ArcGIS, etc., supports caching and runs on windows mobile.[5]

The control has integrated by default a large number of map sources, but any other web mapping service can be added into the free accessible code and then selected to

[5] http://greatmaps.codeplex.com/

be the main map type. The GMap.NET provides features that make the navigation throughout the map (zooming, moving, selecting) direct and natural with both mouse actions and button presses on the visual interface.

It was mentioned that this project is open source, but more specifically it is released under the MIT license which basically states that: "Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files, to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software"[6].

In FlockWatch, GMap.NET was the starting point of the sheep monitoring system. Great Maps provides support for the operations involving the maps and all the other specific functionality is built on top of it. A local map source, statkart has been added to the existing provider list, especially for this project.

Silverlight & WPF Timeline Control: This timeline control[7] is a Silverlight and WPF component, useful in situations where an application emphasizes the progress over time of a specific, interesting feature. The control permits interactive interaction from the user and can be personalized with different styles and templates to give a look-and-feel similar to the hosting application. It also exposes many properties and events to the developers for a better fine-tuning and customization. The code is released under the BSD licence which in short means that any product incorporating the original control, even the commercial ones, can do it without paying any charge or changing the existing license.

In FlockWatch, this component can be found under the Playback tab, located at the bottom of the map and has an important visual role in replaying historical data.

[6] http://en.wikipedia.org/wiki/MIT_License#License_terms
[7] http://timeline.codeplex.com/

2.3 Data Tier

Most of the modern applications with a certain degree of complexity need and utilize a persistence mechanism in order to feed the inside logic or preserve information from one session to the next one. Over the years there were several solutions to achieve this goal and even if today there are places where file storage (plain text, binary, XML) is still used, relation databases represents and probably will represent the main solution to store data for a good period of time.

In the three-tier architecture, the data is accessed and extracted by the logic tier and then manipulated to a suitable form for the presentation layer. And the same reasoning is also valid for the inverse flow direction, presentation - logic - data.

2.3.1 Databases

Databases are dedicated data storage collections where information is sorted by certain criteria in order to interact with them in a logical manner. Habitually, operations with a database like creation, maintenance, use are managed by a software package called database management system (DBMS).

MySQL: MySQL is a client/server RDBMS (relational database management system) that includes an SQL server, client programs for accessing the server, administrative tools, and a programming interface for writing your own programs.[2] MySQL origins are found back in 1979 and since then it has travelled a long and winding path to reach the maturity and reputation gained as for today.

MySQL distinguishes itself through a series of features that recommends it for most of the cases where a database system is necessary. MySQL is fast, easy to configure and administrate, supports the SQL standard, has high capacity of concurrently served clients, provides mechanisms for secure connectivity, is portable and can run on the main operating systems, and in most of the situations the costs for it tend to be zero.

MySQL lies still under the Open Source umbrella (GNU General Public License - GPL), even though commercial solutions are also available on the market. Thus it is

distributed without restrictions and can run freely even on a personal computer. There are no supplementary costs except those related to hardware. Having the support of the open source community was a real aid for the project as it developed extensively and became a real competitor for commercial products.

The reason that stood behind the decision of going for MySQL as the main storage solution of the entire sheep project and not just for FlockWatch was that today this relational database engine is widely spread, easy to manage and access and most of the web hosting services have in their offer database support for MySQL in exchange for a small fee.

SQLite: SQLite is a zero-configuration, standalone, relational database engine that is designed to be embedded directly into an application.[4] This small and "lite" (lightweight) engine is the perfect choice for applications that do not need to share their data with other parties via a central server, but still use the advantages of a relational database.

A single file contains both the database engine and all the tables and indexes. Having everything in one place on the local disk reduces significantly the overhead of having a central server in a remote location. Also the one file strategy simplify operations as "create", "duplicate", "backup" which tend to be pretty complex for traditional RDBMSs.

Even though SQLite lacks the central server feature, it has a wide range of applicability such as "baby" databases, application configuration and preference settings, application cache, data archives, and even as teaching tool due to its uncomplicatedness. In terms of license, SQLite has been placed in the public domain by its maintainers, meaning that the code and the resulting libraries can be used in any way, including selling.

For FlockWatch, SQLite represents a convenient way to build a cache system and avoid communicating intensively with the central server. In fact, the local SQLite database is just a mirror of the MySQL one, synchronized occasionally to keep consistency. The decision of having an intermediate level when accessing the data was dictated by the fact that most of the database operations performed by FlockWatch are readonly and the update frequency of the data on the server is reasonably low.

2.3.2 Object Relational Mapper (ORM)

Object Relational Mapper (ORM) is a framework designed to unify two heterogeneous notions, a table in a database and an object as an instantiation of a class in an object-oriented programming language. Simply put, an ORM gives the illusion of saving and retrieving model objects from a relational database. How the mapper manages to implement the connection between the two worlds is specific to each one and might differ from one ORM to another.

Introduction: Relational databases have tables to maintain data, views to logically organize them so that they are easier to consume, stored procedure to abstract the application from database structure and improve performance, foreign keys to relate records in different tables[8], while in an OOP language, objects are the natural way to represent data and mechanisms as inheritance, polymorphism or encapsulation were decisive in making this technique successful among developers.

But these two universes, relational and object-orientated are structurally and conceptually different, and trying to translate and communicate between them in a transparent manner is a difficult task. This problem is overcome by analyzing each of the primary differences and finding the best way to build a bridge between the initial incompatible sides of the issue. The solution takes most of the time the shape of an ORM.

Accessing Data: A database is usually made up by several tables and each table is structured as a list of rows, with each row composed of fields, known as columns. This tabular manner of keeping the information influenced the ways data is retrieved from the tables.

In the pre-.NET era, ASP and VB6 users could use a Recordset object for any interaction with a database. This object, as the name says, holds a set of records from a table and due to the fact that it tries to be as close as possible to the original source, the database table, a Recordset organizes the extracted data similarly, as rows and columns.

In the context of the .NET introduction, combined with a series of disadvantages

like the strong bind to the database, the old Recordset object was replaced by a suite of new objects capable to sustain a productive exchange with databases. The new API in the form of datasets, datatables, datareaders and dataadapters was everything a developer needed in order to get from or add information to a database. And even if apparently few lines of code were sufficient to achieve most of the tasks related to databases, very soon important obstacles were encountered in the development process as:

- strong coupling - a small change in the database structure might determine a heavy restructuring of the existing application code
- loose typing - retrieving and storing data imply casts and conversions which have an important impact on the overall performance
- general performance issues - the used classes have complex implementation that can affect negatively multi-user applications

Object Relational Impedance Mismatch: The object relational impedance mismatch is a set of conceptual and technical difficulties that are often encountered when a relational database management system (RDBMS) is being used by a program written in an object-oriented programming language or style; particularly when objects or class definitions are mapped in a straightforward way to database tables or relational schemata[8].

The following items are some of the most important aspects of the Object/Relational mismatch:

- the datatype mismatch - the datatype of a column in a table rarely coincides exactly with a basic datatype specified by the programming language utilized in the object world. Also, it is extremely difficult to explicitly set in the code a length restriction to a value in the database.
- the association mismatch - this mismatch refers to the association among classes

[8] http://en.wikipedia.org/wiki/Object-relational_impedance_mismatch

in the source code when the relational database uses foreign keys to mark connections among tables. By cardinality, these relationships are divided into:

  - one-to-one relationship - adding a new field to a table might be a real hassle from the application's point of view in case of using an object representation; this can be avoided by creating a new table with the single field and the same primary key as the considered key; even if this may seem a minor change, for the object model it is superfluous.
  - one-to-many relationship - this relation is expressed in a relational database through foreign keys and these associations between tables are silently unique and bidirectional due to SQL internal mechanisms. On the other hand, in an object-oriented environment, these bindings have to be explicitly formulated.
  - many-to-many relationship - the many-to-many relationship represents an association where each of the endpoints has a multiple relationship with the other one. This means that there is not a master-detail relationship between tables, but both of them are at the same level.[8] Translating this relationship into the object world results in an apparently absurd construction.

- the granularity mismatch - means that sometimes mapping a given number of tables to classes might create either more or either less classes.
- the inheritance mismatch - this mismatch originates from the special characteristics of the OOP where is a common practice to extend a base class via the inheritance feature; but this peculiar behaviour is almost impossible to be imitated in a database.
- the identity mismatch - databases and objects perceive the term equality differently: databases rely on the primary key of the records when making the comparisons, whereas objects use references.

Handling the object relational impedance mismatch is a difficult job and a system capable of doing it tends to become complex in terms of written code and implicitly

spent time. Here is where an ORM (as an out of the box solution) comes into play as a middle layer between the two worlds, transforming accordingly the traffic between those two and hiding the specific features of data layers from the logic and presentation.

Adopted Solution - Telerik's OpenAccess ORM: Telerik's OpenAccess ORM is an industrial strength ORM ready to bridge the gap of the object relational impedance mismatch as best as possible. It is a .NET mapper that provides tight integration into Visual Studio and through simple wizards, forward and reverse mapping to many major database engines can be performed.[11]

OpenAccess has a decent support for LINQ queries and even if during development some of them could not be run on the server, eventually the available ones were sufficient for the necessary database interrogations. The LINQ support is specific to OpenAccess, not to the backend database.

This product comes in two flavours, a commercial and a cost free edition. The commercial solution supports an extensive collection of databases, while the free product offers only a subset of the complete list. But fortunately both MySQL and SQLite are included in the free list, hence the free edition was sufficient for the current project's needs.

OpenAccess was the first library to be tested as an ORM for FlockWatch. It proved to be an excellent solution for both MySQL and SQLite databases, therefore an alternative was not tried further. Probably similar ORM frameworks as NHibernate and ADO.NET Entity Framework would have been capable of analogous accomplishments. The code written for interacting with the databases via Telerik OpenAccess ORM is simple and straightforward, mostly consisting of LINQ queries.

Chapter 3

Technologies II - Waspmote

The research project this application is a part of uses sensor nodes provided by Libelium[1] called Waspmotes. Finding an alternative to Waspmote was outside the scope of the current research due to the fact that this technology was already selected for the whole project when FlockWatch was started.

