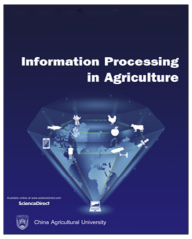
Available at www.sciencedirect.com

INFORMATION PROCESSING IN AGRICULTURE 4 (2017) 1–17

journal homepage: www.elsevier.com/locate/inpa



Wireless data management system for environmental monitoring in livestock buildings

James Gray[a], *Thomas M. Banhazi*[a,b,*], *Alexander A. Kist*[a]

[a] *Faculty of Health, Engineering and Sciences, University of Southern Queensland (USQ), West Street, Toowoomba, QLD 4350, Australia*
[b] *National Centre for Engineering in Agriculture (NCEA), University of Southern Queensland (USQ), West Street, Toowoomba, QLD 4350, Australia*

ARTICLE INFO

Article history:
Received 18 April 2016
Received in revised form
19 October 2016
Accepted 5 December 2016
Available online 12 December 2016

Keywords:
Sensors
Environmental monitoring
Networking
Software engineering

ABSTRACT

The impact of air quality on the health, welfare and productivity of livestock needs to be considered, especially when livestock are kept in enclosed buildings. The monitoring of such environmental factors allows for the development of appropriate strategies to reduce detrimental effects of sub-optimal air quality on the respiratory health of both livestock and farmers. In 2009, an environmental monitoring system was designed, developed and tested that allowed for the monitoring of a number of airborne pollutants. One limitation of the system was the manual collection of logged data from each unit. This paper identifies limitations of the current environmental monitoring system and suggests a range of networking technologies that can be used to increase usability. Consideration is taken for the networking of environmental monitoring units, as well as the collection of recorded data. Furthermore, the design and development of a software system that is used to collate and store recorded environmental data from multiple farms is explored. In order to design such a system, simplified software engineering processes and methodologies have been utilised. The main steps taken in order to complete the project were requirements elicitation with clients, requirements analysis, system design, implementation and finally testing. The outcome of the project provided a potential prototype for improving the environmental monitoring system and analysis informing the benefit of the implementation.

1. Introduction

Air quality is an important factor that needs to be considered in livestock production, especially when livestock is kept in enclosed buildings. There are a number of airborne pollutants found in piggery buildings that are of concern. The most notable of which are ammonia (NH_3), carbon dioxide (CO_2), airborne particles, and microorganisms [1]. When these pollutants are above certain levels, they can have detrimental effects on the health and welfare of exposed humans and livestock, can reduce the production efficiency of livestock, and can have a negative impact on the external environment [2]. In order to determine if air pollution is an issue, as well as what actions need to be taken, the air quality within a

[*] Corresponding author at: National Centre for Engineering in Agriculture (NCEA), University of Southern Queensland (USQ), West Street, Toowoomba, QLD 4350, Australia.
E-mail address: thomas.banhazi@usq.edu.au (T.M. Banhazi).

Peer review under responsibility of China Agricultural University.
http://dx.doi.org/10.1016/j.inpa.2016.12.001

livestock buildings need to be monitored. In 2009, an *Environmental Monitoring System* (EMS) was designed, developed and tested in Australia. The development of the EMS was born from the need for an accurate, low-cost user friendly kit for monitoring airborne pollutants, which would allow for air quality improvement strategies to be implemented [3]. The routine use of the EMS has the potential to improve building environments and reduce pollutant emissions by creating awareness of air quality issues amongst livestock farmers [2]. The EMS was developed with the key airborne pollutant found in piggery buildings in mind. Sensors allowed the recording of air quality components and portability was maintained in order for the unit to be taken to different livestock buildings. The unit was also field tested in working piggery buildings, which allowed for the assessment of operational/labour requirements as well as the capabilities of the system. Improvements in effectiveness and efficiency and reduced cost of the system gave it the potential to become a routinely used tool for the monitoring of air quality in the future.

However, there are a number of key limitations of the current EMS implementation. The main problem is the labour and time required to collect, collate and store data recorded by EMS units. An increase in ease and efficiency of these processes will increase usability of the system and therefore the potential for deployment and widespread use. In order to resolve the problem, telecommunication technology can be used to create a system that will automate the data collection and management process. This study investigates how the current EMS can be improved using telecommunication technologies. In the first part current networking technologies are discusses and their suitability in livestock buildings is investigated. In the second part a specific prototype implementation for the existing EMS are discussed and evaluated. The key contributions of this work include an overview of available networking and data management technologies that can be used to build could-based environmental monitoring systems. In particular their application in the context of agricultural engineering is evaluated and practical implications for the design of could-based systems in this context are discussed. The paper also introduces a case study that demonstrates how such a system can be designed and build in practical terms. The discussions and the conclusions that are drawn are generic and apply to other systems in similar environments as well. The remainder of the paper is organised as follows: Section 2 discusses an example of an environmental monitoring system to provide the necessary background to provide constraint that inform the selection of suitable communication technology. Section 3 provides a tutorial style overview of technologies that can be used to provide network connectivity for environmental monitoring systems. Section 4 address alternative approaches to data management and Section 5 highlight implications for the overall system design. Section 6 introduces a requirements analysis and the system design. The prototype and testing are discussed in Sections 7 and 8. The paper concludes with results in Section 9.

2. Background and literature review

This section introduces current environmental monitoring system in more detail. The system has evolved over time undertaken a variety of upgrades and modifications since its origins as the BASE-Q system, such as those described in Clements et al. [4] and Saha et al. [5] The latest working EMS is discussed and its requirements and limitations are highlighted.

The *hardware* configuration of the current EMS unit is depicted in Fig. 1 and in detail described in Clements et al. [4]. The EMS unit is all contained within a small plastic hard case, which allows for ease of portability during transport and mounting when in use, as well as protecting the internal components from the operating environment. The gas monitoring subsystem allows for the monitoring of ammonia and carbon dioxide both inside and outside the livestock shed. The small size of the gas sensor modules allows for them both to be contained in a single air-tight chamber, which is supplied air by the single vacuum pump. Because the chamber has a transparent lid, it also allows for easy inspection of cleanliness or other physical issues.

The air monitoring subsystem allows for the monitoring of temperature, humidity and particle concentration within the livestock shed. The air sampling system requires a continuous flow of air, which is achieved using a small fan within a straight-through vent from one side of the case to the other. Within the vent are two sensor modules, a temperature and humidity sensor, and a dust particle sensor. As with the gas monitoring subsystem, the vent is mainly transparent,

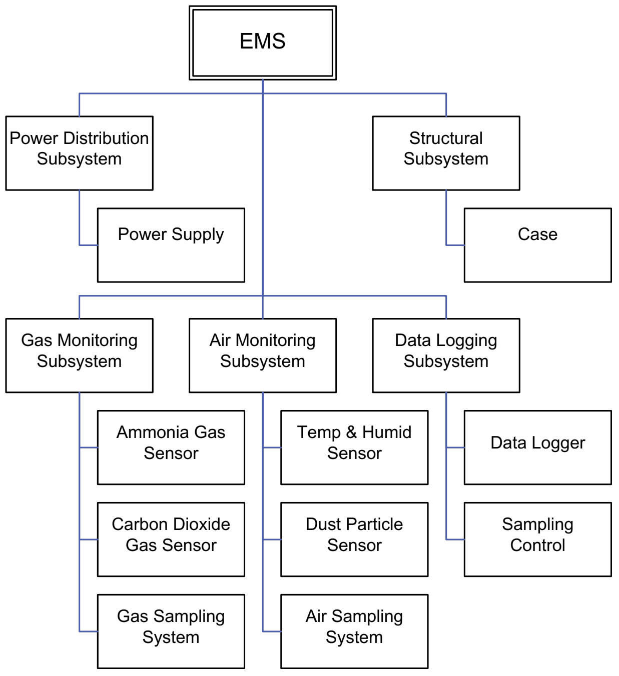


Fig. 1 – Physical architecture of the EMS (Clements et al., 2011).

allowing for visual inspection. The power distribution subsystem provides power to all the other subcomponents of the unit. The data logging subsystem records sensor data, controls the fan and vacuum pump, and provides a user interface for the unit. The custom circuit board includes a microcontroller, real-time clock, LCD screen, SD card interface and three push buttons. The LCD and push buttons provide basic user I/O for configuring, starting and stopping the system. The real-time clock is used to time-stamp records obtained from sensor modules, which are saved locally to the non-volatile memory.

The BASE-Q *software* is a database application used to capture, store and retrieve air and environment quality data. It was designed for use on a Windows 2000 operating system and has a desktop and pocket-based version. The desktop BASE-Q software maintains data records, provides data security, accepts data from external sources and from manual input and provides search and reporting functions. Using the software, farms and sheds can be added/removed, as well as details provided such as address, etc. The desktop BASE-Q software is the main repository for data obtained from EMS units, and provides functions to manage, store and view the recorded data. The pocket-based BASE-Q software (PBQ) can be run on portable PC's running Windows CE. PBQ allows for information to be easily retrieved/supplied in the field, such as from SD cards out of EMS units, before being returned to the desktop BASE-Q for synchronization.

