A MIDDLEWARE FRAMEWORK FOR WIRELESS SENSOR NETWORK

By

Huma Javed

A thesis submitted in partial fulfilment of the requirements of the
Liverpool John Moores University
for the degree of Doctor of Philosophy

Supervised by

Prof. Madjid Merabti and Dr. Bob Askwith

School of Computing and Mathematical Sciences
Liverpool John Moores University

This thesis is dedicated to my mother
&
my daughter Leena Javed

ABSTRACT

Advances in wireless and Micro-Electro-Mechanical Systems (MEMS) technology has given birth to a new technology field sensor networks. These new technologies along with pervasive computing have made the dream of a smart environment come true. Sensors being small and capable of sensing, processing and communicating data has opened a whole new era of applications from medicine to military and from indoors to outdoors. Sensor networks although exciting have very limited resources, for example, memory, processing power and bandwidth, with energy being the most precious resource as they are battery operated. However, these amazing devices can collaborate in order to perform a task. Due to these limitations and specific characteristics being application specific and heterogeneous there is a need to devise techniques and software which would utilize the meager resources efficiently keeping in view the unique characteristics of this network.

This thesis presents a lightweight, flexible and energy-efficient middleware framework called MidWSeN which combines aspects of queries, events and context of WSN in a single system. It provides a combination of core and optional services which could be adjusted according to the resources available and specific requirements of the application. The availability of multiple copies of services distributed across the network helps in making the system robust. This middleware framework introduces a new Persistent Storage Service which saves data within the sensor network on the nodes for lifetime of the network to provide historical data. A Priority algorithm is being also presented in this thesis to ensure that enough memory is always available. A novel context enhanced aggregation has also been presented in this thesis which aggregates data with respect to context. Application management service (AMS) provides Service optimization within the network is another novel aspect of the proposed framework. To evaluate the functionality of the work presented, different parts of the framework have also been implemented. The tests and results are detailed to prove the ideas presented in the framework. The work has also been evaluated against a set of requirements and compared against existing works to indicate the novel aspects of framework. Finally some ideas are presented for the future works.

ACKNOWLEDGEMENTS

I am really thankful to Allah the most gracious and most merciful. I would like to express my deep felt gratitude to both my supervisors Prof. Madjid Merabti and Dr. Robert J. Askwith for their patience and guidance. I would also take this opportunity to thank my mother without whose guidance and support I would not have reached this stage of my life. I am also very thankful to my husband Javed Anwar and three children Saad, Fahad and Leena for their continuous love and support through a difficult and trying period of our lives. I want to appreciate the efforts of my brother and sisters to help take care of my children during the last part of my studies. I also want to acknowledge my office mates Gurleen Arora, and Jasim Saeed for their encouragement and support. I want to thank Dr. Paul Fergus, Dr. David Llewellyn-Jones, and Dr. Faycal Bouhafs for those long discussions which helped me a lot in my difficult moments in studies. I would also take this opportunity to thank my fellow student Muhammad Arshad for his support through out my stay in Liverpool. I also want to acknowledge the cooperation of Amjad Shaheed, Sareer Badshah and Kashif Kifayat. I am really grateful to all these people mentioned above in particular and all others that I have not mentioned but have helped me a lot for their sincere and continuous support through out the course of my Ph.D.

I want to dedicate this thesis to my mother who has been a source of inspiration, selfless love and devotion all my life and my daughter Leena who was born at the start of Ph.D studies, grew up with it and is joy of my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 INTRODUCTION

1.1 Preamble

Internet technology has connected people to the wealth of knowledge in almost every field of life but there was still one important aspect missing the physical dimension because of their ability to sense small changes in the physical quantities. Sensors have been able to fulfill this criterion and have provided the physical dimension missing from this existing wealth of knowledge. The existing technologies combined with sensors have made it possible to make the dream of a smart space come true. Terms like ubiquitous computing and wireless technology is becoming very common and is being used in consumer products, for example, mobile phones, broad band, Internet.

There was a pre Internet phase and the world is going through the Internet phase in the recent years. After further advancements in the field of networking and computing there will be a post Internet phase which is leading us to the Sensor Net which means sensors connected to the Internet providing real time physical quantities (Tavakoli et al. 2007). One of the important events of the pre Internet phase, apart from the advent of computers, which starts the era, is wireless technology because it leads the world to the post Internet phase.

The history of wireless technology can be dated back as far as the 19th century when Marconi was able to conduct his first experiment(Carovillano 2003). It has led us to the present day technologies and terms which are widely used, for example, wireless LAN (Local Area Networks) (Cisco Systems 2004) in the 1990s which was a breakthrough for wireless networks, Global System for Mobile Communication (GSM) (GSM 2007) which made mobile phones possible. In addition to the previous works we have Wi-Fi, enables connection to the Internet or other machines that have Wi-Fi functionality through a wireless network. It conforms to the IEEE 802.11 standards. The most recent addition to wireless telecommunications is WiMAX, defined as Worldwide Interoperability for Microwave Access to promote conformance and interoperability of the IEEE 802.16 standards. WiMAX provides broadband access as an alternative to DSL or cable for the final phase or mile delivering connectivity from a service provider to a customer. WiMax can connect several Wi-Fi hotspots which cover one or more acess points. Through these technologies the world becomes well connected and people can connect to each other easily no matter which part of the world they are placed.

The history of networking can be connected to the creation of ARPA, or Advanced Research Projects Agency (ARPA's now known as DARPA) (DARPA 2007) in 1957 which led to the advent of networks and later the Internet which was marked by the first packet sent over the network in 1969 starting the exciting phase of the Internet era. ARPANET (Hauben 2006) stands for ARPA-network is a network that was specially designed to connect universities and other research centres on a network. The concept conceived in the 1960s was implemented in 1969 on four nodes (Griffiths 2002; Griffiths 2002; Hauben 2006). One was at the University of California Santa Barbara, one at UCLA, one at SRI International, and the last one at the University of Utah. The Internet is a worldwide collection of publicly accessible interconnected computer networks. The advent of internet led to email in 1972 and World Wide Web (WWW) the worlds first browser in early 1990s. Since its advent, the Internet has grown at a phenomenal speed both in volume and complexity. It has made it possible to connect anyone to almost anything for example other computers, telephones, mobile phones and even common house hold appliances. The Internet has been able to realize the dream of connectivity and has turned the world into global village.

As mentioned earlier there was one important aspect still missing, the physical dimension. Imagine while en route to your office in the morning, there is an accident ahead and that information is automatically transferred to your car which can then take an alternate route. In a different scenario while shopping in the super market you can connect to your refrigerator to find out what items you need to pick. This is the dream of a smart world or Smart space in which one can interact with the environment and help people in finding what they require easily. This dream is possible by combining the present day wireless communication technologies and sensors. The next section provides a historical perspective of sensors.

1.2 Sensors and Wireless Networks

The sensor age starts in 1967 when Honeywell Research Centre in Minneapolis, USA applied for patent for the edge-constrained silicon diaphragm by ART R. Zias and John Egan (Copidate Technical Publicity 2006). In 1999 the industry realized that sensors can go wireless which opened a new world of applications that was never imagined before. A Wireless Sensor Network (WSN) is a collection of small, low-cost devices capable of sensing, computation, storage and communication, called sensors. These sensors, although limited in their resources, can be combined together to perform complex operations.

Wireless sensors first use was for military applications (Chong et al. 2003). Sensors have been used in the wired environment previously, with success in industrial applications like automobile manufacturing, but the combination of wireless technology and sensors has opened a whole new exciting avenue of applications. It has made possible to access difficult terrains where human intervention was impossible or very difficult. These sensors can be deployed anywhere and can work independently for months without any maintenance. It is possible to throw these small compact devices called sensors into difficult terrains in large quantities where they organize themselves and start functioning by establishing contact with their immediate neighbours and start sending data for months or as long as their batteries last.

Sensors have made the dream of a smart world come true through embedded technology being small in size they can be embedded on any object or creature. This will make it possible to connect anything with everything any where. For example, our body network will be connected to our car network or home/office network and they can communicate with each other. It means a fully integrated and connected environment.

The different prospects of wireless sensors are exciting and are possible with existing wireless technologies but all this makes WSN very unique and difficult as well. Real world applications mean that sensors have to operate in harsh conditions where they could easily be made unavailable by physical forces of nature. Wireless sensors are battery operated so the batteries may run out very easily. As there will be no maintenance or network administrator, the network will have to manage itself. Presently the wireless sensors are sensing and sending raw data which is difficult with limited resources. Handling such a huge amount of data is a big issue and needs to be resolved. There are security issues especially in military applications and more generally because of the data-centric nature of this network. This network also communicates with wired or traditional networks which means that the protocols developed for this network will also have to consider issues related to communication with wired networks too.

In general, WSN is a very challenging and promising field for researchers because it has so much potential but there are a lot of issues that still need to be addressed. The aim of this research is to address some of the above mentioned issues in order to facilitate the dream of Smart Space in which humans can interact with the surrounding environment easily.

Looking at the previous history of these technologies shows how fast they have grown individually, especially in the later half of the 20th century, and worked collectively to give birth to new fields, for example, grid computing (Grandinetti 2005), peer-to-peer (Subramanian et al. 2005), ubiquitous/pervasive computing (Burkhardt et al. 2002) or ad hoc (Ferrari 2006) and sensor networks etc. The idea of pervasive or ubiquitous computing is to connect anything with everything seamlessly without the user being even aware of it. The goal of pervasive computing is to provide unobtrusive and continuous connectivity using the current networks and wireless technology. It has not only improved interconnectivity and transfer of knowledge but helped identify new avenues like smart spaces and embedded technology. This technology takes computing beyond the traditional desk top computer to consumer devices like the fridge, TV, our home security systems or electric systems, cars or even our clothes or coffee mugs (Burkhardt et al. 2002). First it was just data and information which was flowing now sensors have added physical dimension to it directly from the real world. It's like creating a virtual world using the real world parameters.

Taking a short tour of history in the previous sections also reveals that a lot of different types of hardware and technologies are working together which means a heterogeneous environment. Networking not only means connecting computers but also electronic devices and through embedded technology everything or every device. As the size of a chip is becoming smaller it is possible to embed them on any object or creature, which makes it possible to connect everything. It is not only possible to connect devices but bodies, clothes, etc. anything that can be thought of. As already mentioned above connectivity means heterogeneity too because everything is connected to anything meaning different types of hardware and software. It also means heterogeneous technologies communicating and working with each other. For all these heterogeneous technologies to work together seamlessly, smoothly hiding the complexities of underlying platform details there is a need for another layer of software which can relieve the user of the burden of resolving complex issues related to heterogeneous hardware and software. The developer does not have to be aware of the underlying system because everyone cannot be an expert or computer scientist. The end user is only interested in getting his work done with out hindrance. However, the underlying system might be very complex combining different technologies, software and hardware. Operating systems could not handle this complexity because there may be more than one operating system involved. Therefore to handle heterogeneity and its related issues researchers came up with another layer of abstraction called *middleware*. The next section introduces this aspect of research.

1.3 Middleware

The term middleware gained popularity in the 1990s although there are claims that it has been around since 1968 (Naur et al. 1968). The popular definition of the term is that it is an additional layer of software between the application and the operating system which deals with issues not being handled by the operating system. Middleware is used to hide the low level system programming from the developer so that he can concentrate on his actual application. With the advent of the Internet and World Wide Web, so many different types of new software are being introduced in a short period of time. On the other hand technology is also advancing fast as we have already observed in the previous section, therefore new and more sophisticated hardware is flooding the market. To bridge the gap between heterogeneous software and hardware, middleware comes in which hides the lower abstractions from the user.