Waspmote is a device based on the Atmel 1281 microprocessor and a very precise internal clock with a maximum loss of accuracy of 1 minute per year. The latter property is essential for real time applications or applications alternating dormant periods with exactly determined wakefulness states. The mote is designed to have a modular architecture, thus extra parts can be attached to or removed from the main board. The modules available for integration with Waspmote are categorised as:

1. ZigBee/802.15.4 modules (2.4GHz, 868MHz, 900MHz). Low and high power
2. GSM/GPRS Module (Quadband: 850MHz/900MHz/1800MHz/1900MHz)
3. GPS Module
4. Sensor Modules (Sensor boards)
5. Storage Module: SD Memory Card[14]

[1] http://www.libelium.com

From the above list, only the first three modules are used in the current implementation, namely ZigBee (more specifically the 868MHz frequency version for increased range), GPRS and GPS. Different aspects of these modules are detailed in the following sections.
Additionally, a series of other sensor boards can be connected to the main mote (gases, event detection, prototyping, agriculture, smart metering), extending the original applicability area and making it suitable for a larger number of purposes.

3.1 ZigBee (802.15.4) Module

The subgroup 4 of 802.15 had the task to establish the guidelines physical layer and media access control for low-rate wireless personal area networks (LR-WPANs). The aim of this standard is low power consumption with the cost of low data transfer speed and short ranges. This makes 802.15.4 to target embedded devices which rely on scarce resources, therefore capable of not so high performances.

Regarding the numbers, the standard was designed to support an average range of 10 meters and maximum 250 kbps as data rate. The specifications of the standard refers only to the MAC and PHY layers, leaving the superior levels to be implemented in connection to the market trends. The physical layer can operate in 3 different open frequency bands, included in the ISM (Industrial Scientific Medical) bands:

- 868.0-868.6 MHz: only Europe, allows one communication channel (2003, 2006)
- 902-928 MHz: only North America, up to ten channels (2003), extended to thirty (2006)
- 2400-2483.5 MHz: worldwide use, up to sixteen channels (2003, 2006)

This standard formed the base for other specifications like ZigBee, WirelessHART or MiWi.

ZigBee is a commercial implementation of the IEEE 802.15.4 standard, stated by ZigBee Alliance, an association of companies cooperating to develop interoperable

products. The main purpose of ZigBee is to be a very cheap and light technology and is targeted toward applications requiring low data rates, long battery life and secure networking. Few common examples of the ZigBee usage are applications like industrial control, embedded sensing, medical data collection, smoke and intruder warning, building automation, home automation.[2]

The Waspmote equipment is compatible with the Digi XBee modules and a series of such modules are distributed by Libelium for integration in Waspmote. The area of selection is rich so developers can have the opportunity to choose the module that fits best the operating environment of their product.[14]

Model	Protocol	Frequency	txPower	Sensitivity	Range
XBee-802.15.4	802.15.4	2.4GHz	1mW	-92dB	500m
XBee-802.15.4-Pro	802.15.4	2.4GHz	100mW	-100dB	7000m
XBee-ZB	ZigBee-Pro	2.4GHz	2mW	-96dB	500m
XBee-ZB-Pro	ZigBee-Pro	2.4GHz	50mW	-102dB	7000m
XBee-868	RF	868MHz	315mW	-112dB	12km
XBee-900	RF	900MHz	50mW	-100dB	10km
XBee-XSC	RF	900MHz	100mW	-106dB	12km

Table 3.1: The XBee modules distributed by Libelium for integration in Waspmote

The XBee-868 was elected as solution for the considered project bearing in mind that the modules are intended to be used in Norway (Europe) and it is preferable to have the best possible transmission range as it is impossible to keep the animals in a limited area. The module uses a single channel, can assure an AES 128b encryption of the data and the most common type of topology realised with XBee-868s is P2P, with each node making contact with neighbours through parameters like MAC or network addresses.

This radio module is used in the project as the only way for nodes lacking the GPRS module to log their position via other nodes capable of data transfer. Also, it becomes important in the collection phase when the central server is bypassed and all

[2] http://en.wikipedia.org/wiki/ZigBee

the Waspmotes transmit their coordinates directly to a special mote called sniffer and connected to a portable machine capable of showing live the locations on the map.

3.2 GPRS Module

General Packet Radio Service (GPRS) is a standard built on top of Global System for Mobile Communications (GSM) meant to provide superior data transfer without making important modifications to the existing infrastructure. It was considered a fundamental step forward, toward the next generation of cellular systems, 3G (third generation), and for this reason, it is sometimes referred as 2.5G.

This data overlay network provides packet data transport at rates from 9.6 to 171 kbps. Additionally, multiple users can share the same air-interface resources simultaneously[3]. GPRS is using the time-division multiple access (TDMA) channels of the GSM system for superior data rates.

The relatively decent data transfer rate supported by GPRS makes the technology adequate for several uses as e-mail, web browsing, instant messaging, multimedia messaging, telemetry, broadcast services and in general applications that are not greedy when it comes to bandwidth.

The low demanding characteristics of this technology recommend it not only for mobile phones but also for other embedded systems with several types of constraints. This is also the case for a Waspmote which provides a special socket for a GSM/GPRS module, namely the HiLo model from Sagem. It supports all 4 main GSM frequency ranges, being usable in both Europe and America. The module is capable of performing most of the functions a normal cellular phone can do and even more in some respects:

- Making/Receiving calls
- Making ‘x‘ tone missed calls
- Sending/Receiving SMS
- SMTP Service (sending emails)

- POP3 Service (receiving emails)
- FTP Service (downloading and uploading files)[14]

Each mote placed on the sheep utilizes the GPRS module solely for sending the current position determined via GPS to the central server for storing and later use. Obviously, this task can be completed only if the sheep is situated in a region with coverage from the chosen mobile network operator.

3.3 GPS Module

Global Positioning System (GPS) was presented briefly in the introductory chapter as an US project started for military purposes which eventually led to a public service, accessible freely by everyone. The system consists of a constellation of 27 satellites (3 as backup) orbiting circularly the Earth at an altitude of around 20.000 km and distributed uniformly so that always and everywhere on the planet at least 4 satellites are seen by a receiver.

The GPS receiver applies a technique called trilateration in order to determine its current position (in the form of latitude, longitude, altitude) based on the readings it gets from the visible satellites. Starting with this primitive numbers, smart devices having a GPS receiver can derive secondary measurements (ie. speed, bearing, etc.) useful in specific situations. Depending on the available satellites and the receiving device capabilities, an accuracy down to 10 meters can be achieved, good enough for the majority of application.

In the considered project, intuitively, the GPS module serves to provide information regarding the estimated location of the node installed on a certain sheep when moving in an area covered by the satellites.

The GPS receiver, model Vincotech A1084 needs a certain time to obtain and structure the information that the satellites send. This time can be reduced if there is certain prior information. This information is stored in the almanacs and ephemerids. The information that can be found out is relative to the current position of the satellites (ephemerids) and the trajectory they are going to follow over the next days (almanacs).

The almanacs indicate the trajectory that the satellites are going to follow during the next days, having a validity of some 2-3 months. The ephemerids indicate the current position of the satellites and have a validity of some 3-5 hours.[14] The module can be attached to the existing Waspmote and even in the worse conditions can offer good time performances for data availability:

- Hot Start time (after the time and date are determined and having ephemerids and valid almanacs already in the memory) - under a second
- Warm Start Time (after the time and date are determined and having valid almanacs already in the memory) - under 32 seconds
- Cold Start Time (no time, date, ephemerids nor almanacs) - under 35 seconds

The information delivered by the GPS module as Waspmote API‘s primitives is not limited only to the geographical coordinates (latitude, longitude, altitude), but additional useful values are given, such as speed, direction, (precise) date/time or ephemerids.

3.4 Programming Waspmote

Waspmote is based on the larger open source, cross-operating systems project, Arduino (an electronics prototyping platform based on flexible, easy-to-use hardware and software[3]). The code for Waspmote is written in the Arduino language, a derivative of C/C++. The language is linked against the AVR Libc library and allows the use of any of its functions.

Libelium provides an IDE (integrated development environment) for development with Waspmote, based on the Arduino open source IDE and compiler, but it is still in an incipient stage, lacking important features as code colouring and autocompletion. The alternative used for the implementation of the Waspmote applications for the considered project is the Eclipse environment (the CDT Project) configured to work with the AVR toolchain and the Arduino libraries and hardware. Additional API

[3] http://arduino.cc/

libraries are added on top of Arduino, each hardware modules exposed in the above sections having a corresponding software library.

Any Waspmote programs has at least two parts: the setup function invoked first after the board reset and an infinite cycle running the loop method that contains the main logic. The rest of the program may be populated with any other lines of code necessary to complete the desired functionality. After compiling and linking the original source, an .elf binary file is outputted, ready to be written on the flash memory. This file can not exceed 128Kb, the maximum size of the flash chip equipping the Waspmote board.

Chapter 4

Implementation I - FlockWatch

As mentioned before, FlockWatch is an application targeting the Windows platforms and will run strictly on operating systems released by Microsoft that have .NET Framework version 3.5 or newer installed. A general screenshot of the application can be seen in the figure number 4.1.

The chosen approach to detail how FlockWatch has been implemented is similar to the one adopted in the user manual attached to the last part of this report. There, each significant section of the application is presented as for the final user, focusing on the different features and functions. In this chapter, the technical aspects of each section of the application are discussed.

4.1 Prerequisites

The entire application was built exclusively in the integrated development environment provided by Microsoft, Visual Studio, 2010 version. After a relatively long installation time, Visual Studio was ready to be employed in developing a wide range of application types like console, web, desktop, etc.