In comparison to the earlier BASE-Q units, for latest revision of the EMS, the cost was reduced by approximately 80%, the size reduced by approximately 75%, the weight reduced by 85% and the lifespan of the ammonia sensor was increased by approximately 200% [4]. The main operational/labour requirements for the system can be broken into three categories: pre-farm, which includes preparation and calibration; on-farm, which includes deployment of equipment and sampling procedures; and post-farm, which includes cleaning of equipment and downloading. According to results obtained whilst testing the BASE-Q unit, both pre- and on-farm operational/labour requirements were within approximately 1 and 2 h, whereas post-farm was between approximately 2 and 4 h [3]. Due to the improvements made in the EMS unit, and without extensive testing, an all-round improvement can be assumed. For instance, calibration requirements (pre-farm) have been reduced, deployment effort has been reduced (on-farm), and cleaning requirements have been reduced (post-farm).

Taking this into account, the greatest amount of operational/labour requirement remains in the post-farm processes, which predominantly consists of downloading and managing recorded data. Two main limitations of the EMS system are that it does not incorporate an airborne bacteria monitoring system, and that data needs to be manually acquired from each sensor. Difficulties such as techniques for measuring and accuracy have been discussed with regard to monitoring airborne bacteria [3]. The automation of record downloading has often been considered as a future possibility in the development of the EMS system, and the development of such automation was commenced, as documented by Saha et al. [5], however, the design was not implemented. In order to reduce the limitations identified, a solution should ideally provide the following:

- Automatic data retrieval from EMS units on a farm.
- The ability to easily increase/decrease the number of EMS units logging on a farm.
- Automatic collaboration of data from different farms to a central location.
- Data analysis and viewing from the central repository.
- Security features for privacy and protection.

There are two main sections of networking technology that will be considered: sensor networking implementation and data collaboration and storage. In the following two sections, a review of literature will establish technologies and methods that will be most appropriate to solve the problem previously outlined.

3. Alternative technologies to network environmental sensors

As stated by the requirements, a means of automatically retrieving records from the EMS units must be designed using wireless communication technology for greatest effectiveness and ease of use. Wireless communication has been used successfully for a multitude of agricultural monitoring situations such as in greenhouses and fields to monitor conditions and plant growth [6].

3.1. *Wireless technologies*

Four networking technologies are widely used. This section provides a comparison and most efficient uses, determined from the literature. The discussion focuses on technologies that use high frequency radio as communication. Other alternatives such as microwave communication and infrared (IR) have been omitted as these do not work well in a farm shed environment.

Wi-Fi is a popular name for Wireless Fidelity, which is standardized as IEEE 802.11a/b/g/n. Wi-Fi is used mainly for Wireless Local Area networks (WLAN) such as providing internet access to devices throughout an entire household. The transmission range of Wi-Fi is nominally 100 m, with a maximum signal rate of 54 Mb/s and bandwidth of 22 MHz. This makes it an ideal wireless technology for general networks which provide communication between personal computers, laptops and routers. The transmission time of Wi-Fi is relatively high and the data coding efficiency is high with large data payload sizes. *Ultra-wideband*, or UWB, is standardized by IEEE 802.15.3 and is becoming a popular technology for short-range high-speed communication, such as in multimedia applications [7]. This is due to the very high signal rate and channel bandwidth. These characteristics mean that UWB has a very fast transmission time; however, out of the four discussed wireless technologies it has the greatest power consumption.

ZigBee, as standardized by IEEE 802.15.4, is used to support simple devices that consume minimal power and operate within a 10 m range [8]. It provides self-organized, multi-hop mesh networking with low power consumption [9]. Of the discussed technologies, ZigBee has the lowest signal rate and channel bandwidth, which also gives it the highest transmission time and makes it inefficient when transferring large data payloads. On the other hand, ZigBee is second in efficiency only to Bluetooth when transferring small data payloads, has the least complex protocol stack and consumes the least amount of power. ZigBee has been implemented successfully in customized wireless sensor networks, an example of which is shown in Zhang et al. [6], where ZigBee was used to communicate from wireless sensors from within greenhouses. The results of the implementation determined that the ZigBee technology provided the capabilities of self-organization, self-configuration, self-diagnostics and self-healing.

Bluetooth, standardized as IEEE 802.15.1, is designed for short range and cheap devices such as keyboards and headsets. The wireless range reaches a maximum of 10 m, known as the wireless personal area. Bluetooth communication occurs as a master and slave relationship between devices, where a master can communicate with multiple slaves but a slave can only communicate with one master. Bluetooth also has small bandwidth, a small signal rate and a complex protocol stack, meaning that it has a relatively high transmission time. On the other hand, Bluetooth has a very low power consumption, which makes it ideal for small battery powered devices. As discussed in Lee et al. [8], Bluetooth is the most efficient technology for transmitting small data payloads (less than 339 bytes).

Table 1 shows a comparison of the wireless technologies. This includes test results reported in Jin-Shyan et al. [8]. The key conclusions of this study are that Bluetooth and ZigBee are suitable for low data rate applications with limited battery power, such as mobile devices and sensor networks, whereas UWB and Wi-Fi should be used for high data rate applications such as audio/video surveillance.

3.2. *Ad-hoc networks*

Ad hoc wireless networking is a technology that allows wireless networking in environments where there is limited or no wired or cellular infrastructure. Typically, ad hoc networks are set up for a limited amount of time and established for a specific service or application. This kind of networking allows for continued network communication whilst still allowing mobility of nodes. There are a number of design requirements that must be considered due to the mobile, non-infrastructure nature of wireless ad hoc networks, which includes self-configuration, multicasting and mobility. Ad hoc networks have had difficulty emerging as generalized standards; however they are becoming increasingly more successful when designed for specific purposes or applications [10].

There are a number of key characteristics that define an ad hoc network. One of the main characteristics is mobility. Because nodes can have random mobility, the network must be able to cater for nodes that enter and exit the network area. Multi-hopping is a term used when a path from a destination node traverses one or more other devices before reaching the source device. Multi-hopping requires routing protocols and technologies to be used, which may either require each node to be a router as well, or some form of designated infrastructure. Self-organization and scalability are also important characteristics, meaning that nodes should be automatically configured when added to a network. Finally, energy conservation is often a characteristic required of ad hoc networks, due to nodes most commonly being completely wireless and therefore relying on batteries for power. There are a number of different classes of ad hoc network [11], the most relevant of which are explained below.

Mesh Networks can be defined as a short-term evolution of multi-hop ad hoc networks [10]. They use a form of infrastructure which defines it as a hybrid ad hoc wireless networking style. As opposed to pure ad hoc networking, mesh networks use fixed routers (mesh routers), which reduces node mobility but also simplifies the communication protocols and configuration required. A wireless "backbone" is formed by the mesh routers. In a typical mesh network, all routing is undertaken by the mesh routers; therefore communication is only maintained in their coverage range. Mesh networks can be used with traditional wireless technologies such as 802.11 (Wi-Fi) and are therefore capable of general purpose use. A major use of mesh networks currently operating is in public internet access, where internet connectivity is provided in a populated area via a wireless backbone.

Opportunistic Networks differ from pure ad hoc networks by removing the notion of communication failure when a node moves out of range of another node/provider. This is a beneficial characteristic when the number of nodes is small. The concept of opportunistic networking is that a node will transmit packages whenever an opportunity arises, and when it

Table 1 – Comparison of specifications between Wi-Fi, UWB, ZigBee and Bluetooth [1].

Specification	Wi-Fi	UWB	Zigbee	Bluetooth
Max signal rate	54 Mb/s	110 Mb/s	250 kb/s	1 Mb/s
Nominal range	100 m	10 m	10–100 m	10 m
Channel bandwidth	22 MHz	500 MHz–7.5 GHz	0.3/0.6 MHz; 2 MHz	1 MHz
Number of cell nodes	2007	>65,000	8	8
Coding efficiency	97.18%	97.94%	76.52%	94.41%
Power consumption (in test)	700 mW	750 mW	80 mW	90 mW

loses connectivity it simply waits until it returns, as opposed to failing. This method of transfer can be used for low priority applications such as email and messaging. Multi-hopping, and therefore routing protocols, are an important characteristic in opportunistic networks, as the node will need to quickly determine if another node it comes in contact with can reach the required destination. An example of general purpose opportunistic networks being developed is in villages in India. There are plans to use mobile relays such as buses and motorcycles that pass through the village to wirelessly exchange data with local nodes [10].

Wireless Sensor Networks (WSNs) are a type of ad hoc network that has evolved for the specific use of monitoring a large number of sensors. In a WSN, the sensors contain integrated computing and networking capabilities and are themselves the nodes. A sink is a wireless routing device that the sensors relay data to, which routes it to the final destination. Sensor nodes can either communicate only with a sink, or multi-hop through other sensor nodes, depending on the category of WSN used. WSNs typically have a large number of sensor nodes (possibly thousands) that are densely located. There are also a number of other significant differences between other ad hoc networks and WSNs [12]: Sensor nodes are normally not mobile, therefore protocols do not need to compensate for this. Sensor nodes are often monitoring common phenomena; therefore data redundancy can be a significant factor. Sink devices can process data in order to remove redundancies and reduce transmissions, at the cost of processing power. And most sensors are small and run on batteries, therefore power consumption is even more important to limit.