The term middleware has also been referred to as a distributed platform of interfaces and services that reside between the application and the operating system in order to facilitate the development, deployment and management of distributed applications. It can work as glue between two applications, an application program and a network, more than two software systems (applications, operating system and network). Generally it is a communication link between two or more software and /or hardware systems (Coulson 2004) e.g. OMG'S CORBA (Vinoski 1997; Wang et al. 2001), Microsoft's DCOM (Karp et al. 2000) or Java RMI (Satyanarayanan et al. 1999). These are the most popular middleware software available in the market since the advent of the term middleware.

Common Object Request Broker Architecture commonly known as CORBA was introduced by the Object Management Group (OMG). It is an open, vendor-independent architecture and infrastructure to be used by computer applications to work together over networks. It details the Object Request Broker (ORB) component of the object management architecture (OMA) (Vinoski 1997). The ORB component facilitates communication between clients and objects.

Distributed Component Object Model (DCOM) extended Microsoft's COM and has now been replaced by .NET in 2002. The .NET framework (Microsoft 2007) included in both Windows Server 2003 and Windows Vista is software that has been developed for Microsoft Windows. It provides solutions to common program requirements, and manages the execution of programs written specifically for the .NET framework

Java Remote Method Invocation (Java RMI) helps in creating distributed Java to Java applications. It is based on Remote Procedure Call (RPC) mechanism which allows a program to execute a subroutine in a different address space with out explicit instructions (Grosso 2001).

All the above are examples of popular middleware systems used in traditional or wired networks. The existing middleware techniques for wired or traditional networks cannot be applied to WSN due to the following reasons.

- The existing wired and traditional networks have more resources therefore the middleware developed for these are resource intensive whereas in sensor networks the resources are very limited. Energy is probably the most precious resource because wireless sensors are battery operated. Similarly, other resources like memory, processing power, bandwidth etc., are all very limited and constrained.
- Sensor networks usually works without any human intervention or network administrator. Self-organization and self-management is one of the characteristics of this type of networks. Sensor networks might have to operate in difficult terrains with limited resources without the help of any kind of human intervention or infrastructure. These sensors could just be thrown into harsh terrains randomly and have to self-organize and start functioning as a network. Where as the traditional networks are operated by humans and needs constant human intervention.
- Usually sensor networks are connected to the traditional wired networks also or they might be part of a heterogeneous network having different capabilities, functionalities and characteristics.
- Deployment in real time situations in difficult terrains and random deployment means that these sensors are exposed to harsh conditions with limited resources. Therefore they could be easily damaged or unavailable or run out of batteries which mean dynamically changing topologies and reorganization of the network.
- Sensor networks are application specific which means application requirements play a major role in the dynamics of the network organization and functioning.
- The operating systems for sensor networks are not well developed. Therefore they do not provide the full functionality of operating systems as in traditional networks. There may be different reasons for not having fully developed operating systems which is beyond the scope of this text but due to these reasons developers have extra burden of worrying about the underlying hardware.

Therefore, due to the above mentioned reasons the researchers and scientists face a number of challenges in developing lightweight middleware which could cope with the unique characteristics of a WSN.

In order to realize the dream of integrated Smart world in which humans can interact with the environment any time any where the researchers are trying to use the limited resources of WSN in an efficient way to make the most of this network. The work done so far has mainly concentrated on how to transfer most of the data to a powerful base station and then manipulate or process that data to gain useful information. However, the problem is that transmission is the most expensive activity in terms of resources in WSN. The techniques used so far focus on simple data gathering-style applications, and in most cases support one application per network. Therefore, network protocols and applications are usually closely coupled. These procedures are usually ad hoc and do not provide a good solution to the vision of Smart World in which the sensors are able to talk to each other and dynamically discover services whenever required.

The goal of this work is to systematically design services and portable abstractions of the system for diverse applications and sensor reuse. In addition, the system should also be able to support and coordinate mechanisms to efficiently adapt to changing environment. In order to achieve this •goal there is a need for middleware which can interconnect wireless sensors quickly requiring minimum software and hardware without any problem. The middleware presented in this thesis will adapt and provide services under different circumstances with minimum hardware or software requirements. This middleware will sit between the operating system and application in order to support the development, maintenance, deployment, and execution of sensing-based solutions (Shen et al. 2001). In addition it will also provide abstractions and mechanisms for dealing with heterogeneity of sensor nodes. Middleware for WSN should be energy efficient, easy to use, robust and scalable.

1.4 Problem Definition

The challenge facing the researchers in WSN'S is how to seamlessly integrate and bind the sensors together with limited resources. Sensors are light, low cost and can be deployed easily. Stringent resources mean that they have limited power being battery operated, small memory, low bandwidth and less processing capabilities. With limited resources the problem of handling a huge amount of data is a big challenge because WSN'S are data-centric. The whole network activity is focused on gathering data and transmitting it to the base station.

The middleware systems developed so far are restricted because they are application specific in order to make them efficient. They are either event-based (Dunkels et al. 2004; Dunkels et al. 2004b; Janakiram et al. 2005; Nicopolitidis et al. 2003), query-based (Bonnet et al. 2000; Gummadi et al. 2005; Madden et al. 2005; Woo et al. 2004; Yao et al. 2003), or context-aware (Choi et al. 2005; Gu et al. 2004; Huebscher et al. 2004) etc.. The sensors are only used for specific applications and the same sensors cannot be reused for different applications. Application dependant middleware is problematic because, for each different application, new middleware has to be implemented. For more than one application working together there will be more than one middleware. This makes it difficult for different sensors to talk to each other. In order to bind and integrate the sensors a strong need is felt for a generic and flexible middleware which would be able to utilize the same sensors for different applications and also be able to provide context aware information to the end user within the restricted resources of WSN.

Middleware for WSN should be event-based because they may be deployed in difficult terrains to listen to different events which are likely or expected to happen. In addition it saves energy because unless the event occurs they are in a sleep mode or low energy mode which conserves energy the most precious resource being powered by battery. Therefore it is also close to WSN characteristics (Römer et al. 2002). The main problem is that if there is an emergency or real time situation where the user might want to investigate further there is no provision in the existing event-based systems which would enable the user to immediately query the sensors directly or indirectly for further information. Therefore there should also be a provision for querying the system in a real time situation or any time the user wants to investigate or collect additional information.

Another challenge that is confronting the sensor community is how to handle data within the network. Previously two approaches have been deployed to handle data. The first approach was to sense and just send all the data to the base station or sink. The second approach which is mostly employed by the current systems is to store data temporarily and perform certain calculations within the network and then this semi processed data is sent to the base station for further processing. The reason for this shift is the fact that research has proved that transmission is the most costly activity in terms of resources, especially energy which is the most precious resource in this kind of network. Although the second approach is an improvement but still all data is transferred, stored and mostly processed outside the sensor networks which means a lot of transmissions.

Researchers found that in-network processing was more cost effective in terms of resources. For example, in a real time deployment of wireless sensor networks on a volcano the researcher had to face real difficulty in transferring data from the sensors to the laptop which acted as a base station because one minute of collected data took several minutes to be transmitted along the radio channel (Werner-Allen et al. 2006). The authors proposed local buffering of event data to overcome the problems they faced. Now further research has proved that storage utilizes even less energy (Mathur et al. 2006) therefore researchers are exploring this very new and exciting prospect of saving the data within the network permanently or for longer period of time.

This new development is possible with the advancement of technology, for example, the advent of new NAND flash memory technologies which has increased the capacity and energy efficiency of local flash storage dramatically (Mathur et al. 2006) and also other advances, for example, Imotes introduced by Intel which has 32 MB memory or SPOT developed by Sun Micro Systems which has 4MB of flash memory or the tiny wireless memory chip developed by Hewlett-Packard (Hewlett-Packard 2007) which can store up to 512KB, only 2-4mm square of size and can transfer data at a speed of 10 megabits per second. Therefore it now seems possible to save the data within the network using flash memory process it and then send out processed information on request. This means less transmissions and also availability of data whenever required which ensures better utilization of the limited resources.

1.5 Design Goals

Sensors have so many indoor and outdoor applications which are beyond any one person's imagination. Such a varied scenario requires a middleware which would be able to adapt to changing circumstances and enable the heterogeneous sensors to communicate with each other to provide services under different conditions with the minimum requirement of hardware or software. It should interconnect wireless sensors quickly without any problem.

The main objective of this research is to create a middleware that would work with different applications. This middleware will sit between the network and application in order to support the development, maintenance, deployment, and execution of sensing-based applications (Römer et al. 2002). It should be generic and flexible enough to facilitate different applications and acquire context-aware data. In addition it should also provide abstractions and mechanisms for dealing with heterogeneity of sensor nodes. Main characteristics of WSN's are heterogeneity, small-scale devices, low-energy-usage, dynamic, restricted-resources. In view of its special characteristics middleware for should, however, be designed to be energy efficient, robust and scalable.

In order to achieve the above mentioned goals we are introducing a novel generic and flexible middleware framework for WSN'S which will not only provide context-aware information but also be both event-based and query-based. This framework can be implemented as a whole or in parts depending on applications requirements. It has a distributed, service-oriented architecture which means services can be discovered dynamically within the network. No other framework to the best of knowledge of the author has combined both events and queries. The use of queries also provides an opportunity to handle real-time situations. Giving the concept of sensor reuse for multiple applications and application independence because so far sensor networks are only restricted to a single application.

1.6 Novel Contributions

Motivated by the advances in sensor networks and based on the above mentioned problems this thesis presents a novel middleware framework called MidWSeN. The contributions provided in thesis are detailed as follows.

- A detailed literature survey was conducted to have a thorough understanding of the area and provide a set of requirements. Based on the literature survey a fresh set of requirements has been proposed in the thesis in order to address the issues which are important to develop a better middleware for WSN.

- This thesis presents a lightweight and energy efficient middleware framework (Javed et al. 2005) which combines most important aspects of WSN in a single system. No other work to the best of author's knowledge so far has presented such a holistic view of different aspects of WSN. This framework can be used by both query based and event based applications. This makes it more generic because applications can both query and register events as well. It also has the ability to work efficiently under normal as well as emergency situation. Due to service–oriented architecture it can be used in heterogeneous environment as well as pure sensor based environment because the framework does not bind the user to any specific protocols or data structure. Furthermore, the user does not have to worry about heterogeneous underlying platforms. Simple and easy to use framework in addition to loosely coupled architecture, provides a layer of abstraction between the application and the operating system which relieves the user from worrying about the under lying hardware. The framework does not concentrate on a single aspect of middleware it is providing query, event and context. The framework is flexible and can be customized to an application's needs, keeping in accordance with the application specific character of WSN.
    - The framework provides a combination of core and optional services which could be adjusted according to the resources available and specific requirements of the application. The availability of multiple copies of services distributed across the network helps in making the system robust. Due to the loosely coupled architecture new services could easily be added to the system which makes it more scalable.
- Storage and processing of data within the network consumes fewer resources (Javed et al. 2007b), as has been already mentioned above. Therefore our framework provides services and techniques which ensure that data is not only stored but also processed within the network in order to make it more energy efficient. This middleware framework introduces a new persistent storage service

which saves data within the network. This service stores data within the network for lifetime of the network which provides historical data. This not only helps in creating spatial and temporal relationship between data but also provides implicit relationships. It has been mentioned that previously data was either directly sent to the base station or kept temporarily for initial processing and then transmitted back to the base station which consumed a lot of precious resources. Storing and processing data within the network saves precious resources, for example, bandwidth and energy.