One of the first initiatives after FlockWatch was officially started was to define the database structure of the future application. MySQL tables were added to the database and then through the Telerik OpenAccess ORM wizard, equivalent C# classes were generated. Later, the same structure was the starting point for producing the SQLite

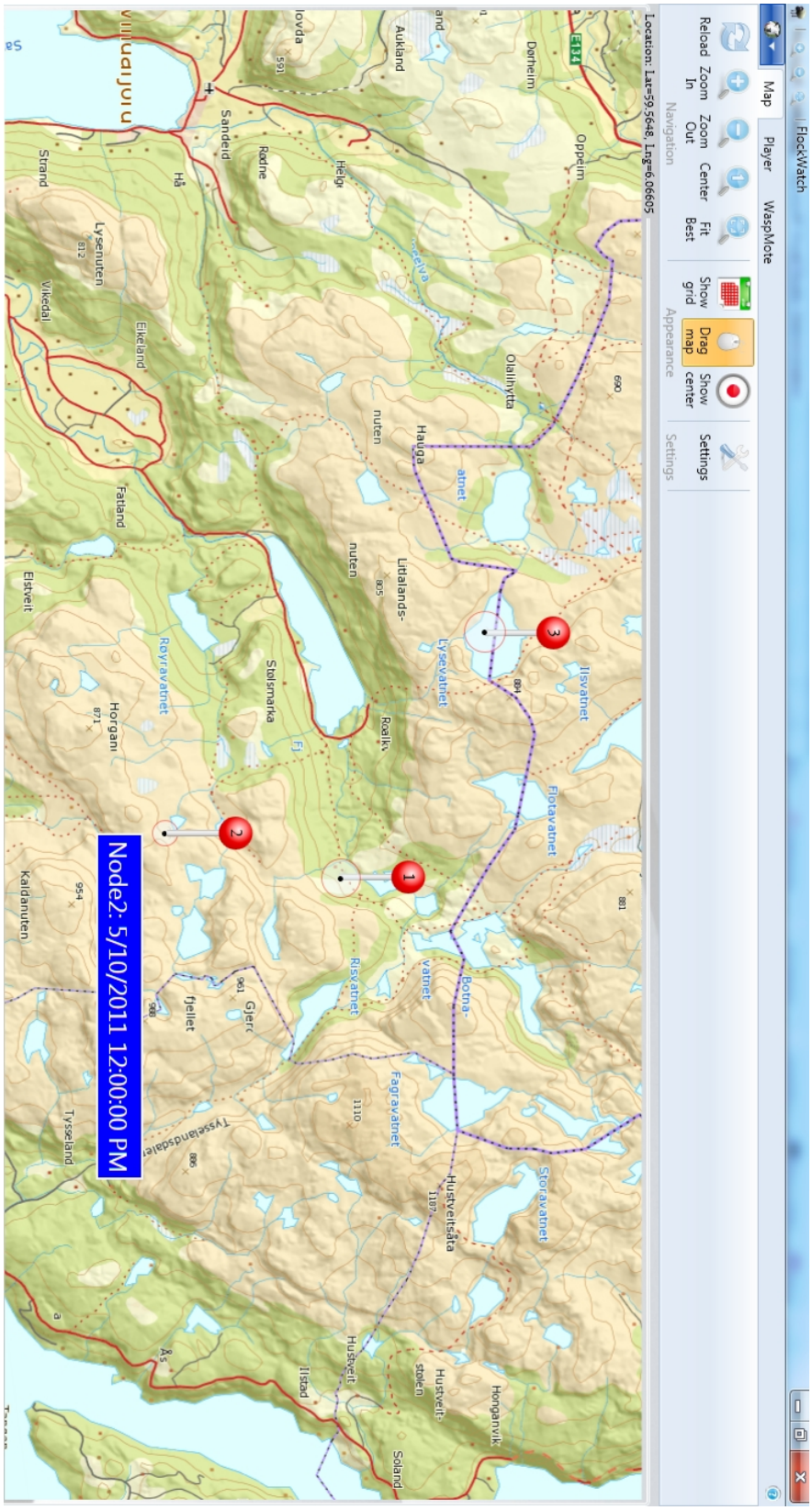

Figure 4.1: FlockWatch's main window

database. Additionally, specific drivers had to be installed and DLLs files added to the project references in order to make the ORM solution to work with MySQL and SQLite. Currently, the database comprises just two tables, Node and Record. The node symbolizes a Waspmote carried by a sheep and the record represents a single position value retrieved from a node at a certain point in time. The rendered diagram of the tables as displayed in Visual Studio is depicted in the image 4.2:

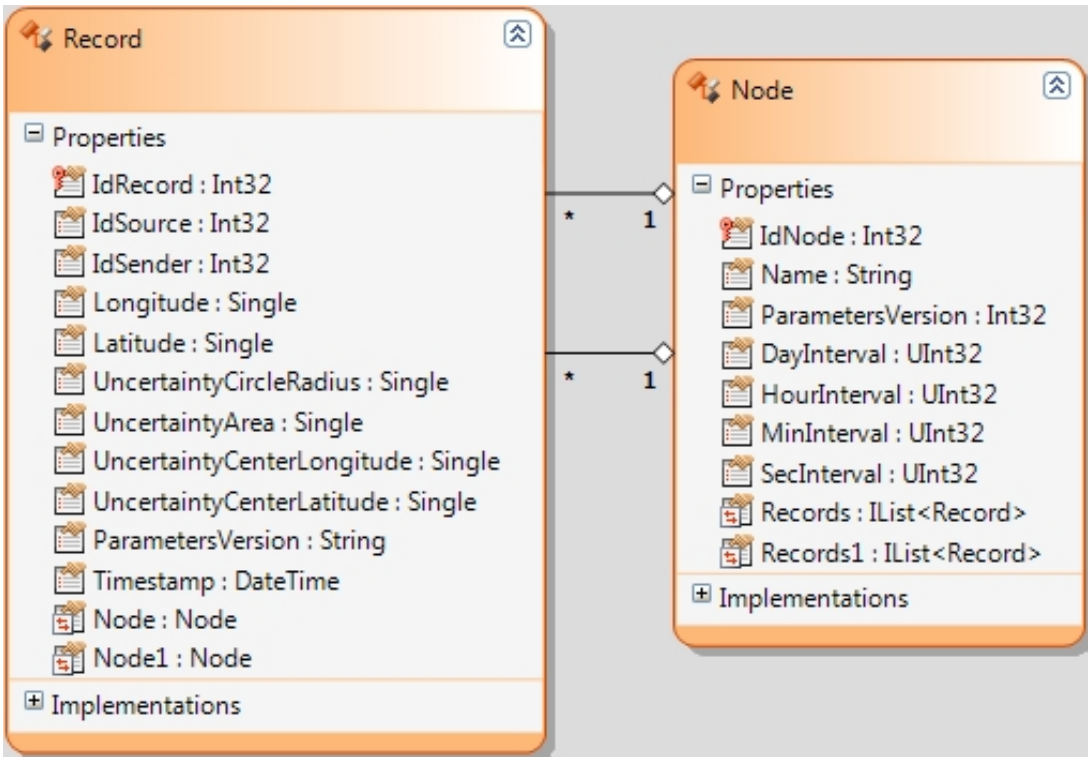

Figure 4.2: FlockWatch's database tables

Also, in these initial stages of the project, a research procedure was carried out to investigate possible third party solutions which could have facilitated the development. The best candidate for displaying a map, the GMap.NET component, needed an even deeper analysis in order to be integrated in the FlockWatch aggregate. A good amount of effort was invested in understanding both the code and the provided features and deciding which of them can be useful in the implementation, more precisely in the abilities of showing the map.

Another preliminary step performed later along the development timeline was to

beautify the Ribbon controller with suggestive icons. These images were carefully picked from a large open source collection, Open Icon Library. The main aim was on those in SVG format (Scalable Vector Graphics), easy to be processed with an external helper application and then translated to XAML content. This conversion is possible because the features provided by XAML are a superset of the primitives in SVG. Having the icons constructed with XAML objects is advantageous from several points of view:

- resizing the window does not affect the resolution of the icons
- small changes to the appearance can be done directly in XAML avoiding the use of other tool as an image editor
- shorter loading time as the icons are compiled into the application in a known format as opposed to having them stored in files on the disk encoded in another graphical format

4.2 Initialization

Before the application is actually ready to perform its duties, a series of preliminary tasks are carried through, hidden under the start splash screen. During this time the normal .NET initializations are performed, but also the remote and the local databases are synchronised by transferring nodes both ways in case there are mismatches and by updating the local SQLite table with the newest position records retrieved from the MySQL server. Another part of the warming up process is reading several user settings stored in the application config file and binding them to internal variables. These variables can be modified mainly in the configuration windows of FlockWatch and have an important role in customizing the application. After this initial step, the splash screen is replaced by the main window. The primary window is divided in two distinct parts that are analysed next.

4.3 Map Display

This part of the display is based massively on the GMap.NET component, briefly described in the technologies chapter (2). However, the existing functionality was not simply put into play in an out of the box manner. Changes and adaptations had to be made before the map control was ready for integration within FlockWatch. In fact, the task with the highest priority in the todo list of the current project was to be able to show a topographic map, the preferable type of map when navigating in mountain areas. The map server was chosen to be statkart.no as it provides a wide range of maps with excellent representation of the Norwegian territory.

First, both the way GMap.NET supports different map providers and how statkart gives access to tiles for their maps were investigated. With these two sets of details, the open source code of the map component was modified to make room for the maps supported by the Norwegian website, eventually appended to the already existing list of well-known map sources. The most difficult task was to get one of the statkart maps in place, any of them, as adding the rest of 11 different types was a mere formality, with copy, paste actions and string replacements.

The original control had to suffer even further changes to meet all the requirements of FlockWatch. Another part of the adjustment process was to include a new marker symbolizing the sheep. It was defined as a WPF user control that more than the visual appearance includes a dynamic mechanism to have printed on it a number equal to the node‘s ID. This way, the user can distinguish easily between signs especially in the playback mode where they move continuously on the map.

Also, the system of events present in GMap.NET had to be redesigned in order to permit to the new layer in FlockWatch to have access to the activities below and metamorphose the behaviour accordingly. Events like selecting a map area or zooming operations already implemented in the component were connected to the new added functionality so that the application can be reflected to the user as a whole and not as made of individual modules. In fact, the original implementation behind selecting a map zone has been completely modified. In the previous variant, selecting a rectangular map cut would have resulted in zooming in to that specific zone. For

FlockWatch, this feature was disabled and removed to accomodate another one. in order to do this, the associated event is intercepted in the upper layers and handled there without rerouteing it further down. The way the event is treated currently consists of determining which sheep markers were caught under the selection and building a table with the related nodes settings that are showed inside a new pop window. The user can update the displayed values and can save back to the databases the newer information by pressing the Save button.

4.4 Ribbon Area

The ribbon area is found in the upper part of the main window (4.1). The developer has control only over which elements are placed in certain zones and the logic below the inserted commands. The content is populated by simply defining the menu items in the XAML part of the main window. This will only have a visual effect by placing the buttons on the layout, accordingly to the predefined theme of the Ribbon, but unresponsive when clicking them.

The functionality is defined separately by using specially dedicated objects. These objects implement the ICommand interface which is the exponent of a design pattern known as the command pattern. This pattern encapsulates a request as an object and gives it a known public interface. Command Pattern ensures that every object receives its own commands and provides a decoupling between sender and receiver. A sender is an object that invokes an operation, and a receiver is an object that receives the request and acts on it.[1]

The ICommand interface exposes two methods, CanExecute and Execute that have to be defined in the classes implementing the interface. The implementation of CanExecute dictates if and when the command is enabled while Execute contains the functionality run when the command is invoked.