Wireless Sensor Networks moved away from general purpose networking and their focus has shifted from node-oriented to data-oriented. WSNs have been successfully applied in many areas, including scientific research (habitat/environment monitoring) and battlefield scenarios (battlefield surveillance, attack detection). It can be seen from the number of successes, as well as current commercial usage, that while general ad hoc networks are becoming more popular, WSN are most prolific in their diversity. According to Conti & Giordano [10], the success of the WSN can be accredited to the specificity in which each network is designed. Because they are tailored to a specific need as opposed to providing general networking communication, they are currently more reliable.

4. Alternative approaches to data management

Use of a file-based approach to storing large amounts of data has many detriments, including separation and duplication of data, data dependence and fixed accessibility. In order to overcome these issues databases are commonly used. A database is a shared collection of logically related data and its description, designed to meet the information needs of an organisation [13]. Because of the large amounts and varying nature of the information recorded by EMS units, a database system is more appropriate than implementing flat-files. Before integrating a database with a system in development, there are a number of factors that must be considered. Four important factors to consider are the data model, architecture, distribution, and languages used.

4.1. *Data models*

A data model determines how data is stored, organized and manipulated in a database system. Data models are used to describe data at different levels of the system. The conceptual level represents how the information is stored and related logically, as opposed to the external level representing how the users see the data, and the physical level representing how the data is physically stored. Hierarchical and network data models are earlier developments of data models than the relational model and are often replaced by relational models. In these models data is represented as a collection of records. Sets are used to define relationships between records. In the hierarchical model, a record may be an owner of any number of sets but a member in only one. In other words, records can have only one parent. The network model, on the other hand, allows for records to be members in any number of sets. Due to the surpassing of these data models with more current systems they are not considered to be viable options for EMS data storage.

Relational data models use tables to represent both data and relationships. Each table has a number of columns that hold data for a specific field. Entries are added to the table as new rows. Every table must have at least one column (or combination of columns) that has a unique entry for every row, to allow for individual identification. Relationships are not explicit in this data model and are defined by relating columns in multiple tables. The relational model is currently well used and has had innumerable implementations with satisfactory degrees of success. Two successful implementations of relational databases used to record environmental data can be found in Peacock et al. and Wösten et al. [14,15]. In Wösten et al. [15], a relational database was developed in order to store various soil properties from different sources in a central area for ease of use. A relational data model was selected in order to provide flexibility in data extraction with the diverse range of data being collected. This successful implementation relates to this project both in aim and in requirements. Peacock et al. [14] documents a database development for the storage of large amounts of data relating to trees in the Amazon forest. A user interface was provided to both view and manually add data to the system. The successful implementation of this system provides further encouragement that a relational database is an appropriate option for the proposed system.

Object-oriented data model approaches to software construction have shown considerable promise for solving some of the classic problems of software development. The underlying concept behind object technology is that all software should be constructed out of standard, reusable components wherever possible [13]. The object-oriented paradigm has been applied to database technology and created models that attempt to bring the database and application worlds together. Object-oriented database models aim to avoid overhead required to convert information stored in the database to the application form. The object-oriented data model uses the concept of classes and objects in a similar fashion to object-oriented programming languages. Object-oriented databases have been used successfully to record data from sensor networks, providing benefits over other data models due to the use of abstract data types. These databases, such as the one developed in Bonnet et al. [16], are designed for large sensor networks and vast expansion. In order to achieve this, the complexity of the database design, development and use is increased.

4.2. *Multi-user database architectures*

There are three common database architectures that allow multi-user system use: teleprocessing, file-server and client-server [13]. *Teleprocessing* is a traditional form of multi-user system where lightweight terminal computers connect to a central computer for database access. This architecture causes a large burden on the central computer because it is required to do all of the processing. Because of this, the other two architectures will be discussed in greater detail. *File-server architecture* implements a database on a computer which is used solely as a repository for raw data. Other computers that have access to the repository run a database system locally, which obtains required data from the database and returns it via a network connection. This architecture can be related to a simple shared network drive or folder. All of the processing is undertaken by the client computer and the file-server database is used only for storage. This has various detriments, the most notable of which are the large amounts of network traffic and the need to have a complete database system running on each client machine.

Client-server architecture is an implementation that separates software components into client processes that require a resource, and servers which provide the resource. Using such modulation allows for clients and servers to be on different machines, providing remote and multi-user functionality. The traditional client-server, known as two-tier, architecture consists of two machines, the client and the server. The database management tasks are split between the two machines; the client machine is responsible for user interface and the main data processing logic, whereas the server is responsible for validation and database access. Splitting the tasks between the two systems means that the server has a greater burden than in the file-server architecture, but less than a teleprocessing architecture. Likewise, client computers still require instances of the database system, however the amount of tasks it is required to undertake means a reduction in resources used. The traditional client-server architecture can be expanded into three-tier, and sometimes even more, architectures. The concept of increasing the tiers is that additional machines are added in order to undertake part of the processing required. For example, in a three-tier architecture, the client machine is responsible only for user interface, a second tier machine is used as an application server which is responsible for main data processing and the server has the same responsibilities as in the two-tier. As expected, as the tiers are increased, the tasks undertaken are distributed in order to reduce burden on each machine, allowing for a larger number of users.

Within the client-server architecture, there are a number of models that differ by the functionality assigned to each component [17]. Three models are the object server, page server and database server. In each model the server is on the same machine as the database and the client can be on the same or a different machine. In an object server architecture, the processing is distributed evenly (approximately) between the client and server. In a page server architecture, most of the database processing is performed by the client. On the other hand, the server is responsible for most of the data processing in a database server architecture, and the client simply passes requests to the server and receives results. The object architecture is more appropriate for object-oriented uses, whereas the database architecture is more commonly used for relational databases [13]. A further factor to consider in database design is distributing the system. In a distributed database system, there are multiple servers each with their own database and network of clients. Clients access data from the servers database, as well as from the other distributed database systems. The entire database is a logical collection of all of the databases in the distributed system. Distributing a system has the benefit of reducing server load and preventing performance bottlenecks or security issues caused by large amounts of data being stored in a single place.

5. Implications for the wireless data management system design

The networking technologies that have been introduced above have different degrees of potential for application with the environmental monitoring system. By looking at the benefits and detriments of the technologies with respect to their application, a theoretical basis for selection of technologies for future improvements to the EMS can be determined.

5.1. *Sensor network implementation*

For the wireless transfer, a number of options are available that can be utilised to implement the objective of automatic data transferal from EMS units. Because each EMS unit would require an interface to be integrated into it, power consumption is a significant factor that must be minimized. This makes ZigBee and Bluetooth preferable options, however both of these have low signal rates compared to Wi-Fi and UWB. Considering that the data being transferred will only be sensor recordings and that immediate transfer is not crucial, the lower transfer rates are considered to be sufficient. The ranges of UWB and Bluetooth would be excessively inconvenient in a practical environment, meaning that units would have to be set up very close to each other or additional infrastructure would need to be established, resulting in greater expense and effort. The number of nodes possible using the networking technology is also a factor to consider: ZigBee and Bluetooth only allow for eight nodes, which is minimal compared to the other two technologies. At this stage of design, the deployment of eight nodes per farm is considered satisfactory. For these reasons, ZigBee has been identified as the first choice of networking technology to establish data transferal from EMS units. It has the lowest power consumption of the technologies discussed while maintaining an effective transferal range. It should also be noted that Wi-Fi is a very popular networking technology and may be used for this reason; software development and hardware interfaces can easily be arranged.

The ad hoc network styles relate to the topology of the network devices and will affect the requirements of routing capabilities and infrastructure needs. Each of the styles have benefits and drawbacks in relation to implementation with the environmental monitoring system. Two important factors to consider when selecting a network style are cost and ease of implementation/deployment. The most ideal configuration would require no additional infrastructure deployed on a single farm other than one or more EMS units and a single computer with wireless interface to receive the readings. Although it will depend on the layout of each individual farm, the ability for multi-hopping amongst the EMS units is considered to be beneficial, as it means that the main interface does not need to be located centrally to all sheds on a farm. Wireless sensor networking is very appropriate for large and individual networks, however the increased complexity and currently limited deployment of the environmental monitoring system means that it is not considered the most advantageous option. Additionally, because each EMS unit would have to be configured before being deployed on a farm, the opportunistic network style does not hold any additional benefit. This means that at this stage of development, the mesh network style would provide the best foundation for designing communication between devices on a farm. Using this foundation, additions can be made, such as multi-hop capabilities, as required.