  - Memory in WSN'S is very limited and storing data within the network might result in utilization of all the memory available in the network therefore to handle such a situation a Priority algorithm is being also presented in this thesis. This algorithm will prioritize the data and whenever the need for more storage space arises it will aggregate the lowest priority data to create more space. This will ensure that enough memory is always available when required and not all of the data is erased.

- Aggregation is a very important technique which helps in combining redundant but related data (Javed et al. 2007a). It also helps in reducing the amount of data being transmitted back to the base station. A context enhanced aggregation has been presented in this thesis. The data is being aggregated with respect to context and is sent to the Persistent Storage to be stored along with the context. This service also has an added advantage in our framework that it helps in saving memory or storage space. This service also provides the opportunity to the developer to tailor or customize it to applications specific needs thus making it more flexible.

- Application management service (AMS) is an important service which not only works as a link between the sink and WSN but also helps in managing the framework. The novel aspect of this service is that it provides Service optimization within the network. When a query or event is sent to the network it will help in locating the correct service and deciding which are the most appropriate location and efficient path through which to send it. This service will also inform the user if an emergency situation occurs with the help of the Rule service if a rule or set of rule fires.

1.7 Thesis Structure

The rest of the thesis is structured as follows:

- Chapter 2 gives an overview of sensors and sensor networks. It describes in detail the characteristics, design principles, applications sensors. It also gives a brief description of the hardware, technologies and software related to WSN.
- Chapter 3 gives the background of the significant works already done in middleware in WSN and also why traditional networks middleware cannot be applied to sensor networks. Further, it discusses the existing works and their drawbacks in terms of flexibility, ease of usage and efficiency.
- Chapter 4 gives a new set of requirements considered important for a better and efficient middleware. It also presents the design of a novel middleware framework. It further gives some detail of the novel core services and optional services included in the framework and their individual contribution to the overall structure.
- Chapter 5 explains the persistent storage service in greater detail. This novel service is used to store data within the network on the sensors. This chapter explains why and how the data is being stored within the network. It presents a novel algorithm for prioritization. It also explains how other services within the framework interact with this service in order to provide an efficient system.
- Chapter 6 explains the aggregation service in detail. The aggregation service is also used to save memory and add context to data in addition to the traditional way of compressing or filtering data and saving resources which is a new concept. This chapter also explains why it is necessary to keep aggregation as a middleware service and how it functions within the framework to provide efficient use of resources.

- Chapter 7 provides the evaluation of the framework and the implementation details. It gives an evaluation against the requirements given in chapter 4 and a comparison against the existing works.
- Chapter 8 provides the conclusion and suggestions for future work.

1.8 Summary

This chapter has underlined the need for having a generic middleware for WSN's. In the beginning of the chapter the evolution process of the existing technologies has been given. Through the evolution process of these technologies now a stage is reached where dreams like smart homes or smart spaces can be realized. It has also been iterated that wireless technology and sensors combined together with other existing technologies is able to realize this dream of Smart environment which means humans can interact with the environment intelligently.

Taking a short tour of history also reveals that a lot of different types of hardware and technologies are working together which means heterogeneous environment. Therefore, to handle heterogeneity and its related issues the researchers came up with another layer of abstraction called middleware. The existing middleware techniques for wired or traditional networks, however, cannot be applied to WSN'S because they are resource intensive. Wireless sensor networks have unique characteristics, for example, they have very limited resources and operate without human intervention in harsh terrains. They have to be self organized and self manage over long periods of time. Therefore there is a need for new middleware strategies in order to cope with the unique characteristics of the WSN.

The challenge facing the researchers in WSN's is how to seamlessly integrate and bind the sensors together with limited resources. Sensors are light, low cost and can be deployed easily. The middleware systems developed so far are restricted because they are application specific in order to make them efficient. They are event-based, query-based or context-aware. The sensors are only used for specific applications and the same sensors cannot be reutilized for different applications. With limited resources the problem of handling a huge amount of data is a big challenge because WSN'S are data centric. Researchers found that in-network processing is more cost effective in terms of resources. Further research has proved that storage utilizes even less energy therefore researchers are exploring this very new and exciting prospect of saving the data within the network permanently.

Chapter 2 WIRELESS SENSOR NETWORKS

2.1 Introduction

Data networking technologies have a come along way since the 1980s. It has changed by leaps and bounds in the last few decades and the coming decade or two promise more changes thanks to wireless sensor networks. A brief tour of the history of these technologies was given in the last chapter. This chapter gives a comprehensive overview of what the related technologies are and how WSN's progressing. This chapter talks about wireless networks in general and Wireless Sensor Networks (WSN's) in particular. The following section introduces wireless networks and its different types. Section 2.3 explains WSN's in some detail the basic operations, hardware, characteristics and technologies. This section also explains the different operating systems, applications, design principles and also different research areas.

2.2 Pervasive/Ubiquitous Computing

Pervasive computing, also called ubiquitous computing is taking the world beyond computers to embedded technology (Estrin et al. 2002). Pervasive computing can be defined as embedding computing into the environment in such a way that the user will not have to think for utilizing this technology. Pervasive computing is making use of the existing wireless and other technologies (Ou et al. 2007). Research is going on in the area of embedded technology and it is leading towards an integrated Smart world where there may be a chip everywhere on human body, clothes, and equipment that is being used in daily lives at home, office, car etc. The goal is that each and every thing used in daily life is always connected to a network of devices without any hindrance and can be used casually. The integration of already existing technologies like wireless communications and sensors has made it possible to make this fantasy real (Microsoft 2007).

2.3 Wireless Networks

Wireless technology helps two or more computers to communicate using standard network protocols, without cabling. Any technology that does this could be called wireless networking but often, however, the term refers to wireless LANs. This technology has produced a number of popular wireless solutions using the IEEE 802.11 standards. These solutions are being used in business and schools as well as sophisticated applications where network wiring is impossible, such as in warehousing or point-of-sale handheld equipment.

2.3.1 Wireless Architecture

Wireless Networks includes different components which could be broadly divided into computer devices and a wireless infrastructure.

- Computer Devices:

There can be many different types of computer devices ranging from small to large devices for example sensors, PDA, laptop, desktops etc. These computer devices are equipped with wireless networking interface cards which provides an interface between the computer device and the wireless network infrastructure. Individual computers can communicate directly with all of the other wireless enabled computers. The computer devices can work as clients or servers. The client requests for a service and the servers which are also sometimes referred to as end systems satisfy the clients request or provide the service. They can share files and printers dynamically with the support of appropriate software. They can also access wired LAN resources, using one of the computer devices as a bridge to the wired LAN having special software. (This is called "bridging").

- Wireless Infrastructure:

In the second type of architecture the wireless network can use an access point, or base station to access data and resources of the wired network. The access point acts like a hub, providing connectivity for the wireless computers and connects the wireless LAN to a wired LAN. This, with appropriate networking software support, allows users on the wireless network to access wired LAN resources, such as file servers or existing Internet connectivity and vice versa (Akkaya et al. 2005; Frodigh et al. 2001; Nicopolitidis et al. 2003; Varshney et al. 2000).

2.3.2 Types of Wireless Networks

This section briefly explains that wireless networks are networks without connecting cables that rely on radio waves for transmission of data. The previous section explained the two main components of wireless networks. One of the two main components are computer devices which uses interface cards to communicate with other devices whereas the second component is the wireless infrastructure which uses a common access point to communicate.

While the term wireless network may technically be used to refer to *any* type of network that is wireless, the term is most commonly used to refer to a communications network whose interconnections between nodes is implemented without the use of wires, such as a computer network (which is a type of communications network). Wireless telecommunications networks are generally implemented with some type of remote information transmission system that uses electromagnetic waves, such as radio waves, for the carrier and this implementation usually takes place at the physical level or "layer" of the network. (For example, see the Physical Layer of the OSI Model).

- Personal Area Network

A personal area network (**PAN**) is a network to communicate among computer devices, for example, computers, phones or PDAs within a few meters space. With the help of wireless network technologies, for example, Infrared, Bluetooth, and ZigBee wireless personal area network (WPAN) is also possible.

  - One type of wireless network uses Infrared to connect small devices over very short range in WPAN (Wireless Personal Area Netork) having a small range. Infrared is limited to line of sight and used for point to point communication. One example of Infrared communication protocol is IrDA (Infrared Data Association) which is physical specification standard for short range data exchange over infrared light.

  - Another type of network used for WPAN is Bluetooth (Bluetooth 2007; Ferro et al. 2005) for short range communiction of data. Bluetooth sends and receives signals from all directions and has greater communication range than infrared. It follows the IEEE 802.15 standards.
  - Another type of network technology used for WPAN which is simpler and cheaper than other similar technologies mentioned above is ZigBee. ZigBee uses small, low-power digital radios conforming to IEEE 802.15.4 standard.
  - Ultra-wideband (UWB, ultra-wide band, ultraband, etc.) is also used for WPAN as a radio technology. It can also be used for short-range high-bandwidth communications at very low energy levels. Ultraband is using pulse coded information with sharp carrier pulses at a bunch of center frequencies in logical connex.

- Wireless Local Area Network
  - One type of wireless network is a WLAN or Wireless Local Area Network. Similar to other wireless devices, it uses radio instead of wires to transmit data back and forth between computers on the same network. It is based on IEEE 802.11 standards for example IEEE 802.11a/11b. Wi-Fi (Ferro et al. 2005) is a commonly used wireless network in computer systems which enable connection to the Internet or other machines that have Wi-Fi functionalities. Wi-Fi networks broadcast radio waves that can be picked up by Wi-Fi receivers that are attached to different computers.
- Wireless Metropolitan Area Networks
  - This type of network that could connect many wireless LANs. WiMAX, the Worldwide Interoperability for Microwave Access (Agapioy et al. 2006), is a telecommunications technology aimed at providing wireless data over long distances in a variety of ways, from point-to-point links to full mobile cellular type access. It is based on the IEEE 802.16 standards, which is also called WirelessMAN.

- Wireless Wide Area Networks
  - This type of network can cover a wider range than a LAN and can make use of cellular technologies, for example, GSM. The proposed IEEE standard for WWAN is 802.20. The WWAN could also be called mobile device networks.
  - Global System for Mobile Communications (GSM) network is divided into three major systems which are the switching system, the base station system, and the operation and support system. This is used for cellular phones, is the most common standard used for a majority of cellular providers.

This section briefly explained that networks without cables are known as wireless networks and mainly has two types of architecture. The first architecture uses network interface cards for communication whereas the second uses a common access point to communicate. Sensor networks can make use of both types of architectures.

Different types of commonly used wireless networks have also been mentioned above. There are many other types and varieties of wireless networks therefore it is difficult to compare and categorize all of them. The main focus in this chapter is wireless sensor networks which are a network of small devices connected over a short range as already mentioned above. The next section gives detailed information about these types of wireless networks.