All the buttons arranged inside the Ribbon sections are declared as RoutedUICommand, objects that implement ICommand. Therefore they are obliged to comply with

[1] http://www.dotnetheaven.com/Uploadfile/cupadhyay/CommandPatternsInCS02012006014005AM/CommandPatternsInCS.aspx

the interface and add a body to both methods. These methods were implemented through lambda expressions to save lines of code and thus have all the commands in one file for easy access.

4.4.1 Application Menu

The application menu is positioned on the top part of the Ribbon panel and usually contains the most common used commands of the application, making them accessible at any moment with a minimum effort. Once again, the application menu layout and appearance are dictated by the internal specification of the Ribbon component.

This area contains a series of elements such as the application title, application icon, frequently used commands, help button, application button. Only the last two from this enumeration required roughly more work to be completed. First, the help command is supported intrinsically by the Ribbon, but the logic behind must still be included. It was also implemented as an ICommand with the CanExecute returning always the true value and the Execute method opening the user manual as PDF. The method checks first for a valid installation of the Adobe Acrobat Reader in the Windows registry tree. If the expected entry is found then the document is open, otherwise the user is advised through a message box to install it. Secondly, the application button; at the moment it hosts only two options, but the list can be extended in the future with new features. Those two possibilities are Export Image and Close. Close button is simply triggering the application to shut down. The other one, Export Image is slightly more complex: its task is to take a snapshot of the map status, encode it to one of the available formats (jpg, gif, png or bmp) and save it to the disk in a new file for later usage.

4.4.2 Map

The Map tab of the Ribbon was designed for the configuration, appearance and navigation of the map display presented in the 4.3 section. The first group of commands (snapshots published in the user manual) are essential in navigation within the map, therefore are also present in the other tabs. Their logic behind relies mostly

on the primitives existing in GMap.NET component. Functions as Reload, Zoom in, Zoom out, Center had already support in the provided code. The Fit Best option had to be implemented separately with the available features in the map controller plus additional functionality. A new method determining the minimum rectangle area covering all the generated markers has been added. Based on its computations and the zooming and translating capabilities embedded in the component, the map is rearranged so all the markers become visible to the user. This way, the user can get automatically a general view of the current status of the flock.

A second set of buttons are just decorations to the way the map is displayed. The on/off commands Show Grid, Drag Map and Show Center are also connected to the existing code in the GMap.NET and using them will automatically have an effect on the map. The WPF commands in the back are just calling member methods on the map controller with the right parameters.

The final group in the Map tab contains only a single choice, Settings. Using it will open a pop-up window containing some configuration details related to the map object. These settings are loaded during the initialization phase and later populate this window, in categories delimited by surrounding boxes. Changing any of them here will not have any effect until the Save button is pressed and implicitly the window closed, so that there is always the option to invalidate a selection. This behaviour is achieved through buffer variables that are not linked to the real settings until the user confirms them. Saving them will also update the settings stored in the config file so next time the application is restarted, the user finds it with the same configuration.

The map zoom levels have an impact on the range a map permits the zooming but also the predefined zoom value the map is set on loading. These zooming parameters were restricted so that the minimum level is always lower that the maximum one and the default level is always in between.

The next settings are simply the geographical coordinates (latitude and longitude) of the point where the farmer can expect to localize the flock. This position is used as parameter to a “goto“ method of the map controller when pressing the Center button on the Ribbon’s navigation group.

The last group of settings of this window refers to the map origin. The first option is a

long list with map sources from which the user can choose the tiles server. The second specifies from where the tiles should be loaded, local cache or the remote server. Once again these two settings correspond to variables belonging to GMap.NET.

4.4.3 Playback

The Playback tab is useful in replaying historical data about the past locations of sheep. It needs the following inputs before it can play: an initial date, a final date and a current date, all settable through calendar pickers placed under the Ribbon area.

The playback functionality is built on the foundation of a specialized .NET object called BackgroundWorker that can execute an operation on a separate thread in a very handy way. But first of all a description of this class is provided, as it is also used in the final section of the application, both in the sniffer and RSSI locator commands. The BackgroundWorker class allows to run an operation on a adjacent, dedicated thread. Time-consuming operations like downloads and database transactions can cause the user interface to seem as though it has stopped responding while they are running. When the expected outcome is a responsive UI in the conditions of long delays associated with such operations, the BackgroundWorker class provides a convenient solution.[2]

From the main thread, the worker can be started in two modes, synchronously or asynchronously, but the latter is preferred in most of the cases. Starting execution of the background operation is equivalent to call a run-like function, where the specialized job of the thread is performed, as in any general thread implementation. This method is called intuitively DoWork and is usually a loop where repetitive tasks are solved. The background worker can still keep in contact from the main loop with the starting thread through a method called ReportProgress. In the main thread an event is triggered and this can be used to manipulate any user-interface objects. The changes can be controlled with the two transmitted parameters, an integer, habitually a progress percentage and optionally an object storing the current state of the worker. The BackgroundWorker operations can be cancelled asynchronously from

[2] http://msdn.microsoft.com/en-us/library/system.componentmodel.backgroundworker.aspx

the principal thread or can exit normally when the DoWork method returns. In both situations, another event is called in the main thread to notify that the job is done (RunWorkerCompleted). Once again, the user interface thread can use this information for any additional work on the visual appearance.

For the particular case of the player, the class BackgroundWorker was extended so that the new object could be paused and resumed from the main thread by interrupting the main execution of the background thread and avoiding busy-waiting. Except this small improvement, the new class was basically identical to the base. The core functionality of the player is in the DoWork method. The main loop of the method moves the current date until the end date is reached, incrementing with one day in each cycle. The current day is used as a parameter for extracting from the database all the records from that specific day. This list is then transmitted through the ProgressChanged event to the GUI thread that replaces the old markers from the previous day with the new ones.

A normal playback process can be manipulated with the help of the Ribbon's commands especially dedicated for this purpose. They are grouped under the same cell (Media) and are similar to the icons of a media player. The Play button is in charge with initiating the background worker and the replay process by running the DoWork method. The background operation can be easily halted by pressing the Pause button. This will just interrupt the background thread until the Play button is clicked again. The playing can be stopped permanently using the Stop button. The command is also resetting the current date and clears the last displayed markers. The player has a certain speed that dictates how fast the transition to the next day should be made. It is in fact a sleeping period inserted at the end of the cycle in the DoWork method. This delay can be adjusted with two buttons, Slower to reduce the speed and Faster to increase it with a predefined time unit. The First and Last options simply move the current date to respectively start date and end date while the playing is still going.

4.4.4 Waspmote

This tab assumes the presence of a Waspmote device attached to the USB/serial port of the PC hosting FlockWatch. In case there is no sensor node connected to the expected port then any of the following functions will fail to work.

All three boolean commands (Sniffer, RSSI Locator and Test Communication) situated in the Communication group are implemented in a similar manner with a background worker performing a specialized task. The fact that they can be only in two states, on or off forces the worker to try cancelling any ongoing activities in other two before being able to start its own job.

The implementations of the Sniffer and RSSI Locator threads is even more alike as they execute the same list of operations until the very end, when each of them displays differently the obtained data. This sequence of identical steps contains: listen for activity on the serial port, read the incoming message, parse it and extract the tokens, select the parts of interest and send them to the visualization method. The sniffer is using the read node ID and location latitude and longitude to add a new marker to the map with the timestamp set to now. On the other hand, the RSSI Locator needs only the node ID and the RSSI reading to place them in a table inside a new window.

The Test Communication functionality is relatively simple and closely related to the parameters set in the Communication Settings window. This command tries to establish a connection to a serial port based on the specified configurations. If the attempt times out or an exception is raised then the setup is considered incorrect and the user advised to change it, otherwise the settings are validated. These values are published in the Communication Settings window and are specific to all serial ports.

The last option in this tab, Initial Setup, is designed to make the experience of initial configuration smoother. The user has to set a series of parameters such as node‘s name and ID, GPS and GPRS modules, Synch before the configuration event can be triggered. The signification of these inputs is detailed in a dedicated section in the user manual. Pressing the Configure button will make FlockWatch connect to the serial port. The code will enable the DTR (Data Terminal Ready) pin of the serial wire, pin that allows the target to auto-reset and to step into setup mode. After reset, the board

initiates a dialogue with the other end of the serial link, in this case, FlockWatch. The application delivers the configuration properties one by one to the Waspmote in a pre-known order by both parties. This exchange is concluded with a verification message regarding the setup validation, received by FlockWatch and popped-up on the screen. There is a second façade of this process: FlockWatch uses the node‘s name and the ID typed by the user to insert in parallel a new row in the Node table. This way, the farmers can have a clear image on the sensors that are deployed on the sheep.

Chapter 5

Implementation II - Waspmote

This section is dedicated to the pieces of code written in the C language for the Waspmote device. One of these small embedded software programs controls the configuration process scheduled for each Waspmote to happen before being attached to a collar on sheep. The initial configuration can be done both from any serial terminal or with FlockWatch in a more user-friendly approach. The last two functionalities have many parts in common so they were coded in a single file. By only changing a couple of pound defines, either the sniffer or either the RSSI locator binary files are compiled and generated. The sniffer works only in connection with FlockWatch while the RSSI locator was designed to work autonomously. The contact points between FlockWatch and the functionality written for Waspmotes were also described in the previous chapter (4.4.4).

5.1 Initial Configuration

All the sensor nodes planned to be placed on the sheep have to go through this initial configuration prior to any other activity. Each Waspmote needs to know a few basic details before starting to run their normal main loop. These settings are stored and later read in the code from a persistent memory zone located at the start of the microcontroller's EEPROM (4KB) non-volatile memory, the first 512 bytes, with few exceptions for reserved features. This chunk of ROM is specially dedicated for such

usages, when the code uploaded to Waspmote depends on information saved between successive runs.