5.2. *Data collaboration and storage*

Data collaboration and storage identified options regarding the access and storage of recorded data. This comes into play when developing a central server for storing and viewing data from all/multiple farms. More specifically, the data models will determine how the data is stored, organized and manipulated in a database system, while the database architecture will determine how users will access the data. Of the data models presented, the relational and object-oriented are two obvious choices. As shown earlier, the object-oriented data model has been successfully used to record large amounts of data recorded from sensor networks, however the increased complexity would cause detrimental development overhead. The relational data model is widely used and numerous successful databases have been developed using it to record sensor data. The combination of these observations concludes that the relational and object-oriented database models are both satisfactory, the latter being preferred, and the choice would rely on the effectiveness of the available design and development team. The nature of the desired setup for establishing networking capabilities amongst the environmental monitoring system means that the client-server architecture is optimal. This allows for the server maintaining the recorded data can be accessed by multiple clients, being both users to view data and farm computers to contribute data. At this stage of design, a distributed database system is not necessary, however with expansion, distribution can easily be included if required.

6. Materials and methods

In order to design and implement the project objectives, software engineering processes and methodologies have been utilised. Due to the limited resources, some simplifications have been made. The entire software engineering process undertaken is not documented here; however, major design decisions and notable milestones are included. The processes and templates used during the software engineering cycle have been simplified from the text Bruegge and Dutoit, & Sommerville [18,19], which contain more detailed descriptions.

6.1. *Requirements elicitation and analysis*

Requirements elicitation and analysis is the first step of software design and consists of determining and modelling system functionality. System requirements are provided in natural language and refined to with respect to the purpose of the system. *The purpose of the system* is to automate the collection and storage of air quality data from EMS units. The system consists of multiple subsystem components that allow for measurements from a number of EMS units to be transferred to a single location, which can be remotely accessed for visualisation and analysis. Overall, the system reduces the required effort in obtaining and maintaining recorded data. *The scope of the system* extends from the networking and transferal of data from the EMS to the storage and accessibility of that data on an online server. This includes a number of subsystem components, but most specifically it requires a web based user interface, networking and transferal methods and storage methods. Due to the large scope of the system, and considering that the project was undertaken by a single student with limited resources, a partial implementation was undertaken. The final design and implementation considered only the online storage and receive subsystem, which will be explained in the next sections. The overall objective of the system was to dramatically reduce the effort required to collect, store and maintain data recorded by EMS units. A prototype system was developed to test the effectiveness of the approach.

In order to eliminate or reduce shortcomings of the current system processes, an online data repository and a number of automations were introduced. In its entirety, the data logging was automated to the extent that multiple EMS units can run on a single farm, and the data was made available to authorised users online. In order to achieve this, the EMS units required integrated networking functionality that allowed measurements to be transmitted to a computer for uploading to an internet server. Fully automating the data collection and collaboration process ensured data consistency and availability, and prevented any possible human errors. In addition, a storage server can easily implement data redundancy techniques to prevent data loss. Therefore, the system consists of an online server that receives data via the internet from computers located on a number of farms. The computers local to the farms which transfer data to the online server

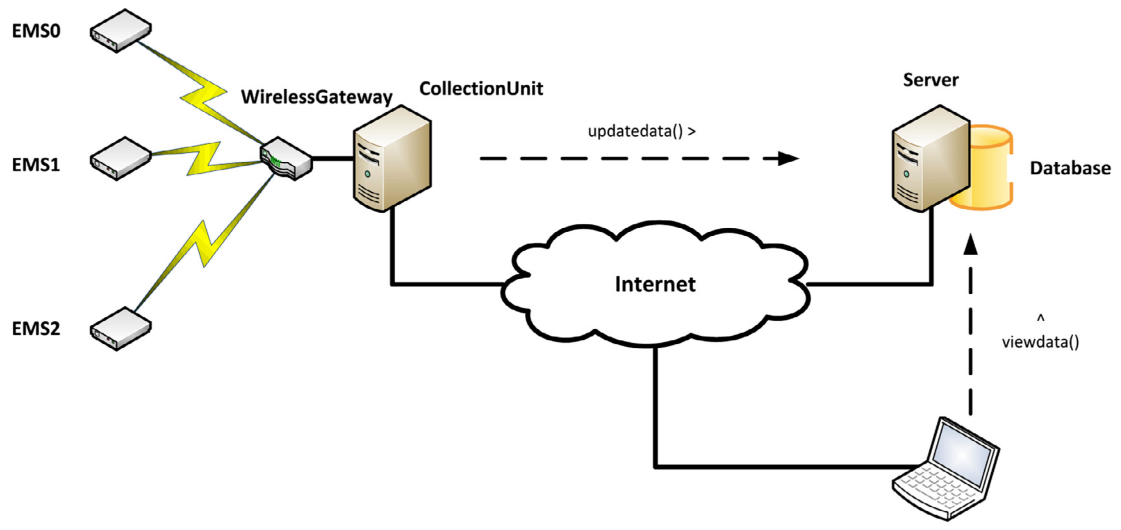


Fig. 2 – Basic overview diagram of high-level proposed system.

receive the data from one or more EMS units via wireless communication. Finally, the server provides an online interface for authorised users to view and analyse recorded data. Additionally, data redundancy and fault tolerance, such as occurs in the loss of a network connection, are also handled. A basic overview of the system is shown in Fig. 2.

Functional Requirements: Below are a number of functional requirements for the system. The functional requirements explain what functionality the system is required to have, which are the interactions and results of actors using the system. Using these, the functionality of the system was determined, leading to design and development.

EMS Data Transfer

- Each farm has a local computer (henceforth called the Collection Unit) that receives EMS data.
- The Collection Unit has a wireless interface that is used by the EMS unit(s) to transfer data.
- A farm has to easily be able to add more EMS units, which after being configured will use the setup interface without modification.
- If an EMS unit is not within range of a configured network, it will record data until the local storage is full, or the network becomes available.
- When the network is available an EMS unit periodically send data to the Collection Unit.
- The Collection Unit maintains records of all data received from the local EMS units.
- When configured to do so, the Collection Unit periodically send new data to the online Server Computer.

Server Functionality

- The Server Computer maintains a database of recorded data from multiple farms.
- The server is updateable only via remote Collection Units, or via manual input by a user.
- A web interface is provided to users with the ability to remotely view and analyse data.
- The web interface provides, at a minimum, the functionality of the current BASE-Q system.

Table 2 outlines the *non-functional requirements* determined for the system. Non-functional requirements affect design decisions such as architectural styles and implementation languages. They do not alter the functionality of the system. Using the functional requirements defined above, a system model was defined.

6.2. *System design*

System design is the final step before implementation, where architectural design decisions such as hardware/software mapping, data management and access control are made. The *purpose of the system*, as stated in the previous section, is to provide a user-friendly and effective way to collect and store data obtained from EMS units. The Collaboration System has focused on functionality of the farm and server machines. The system was developed for the purposed of undertaking research. Therefore the complete functionality was not implemented. *Design goals* were obtained from the non-functional requirements and were used to make important decisions throughout the system design process. The main design goals for the Collaboration System, including reasoning for each, are listed below.

- Low operating and development cost – This is required due to the limited resources involved in the project. This design goal leads to the selection of free or open-source components.
- Scalability – The number of farm machines supported by the system must easily be increased without degradation to system performance.
- High usability – The interface must be easy to use in order to accommodate users unfamiliar with the system.
- Rapid delivery time – The system is being developed for research purposes and delivery time is more important than full functionality.
- Security – As the system will be utilised online, security measures must be taken in order to prevent malicious and unauthorised access or modification.

Table 2 – Non-functional requirements for the system.

Usability	Users have to be able to understand the layout and options of the user interface
	The user interface shall be comparable to that of the current systems
Reliability	The Server Computer shall implement data redundancy procedures in order to prevent loss of data and increase availability
Performance	The performance must be better, if not similar, to that of the current system
Maintainability	The storage capacity of the Server Computer must be easily increased if required
Adaptability	The system must have the ability to easily integrate future additions such as off-the-shelf analysis tools, etc
Security	Only authorised users are able to access or modify the data on the server machine
	Security should be used to ensure that only registered farm machines are used to update the server machine

6.3. *Proposed software architecture*

There is currently no software system in use to adapt or modify in order to meet the requirements. A new system is being designed and developed, using previously undertaken research and the authors experience and knowledge in order to make design decisions. The main task in system design is decomposing the whole system into a number of manageable subsystems. The subsystems are determined by grouping classes together, ideally in a way that maximises cohesion while minimising coupling. This process allows for the identified subsystems to be developed independently of one another, improving maintainability through modularity. Considering the design goals, the system is decomposed into a number of subsystems. Additionally, these systems have been arranged into a layered architecture. Fig. 3 shows the subsystem decomposition using a UML component diagram.

The system has a number of persistent data objects that must be recorded. The main persistent objects are the recorded data that must be saved on both server and farm machines, the remote user login details, and the general settings required by various subsystems. Different storage strategies are used for the different objects. The general settings are stored in flat files using the configuration files supported by the selected framework. The user login details are managed by the chosen pre-developed system, and implementation details need not be considered. When determining the storage style of recorded data, it is important to consider applications provided in available literature. Due to the availability of successful examples and appropriateness of the relational data model, it is used to store the recorded EMS data. MySQL is used due to the free availability and the ease of integration with the chosen framework. A MySQL JDBC driver provides an interface between the database and the application. Microsoft's SQL Server has a multitude of tools that provide powerful functionality that would be beneficial for this system; however it has not been used due to the commercial nature of the product and the large amount of resources it would require.