2.4 Wireless Sensor Networks

This section provides information about wireless sensor networks in detail which is the main focus of attention in this research. It explains the architecture, its unique characteristics and applications which make this research more exciting and challenging. This network basically consists of a base station and large number of small-scale cooperative nodes capable of limited computation, wireless communication and sensing. By correlating the sensor output of multiple nodes, the whole network can provide functionality that an individual node cannot (Römer et al. 2002). As already mentioned in the previous section WSN can make use of both architectures. It can connect with it neighbors using on-demand multi-hop communication as in peer to peer communication to reach a base station to communicate with wired networks. On the other hand it can also have a single hop communication with its base station which in turn may be connected to a wired network.

2.4.1 Basic Operation

The basic operation of a sensor network is to deploy sensors in a target area to collect information or monitor and track certain specific phenomena. This deployment could be of a continuous process in order to replace nodes which have been unavailable or have depleted batteries due to environmental effects. These sensors can equip any object, creature or place with information-processing capabilities. They could either establish associations by a one- to-one relationship or by just throwing these nodes into an area of interest, where they have to function on their own.

After the successful deployment of nodes they can start functioning. They can either be assigned a task by an external source (for example. a vehicle passing by) or they could be self sufficient isolated networks which are capable of fulfilling a certain task themselves (for example sensing temperature) sending the information to other nodes which are controlling the results as part of the network. However, these controlling result sensing nodes might also be reporting to some external source for a more detailed monitoring of certain phenomena.

In order to obtain a high level result the sensor nodes have to coordinate and divide the task among themselves because they have limited functionality taking into account the individual characteristics of the nodes (for example attached sensors, location, and energy level).

2.4.2 Hardware

The hardware structure of the most commonly used motes or sensors consists of the following four parts.

- Sensing unit

The sensing unit is the main part and is made of two components sensors which are used for actual sensing and analog to digital converters (ADCs). The sensor unit senses in the form of analog signals produced by the sensors based on the observed phenomenon which are then converted to digital signals by the ADC, and sent to the processing unit (Akyildiz et al. 2002).

- Processing unit

The processor runs or processes the procedures that make the sensor node function, for example, collaboration with other nodes to carry out the assigned sensing tasks. It also has a limited storage or memory to, facilitate in the task of processing by storing intermediate results of a query or, store the programs associated to application management.

- Transceiver unit

A transceiver unit or Radio is used for communication and connects the node to the network. Communication which is mainly sending and receiving depends on at least three different layers: physical, media access control (MAC), and network. The physical layer handles the communication between transmitters and receivers which includes signal modulation and coding of data. The MAC Layer is responsible for efficient use of energy, changes in network size, node density, topology and bandwidth. MAC (Demirkol et al. 2006). Finally, the network layer is responsible for determining the routing path that a message has to take through the nodes of the network in order to travel from its source to the destination (Feng et al. 2002).

- Power unit

Energy is most vital for sensors therefore one of the most important components of a sensor node is the power unit. Two types of batteries might be used for the power unit one can be rechargeable and the other may be normal Alkaline batteries. Power units may also be supported by a power scavenging unit such as solar cells (Feng et al. 2002).

The above mentioned units are the basic parts which any sensor node will have however there may be other parts depending on the application they are used for.

2.4.3 Characteristics of Wireless Sensor Networks

A WSN has different unique characteristics which makes it more challenging and different from other types of networks. They consume less energy because they are small and therefore can be used in difficult terrains where human intervention is not possible. The most important characteristic is that they are small-scale devices with restricted resources like energy, CPU performance, memory, wireless communication bandwidth and range (Römer et al. 2002). This scarcity of resources especially very limited energy resource makes it more challenging. They are battery-operated and die if these batteries run out which means the energy has to be used very efficiently in order for them to work for longer periods of time.

When sensors are used for outdoor applications they are subjected to harsh conditions, for example, they may be carried away by wind or trampled by animals. They are prone to failure being battery-operated and used in difficult terrains. Due to dynamic conditions and power failure being frequent as they are battery-operated the topology can change frequently. So, constantly changing topology has become another important characteristic to consider while designing a sensor network.

They are also application dependent which is another unique characteristic of this network. For example, in indoor applications the sensors are accessible but in outdoor applications the sensors may not be accessible or highly unlikely to access which will have a deep impact on the design of the network in choice of protocols being used for different activities.

They may also be heterogeneous in terms of computing power and memory. Sensors may be used in collaboration with other equipment, for example, PDA, Laptops etc., within the same network. Such a heterogeneous network means a combination of more capable and less capable devices.

2.4.4 Technologies

In this section the different technologies associated with WSN's are introduced. Micro-Electro-Mechanical Systems (MEMS) are the integration of mechanical elements, sensors, actuators and electronics on a common silicon substrate through micro-fabrication technology. The MEMS is an enabling technology which can enable a smart environment with micro-sensors and micro-actuators ubiquitously. It helps in the realization of complete systems-on-a-chip dream.

The most common communication technologies being used for WSN's are Bluetooth and ZigBee. Both of these are used for short range wireless communications. BlueTooth is an industrial specification which uses low-power radio communications to wirelessly connect and exchange data between phones, computers and other network devices over short distances. It is being named "Bluetooth" after a Danish king more than 1,000 years ago Harald Bluetooth. It can transmit signals over short distances wirelessly up to 10 meters and generally communicate at less than 1 Mbps. A master Bluetooth device can communicate with up to 7 more devices which are called piconet in a PAN. It conforms to IEEE 802.15.1 standards (Nachman et al. 2005).

ZigBee (Eady 2007) was created as a new standard (IEEE 802.15.4) to support relatively simple, low power wireless communications, primarily home and industrial automation applications. ZigBee is being used in *wireless sensor* and *grid networks*. Some early examples of ZigBee networking include monitoring your washing machine for leaks or controlling the heating of buildings which are both indoors applications.

2.4.5 Operating System

There have been several attempts to make a specialized operating system for sensor networks. But these efforts are still at an early stage. The difficulty lies in the unique characteristics of sensor networks, especially the resource constraints and their application dependant nature. One of the first and probably the most important attempts is TinyOS (Du et al. 2004; Levis 2006; Levis 2006). It has been developed in nesC (Gay et al. 2003) also specifically designed for embedded systems and programming component based applications. It is an extension of C which is well known for programming at the physical level. TinyOS is based on an event-driven model which is very close to sensor networks. It uses event handlers and tasks to handle an external event or, for example, a sensor reading. These events or tasks will always be processed to completion. Tasks can be posted to be processed later. Race condition can be detected at an early stage through nesC programming techniques.

Other operating systems developed for WSN also use C programming. They are Contiki (Dunkels et al. 2004a), MANTIS (Bhatti et al. 2005), SOS (Han et al. 2005) and NanoRK (Eswaran et al. 2005). Contiki and SOS are both also event driven operating systems where as MANTIS and Nano-RK are based on preemptive multithreading.

2.4.6 Design Principles

A survey of literature reveals three main design principles for WSN; Localized Algorithms (Estrin et al. 1999; Rosenstein et al. 1997; Tsuchiya 1988), Adaptive fidelity (Estrin et al. 1999), and Data-centric communication (Akyildiz et al. 2002; Ratnasamy et al. 2003). Localized algorithms are a distributed algorithm which utilizes only a subset of nodes for a specific task (Meguerdichian et al. 2001; Qi et al. 2002). They are scalable, robust, and can utilize resources more efficiently by distributing tasks among coordinating sensors. They are difficult to design because their behavior depends on global knowledge converting this global knowledge to local behavior means changing it completely. Also, different types of localized algorithms may have different types of parameter choices which may lead to conflicting global behavior. Thus energy insensitive design may utilize more battery power or more resources. In comparison adaptive fidelity the other design principle is more energy efficient because there is a trade-off between the quality (fidelity) of the output and resources, for example, battery lifetime or bandwidth etc.

The adaptive fidelity algorithms are resource efficient but again not without problems, the implication to introduce multi-fidelity algorithm is that, different fidelities consume a different level of resources; the most important ones are time, computing power and network bandwidth, which are extremely scarce in mobile computing environments. Most of the time, especially in the case of interactive applications, people would rather see a less-than-ideal result within constraints of time and power. Multi-fidelity algorithm allows the trade-off between output quality and resources consumed, which is an essential theme in adaptive application design.

In a data-centric (Akyildiz et al. 2002; Ratnasamy et al. 2002) approach the applications are not interested in the identity of a particular node rather it requires the attribute of the phenomena. For example, if an application queries about the temperature reading it will require the location at which it exceeds a certain limit rather than the temperature of a specific node. Therefore it has proven to be a more energy efficient data dissemination method due to its application knowledge.

Another important technique related to design is the in-network (Fasolo et al. 2007) processing which is popular among researchers. In in-network processing some of the calculations are done within the network and only the final result is communicated to the user. This means fewer transmissions to the sink which saves energy because transmission is the most costly activity in terms of resources than processing data within the network especially energy.

2.4.7 Applications of Wireless Sensor Networks

As mentioned before WSN's have a wide range of applications. Examples include military and national security surveillance (Abdelzaher et al. 2004; Lim 2001), geophysical monitoring which involves monitoring air, soil and water, condition based maintenance, habitat monitoring like studying birds and plants (Cerpa et al. 2001; Mainwaring et al. 2002), smart spaces, traffic surveillance, medical application (Heinzelman et al. 2004), business processes etc. There is a long list of applications in short sensors will be used everywhere. Below a list is given of a few real world examples of WSN's already being deployed and working.

For example in the medical field, researchers are exploring applications of wireless sensor network technology to a range of pre-hospital and in-hospital emergency care, disaster response, and stroke patient rehabilitation (Lorincz et al. 2004). Recent advances in embedded computing systems have led to the emergence of wireless sensor networks, which permits data gathering and computation to be deeply embedded in the physical environment. This technology has the potential to impact the delivery and study of resuscitative care by allowing vital signs to be automatically collected and fully integrated into the patient care record and used for real-time triage, correlation with hospital records and long-term observation (Malan et al. 2004).

These devices collect heart rate (HR), oxygen saturation (SpO2), and EKG data and relay it over a short-range (100m) wireless network to any number of receiving devices, including PDAs, laptops, or ambulance-based terminals (Thaddeus et al. 2004). The data can be displayed in real time and integrated into the developing pre-hospital patient care record. The sensor devices themselves can be programmed to process the vital sign data, for example, to raise an alert condition when vital signs fall outside of normal parameters. Any adverse change in patient status can then be signalled to a nearby EMT or paramedic. Figure 2-1 (Malan et al. 2004) and Figure 2-2 (Malan et al. 2004) depicts phenomenas explained above and these pictures have been adapted from codeblue project website.

Another real time example, which has been conducted successfully since 2002 working with biologists on studies of flocks of about 18,000 Petrels that live at sea but fly inland every summer to lay eggs and rear their chicks on Great Duck Island, a small, uninhabited isle off the coast of Maine(Mainwaring et al. 2002). The birds nest in underground burrows, which cluster around particular places on the island. Sensors called motes are being used to study the Petrels' nesting behavior. In this case, temperature, atmospheric pressure and humidity sensors record micro-environmental conditions, while passive infrared sensors detect the presence of warm birds and eggs.

An example of an industrial usage of sensors is at Jones Farm fabrication plant in Hillsboro, Oregon. About 4,000 places in such a facility hold equipment that should be monitored for signs of wear and failure (Culler et al. 2004). So many locations are impossible to be checked manually and therefore currently engineers can check only selected pieces every one to three months which was not good enough. A device failure between two vibration inspections at a plant may cause a costly interruption of operations. An entire system of 4,000 IMotes has been created that could provide hourly updates on the health of the plant's infrastructure.