- node ID - a unique positive integer to unambiguously identify each node. When the setup is executed via FlockWatch, this ID becomes also the ID (primary key) of a new node inserted in the database for the newly configured Waspmote
- has GPS module - a boolean value marking the presence of the GPS module on the Waspmote platform
- is synchronizing - this option is applicable only if the precedent ("has GPS module") was set. It is also a true/false setting and its "raison d'être" is to force the update of Waspmotes' internal clock with the GPS reading and thus keeping it synchronized with the very precise time on the satellites. Also a node with this mode activated, after updating its own clock can then send the value to other nodes lacking totally the GPS module, for further synchronization.
- has GPRS module - a boolean value marking the presence of the GPRS module on the Waspmote platform

These values are read one by one from the serial port, through a regular serial terminal or even better for the person executing the configuration process through FlockWatch and then written to their specific memory slots. The memory map was defined by convention, within the dedicated range (11-511) with the indexes as following: ID - 100, GPS - 110, GPRS - 120, SYNC - 130. In order to test this operation was successful, the values are read back from the memory and concatenated into a single summary message that is returned via the serial connection to the configurer.

The configuration application can exist independently as a singular file or can be inserted into the preamble of the main software running on the Waspmote, as part of the setup method. In the first case, the setup binary is uploaded to the board, then the initial configuration performed and in the end the final .elf file will overwrite the flash content. With the second possibility, the initial step of having a separate program is

skipped, but the total size of the binary increases and a delay is added to the total time span the main application needs to start up. In the end, the decision on choosing one solution or the other will depend strictly on the amount of remaining free space for the compiled code.

The setup process can be run in two ways, either from any serial monitoring tool by typing manually the setup parameters consecutively after each message received from the board or either through FlockWatch. The latter was exhibited in the above chapter and requires just few mouse clicks to get the Waspmote configured.

5.2 Broadcaster

The Broadcaster is the generic name of an application that can play a dual role, of sniffer, gathering the coordinates from the other nodes in the field and of proximity RSSI (received signal strength indication) locator, being interested in the strength of the signal transmitted by others as an indication of the distance between. These two separate behaviours will be discussed in detail in the next sub-sections, but first the reader is invited to have a look over the core functionality, the code shared by both parts.

Normally, the code of the broadcaster would run on a Waspmote connected to the USB port of a computer. In this state, the Waspmote is sending regularly messages with the data received through the radio antenna from the others, with one exception, when the device is used independently as a RSSI locator, tracking a specific node. More about this special case in 5.2.1. Returning to the general situation, in the main loop of the program, the Waspmote is broadcasting a message to notify the others of its presence and then waits for replies in repetitive sequences of 20 seconds each.

For an ordinary Waspmote placed on a sheep, it does not really matter in what mode the broadcaster is working (sniffer or locator), as the answer delivered back has the same structure in both cases. This response is supposed to be a simple string message, containing data like the Node ID, current latitude and longitude, all separated by delimiters (';'). The sensor nodes missing the GPS module still have to send the reply for RSSI purposes, with invalid values in place of latitude and longitude and it

is up to the final application (FlockWatch) to use only correct data. The broadcaster gets the text, appends the RSSI parameter determined from the signal quality of the last packet received and sends everything to the serial port for further processing.

5.2.1 RSSI Locator

The RSSI locator will communicate with a single sensor node in the flock, trying to give an evaluation of its current location. This part of the code is in charge of delivering information about the signal intensity as an estimation of the distance between the two points making the radio transmission. It can be expressed in two ways, first as explained previously, with the value added at the end of the message and returned by the broadcaster over the serial link for all the nodes sending the message and second, when the Waspmote is disconnected from the computer and a particular sensor node needs to be located. This situation can occur in the collection phase when a particular sheep is dislocated from the group and farmers have trouble finding it with FlockWatch as there might be no immediate radio signal.

This second mode of working is quite distinct from the main code and extra functions had to be written. The application on the Waspmote takes a node ID as a supplementary configuration parameter when set to act as an RSSI locator. This value is used to treat in a special manner the messages originating from that specific node. The RSSI recorded for it is not only sent to serial port, but it is also displayed through the two LEDs (red and green) on the board. In this way, the Waspmote can perform a "Geiger counter" function, however in this case directed to localize a certain sheep of interest.

Printing a number with 2 to 3 digits by having just two lights can be a very challenging task and it asked for some effort to investigate and find the best way to do it. Having available two possible states, red and green, using the binary representation of a number would have been a solution, perhaps for a computer addict, but the regular user can not be expected to be familiar with binary numbers. Thus, a simpler method was proposed: once the RSSI had been obtained, its value is decomposed, the digits extracted and saved to a list. Each of these single digit numbers are transposed to the

Waspmote outputs by blinking the green LED an amount of times equal to the 0-9 integer taken into account. The transition to the next digit is made by blinking rapidly the red light. After the last digit is showed, there is a longer break until the coming up RSSI reading goes through the same cycle.

5.2.2 Sniffer

The sniffer broadcasts a short message and then listens for activity in the field from all the Waspmotes on the sheep placed within the radio range. Its job is pretty simple, to collect the geographical coordinates from the nodes and send them in a known format to the USB port. From there, the string is parsed and the retrieved information can be used in any possible way for showing to the final user. FlockWatch provides already a solution to process and display the sniffer's inputs. The code for the sniffer is almost identical to the base one, as there are no additional features requiring special attention.

Chapter 6

Tests and Results

6.1 Tests

FlockWatch was tested continuously during the development phase, verifying the functionality and detecting the potential bugs. Simulation data had to be added to the database in the initial stages in order to achieve this, as real data from the field were not yet available at that time. These records were created artificially with a small piece of software, specially written for this purpose. A record contains random, but valid data, similar to the real-time ones, with the geographical coordinates generated around the already known flock center. These inputs sufficed to implement the features presented today in the application.

Another side of the testing procedure was to check the Waspmote capabilities from FlockWatch. A sensor node had to be hooked up at all time to one of the USB ports on the computer running the application. This setup was mostly useful in the first phases of implementation and debugging when it was important to analyse the traffic coming from the sensor and make sure that the correct bits and pieces are imported into FlockWatch.

A single **Waspmote** microcontroller was used in testing the initial configuration functionality. At first, a serial terminal had the job of displaying the messages from the device and sending back commands to it. This tool was employed mostly in developing and fixing bugs. Later, the serial monitor was substituted by the configuration window

in FlockWatch where all the setup can be performed by simply typing short words and ticking a couple of checkboxes.

The Broadcaster capabilities were tested with up to two Waspmote sensor nodes, one having just the radio module while the other was also equipped with the GPS module. One separate check was to see the capacity of the system to exchange packages from the broadcaster to the second node and the other way around. The test completed successfully with data including the GPS position arriving intactly at destination. At the same time, a special attention was giving to the integration of the broadcaster with FlockWatch, therefore the serial connection between the two entities has also been part of this testing procedure, but in this case focusing more on tracking down and fixing bugs.

A totally separated part of the process of testing was to figure out how is the RSSI locator handling the conditions of different surroundings and how the RSSI readings are affected by them. Two tests were performed, one with the broadcaster indoor and the other node outdoor, in motion, while the second test was with the two together outside but again the broadcaster in a fixed position and the other Waspmote moving. Both of the tests were done in a urban environment with relatively short buildings. The results are depicted in the following section.

6.2 Results

The immediate effects of the previous tests were analysed and the drawn conclusion are summarized in the current section.

FlockWatch had to follow a requirements document and all the development and testing phases had to be done with it in mind. In general, with applications trying to adhere to a predetermined plan it is difficult to expose other results than the conclusion that all the specified features were implemented or not, and FlockWatch does not make an exception. In the early days of this project, a list containing the wished features had been compiled and based on them a Gantt diagram had been drawn to be taken as reference along the development line. Today we can affirm that most of the intended functionality is in place, with a certain deficit in the area of testing with real data and

involving the final users.

The tests of the different embedded applications for **Waspmote** are presented as follow. The results of testing the original configuration tool were quite obvious as the configuration process itself is simple and straightforward. After the preliminary stage of testing for anomalies in behaviour, the application was considered ready to be employed for the purpose it had been designed for and started to set up Waspmotes.
Initially, the Broadcaster was tested for the basic functionality to make sure that the packages are delivered as expected through the radio link between the transmitter and receiver. The results of this tests were satisfactory and as their consequence, the work on integrating it to FlockWatch has begun.
As mentioned in the "Tests" section (6.1), the RSSI locator has undergone two distinct categories of tests, in two different environments. In both cases, the sensor node sent to the broadcaster a package containing its current location determined through the GPS receiver. The broadcaster added to the received string the measured RSSI value and saved all together. These pieces of information were then used by a small helper application developed in C# to compute the distance between the broadcaster and the second Waspmote and associate the RSSI value with the calculated distance. This external software takes two pairs of geographical coordinates and calculates the distance between, by using the great-circle distance formula (the great-circle distance or orthodromic distance is the shortest distance between any two points on the surface of a sphere measured along a path on the surface of the sphere, as opposed to going through the sphere's interior[13]).

Before stepping to perform the main tests, the accuracy of the GPS receiver had been verified. In order to do this, the sensor node was placed in a fixed position with opening to the sky so the GPS could get a lock on the satellites. Once the GPS locations were available, the sensor node started to transmit repetitively them to the broadcaster, which saved them in a log file with a timestamp attached. This file was used as an input for the external software that computed the distances between each reading from the Waspmote and the "correct" pair of coordinates. The test run for almost 1000 seconds and a step of 13 seconds after each new data transmission. The obtained results are showed in the following plot (6.1):

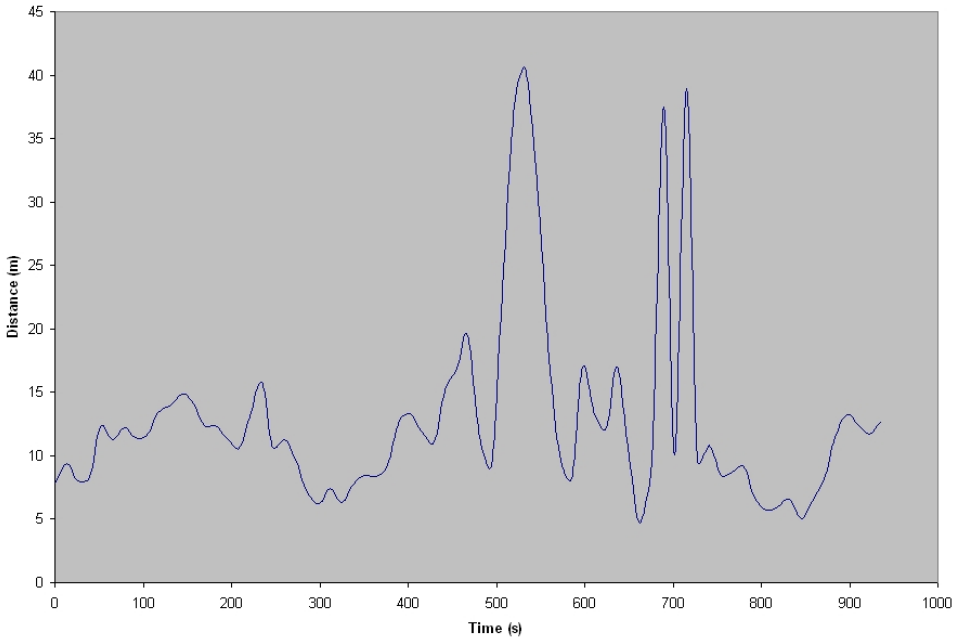

Figure 6.1: GPS error margin

It can be observed that the read coordinates oscillate quite significantly for consecutive values when it should be a straight horizontal line close to 0. This lack of precision of the GPS receiver will automatically influence the final results of the considered tests. The noise characterizing the GPS readings will overlap the already present radio interferences and the final results will be affected by errors further more.