Hardware/software mapping: The system is inherently distributed as users remotely access server data from their own machines, and the collaboration of data occurs in stages. Therefore there are three nodes identified, and a distinctive client–server style is adopted. Fig. 4 shows the hardware/software mapping for the system. Using a client-server architectural style assists in the scalability goal by making it easy to add farm machines. A specific architectural style for interaction with the database system must be considered. Due to the nature of the system, as stated earlier, the client–server architectural scheme is most appropriate to be used for the development of this system.

The specific architectural model used is the database model, meaning that the server undertakes most data processing. Again, this has been chosen in order to reduce the resources required by the client machines. Due to the current experimental use of the system, overloading the server by having too many client machines is not considered an issue. For the realization of the system on each node, there are a number of viable frameworks available to choose from. In order to allow for effective web interfacing as well as provide system functionality, a web application framework is used. Additionally, considering the vast use of the internet in today's society, providing a web interface increases the usability of the system. Three popular web application frameworks are PHP, ASP.NET and JavaServer Pages (JSP).

For the Collaboration System, JSP is used to provide web based services founded on a Java framework. This allows for effective web interfaces and services to be developed in order to implement the client-server nature of the system. In order to host the application, a server system is required. A popular web server for Java applications is Apache Tomcat (http://tomcat.apache.org/), however because this project includes a web service that is used by an application client, an application server is required. Oracle Glassfish Server (https://glassfish.java.net/) is an open-source application server that supports all the required functionality, some of which is achieved using a version of Apache Tomcat, therefore it has been used on the server machine to host the application. Communication between all nodes uses HTTP or XML. The framework and protocols chosen are free to use and readily available, which meets the criteria set by the rapid delivery time and low cost design goals.

Access control and security: The system allows for multiple users to remotely view or modify data. As specified in the design goals, although the data is not specifically confidential or sensitive, security is still be used to monitor the access and activities. This means that access control must be included on the server machine in order to prevent unauthorised

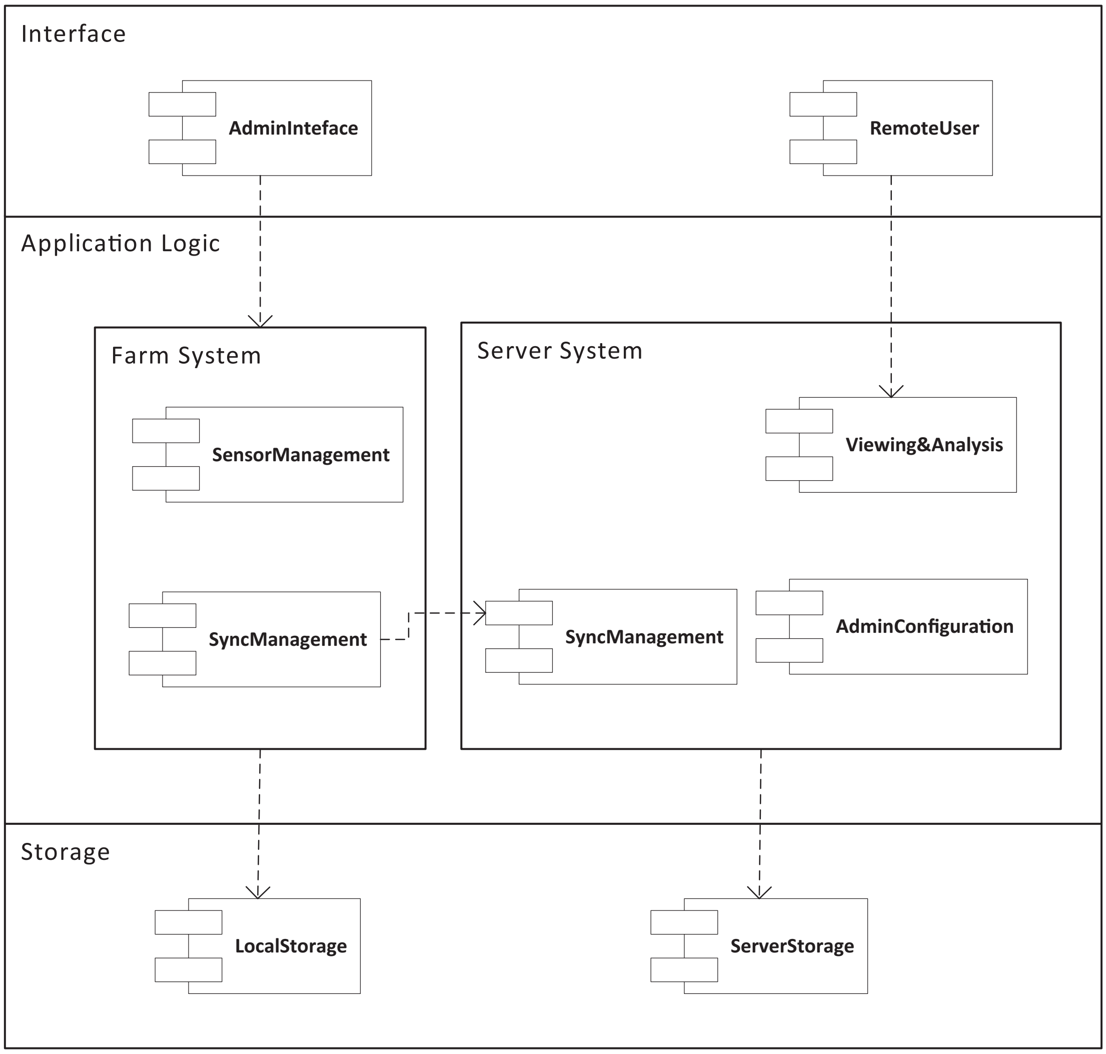


Fig. 3 – Subsystem decomposition, shown using a UML component diagram.

actions. Access control is implemented primarily by the pre-developed user management system chosen. *Global control flow* is event-driven, allowing for simpler system structure, which therefore reduces the development time required. Thread-driven control is used by the web-hosting system chosen, however implementation does not need to be considered in the development of the server functionality. Simple threads are utilised in the *FarmSystem* subsystem in order to "simultaneously" record readings and update them.

7. Prototype development

This section documents the process of creating a system implementation from the design. There are a large number of integrated development environments (IDEs) for use in developing applications in the provided framework. NetBeans is a free, open source, multi-platform IDE for developing desktop, mobile and web applications in Java, PHP, C and more. The Collaboration System was developed on a computer running Microsoft Windows 7 using the NetBeans IDE 7.2. Oracle GlassFish Server 3.1.2.2 was setup and used on the local machine, which was integrated with NetBeans in order to automate testing and deployment. MySQL 5.5.28 for Microsoft Windows was used to provide database functionality to both the farm and server systems. After installation on the development machine, the databases to be used by the systems were created and authorization options specified. This was achieved via the MySQL command line client. The FarmSystem is responsible for maintaining a database of readings that are used for redundancy while updating the server and not for viewing or analysing data. Because of this, a single relational table is be used to maintain readings at a given farm. The FarmSystem is given access to a database and use the configured farm identification to create a new relational table, or append to the relational table if it already exists. The relational table looks as follows: *FarmId (ReadingId, Timestamp, ShedId, SensorType, Reading, IsSynched)*. It uses the following definitions:

- *FarmId* – Table name. A new relational table will be created by the application for each farm it is configured to record on. A unique farm identification string will be used as the name of the relational table.