These are just a few examples of the many exciting applications both indoors and outdoors of sensor networks that are achievable through the combination of existing technologies.

2.4.8 Research Areas in Wireless Sensor Networks

Wireless Sensor networks is a new area therefore there is a lot of potential from a research point of view. Currently research is going on in a lot of diverse directions in this field. This section gives a few major areas which are important. Every field described in this area has some common problems which relate to the common characteristics mentioned above in section 2.3. The common problems are of course very limited resources which dictate every effort done in this area.

- Security

  Security is a very important issue in wireless sensor networks from different perspectives. Some applications might be sensitive, for example, military applications in which sensors are used for collecting and communicating sensitive data. It is also important because sensors have to be deployed in harsh terrains and are subjected to harsh conditions which may destroy them easily. This means they are very vulnerable to physical attacks. One peculiar characteristic is that these networks may have no human administrator and have to self manage or self administer therefore it is important for this network to defend itself from enemy attacks. Another security problem is that since it is a distributed network and has to collaborate with other nodes to perform a task, each node is important and has to be protected. Compromising a single node could mean compromising the whole network. The problems mentioned above added to meager resources make it a very challenging research field which requires fresh approach and new ideas.

In addition to the above mentioned problems which make it an interesting research area there are other problems like key management (Du et al. 2004), secrecy (Perrig et al. 2004), authentication (Liu et al. 2005), privacy (Sang et al. 2005), robustness to denial-of-service attacks (Perrig et al. 2004), secure routing (Karlof et al. 2003), security in mobile networks, and node capture(De et al. 2006).

- Management

An important advantage of sensor networks is that they can be deployed in harsh terrains and can operate without human intervention. The network typically has to self-manage without human administrators. Therefore management is a very important issue in this type of networks considering its meager resources (Marrón et al. 2005). Topology changes frequently as sensors can easily be unavailable being subjected to harsh conditions or die due to depleting batteries. In addition the environmental obstacles also lead to issues of connectivity interruptions and fault management. The dynamic nature of this network in which the nodes have to collaborate in order to achieve a common goal in a distributive manner poses a lot of open research issues (Lee et al. 2006). Therefore self-organization and self-management with such minimal resources is a monumental and challenging task which demands extremely skilful solutions.

- Mobility

Mobility means that sensors also have the capability of locomotion.(Light et al. 2006). This mobility means that the sensors are attached to objects which move in a specified area. The ability of locomotion introduces new problems and areas of interests. For example, the localization problems (Hu et al. 2004), in which nodes determine their exact position every time they move or change its position. The problem of energy efficiency in mobile networks (Hwang et al. 2004). Deployment is also a problem (Li et al. 2006). Group management is another issue in which nodes will be joining or leaving cluster. Security will also be an issue because every time a new node joins the group it has to be authenticated and there would be secrecy and privacy issues. The network might be a combination of static and mobile nodes which is another challenging area of research (Giordano et al. 2006).

- Routing

Routing is another major field of research in WSN's and a considerable amount of work has already been done in this field ((Heinzelman et al. 2000), Kulik *et al.* 2002, Intanagonwiwat *et al.* 2003). The major issues are the same as already mentioned above, limited resources, for example, energy and bandwidth which only requires essential minimal routing (Tubaishat et al. 2003). Another major difference from traditional networks is that sensor networks are data-centric networks therefore the unique addressing techniques used in the typical IP based protocols are not useful in these networks (Al-Karaki et al. 2004). Another unique characteristic of these networks is application dependence. The type of application for which a sensor network is used has a greater impact on the whole network as compared to wired or traditional networks on the routing technique being used (Akkaya et al. 2005). These are some of the main issues which should be considered at design time.

- Middleware

This is also an active research area for wireless sensor networks and as mentioned in the introduction chapter the focal point of research presented in this thesis. Middleware for sensor networks (Boulis et al. 2007; Fok et al. 2005; Light et al. 2006) is very important from a lot of aspects. The operating systems developed so far have not been able to fully provide the functionality required to talk to sensors and mostly the application developer has to communicate with hardware directly (Hadim et al. 2006). Another important aspect is that this network very rarely works as a stand alone system; it usually works in conjunction with wired networks. Therefore due to its heterogeneous nature it is important to have middleware. In addition to the above the unique characteristics of sensor networks also makes it eligible for special consideration. All this makes this area of research very challenging and therefore the focus of the research presented in this thesis has been directed to this particular area of wireless sensor networks. The next chapter is dedicated to this very important area of research in WSN's.

2.5 Summary

The above sections have discussed wireless networks in general and WSN in detail. As the name implies wireless networks are networks without cables. They have two different architectures one uses network interface cards and the other uses a common access point to connect to other computer devices or wired networks. Wireless sensor networks are also an example of wireless networks. This type of network collects information in a target area or monitors a specific phenomenon. These networks consist of small devices deployed in short range and establish wireless communication with its neighbours. Sensors have many exciting applications in the field of medicine, biology, traffic surveillance etc. The most distinguishing characteristics of sensors are limited resources like battery life, very limited memory and processing power. Other unique characteristics of WSN include application dependency, frequently changing topologies and heterogeneity. The unique characteristics of this exciting field make it very challenging.

The different design principles have also been discussed which are Localized Algorithms, Adaptive fidelity, and Data-centric communication. The different technologies associated with sensor networks like MEMS and communication technologies Bluetooth and Zigbee have been discussed briefly in this chapter. Section 2.3 describes the parts of a basic sensor node which are sensing unit, processing unit, transceiver unit and the power unit. This section also gives the different types of operating systems developed so far for WSN out of which the most important is TinyOS event-driven operating system developed in nesC. The NesC language is an extension of C which incorporates special features suitable for embedded systems. Some of the important and currently active research areas in wireless sensor networks have also been mentioned to give an overview of research activities going on the field of WSN's. They are security, management, mobility, routing, and middleware.

As mentioned above middleware is the main focus of attention in this research therefore the next whole chapter has been dedicated to explain research activities in this active field of research. In addition types of WSN's middleware and the difference between traditional and WSN's middleware has also been explained in the next chapter.

Chapter 3 BUILDING SENSOR NETWORKS MIDDLEWARE

3.1 Introduction

The previous chapter gave an overview of the general characteristics and applications of WSNs. In this chapter the overview of the role of middleware in building WSNs is being discussed. This chapter also highlights differences between traditional and sensor networks which leads to the fact that a whole new approach is required to address the challenges of developing middleware for WSNs. The current middleware approaches in WSN have also been discussed in some detail to create an awareness of the existing problems, which the proposed work in this thesis tries to address.

Middleware is used to hide the lower level abstraction from the developer so that he can concentrate on his actual application. With the advent of the Internet and World Wide Web so many different types of new software are being introduced in a short period of time that it becomes difficult to keep the systems updated in order to utilize the new softwares available in the market to their full capacity due to cost and interoperability with other existing software. On the other hand technology is also advancing, therefore new and more sophisticated hardware is flooding the market. To bridge the gap between heterogeneous software and hardware, middleware comes in which creates a bridge not only between heterogeneous software and hardware but also hides the lower abstractions from the user.

Middleware can be referred to as a distributed platform of interfaces and services that reside between the application and the operating system in order to facilitate the development, deployment and management of distributed applications. It can work as glue between two applications or an application program and a network or more than two software components/systems (application, operating system and network). Generally it is a communication link between two or more software and/or hardware components/systems (Coulson 2004), for example OMG'S CORBA (Vinoski 1997), Microsoft's DCOM (Karp *et al.* 2000) or Java RMI (Satyanarayanan *et al.* 1999). The next sub-section discusses these systems in some detail.

3.1.1 Traditional Middleware Approaches

This sub-section explains some of the commonly used or popular middleware approaches used in traditional middleware.

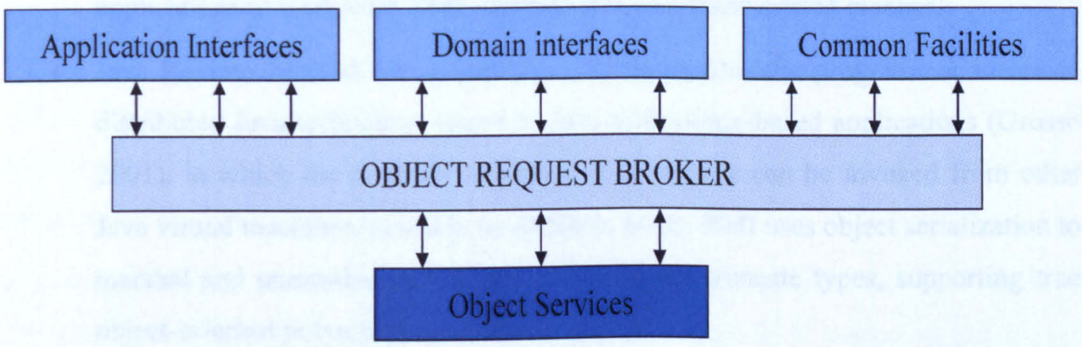


Figure 3-1 OMG Reference Model Architecture

- The Common Object Request Broker Architecture CORBA automates many common network programming tasks such as object registration, location, and activation; request demultiplexing; framing and error-handling; parameter marshalling and demarshalling; and operation dispatching (Vinoski 1997). Figure 3-1 OMG Reference Model Architecture shows the OMG Object Request Broker services.
- Distributed Component Object Model (DCOM) was introduced by Microsoft as a competitor of CORBA for software components distributed across several networked computers to communicate with each other. However, now Microsoft has introduced .NET (Horovitz *et al.* 2007) instead of DCOM which provides pre-coded solutions to common program requirements. The Common Language Runtime (CLR) environment runs .NET (Conard et al. 2001) applications that have been compiled to a common language, namely Microsoft Intermediate Language (MSIL) Figure 3-2 (Conard et al. 2001) shows the .NET framework. Windows Forms allows users to create standard desktop applications, based on the Windows Foundation Classes (WFC). Web Services, allows programs to communicate over the Internet. This framework allows the developer to write less code by providing underlying functionality in a consistent and standardised

manner. The Execution Support module shown in the figure Figure 3-2 NET new framework, contains most of the language runtime CLR capabilities. The .NET framework also provides built-in memory management functions. It's also important to note that Microsoft is not restricting use of the CLR to Microsoft languages. ADO.NET, provides data type support, whereas, Windows Forms provides graphic support for Window applications. System XML, enables applications to work with XML-defined data in a standardised manner.

- Java Remote Method Invocation (Java RMI) enables the programmer to create distributed Java technology-based to Java technology-based applications (Grosso 2001), in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts. RMI uses object serialization to marshal and unmarshal parameters and does not truncate types, supporting true object-oriented polymorphism (Grosso 2001).