Another considered test was interior with the broadcaster in a regular room close to the window. Meanwhile the second Waspmote, simulating the sensor node on a sheep was transported from the same room until it was out of the broadcaster range and no longer communications occurred. In that point, the sensor node was taken the way back and further records were logged. The second phase of the test was to process the data with the same helper tool and save the computed distances together with the read RSSI values. All these pairs were then sorted ascendingly by distances and used to draw the plot below with the distance expressed in meters on the abscissa axis while on the ordinate were placed the associated RSSI values expressed as negative dBm (power ratio in decibels (dB) of the measured power referenced to one milliwatt (mW)). The

resulting values were condensed in the following plot 6.2:

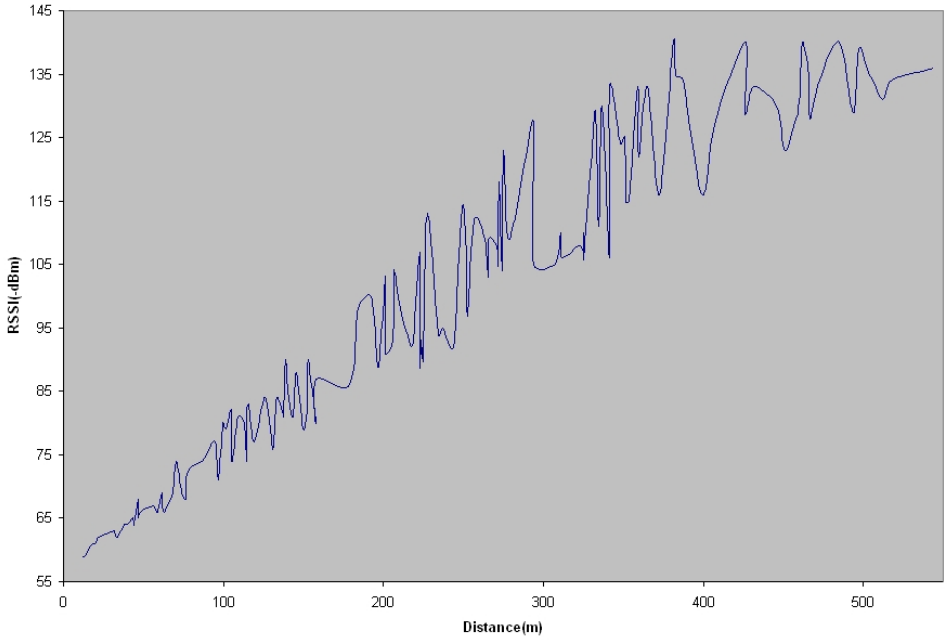

Figure 6.2: Broadcaster indoor

An immediate observation is that even if the overall trend is ascending, the individual values tend to be quite unstable which may hinder the tracking down attempt. Also, at the superior edge of the curve, where the signal quality becomes pretty weak, a difference of one measured RSSI unit can mean a gap of 25% in terms of distances which is high enough not to trust the result.

A similar shape was obtained for the second test performed exclusively outdoor (6.3). Obviously, the recorded range is higher in this case with better transmission-reception conditions, but the local fluctuations persist in the context of the same constant slope. The general impressions made for the previous graph are also valid for this one.

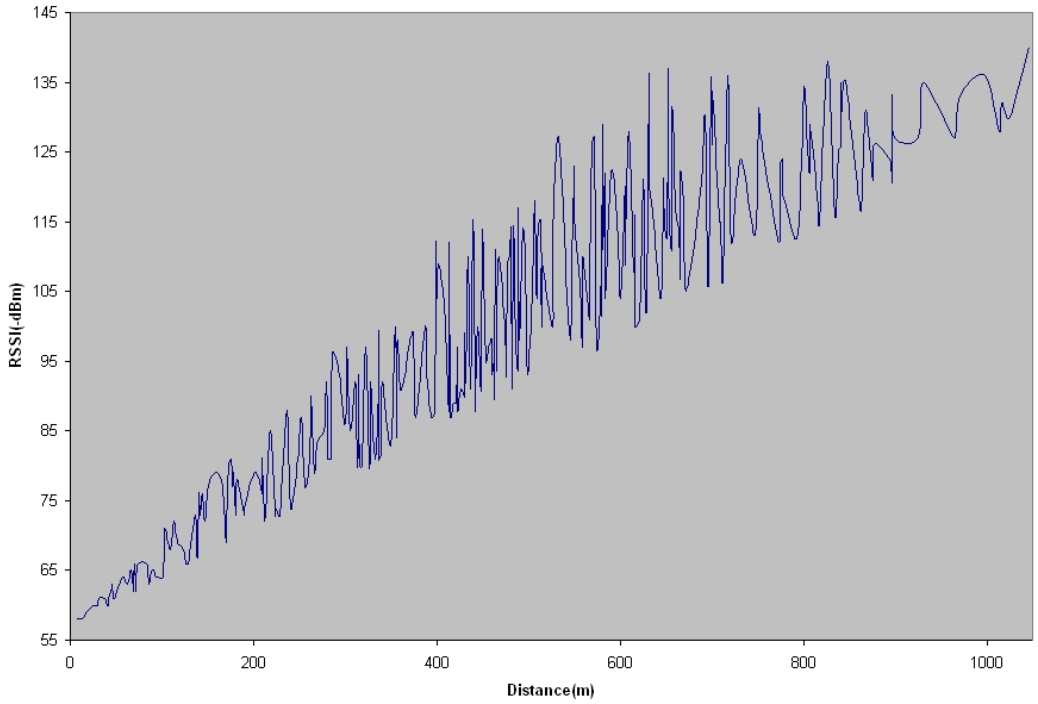

Figure 6.3: Broadcaster outdoor

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The conclusions chapter is also divided in two, one section for each category, as the two main parts of the project were quite distinct.

Usually, in this kind of reports, there is a section describing the previous work in the considered domain. Due to the special nature of this project with an application customized for the given situation, it was decided that such a chapter should be skipped. It is likely that currently there are other products on the market with a similar role, monitoring various targets, but in this case it was preferred to develop **FlockWatch** from scratch to cope successfully with the cases specified by the project requirements.

FlockWatch exposes a state-of-the-art visual interface that enhances greatly the user experience with the tool. The task of tracking and monitoring in an accurate and precise manner is eased by this user-friendly environment. The application provides several means of observing from a distant location how the flock progresses. There are two complementary methods of operating FlockWatch, one via the MySQL database server for displaying the latest position or to review historical data, and the other assumes a Waspmote sniffer set up with a radio antenna. The first one is used primarily to get daily updates during the period the sheep are left unattended in the mountains, while the other is intended for the collection phase when it is important to get real-

time readings for a faster localization. In both cases, FlockWatch reveals unitarily to the user the same familiar interface, configurable with a wide selection of maps.

As for **Waspmotes**, two separate parts had to be considered, hardware and software.

First, we analyse the hardware provided by Libelium, consisting of the main board, battery and optionally the GPS and GPRS modules and also the radio antenna. The first impressions about the developing kit were that it seems relatively fragile with the battery and the GPS receiver hanging in a couple of thin wires. As a matter of fact during the testing phase, these binding elements broke under the weight of the extremities and needed to be fixed by soldering them meticulously.
Also, by the way the Waspmote PCB was drawn, the total amount of UART channels is limited. This restriction led to the annoying situation of removing the Xbee module from the main board each time a new version had to be uploaded, as the two processes share the same transmission line. This odd procedure made the development routine to become arduously and also costly in terms of the time spent to check just a small change performed to the source code.
Another aspect observed during coding and testing stages was that the hardware was not behaving always as expected, apparently with no good reason. This baffling conduct manifested especially with the GPS module which chose randomly between working properly or not when placed in the same position and in similar conditions. This instability has slowed down significantly the process of getting the final results of the above tests where the GPS receiver had an important role.

Then, the writing code side of the project was not the most pleasant experience. First of all, the API provided by the Waspmote manufacturer is still in an immature, incubation phase, insufficiently tested and hiding a good amount of bugs and issues. The development of the Waspmote programs was started with the version 0.14 of the library which had a serious memory corruption problem in transforming a MAC address from an array of bytes to string. This required days of investigations and debugging until finally the cause was identified. The bug was fixed in the latest version, 0.18 but it was a frustrating experience to suspect our own code of malfunctioning when the real flaw was in a part that should be considered stable and reliable.

As mentioned in 3.4, the IDE accompanying the API and the hardware parts from Libelium is primitive and lacks many of the nice features present in the modern editors that makes the programmers life easier. From the very beginning, it was not considered as a decent candidate suitable for the development process so soon it was substituted by a superior environment, Eclipse.
Another important issue with the provided materials was that the only method to debug the written code is rudimentary, simply sending text messages to the serial port and based on them trying to figure out what are the eventual problems. Even though it is still embedded development, a bit more challenging than the conventional one, writing code for the Waspmote is equivalent to the stone age of programming when tools like a proper development environment or a JTAG debugger were not yet invented.

In summary, we can conclude that probably today Waspmote is one of the few products capable of performing so many function (GPS, GPRS, radio, etc) on one board, but it still has a long way to go until reaching a state that can recommend it as an excellent solution.

Referring strictly to the results exposed in the previous section dedicated to tests, a definite conclusion is that the RSSI value is a good indicator of the radio link quality, but when it comes to estimate distances it does not give definitive answers. Perhaps in the location where the flock will graze the results will be more stable than the urban environment considered in the test and the relation RSSI - remaining distance will be more exact, but the sheep's bearing still has to be guessed by wandering around. Even in a medium without most of the known interference sources, the results can vary substantially as the signal level still suffer distortions.