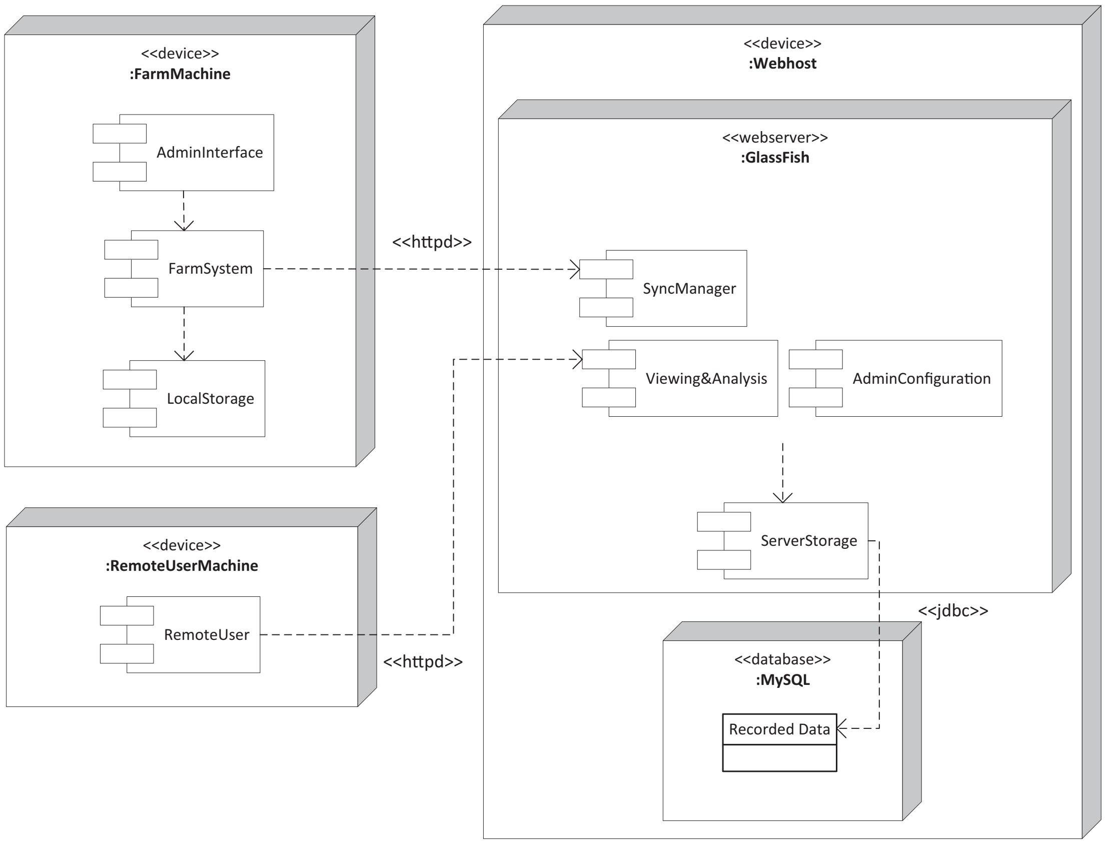


Fig. 4 – Hardware/software mapping of the system, shown using a UML deployment diagram.

- *ReadingId* – Integer column. Automatically incremented unique identification number for each reading. This value will be used in the synchronising process. It is also used as the primary key of the relational table.
- *Timestamp* – Date-time column. Timestamp indicating when the reading was taken.
- *ShedId* – Integer column. Unique identification number of shed on a particular farm.
- *SensorType* – String column. Type of sensor the reading was taken by (e.g. CO_2, Temperature, etc).
- *Reading* – Decimal column. The reading taken by the sensor. This will have varying magnitudes for different sensor types.
- *IsSynched* – Boolean column. Flag used to indicate whether the entry has been synchronised with the server. This value will be used in the synchronising process.

Unlike the FarmSystem, the ServerSystem requires relational tables for maintaining the details of sheds and farms as well as the readings. This database consist of the following tables: *Farms (Id, Name, Owner, Contact, Address)*, *Sheds (Id, FarmId, Description, Location)* and *Readings (Id, Timestamp, ShedId, FarmId, SensorType, Reading)*. These tables allow the addition, deletion and modification of farms and sheds, and their respective details. The relations between these tables (foreign keys) are easily identified: *Readings.ShedId* → *Sheds.Id*, *Readings.FarmId* → *Farms.Id* and *Sheds.FarmId* → *Farms.Id*.

The creation of the databases required the use of basic MySQL commands. During implementation, it became apparent that upon further development the database would be required to maintain further persistent data such as authorisation details for remote users and service clients. These additions can easily be made via the creation of more tables within the database as required.

7.1. *Overview of implemented functionality*

The developed system prototype does not provide complete functionality as defined in the requirements; however, it provides the main functionality and demonstrates how the completed product works. This section provides an overview of what functionality the prototype demonstrates for each project, followed by what functionality it is lacking and how it has to be extended for a production environment.

Collaboration System Server: The web pages provided by the web application are not styled within the prototype and are solely for the demonstration of functionality. Within the administration subdirectory the user is provided the options to manage farms or configure the database details. These actions are undertaken by the JSP pages with the aid from the java classes. This process is very similar for the *Sheds* and *Readings* management UI. Therefore only the farm management has been implemented for the prototype, as it will provide sufficient demonstration. For testing, entries can be added to the other database tables via the MySQL client if required.

Table 3 – Command line arguments.

−c	Configuration mode – Allows the user to configure details such as database connection, farm ID and sheds/ems relationships. Multiple interfaces can be produced for the system, such as console and GUI, however neither have been implemented in the prototype. Currently, when this command-line argument is provided, a hard-coded configuration is set
−r −tXX	Read mode – The application will record readings from configured sensors for XX seconds, then terminate. Currently, as the application does not actually interact with any EMS units, the read function adds random entries to the database until the time expires
−w	Write mode – Update all unsynchronised records. The application will synchronise readings with the server until they have all been synchronised, then terminate
−r −w −tXX	Read-Write mode. The application will both read and write. Reading will stop after XX seconds, and the application will terminate when all records have been synchronised

The *SynchManager* java class provides a web service for updating the server's database. The service methods arguments are all lists of values, each of which is a column within the *Readings* database table. These arguments are cast into *ReadingsEntry* objects and added to the server database. The web service returns a value to the client indicating the success or failure of the synchronisation. During implementation it became apparent that there were two distinct ways to add a collection of entries to a database table. The first method creates a loop that executes an INSERT database command for each entry until they were all added. The second method executes a single large database command to insert all entries in a single attempt. Both methods were implemented for analysis.

Collaboration System Farm: The main functionality of this system is the use of the web service provided by the server system to update entries within the local database. The command line arguments of the application are shown in Table 3. The developed prototype can be executed in the read-write mode described above and it will successfully synchronise data with the configured server system. This will continue until execution is manually terminated or the time provided expires.

A working prototype has been developed for initial testing and analysis. However, if the system is to be deployed and released for further testing and approval, there is still a considerable amount of functionality that needs to be added, including:

- User management for accessing online resources. This security requirement encompasses both the web page UI and the web service. In order to prevent unauthorized access, modification or deletion of data a user management system must be developed to manage users who are able to use the website. This would require the inclusion of a username/password authorization scheme, which could be implemented via an external library as planned. During implementation however, it was determined that user access could be managed relatively easily with the addition of another database table and the use of JSP functionality. Security should also be considered with respect to the web service that the server provides. Although it is unlikely that an external party would attempt to use the service, it should be prevented, regardless of their intent. There are methods available for this, however further research would be required to implement it.
- Model-View-Control (MVC) design pattern should be adhered to throughout the entire server project. The MVC pattern separates the representation of information from the user's interaction with it (Reenskaug & Coplien, 2009), which allows for greater maintainability of program code. The prototype was designed with production time in highest regard and therefore the MVC was not fully adhered to. This would need to be changed if the application was to continue development, which would not require considerable work.
- Develop web interfaces for configuring Shed and Readings entries on the server machine. These interfaces would be very similar to the Farm management interface already implemented. Additionally, the web pages' content should be completed to include efficient navigation, etc.
- Develop GUI and/or console interface for the farm system in order to allow users to configure it.
- Develop and integrate more viewing and analysis interfaces for displaying recorded data.
- Overall improved error handling and gracious failure. This is required especially to support users unfamiliar with the system.
- Comment code and supply complete javadoc for all classes. Commenting is an important aspect of programming and if the systems are to continue development it must be included.

The research design and methodology stage of the project has produced a functioning prototype as proof of concept which can be used for testing.

8. Deployment and system testing

This section demonstrates how the prototypes were deployed to environments for testing. The procedures undertaken for this project were done using a single local machine, in order to reduce the amount of time required. Although the process is very similar to how a release version could be deployed to real hosting environments, actual deployment would have some minor differences which will be explained.

In order to deploy the *Collaboration System_Server*, a host machine is required that is running both GlassFish 3+ and MySQL. An easy method of deploying to a server host is through the NetBeans IDE, which allows for remote deployment to a configured server. Therefore the first step is to configure the server in the NetBeans IDE. This requires details such as the URL of the GlassFish server and administrative authorisation details in order to grant permissions to make updates. The application can then be configured to deploy to the selected server. The configurations required for this are simple GUI interfaces, and the details will not be discussed here. After successful configuration, the project can be deployed at any time from within the NetBeans IDE. In order to access the server, the URL of the server followed by the project name can be typed into a browser. This will display the default (index.html) page of the application within the browser.

The *CollaborationSystem_Farm* is a java application. When the project is built, the appropriate files are produced in the required format. In order to distribute the application to another host, the directory named dist in the project folder can be compressed and copied. Once extracted, the application can be run by executing the .jar file that is named after the project. Because the prototype requires command-line arguments, the application will be executed through the console by running the following: *>java -jar fileName.jar arg1 arg2*.

A number of tests have been undertaken with the developed prototype in order to determine if the implementation meets the requirements of the project. One of the main aims of the software project is to reliably reduce the amount of effort required to collect and collate recorded sensor data. A factor in reaching this goal requires determining quantitative performance values. The following tests used java logging and time objects to obtain such measurements. Below is a typical example of how the time required for a process can be determined and displayed.

```
long start = System.currentTimeMillis ();
// Do operation (s)
long finish = System.currentTimeMillis ();
System.out.println (‘‘Time taken: ” +
Long.toString (finish-start) + ‘‘ milliseconds”);
```

These values can then be used for analysis of the prototype. Following are descriptions of the tests undertaken and their motivation.