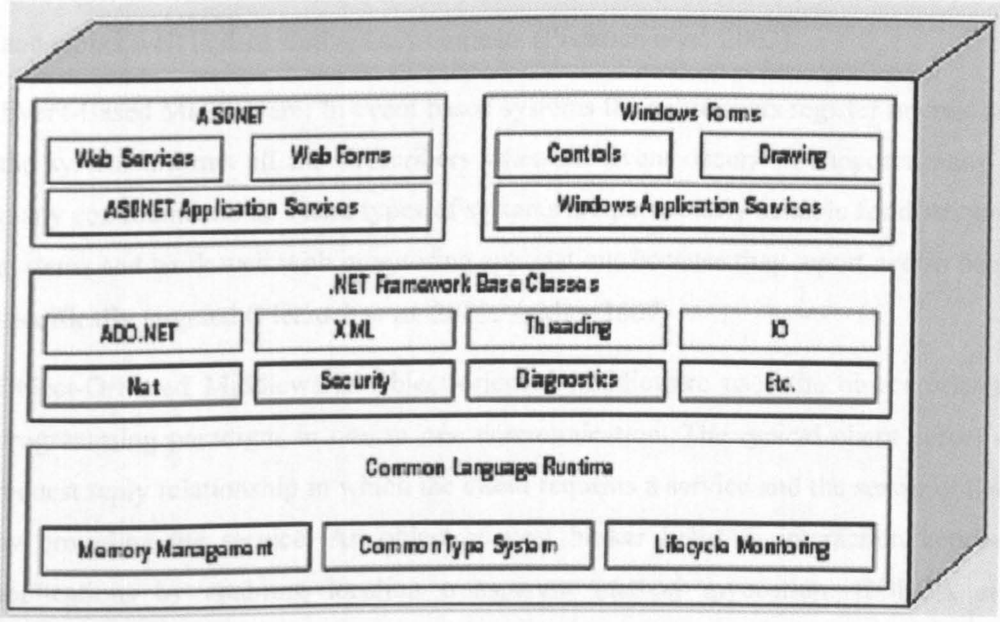


Figure 3-2 NET new framework

None of the above software is without its limitations. CORBA has become too complicated and difficult to maintain and also do not support transfer of objects or code.

DCOM was replaced by .NET due to the same reasons as CORBA but the problem with .NET is that it can fully be utilized only with windows although it is available on other platforms but partially, which makes it limited. Microsoft's .NET is also resource intensive. Java RMI has the same problems it is complicated and restricted to only Java platform. An ideal middleware should be kept lightweight and general. There are many different types of middleware currently available and working in the market. The next sub-section explains the different types of middleware currently available.

3.1.2 Types of Middleware

In literature different types of Middleware have been mentioned which may broadly be divided into the following areas (Coulson 2004):

- Message-Oriented Middleware: As the name implies it is based on Asynchronous message- passing and depends on message queue system. It works on the packet paradigm of communications prevalent in the lower layers of the OSI network model and works well in distributive environments (Pietzuch *et al.* 2003).
- Event-Based Middleware: In event based systems the subscribers register interest and the system informs all the subscribers when the event occurs. It supports many to many communications. These types of systems are particularly suitable for distributed systems and work well with monitoring applications because they report events being specifically targeted (Pietzuch *et al.* 2002; Zeidler 2007) .
- Object-Oriented Middleware: Object-oriented middleware uses the object oriented programming paradigm in one to one communication. The typical client server or request reply relationship in which the client requests a service and the server obliges by providing the service. An object request broker helps in interaction between applications by enabling location transparent method invocation. CORBA and DCOM are examples of this type of middleware.
- Reflective Middleware: Middleware has already been defined above as the software which acts as a glue between two applications or layers etc. whereas reflection is defined as the ability of software to self analyze and adapt itself to changing

requirements. Reflective middleware means applying reflection techniques to make middleware flexible and adaptable (Kon et al. 2002; Kon et al. 2000).

3.2 Differences between Traditional Networks and WSN's

The traditional middleware approaches, for example, .NET or CORBA are all used in distributed systems. They, however, are not well suited for WSNs (Römer *et al.* 2002; Yu *et al.* 2004). This is due to the unique charàcteristics of WSNs as described in section 2.3.3 of the previous chapter. The first and most defining characteristic of a WSN is that it has limited resources when compared to classical networks. In wired networks energy is rarely a problem while it is very challenging in WSN especially when battery operated. Other resources like memory, processing power or bandwidth are also less than a major concern in traditional networks. Whereas in WSN all these resources are very constrained and therefore efficient use of these resources is the biggest challenge. Therefore, it is difficult to use resource intensive functions or store large amounts of data. An external source has to be utilized for resource intensive processing or storage. Thus middleware for WSN has to take into account both WSNs and traditional networks because of a close interaction of processes executing in the WSNs and traditional networks. One such example of external functionality can be seen in (Romer *et al.* 2003).

Another important characteristic of WSNs is that nodes are application aware, which means the design principles and application knowledge are embedded in the nodes. Traditional middleware are designed to hide the application knowledge and give freedom to application developer to concentrate on the application requirements. Usually so far the whole network is designed around the application's requirements and the whole system is very much application dependant.

The third important characteristic of WSNs is that each node does not have a unique identification because they may be deployed in remote inaccessible locations in thousands. Therefore, the traditional request-response schemes may not be efficient for this kind of network. Rather, event–based communication may closely be associated with the characteristics of the WSNs. Data-centric communication matches content-based messaging more closely than the traditional RPC (Remote Procedure Call) communication.

The fourth differentiating characteristic is that WSNs nodes must operate unattended without human intervention or the traditional network administrator. Therefore, they must have parameters or algorithms, which can solve a certain problem to the best of their ability under restricted resources (CPU performance, memory, wireless communication bandwidth and range). These networks have to self-manage within its constrained resources.

Finally, physical conditions play an important role in WSNs as they are deployed to process real world data, in contrast to traditional computing systems. This means they are subjected to harsh conditions and may be frequently unavailable. They are also battery operated and may run out of energy which leads to changing network dynamics. WSNs have to deal with changing topology by self-organising.

The above mentioned unique characteristics illustrate that existing middleware for traditional networks can hardly cope with the unique characteristics of WSNs being resource intensive and requiring human administration. The differences between WSNs and traditional networks clearly require a new approach for WSN middleware architectural design and implementation. The next section explains the current middleware approaches to WSN in detail.

3.3 Current Middleware Approaches for WSN

There are currently various Middleware approaches being used within WSNs. The difference not only lies in the programming paradigms being used but also in terms of scalability, easy of use, significance and overhead incurred (Hadim et al. 2006; Römer et al. 2002). These approaches are being categorised and discussed in the following sections.

3.3.1 Data-Centric Approach

Data-Centric storage means saving relevant data by name at nodes within the network (Ratnasamy *et al.* 2003). In this approach all data having the same name will be stored at one location and all the queries regarding that particular named data would be directed to this location. The main advantage of this method is that network flooding is avoided. This method may only be useful under certain conditions: a large network, where the number of detected events is many and the total number of events detected is much larger than the types of events queried for. It is not useful in a highly mobile or unreliable sensor network. It also raises the issue of how to store data in a sensor network where nodes are highly mobile or unreliable.

Greedy Perimeter Stateless Routing (Karp *et al.* 2000) is being used for data-centric storage. GPSR has been slightly modified to deliver the packets to the node, which is closest to the destination location. A Distributed Hash Table is being used on top of the GPSR by hashing the event name into a key within network boundaries.

According to (Heidemann *et al.* 2001) attribute-based naming and application-specific in-network processing play a crucial role in conserving energy and bandwidth in WSNs. Their low level communication is based on names that are external to the network topology and relevant to the application in a distributed system. Attribute-based naming and in-network processing has two advantages. First, attribute-based naming does not require any overhead of communication for resolving name bindings. Second, reducing data communications by processing data within the network, as data is self-identifying making application-specific processing possible.

The low level naming communications architecture comprises of three main components: directed diffusion, matching rules, and filters. They are using directed diffusion to spread information in the system. Data is being managed as a list of attribute-value-operation tuples and matching rules are used to identify the arrival of data at the destination. They are using filters as a mechanism to help in diffusion, processing and running of application-specific code in the network. Their results show that these techniques can reduce network traffic and conserve energy to a great extent.

Directed diffusion (Intanagonwiwat *et al.* 2003) is a data-centric dissemination paradigm. In this technique attribute-value pairs within the sensor network identify data and all nodes are application aware. A node requests for named data by sending a query or statement of interest to all its neighbors. The matching data is diffused towards the node sending the statement of interest through a gradient or path. The network reinforces one or a small subset of the useful paths depending upon the data rate and/or if new events were recently received from that particular node or nodes. Data can be cached by intermediate nodes and statement of interest can be directed based on this previously cached data. This improves robustness, scalability and also conserves energy.

3.3.2 Database Approach

In this approach the WSN is being viewed as a single distributed database virtual entity which could be queried in SQL-style to assign sensing tasks by the user, examples are (Bonnet et al. 2000; Madden et al. 2002; Shen et al. 2001). Although the database approach is easy to use as it hides the distribution issues by addressing the network as a single virtual entity, it is unable to handle complex sensing tasks, for example, if a user wants to build a relationship between sequences of events in certain geographical areas. Also, adding new operations require extensive modification to the query processor of individual sensor nodes. Due to the tree topology used in the Database approach the network might not give optimal performance in all types of queries, for example, creating spanning tree of the network in TinyDB (Madden *et al.* 2005) will send queries to all nodes. In addition, TinyDB does not have any mechanism for customizing it to application requirements which is an important characteristic of WSNs being application specific. Hence, although the database approach is easy to use it suffers from some scalability issues as well as from limited expressiveness.

3.3.3 Event Based Approach

An action or occurrence detected by a program is an Event. In Sensor Networks an observation is the low-level output of a sensing device during a sensing interval. It is a measurement of the environment (Nicopolitidis *et al.* 2003). Events can be categorized into atomic and compound events (Li *et al.* 2004). An atomic event can be described as a single observation of a sensor, for example, temperature. A compound event may be a combination of several atomic events or an atomic event combined with another compound event. An example of a compound event could be an explosion which is a combination of high temperature, loud sound and light which are all atomic events (Li *et al.* 2004).

DSWare (Li *et al.* 2004) is the most appropriate example of this approach. It has attempted to include in-network processing of compound events but in a restricted manner. DSWare uses data-centric storage modified to replicate the data at several physical locations to improve the robustness of the system. Different copies of the data are synchronized when the system workload is low. They also have included some real time scheduling mechanisms providing flexibility at the cost of complex application development. They are using SQL-query thus restricting and sacrificing more memory and processing capabilities for flexibility and expressiveness.

Another example of Event based systems is Mires (Souto et al. 2006) middleware which uses a publish/subscribe solution. It is designed and implemented on TinyOS, which provides built-in support for event handling and a message-oriented communication paradigm (Active Message). The use of TinyOS (Levis 2006) makes it easier to implement Mires but it also limits the scope of the middleware to a single platform. Mires allow sensor nodes to advertise the types of sensor data they provide and the client applications can select from these advertised services. Then the sensor nodes will publish their data in accordance to the client's subscription. Mires emphasize on architectural and networking issues rather than subscription semantics. Mires is also not clear about the query language being used.

3.3.4 Cluster Based Approach

Another popular approach is the use of clusters (Lim 2001; Yu *et al.* 2004). Cluster heads are selected, usually on the basis of capability in terms of battery life or processing power, and other less capable nodes communicate their data to these cluster heads. These cluster heads can also communicate among themselves and also with the sink. It can be argued that the use of clusters or cluster heads promotes centralization, for example, if a cluster head fails suddenly due to destruction, the data of a whole cluster is lost. In the case of battery failure the cluster head has to transfer the data to another node so it must ensure it retains enough energy to transfer data. Power is the most precious resource in WSN and can be consumed in a more efficient way rather than using it on just data transfer most of the time.