7.2 Future Work

The future work is also scheduled to happen separately for each of the two main directions debated in this report, FlockWatch and Waspmote.

Currently, **FlockWatch** supports a minimum of functions making it able to satisfy the original list of requirements. But considering the fact it was used so far only with test data and in opposite environments than the one the application was build for, it

is expected over the first operational stages to see the users expressing the need for additional features meant to ease their daily activity related to the application.

There are some features that could be inserted into the plan of a future release of FlockWatch as seen today. Among them the following may be mentioned: Implementing an alarm service meant to notify in real time (via SMS or e-mail) the farmer about major events taking place with the flock. Such events could be a sensor logging approximately the same position for days (presume something bad happened to the animal), the Waspmote‘s battery is getting low on power and the list can be extended. Another part of the potential work can be adding an intelligent system able to interpolate a possible route the animals followed and extrapolate the future locations based on the history. Drawing a probable path from the previously recorded position samples could be a challenging task especially if the algorithm should take into account the particularities of the terrain in order to circumvent the areas where a sheep can not physically cross. Also, trying to figure out were the next location might be should not depend solely on the history, but also considering if the sensor node's carrier was part of a small group during the earlier days, as the sheep tend to stay in the same company for a longer period. Indubitably, such algorithms have to be constructed in close relationship with the observed sheep's ordinary behaviour and then fine-tuned with real data from the field.

A distinct point in presenting the future work is referring to the **Waspmotes**. The application for the initial configuration might be extended to support new setup parameters in case the design of the main code will require new adds-in. The changes should be trivial without heavily impacting the current implementation, perhaps just defining new memory zones and add the new settings to the existing list. Obviously, the other external applications depending on the current protocol have to be adapted accordingly.

Regarding the broadcaster, it seems to meet all the today's requirements, therefore no immediate change is foreseen. But in case one might think to an extension of the current implementation then extra features can find easily their place among the existing ones. Such features can include additional GPS details (altitude, speed, direction) and their involvement in a more elaborated analysis on the sheep's route, or

performing a wider study on how the terrain affects the RSSI and add the conclusions to the existing logic on estimating the distance based on the signal quality. The Waspmote might be configured with a predefined type of landform which can be part of the formula calculating the distance in meters and so the final result could be more consistent with the reality in the field.

Bibliography

[1] L. J. Asheim and I. Mysterud. The norwegian sheep farming production system. 1999.

[2] P. DuBois. *MySQL*. Addison-Wesley, August 2008.

[3] *Cisco Mobile Exchange (CMX) Solution Guide*, August 2002.

[4] J. A. Kreibich. *Using SQLite*. O'Reilly Media, August 2010.

[5] L. Letham. *GPS MADE EASY: Using Global Positioning Systems in the Outdoors*. Mountaineers Books, May 2008.

[6] J. Liberty. *Programming C#, Fourth Edition*. O'Reilly, February 2005.

[7] F. Marguerie, S. Eichert, and J. Wooley. *LINQ in Action*. Manning, May 2008.

[8] S. Mostarda, M. D. Sanctis, and D. Bochicchio. *Entity Framework 4 in Action*. Manning, May 2011.

[9] C. Nagel, B. Evjen, J. Glynn, M. Skinner, and K. Watson. *Professional C# 2008*. Wiley, March 2008.

[10] A. Nathan. *WPF 4 Unleashed*. SAMS, June 2010.

[11] N. Rice. Openaccess made easy, March 2010.

[12] P. D. Sheriff. *Fundamentals of N-Tier*. PDSA, Inc., October 2009.

[13] R. W. Sinnott. Virtues of the haversine.

[14] *Waspmote: Technical Guide*, May 2011.

FLOCKWATCH
User Manual

1 Prerequisites

FlockWatch is a Windows application and therefore will run only on the newer operating systems supported by Microsoft, namely Windows XP, Windows Vista or Windows 7. Also, this application relies on the .NET 3.5 framework (available for download at Microsoft website) which has to be present on the hosting machine prior to launching FlockWatch. It has to be installed explicitly just for Windows XP, as the .NET framework is included by default in Windows Vista (version 3, hence an upgrade is required) and Windows 7 (version 3.5).

2 Installation

The user has to run the installer (setup.exe) and follow step by step the wizard (1). This will copy the necessary files to the chosen path on the harddrive and setup the application for the first start.

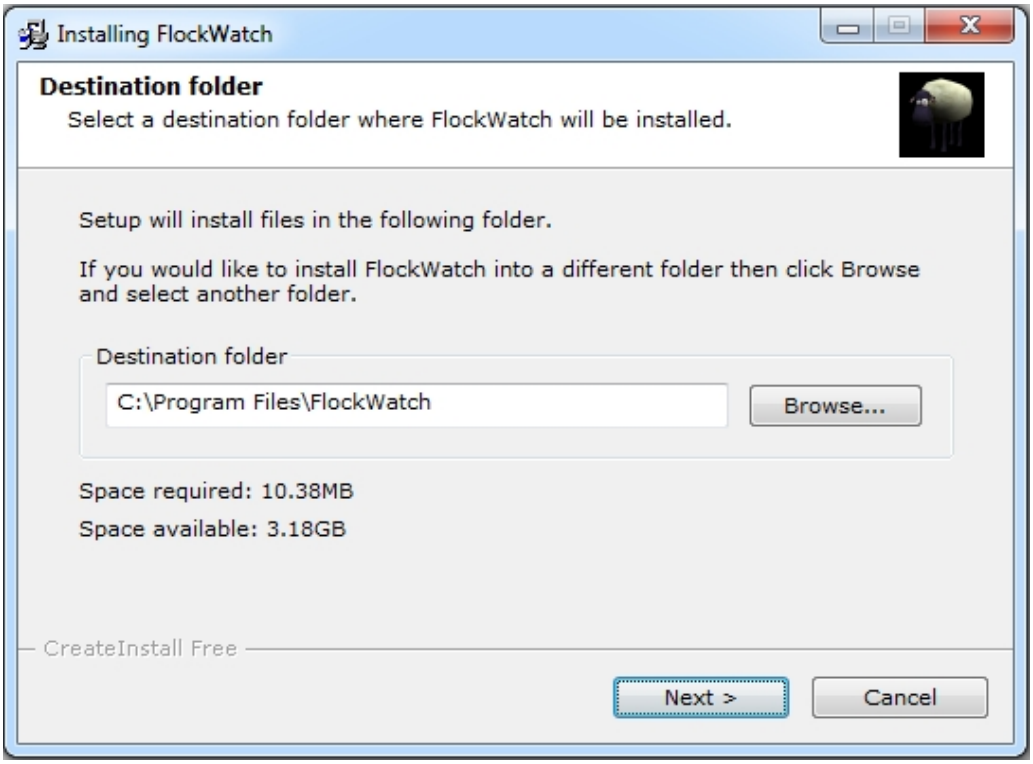

Figure 1: Installation wizard

3 Running

The application can be started from the folder where the installer copied the installation files. By default, this path is "C:\Program Files\FlockWatch". Optionally a shortcut on the desktop might had been created that can also be used. Regardless of how the user starts FlockWatch, on initialization, a splash screen will be displayed for a few seconds, time period used by the application to load the user settings and try to connect and retrieve the latest data regarding the sheep positions from the database server. After this preliminary stage, the main screen appears and the application is ready to be used.

4 Application Menu

The application menu can be found on top side of the main window. The following figure (2) provides an overview of it:

Figure 2: Application menu

Its constituent parts are described as follow:

Quick access toolbar: a collection of the most used Ribbon commands, accessible no matter which tab has been selected. By default, the tool bar contains some of the most useful navigation buttons (Zoom in, Zoom out, Fit best), but the user can add any other one from the Ribbon's tab through a simple right click.

Title: shows the name of the application, FlockWatch.

Application button: a drop-down menu containing a few useful functions that apply to the entire application. Momentarily, this menu has only two options, but the list can be extended in the future with new features.

- Export Image - exports the map content to an image; it is in fact a snapshot of the current state of the flock
- Exit - simply triggers the application to exist and performs some housekeeping work

Help: this manual in PDF format can be opened directly from FlockWatch by clicking the help button (shaped as a circle containing a question mark - the classic symbol for help) or by pressing the F1 key. The application will try first to detect a valid installation of Adobe Acrobat Reader and if successful, then the document is loaded.

5 Sections

The main functions of the application are divided in 3 significant sections. The map component is present in all 3, located in the middle-bottom side of the window and each part has specific commands that are grouped in the Ribbon as tabs. These sections are presented as follows:

5.1 Map

This part is specially dedicated to the map view. When this tab is selected then markers containing the latest records of the sheep are displayed, with the node ID written on the marker. Regardless of which tab is selected, the user can interact with the markers in two ways, by:

- moving the mouse over a marker, the user gets access to additional information such as the node's name and the timestamp of the record (position)
- selecting a group of markers inside a box opens a pop-up window with details about sensor nodes. These settings can be altered and saved back to the database so later the sensor nodes change their behaviour accordingly. An image of this configuration window is showed below: (3)

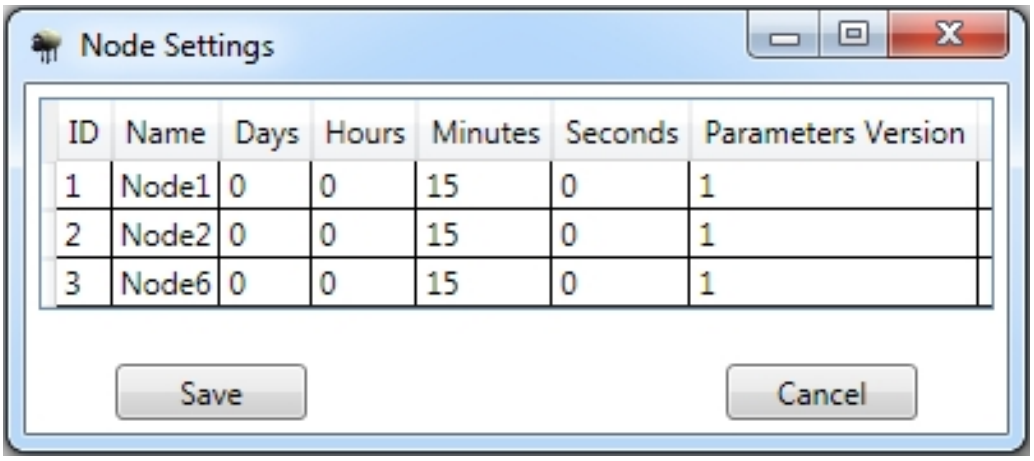

Figure 3: Nodes configuration

5.1.1 Ribbon Commands

A screenshot of the available commands in this section is depicted here 4:

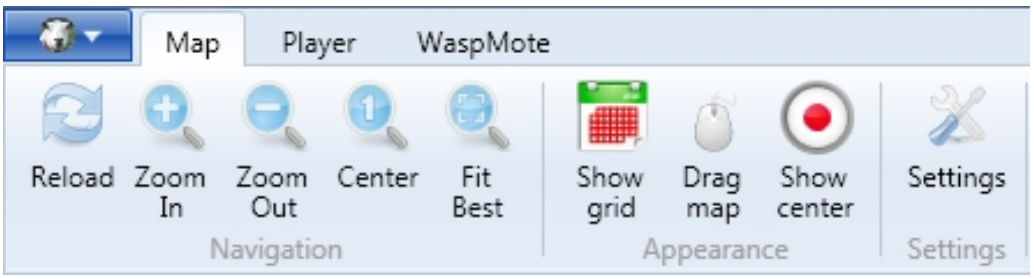

Figure 4: Map commands

Navigation: the buttons in this group are used, as the name suggests, to facilitate the exploration of the map

- Reload - refreshes the map page; the map tiles and the markers are reloaded
- Zoom in - zooms in the map; in case the maximum level of zooming is reached, the button is disabled
- Zoom out - zooms out the map; in case the minimum level of zooming is reached, the button is disabled
- Center - moves the view over the map so the predefined center point becomes the center of the window

- Fit best - moves the view over the map and adjust the zooming level so all the markers (sheep) are visible

Appearance: some on/off commands used to change the details about the style of displaying the map

- Show grid - shows the tiles forming the map as a grid
- Drag map - enables or disables the option of dragging the map
- Show center - shows the center of the map in form of a red cross

Settings: The map settings as defined in the following window (5).

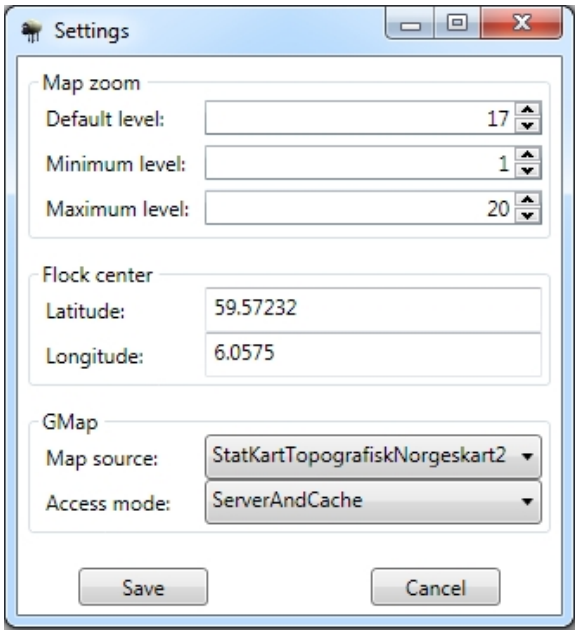

Figure 5: Maps settings

- Map zoom - contains different settings concerning the zoom range.
  - Default level: the map is set to this level by default when the application starts
  - Minimum level: the level to which the selected map can be zoomed out
  - Maximum level: the level to which the selected map can be zoomed in

- Flock center - a point in which the map is centred when the application starts.
  - Latitude: the latitude coordinate of the center
  - Longitude: the longitude coordinate of the center
- GMap - settings of the map source.
  - Map source: select from a long drop-down list the map server
  - Access mode: select from where to load the map tiles, server, cache, or both

5.2 Playback

The Playback section can be used to replay historical data regarding the past locations of sheep.

5.2.1 Ribbon Commands

A screenshot of the available commands in this section is depicted here 6:

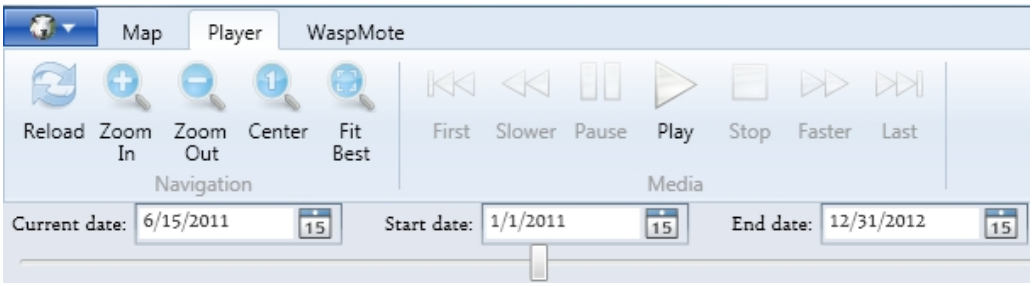

Figure 6: Player commands

Navigation: Identical to the paragraph with the same name in 5.1.1

Media: The buttons in this group are similar to those used in a media player. Their signification is explained below:

- First - brings the playing bar to the start date; if the player is not running, the button is disabled
- Slower - decreases the playing speed with one unit; if the minimum speed has been reached or the player is not running, the button is disabled

- Pause - pauses the player; if the player is not running, the button is disabled
- Play - starts the player; if the player is already running, the button is disabled
- Stop - pauses the player and resets the playing progress; if the player is not running, the button is disabled
- Faster - increases the playing speed with one unit; if the maximum speed has been reached or the player is not running, the button is disabled
- Last - brings the playing bar to the end date; if the player is not running, the button is disabled

5.3 Waspmote

The Waspmote section is dedicated to several jobs involving the Waspmote sensor nodes.

5.3.1 Ribbon Commands

A screenshot of the available commands in this section is depicted here 7:

Figure 7: Waspmote commands

Navigation: Identical to the paragraph with the same name in 5.1.1

Communication: another set of on/off commands meant for starting or stopping different functions involving the serial port and the attached Waspmote

- Sniffer - activates the sniffer; new markers (sheep) appear on the map as new data is received via the serial connection
- RSSI locator - activates the RSSI locator; new RSSI readings are displayed in the showed pop-up window as new data is received via the serial connection

- Test communication - checks whether the settings defined in the “Communication settings“ window are correct or not

Settings: two categories of settings, one for defining the serial communication parameters and one used for the initial setup of the Waspmotes

- Initial setup - The Waspmote initial configuration options are available in this pop-up window Figure 8:

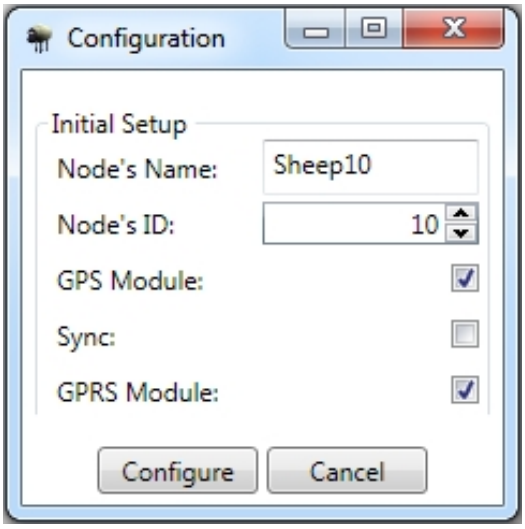

Figure 8: Initial configuration

  - Node’s name: a human-readable name for the node that will be also part of the record inserted into the database
  - Node’s ID: a unique positive integer to unambiguously identify each node
  - GPS module: marks the presence of the GPS module on the Waspmote platform
  - Sync: this option is applicable only if the precedent (“GPS module) was set; It is a true/false setting and it is used to force the update of Waspmotes’ internal clock with the GPS reading and thus keeping it synchronized with the very precise time on the satellites; also a node with this mode activated, after updating its own clock can then send the value to other nodes lacking totally the GPS module, for further synchronization

  - GPRS module: marks the presence of the GPRS module on the Waspmote platform

- Communication settings - The necessary configurations to set up the serial communication to Waspmotes (Figure 9):

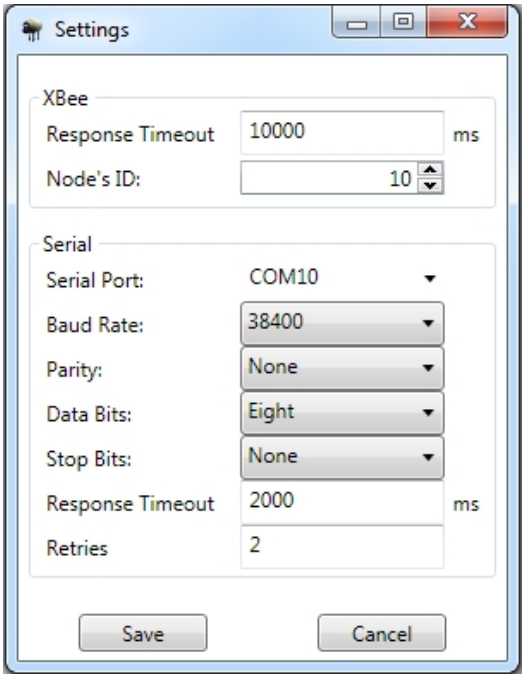

Figure 9: Communication settings

  - Xbee: settings for the Xbee module
    * Response timeout - the time period to wait for a reply via the radio module before continuing
    * Node's ID - the node's ID to monitor for signal quality, with the result displayed on the Waspmote's LEDs
  - Serial: settings for the serial link
    * Serial port - select the correct port allocated by the operating system (COMXX) for the connection to the Waspmote
    * Baud rate - choose the right speed (data rate in bits per second) expressed as baud
    * Parity - choose parity in case the communication needs to detect transmission errors
    * Data bits - select the amount of data bits carried by each character

* Stop bits - have stop bits at the end of a character in case the character stream needs to be synchronized
* Response timeout - the time period to wait for a reply before continuing
* Retries - the amount of retries in case of an unexpected problem, before giving up

6 About

The application this user manual is written for was developed as part of a master's thesis. Any inquires or suggestions about it can be sent via email to the author, Laurenţiu Cocanu.