8.1. *Test 1: Database capacity and entry retrieval*

This test aims to determine if the performance of the application reduces significantly as the number of entries within a database table increase. The EMS units log data at a considerably fast rate and therefore, after a period of recording, the number of data entries in a FarmSystem database can become very large. In addition to this, a farm can be logging from multiple EMS units Therefore it is evident that performance must be maintained as database entries increase. This test determines if the number of entries within a database has a significant effect on the extraction rate of entries.

Method: The CollaborationSystem_Farm project was run a number of times with an increasing number of database entries. A number of entries that are required to be synchronised with the server were added to the farm system database. The application was executed (in synchronise mode), and the times taken for the fixed number of entries to be extracted from the database was recorded. This was repeated with larger numbers of entries in the database. The values contained in the entries of the farm system database are not considered in this test. In order to populate the database with the required number of entries, the CollaborationSystem_Farm in read mode has been used. A loop was written in the SensorManager class that adds the defined number of entries to the table. Table 4 shows the test repetitions.

8.2. *Test 2: Database capacity and entry insertion*

The second test is also concerned with the performance of the application as the number of entries within a database table increase. This test aims to determine if the number of entries within the server system database affects the performance with respect to adding new entries.

Method: This test is similar to Test 1, however, as opposed to timing the retrieval of entries from the local database, the time taken to receive a response from the servers synchronisation web method is recorded. Although this value includes the overhead produced by the transfer protocol as well as the server system application logic, the values remain valid if the other test parameters are kept consistent. The test was executed with both systems on the local machine, meaning that external network traffic did not interfere with results. This test can be performed by clearing all entries from the server database and filling the farm database with a large number of entries. The farm system is then run in synchronise mode and the time taken for each update is recorded. The time taken as the server database increases in size is incrementally recorded.

8.3. *Test 3: Synchronisation entry argument count*

Within the developed farm system, there is a hard-coded variable that defines the number of entries per iteration that are extracted from the database and sent to the server for updating. If this value is small, there is more overhead required for iterations and transfer protocols, but the small data sizes makes it more reliable and easier to monitor. In contrast, if this value is large, less overhead is required, but reliability is also reduced. This test aims to determine the performance trends relating to this variable.

Method: A consistent set of updates have been made from the farm system to the server system with varying entry collection sizes. This was achieved by clearing and repopulating both farm and server databases before running each test. The total time that the farm system took to extract the entries, sent them to the server, and update the local database, was recorded. Table 5 shows the parameters for Test 3.

Table 4 – Parameters for test 1.

Subtest	Entries in farm database	Number of extracts	Entries/extract
1	100	10	10
2	550	10	10
3	1000	10	10
4	5500	10	10
5	10,000	10	10
6	1,000,000	10	10

Table 5 – Parameters for test 3.

Subtest	Entries in farm database	Number of extracts	Entries/extract
1	0	320	1
2	0	320	5
3	0	320	10
4	0	320	20
5	0	320	40
6	0	320	80
7	0	320	160

8.4. *Test 4: Increased number of clients*

The purpose of this test was to determine the performance effects of communicating to the server application from multiple farm systems (clients). A requirement of the software project was the ability to synchronise data from a number of farms simultaneously, therefore performance must not be hindered as a result of multiple client connections.

Method: This test was undertaken on a single machine; however, it would have been ideal to undertake this test with clients running on separate machines. This is because performance may be reduced due to multiple applications waiting for database requests on the same machine. Due to limited resources, this has not been possible; however the results of testing on a single machine still provide beneficial information. In order to undertake this test, a varying number of farm systems were provided with a database of entries and simultaneously updated the server. The amount of time taken for each update on each farm system was recorded and an average was calculated. For initial testing, a range from one to five clients was tested on the single server.

8.5. *Test 5: Synchronisation speed*

In order to determine the benefit of the system, the rate at which it can update data is required. This test aimed to measure this rate.

Method: This was a simple test which times how long it takes to synchronise a number of recordings. It was undertaken from a single machine, using a single farm system.

8.6. *Further test options*

The tests described in this section have provided information regarding the performance of the system, which can be used to determine whether the system should be further pursued. However, before significant further progress is made there are a number of tests that should be undertaken. These tests relate mainly to the performance of the applications when they are on separate physical machines. Tests 3 and 4 described above are both tests that should be undertaken in this manner, as transfer protocols will have the most effect on them. It is important to test the prototypes before significant further development in order to prevent major design modifications.

9. Results and discussion

The test procedures described were undertaken on the same machine that was used to develop the prototype. The machine is running Windows 7 Professional 32-bit and has an Intel Core i5-2500 CPU.

The first test was undertaken to determine if the performance of the farm application would degrade as the number of sensor readings in the local database increased. Fig. 5 shows a graph displaying the average time taken to extract 10 database entries with respect to the number of entries within the database. There is a clear linear relationship between the number of entries within the database and the time required to extract entries. In order to remedy this issue, methods such as deleting synchronised data or archiving should be considered. Because of the current nature of the application, this issue needs to be addressed before deployment for operation.

Test 2 involved determining if an issue such as was found in Test 1 would also arise in relation to the insertion of data into the databases. That is, the test was undertaken to determine if the data insertion performance would degrade significantly as the number of entries within the database increased. Results from the tests were concluded to have no significant trends or relations; the time required to insert entries did not vary consistently with the increase of entries within the database. This has only been concluded with a table of up to 10,000 entries however, which is a relatively small table in database terms. As with the previous test, archiving could be considered to prevent the database from growing too large and causing any potential performance decreases.

The third test involved the modification of an internal variable to determine the relation between performance and the number of database entries extracted and sent to the server at one time. The values tested ranged between 1 and 160, while all other parameters remained consistent for all tests. Figs. 6 and 7 show the results obtained from testing the prototype. From the two graphs it can be seen that the time taken for each upload increases linearly with the number of entries per update, however the total time taken for the update decays exponentially. The choice to continue increasing the number of entries per update beyond that which was tested is not recommended. The test results in this case may be misleading as the test was undertaken on a single machine; the size of the payload may have more effect when it needs to traverse large distances such as when the server is hosted by an ISP. The results conclude that currently a larger number of entries will provide performance benefits, however these conclusions must be re-evaluated if remote deployment is considered.

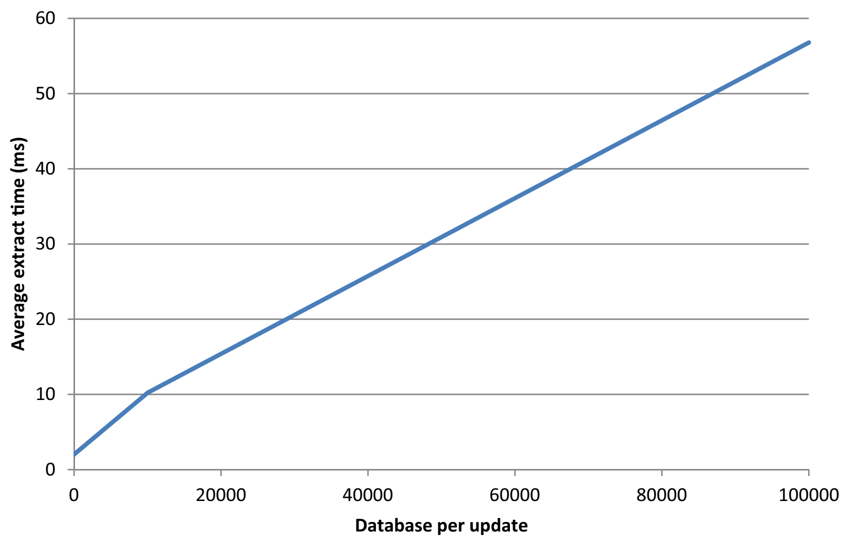


Fig. 5 – Graph displaying entry extraction time with respect to the number of entries in the database.

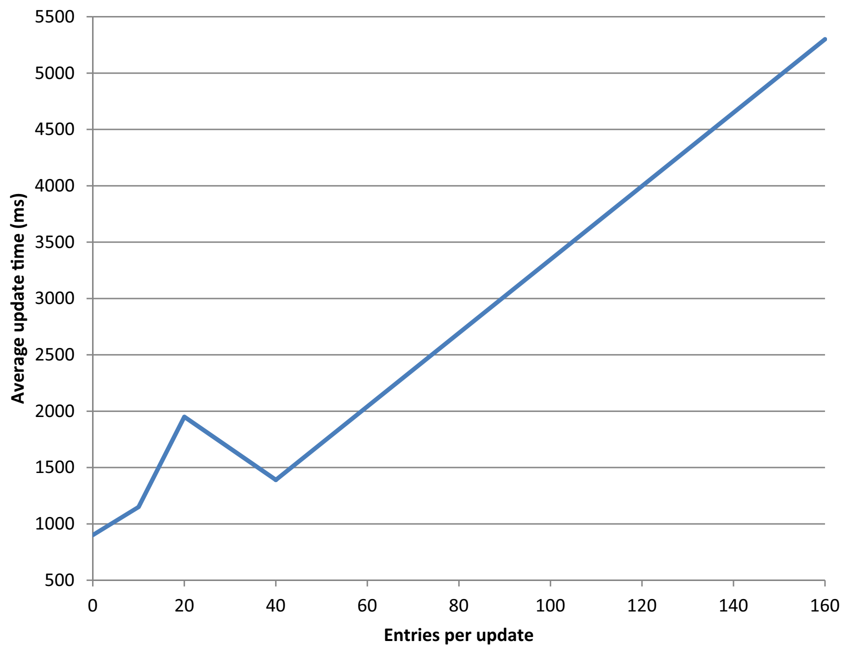


Fig. 6 – Graph showing the average time taken for a single update with respect to database entries per update.