Lim's (Lim 2001) work can be classified under this approach as he is using clusters. Alvin Lim is providing three main services, a lookup service, composition service and dynamic adaptation service. In the application system layer Lim is using a server to store the data within the sensor network which will be a super node having more memory. This may raise a lot of issues. For example being a battlefield surveillance project the network will be in a highly volatile real time environment and there is a high risk of nodes being unavailable. There should be a mechanism, which will automatically start functioning as a backup database server and also a mechanism that could be switched over immediately to decide which node will start functioning as a server. His work does not mention any such mechanism.

3.3.5 Mobile Agent Approach

Mobile Agents are autonomous, intelligent programs that move through a network. They have inherent navigational autonomy and can ask to be sent or copied to some other nodes, communicate with each other and collect local sensor data using mobile code. Significant examples of this kind of middleware are Mate (Levis *et al.* 2002) and SensorWare (Boulis *et al.* 2007). Both are lightweight but Mate suffers from serious memory restrictions, which force it to sacrifice certain important features, which make programming of tasks inefficient or difficult. SensorWare uses identifiers to uniquely identify a node, which raises robustness issues in highly dynamic environments. In addition the programming language used in SensorWare makes use of scripting, which makes even simple sensing events difficult.

3.3.6 Service-Oriented Approach

Service-oriented architecture means a collection of loosely coupled independent well-defined services communicating with each other. A Service is a clearly defined, complete and self-sufficient function. An application can call a service without prior knowledge of its underlying platform using any protocol or technology. The service-oriented architecture is independent of any specific protocol or technology and therefore could be implemented in any software environment.

Impala (Liu *et al.* 2003) is one of the examples of service oriented architecture middleware in WSNs. They are using a layered structure and trying to program sensors dynamically and provide a scheduling mechanism and an event-handling module. In other words it also acts like an operating system. However, Impala is limited in terms of hardware platform since it has only been developed for a single platform, Hewlett-Packard/Compaq iPAQ Pocket PC handhelds running Linux (Hadim *et al.* 2006).

Another example of service oriented architecture is Milan (Heinzelman *et al.* 2004) (Middleware Linking Applications and Networks) that receives a description of application requirements, monitors network conditions, and optimizes sensor and network configurations to maximize application lifetime.

To accomplish these goals, applications represent their requirements to Milan through specialized graphs that incorporate state-based changes in application needs. Based on this information, Milan makes decisions about how to control the network as well as the sensors themselves to balance application QoS and energy efficiency, lengthening the lifetime of the application. Unlike traditional middleware that sits between the application and the operating system, Milan has an architecture that extends into the network protocol stack. As Milan is intended to sit on top of multiple physical networks, an abstraction layer is provided that allows network specific plug-ins to convert Milan commands to protocol-special C language commands that are passed through the usual network protocol stack. However, again it's limited because it is tightly coupled with the application and lacks support for operating systems and hardware heterogeneity (Hadim *et al.* 2006).

3.3.7 Other Approaches

EnviroTrack (Abdelzaher *et al.* 2004) is another middleware specially designed for monitoring mobile targets. It provides middleware architecture for coordinating services to facilitate interaction between groups of sensors monitoring different environmental phenomena. However, EnviroTrack's group management is not very sound and specific because it is only based on small-scale deployment. Further, EnviroTrack needs to incorporate self organization techniques because the target is moving from one place to another in the sensor field and different groups of sensors will be participating in sensing task as the target moves.

Cougar (Bonnet *et al.* 2000) addresses individual sensors. Sensor data is being viewed as tables and the middleware developed optimizes and executes the query plan (Nicopolitidis *et al.* 2003). Cougar has a centralized approach for a large-scale network which could become a bottleneck considering the dynamic nature of WSN. Cougar transfers raw sensor data to a central more powerful base station, which consumes more energy and other resources, such as, bandwidth. Therefore, Cougar not only proves costly in terms of resources but there is always the risk of frequent link failure in case of continuous data communication between sensors and base station.

Both TinyLIME (Curino et al. 2005) and TeenyLIME (Costa et al. 2006) uses tuple space in which applications add and read data from a common tuple space. TinyLIME and TeenyLIME are based on Linda (Gelenter 1985) which also used tuple space shared memory model. TinyLIME is designed to query data from local sensors individually and hence, do not provide multi-hop propagation of data through the sensor network. To obtain data from a remote location the clients have to contact other clients in that particular location. TinyLIME assumes a distributed field of static sensors in which client can move and access local resources. By using the above mentioned techniques TinyLIME avoids complex query routing issues which makes it limited to only few types of applications. TinyLIME is also specifically designed for motes using TinyOS and therefore have predefined formats for the standard motes. The main difference between TinyLIME and TeenyLIME is that TinyLIME targets sensor networks in which sensors only are used for sensing and sending data to other more powerful devices, such as, PDA's whereas in TeenyLIME the applications are distributed and deployed directly on the sensors.

3.3.8 Context-Awareness

Context awareness is vital in WSNs for meaningful interaction between the user and devices. The system should be aware of the environment around these devices. Therefore, more and more context-aware systems are being developed to extract more meaning from data. There are a wide range of applications in various fields such as smart homes (Huebscher *et al.* 2004), smart devices (Gellersen *et al.* 2002), tourism (Abowd *et al.* 1997) etc. which utilize context in order to understand the activities, needs and situations of their users.

Imagine a smart home environment that would know all your requirements and will act accordingly. For example, such a system would know when a user entered their home tired from work and automatically switch on the lights, the coffee maker would start making coffee, and the toaster switch on so that toasts could be put into it or the fridge the middleware as different applications might require different levels of user involvement depending on both the application and the user. Other examples of centralized layered are (Gu et al. 2004; Korpipää et al. 2003).

3.3.9 Aggregation

Aggregation, being considered vital for WSNs has of late had a lot of attention focused on it by the research community because aggregation saves energy by reducing the number of messages sent, thus saving resources. In sensor networks many sensors will be sending same data of the same type, which means redundant data. Aggregation means same data obtained from different nodes can be aggregated and transmitted to its destination. To reduce a large amount of data aggregation can play a vital role in saving limited resources like energy, memory, bandwidth, etc.

The works using in-network aggregation is only going to be discussed here because the proposed middleware architecture in this thesis is processing and storing data within the network. It is a well known fact in the research (Heidemann et al. 2001; Madden et al. 2002) that in-network processing in a distributed environment has proven to be more advantageous than the server-based approach which is being more efficient and less power consuming.

The most popular work in this area is Tiny Aggregation (TAG). In TAG (Madden *et al.* 2002) the authors suggest a service using database query language techniques of selection and in-network aggregation. Distributing the query in the network and then collecting it in a tree structure combining relevant data and sending it to the base station. Another work is Directed Diffusion (Intanagonwiwat *et al.* 2003), a communication paradigm which diffuses a statement of interest from the user by flooding the network and routing the data on the most reliable path called gradients. Data is aggregated as it flows back from child nodes to parent nodes. These works are limited to particular scenarios in many respects. First there is no concept of saving data permanently on nodes within the sensor network and only partially processed data is sent to the user on query which proves more costly in terms of resources , for example, energy or bandwidth because communication is more costly than processing data locally on the sensor node. Second both TAG and Directed Diffusion is restricting the user to a particular query language. Finally they are recommending a particular topology or data structure, TAG uses tree topology, which suffers from limited expressiveness whereas Directed Diffusion only applies to limited scenarios.

3.3.10Discussion

As mentioned in the previous section there has been work on different aspects of middleware in sensor networks keeping in view its constraints. However, previous research has focused on one or few aspects of sensor networks. There is still a need for generic middleware, which could cater for all aspects of sensor networks. As can be seen in the previous section some of the systems are only designed for monitoring a single phenomena, for example, EnviroTrack (Abdelzaher *et al.* 2004) is only developed for tracking mobile targets. Some research is specifically targeted at battle field surveillance (Lim 2001). DSWare has tried to include some real time mechanisms but other systems, such as, SensorWare (Boulis *et al.* 2003) are not providing any such support. Some systems, such as, EnviroTrack (Abdelzaher *et al.* 2004) only provide group management where as others like COUGAR (Bonnet *et al.* 2000) only provide individual sensor mechanisms.

Another problem with most current middleware is that it restricts the user to SQL or SQL-like data query languages, which makes it difficult for the application to query the system in a different way, for example, querying data with context. Aggregation techniques are also restricted to the standard methods like Max, Ave etc. and do not allow the applications or users to apply other aggregation techniques apart from the standard aggregation methods available.

Looking at the above discussion and the previous sections it follows that all middleware available so far has concentrated on a few aspects. In our opinion instead a hybrid approach is the best option. Therefore, the best solution to addressing the problems in WSNs middleware is to take the best options of all the different types of categories.

Keeping in mind the hybrid approach the best available option so far in terms of design is data-centric design as it is not only energy efficient but also scalable and robust. It also improves latency and saves bandwidth by doing in-network processing. In-network processing is more desirable than just sensing and sending to an external source. In-network processing reduces the amount of data being communicated over the network

thus saving energy and bandwidth. It also reduces system latency by processing data quickly near the source. As far as the problem of being unable to cope with highly mobile nodes or unreliable networks where nodes destruction rate is high, this could be solved by redundancy. Redundancy means having multiple copies of data and services.
Event based middleware seems more suitable to sensor networks as it will only notify the system when an event occurs; this saves energy which is a very important factor. Sensors that are constantly sensing and delivering data will wear out their batteries very quickly as compared to sensors which wait and only send data when an event occurs that is of interest to the user. However, queries may also be necessary in case of an emergency, therefore it is also important to add features which allow users to pose queries when ever possible. As sensor networks may be deployed in an ad-hoc manner in a real time environment, it is important to add real time features as well.
There has been a considerable amount of research done in the various fields mentioned in the previous chapter; therefore it is not at all necessary to do everything from the scratch. The next chapter presents a middleware framework, which tries to address the issues raised in this chapter.

3.4 Summary

In the previous sections, the difference between traditional and WSN middleware has been explained to elaborate the point that there is a need for a new approach in WSNs. In addition, detailed literature surveys about the current approaches in WSNs have also been given to illustrate the work already done and its limitations. The main limitation of the existing middleware is to focus on a few aspects or a single feature. Furthermore, the application knowledge of the design principles are tightly coupled to the protocols being used.

The literature survey reveals challenges in terms of efficiency, scalability, limited expressiveness and robustness in the different middleware approaches. Most of the middleware for WSNs are very restricted and limited to a few applications only. Each new application demands a new middleware system, which makes it difficult for sensors in different networks to talk and share information. The above sections under-lines the need for a general and more open approach so that sensor reutilization is possible for different applications. This is a challenging task keeping in view the meagre resources of WSNs.

Keeping in view the discussion presented in this chapter the next chapter introduces a novel middleware framework that tries to address the issues discussed in this and the last two chapters. The next chapter first establishes a set of requirements and then introduces different components of the new framework, MidWSeN, which tries to address these requirements.

Chapter 4 A NEW MIDDLEWARE FRAMEWORK FOR SENSOR NETWORKS

4.1 Introduction

The first two chapters give information about wireless technologies currently available, some details about sensors and a background of different types of middleware in general. The previous chapter provides some details about state of the art middleware in wireless sensor networks. The previous chapters also illustrate that sensors combined with latest wireless technologies can facilitate the dream of connecting everything in our daily lives to a network of devices. Sensors can easily equip any object, creature or place with information-processing capabilities, as they are small, lightweight, and low cost. They can reach not only difficult terrains beyond human reach but can also become an integral part of our daily lives.