The fourth test was undertaken to ensure the server synchronisation performance did not degrade with the execution of multiple clients simultaneously. Only five clients were tested simultaneously on the single machine, however as the project is in its infancy, a large number of clients is not expected. After viewing the results it was apparent that no significant performance change occurred with the addition of more clients. As with the previous test however, this should also be re-evaluated using actual remote calls when possible.

The final test determined that the system can synchronize database entries at an average rate of approximately 420 entries/min. This was determined using the standard configuration used in the other tests. This value is not an exceptionally accurate representation of the time it would take to update the server because there would traditionally be overhead from the long distance package transfer. However, at this point the result is reliable enough to use as an indication of the potential benefits of the system.

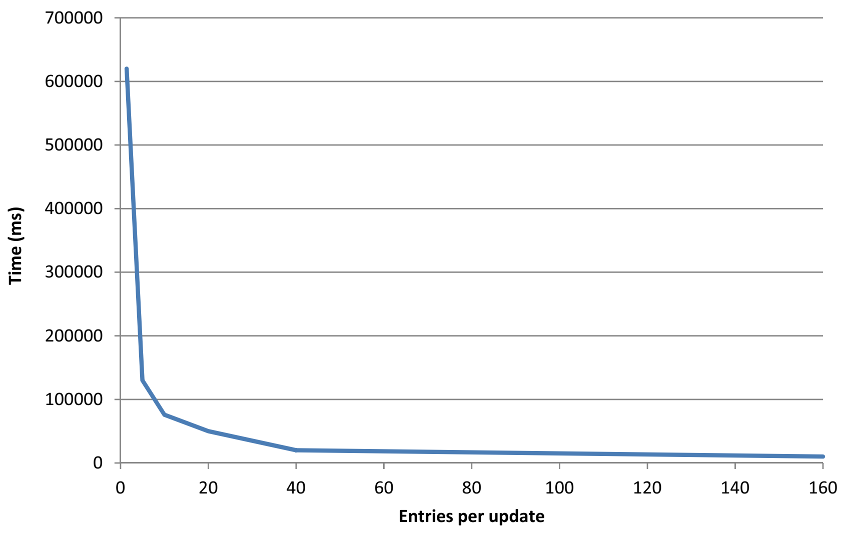


Fig. 7 – Graph showing the total time taken to synchronise with respect to database entries per update.

10. Conclusions

The developed prototype has demonstrated sufficient requirements to be considered a success within the scope of the software design and the prototype is therefore a successful proof of concept for the design of a software system that implements networking technologies to reduce the labour requirements of the current EMS system. The design and implementation can be used as a foundation for the further development of automating the Environmental Monitoring System. Increased automation of the EMS system will improve usability and reliability, which has the potential to increase its overall use. Additionally, the steps taken in this project can provide lessons for similar networking technology ventures. The collection of information into a single location is a common task and automation has the potential to reduce labour requirements. The design process for the software system required many decisions to be made based on experience and knowledge. This is one of the main reasons why there are many different ways to implement a required functionality with software, each implementation having certain benefits and drawbacks. It should be noted that there are a myriad of ways to implement a software application, however if the requirements of the project are fully met they are all equally valid.

As concluded by this project, networking technologies are a feasible option for reducing labour requirements and increasing usability and reliability, through the use of networking, programming and database technologies. In order to achieve these benefits for the EMS system, further research is required. Firstly, this project should be revised and the prototype completed to produce a fully-functioning system. The system must also be integrated with the physical EMS units, which was not possible in this project due to resource limitations, therefore the low-level integration of wireless communication with EMS units, and the interfacing with the farm system must be pursued. This would require further research and design, specifically relating to wireless technologies, protocols and interfaces.

Another area that requiring further work for an operational system is the ability to maintain the system with a low level of onsite support. This also includes the recovery of major power failure events, hardware failures and data corruption. One of the biggest challenges will be to narrow down the type of failure remotely. This will also include safe recovery options and accounting for intermittent internet access problems. These important issues are tightly integrated with the operationalisation of the system and as such left for further study.

Disclaimer

Mention of a commercial product is solely for the purpose of providing specific information and should not be construed as a product endorsement by the authors or the institutions with which the authors are affiliated.

Acknowledgements

The authors would like to thank a number of NCEA staff for providing advice and assistance during the design processes. The authors also gratefully acknowledge the assistance of the European Community via the Collaborative Project EU-PLF KBBE Grant Agreement No. 311825 under the Seventh Framework Programme.

REFERENCES

[1] Wathes CM, Phillips VR, Holden MR, Sneath RW, Short JL, White RPP, et al. Emissions of aerial pollutants in livestock buildings in northern Europe: overview of a multinational project. J Agric Eng Res 1998;70(1):3–9.
[2] Banhazi TM, Seedorf J, Rutley DL, Pitchford WS. Identification of risk factors for sub-optimal housing conditions in Australian piggeries: Part 1. Study justification and design. J Agric Saf Health 2008;14(1):5–20. Epub 2008/04/02.

[3] Banhazi T. User-friendly air quality monitoring system. Appl Eng Agric 2009;25(2):281–90.

[4] Clements MS, Watt AC, Debono AP, Aziz SM, Banhazi TM. A low cost portable environmental monitoring system for livestock buildings. In: Banhazi T, Saunders C, Hegarty R, editors. SEAg 2011: diverse challenges, innovative solutions. Gold Coast, Australia: Engineers Australia; 2011. p. 141–58.

[5] Parkin BJ, Saha PP, Nguyen HT, Göl Ö, Nafalski A, Banhazi TM. A cost-effective monitoring module for the assessment of environmental quality in livestock buildings. In: Banhazi T, Saunders C, editors. The Bi-annual Conference of the Australian Society of Engineering in Agriculture (SEAg 2007) - Challenge Today, Technology Tomorrow. Adelaide, Australia: Australian Society of Engineering in Agriculture; 2007. p. 212–20.

[6] Zhang Q, Yang X-l, Zhou Y-m, Wang L-r, Guo X-s. A wireless solution for greenhouse monitoring and control system based on ZigBee technology. J Zhejiang Univ Sci A 2007;8 (10):1584–7.

[7] Porcino D, Hirt W. Ultra-wideband radio technology: potential and challenges ahead. IEEE Commun Mag 2003;41(7):66–74.

[8] Jin-Shyan L, Yu-Wei S, Chung-Chou S, editors. A comparative study of wireless protocols: bluetooth, UWB, ZigBee, and Wi-Fi. Industrial Electronics Society; 2007. IECON 2007 33rd Annual Conference of the IEEE; 2007 5–8 Nov.

[9] Lu G, Krishnamachari B, Raghavendra CS, editors. Performance evaluation of the IEEE 802.15.4 MAC for low-rate low-power wireless networks. Performance, Computing, and Communications, 2004 IEEE International Conference on; 2004.

[10] Conti M, Giordano S. Multihop ad hoc networking: the reality. IEEE Commun Mag 2007;45(4):88–95.

[11] Danielyan E. IEEE 802.11. Internet Protoc. J. 2002;5(1).

[12] Sohraby K, Minoli D, Znati T. Wireless sensor networks: technology, protocols, and applications. John Wiley & Sons; 2007.

[13] Connolly TM, Begg CE. Database systems: a practical approach to design, implementation and management: international edition. 5th ed. USA: Pearson Higher Ed; 2009.

[14] Peacock J, Baker TR, Lewis SL, Lopez-Gonzalez G, Phillips OL. The RAINFOR database: monitoring forest biomass and dynamics. J Veg Sci 2007;18(4):535–42.

[15] Wösten JHM, Lilly A, Nemes A, Le Bas C. Development and use of a database of hydraulic properties of European soils. Geoderma 1999;90(3–4):169–85.

[16] Bonnet P, Gehrke J, Seshadri P. Towards sensor database systems. In: Tan K-L, Franklin M, Lui J-S, editors. Mobile data management. Berlin Heidelberg: Springer; 2001. p. 3–14.

[17] Loomis M. Client-server architecture. J Object-Oriented Program 1992.

[18] Bruegge B, Dutoit AH. Object-oriented software engineering using UML, patterns, and Java. 3rd ed. Prentice Hall; 2010.

[19] Sommerville I. Software engineering. Addison-Wesley 2007.