The previous chapter also illustrates that such a varied scenario requires middleware that would be able to adapt itself to changing circumstances and enable the heterogeneous sensors to communicate with each other. A middleware should also be able to provide services under different conditions with minimal hardware/software requirements. The middleware should interconnect wireless sensors quickly without any problem. Hence, the main objective of the research presented in this chapter is the creation of such a middleware framework called MidWSeN that would work with as many applications as possible.

Wireless sensor network middleware is usually defined to support development, maintenance, deployment, and execution of the sensing-based applications (Römer *et al.* 2002). A novel middleware framework is being proposed in this thesis, which sits between the network and application.

This chapter discusses the requirements of such a middleware; that it should be generic and flexible enough to facilitate different applications acquiring context-sensitive data. In addition it should also provide abstractions and mechanisms for dealing with heterogeneity of sensor nodes. Analysing the characteristics of WSNs in the previous chapter revealed that these networks are mainly heterogeneous, small-scale devices, having restricted-resources. In view of its special characteristics it is concluded that the main characteristics of middleware for WSN should be energy efficiency, robustness and scalability. The rest of this chapter discusses the requirements of the proposed middleware and also proposes a new framework design.

4.2 Requirement Analysis

This section lists the requirements that are being included in the research presented in this thesis. The literature survey given in the previous chapter reveals that research done so far has only been focused on one or few aspects of the sensor networks. Wireless sensor networks have been looked at from a narrow perspective, which raises robustness, flexibility or scalability issues. Previous research has been trying to solve the issues from a single application (or a few limited scenarios) perspective because sensor networks are application specific. For example, some research only focuses on mobility (Abdelzaher *et al.* 2004) or is limited to scenarios like battlefield surveillance (Ni *et al.* 2005). Application specific means that a system is customised for a particular use rather than general use. In sensor networks, application specific means that low-level communication or in-network message processing is based on names that are external to the network topology and relevant to the application (Heidemann *et al.* 2001) for example aggregation filters. It can be proposed that the middleware designed for WSN should be flexibly adaptive so that it could accommodate the application requirements. In order to achieve the vision of the smart world in which sensors interact with each other and discover services dynamically, there is a need for middleware which should not only be scalable, robust, expressive and easy to use but also adaptable and flexible. The aim of the research presented in this thesis is to either accommodate or facilitate these requirements. The necessary requirements to be considered are as follows:

- Adaptability and flexibility.

Wireless sensor networks are unique in the sense that their versatility is not only in terms of system hardware and software but also in terms of applications, which also affect and play a vital role in the overall network functions. As mentioned in the above discussion, middleware, which is limited to a single type of application, cannot be general. Therefore the required middleware should be adaptable and flexible so that it could accommodate as many situations as possible (indoors or outdoors) at any scale (small /large network). Here it should be noted that there is a difference between limited application and having knowledge of an application. The former means that the network can only be used for a specific type of application whereas the latter means that it can be adapted to an application.

- In-network processing.

In-network processing reduces the amount of messages being communicated over the network by processing data near the source thus saving energy and bandwidth. Communication consumes more energy by utilizing more resources. In-network processing is therefore more desirable in sensor networks because they have scarce resources especially in terms of energy. Just sensing and sending all the data to an external source might seem more appropriate because this raw data can be further processed on more powerful machines with more resources. However, it is considered inefficient in WSN because of the transmission cost in terms of resources. Therefore there is a trade-off between sending raw data and processing it on less powerful processors.

- Robustness and self-management.

Wireless sensor networks can be deployed in difficult terrains where they are subject to harsh conditions and human intervention is impossible or difficult. They can have wireless communication with their neighbours and operate on batteries. The availability of the nodes may be very low due to harsh conditions or devices running out of batteries. Therefore systems designed for them should be robust so that they are able to work even with low node availability. Also the network topology can change frequently because of node mobility or nodes becoming unavailable, for example, in real-time environments therefore the system should be able to self-manage.

- Minimum hardware and software requirements.

Wireless sensor networks can be deployed in ad hoc environments, for example, in battlefields or difficult terrains with no infrastructure. Therefore the network should be able to start functioning with minimum hardware requirements. As sensor networks operate in such constrained environments with stringent resource constraints, the software requirements for network deployment and functioning should also be kept to a minimum.

- Collaboration of resources.

It has already been mentioned in the previous requirement that WSN may have to work in constrained environments with limited resources therefore the middleware should perform different tasks in a distributed manner. The middleware should dynamically distribute processing power between more capable nodes within the network. 'Capable' refers to nodes that have more battery power and other resources such as memory or processing power. Collaboration of resources of all the nodes will not only prolong lifetime of the network but also ensure that data provided by sensors is available for a longer period of time.

- Latency.

The data provided by the sensor network should not be stale, which means the system should be able to transfer the required information in time. Latency is an important issue in real-time and monitoring applications. Turn around time of a query or the processing of data or events should be within the time frame required by the application otherwise the information may become stale. Therefore the middleware should be designed to be able to process and send information in the required time frame.

- Aggregation.

In sensor networks many sensors may be sending readings of equivalent value, which implies redundant data. Aggregation means data obtained from different sensors can be aggregated before transmission. Aggregation is considered vital for WSN because it saves energy by reducing the data sent to the user thus saving resources. Again there is a trade-off because all sensor data could be sent to powerful machines to be processed but due to limited resources it is not possible. Therefore the middleware should provide this service because it is essential for WSN.

- Data Storage.

Middleware for WSN should be able to store data permanently in the network. Storing data within the network will not only help to process the data within the network but also create implicit relations with different events happening at different geographic locations or times and provide historical data. Storing data on the sensors will further help in reducing communications by sending only relevant or required data outside the network thus saving resources and providing meaningful and precise information.

- Context.

Context, as already defined in the previous chapter means any physical or surrounding aspect, for example, location or date and time. Context provides more meaning to data. Middleware should also allow applications to add context to data because it not only gives meaning to data but also helps in establishing explicit relationships between data.

- Event Based and Query Based.

Applications typically either are event-based or query-based. Event-based models are considered closer to the characteristics of sensor networks because sensors only sense events when they occur and mostly remain in sleeping mode thus saving energy. Some applications may require further queries in the case of an interesting event; therefore middleware should be able to handle both queries and events.

The middleware framework in this thesis has been developed to incorporate all the requirements mentioned above because they provide a basis for a more generic approach. The focus of the framework is to make it adaptable so that the user is able to easily register an event or pose a query whether historical or real time and store the data for longer period of time along with some context. The framework is a step forward in terms of adaptability but not the final step. There are other important issues such as security, the intension is not to undermine the importance of security in this framework and therefore discussed in the section future work of the last chapter. Security could, however, be provided at the application level, for example, authentication of user when using the network. The next section is dedicated to the proposed framework and its details.

4.3 An Overview of Proposed Framework

This section introduces the proposed middleware framework called MidWSeN and also describes the services it provides. MidWSeN is based on a service-oriented architecture concept with minimum hardware and software requirements. Service–oriented architectures loosely couple software services and these services can be accessed without prior knowledge of the underlying platform implementation. In addition they are also not tied to any particular technology and both the applications and services can communicate without any knowledge of each other. Therefore this particular architecture is being used as it fulfils some of the requirements of the proposed framework. For example, services can be added or removed without affecting the overall integrity of the network or the framework. Also the framework can adapt to any type of environment without restricting the middleware to particular hardware or software requirements. In addition the framework will also be able to operate in a heterogeneous environment having nodes of different types and capacity.

It is assumed that the nodes will have some operating system and they can communicate wirelessly. The middleware can be added on to the nodes on top of the operating system. The term middleware means by definition that the middleware will exist between layers of software. The middleware will work below a software layer and some software will work below this middleware. Therefore, some design decisions lie outside the scope of the middleware. Sensors could be working in different environments. Sensors could either be working in a pure sensor environment, where only sensors collaborate and form a whole network or they can work in a heterogeneous environment, where sensors are collaborating with other, more capable devices such as desktop computers or PDA. In the case of a pure sensor environment, one or at most two services can be kept on a single node. In the case of a heterogeneous network, more services might be present on a single node depending on the capability of the node. Although no specific type of hardware or software is being recommended for the middleware, the decision depends entirely on the environment being used including hardware and network requirements. Thus choices, for example, what type of network is being used may already be made. Therefore this section only discusses the design decisions that might affect the working of the framework. All other decisions which are not discussed do not affect the working of the framework.

A simple mechanism of message passing is being used for service discovery. The request for a specific service in the network will be broadcast and on availability of the required service on any node or nodes will send an acknowledgment back to the node initiating the request. If there is more than one service available then the service sending the request will decide by selecting the required service on the basis of hop count, location and available resources, for example, energy level or memory. Hop count, location and resources, such as, energy are the most important pieces of information required in the selection of the best available service. Hop count indicates the shortest or longest path; whereas location shows the geographic position and resources will indicate, for example, the energy level of the corresponding node. The decision of which parameters (hop count etc.) to be used for selection of a service depends on the application requirements. The selection criteria can be made based on all three of parameters mentioned above, or any one or a combination of them. For example, if energy level is low then the chances of selecting such a node are highly unlikely, similarly a service situated too many hops away may also be infeasible. There are, however, situations where an application might want data from a specific area of the sensor network in which case location will be considered more important.

There are other service-oriented software platforms also available, for example, Jini (Sun Microsystems 1994) and Jxta (Wilson 2002) that works very well in the traditional networks and can work equally well in a wireless environment. However, from a design perspective they are not well suited to WSN because they require a level of resources, which this kind of network cannot afford. They might work well in a heterogeneous environment where there are more capable nodes available, especially Jxta because it uses a peer-to-peer paradigm (Oram 2001) which can work very well in WSN having a distributed approach in which a peer should only know its neighbours. An alternative is Jxme (Sun Microsystems 2005) which is a micro edition of Jxta developed especially for resource constrained devices, for example, smart phones or PDAs. Jxta works well with ad hoc networks or heterogeneous environments but is still heavy in terms of resources for a network consisting only of sensor nodes. Since one of the design goal of the proposed framework is that the middleware should work equally well in a resource constrained environment consisting only of small sensors, the above mentioned approach of a simple message exchange have been taken.

Data-centric design is being used because data gathering is the primary goal of sensor networks. Data seems to be at the heart of all the applications used in these types of networks. The applications in WSN are collecting, storing, and aggregating data. Everything revolves around managing, integrating and transferring data. Therefore as mentioned in the previous chapter the best option available so far for WSN is data-centric design because of its application knowledge, it proves more energy efficient. Data-centric design provides all or most of the answers required by the applications. This type of design is not only energy efficient but improves latency by processing the data quickly near the source. In the previous section it has been noted that latency is an important issue in real-time environments and monitoring applications. There are also other features of this design, which are important in sensor networks; for example, data-centric design saves bandwidth by doing in-network processing. In-network processing is a desirable feature and mentioned as an important requirement in the previous section within a network having scarce resources. Furthermore, data-centric design has application specific knowledge, which makes it closer to WSN characteristics. It should be noted here, however, that there is a difference between having application knowledge and being application specific. In data-centric design the features of interests are known in advance, therefore, it can save data using the data names, for example, temperature or pressure and store data accordingly. Whereas application-specific means that the whole network is designed to consider only a single application or only one type of applications which makes the network design limited. Therefore, having application knowledge does not mean that the network design cannot be generic.

Data-centric design has a disadvantage, however, that it does not work well in environments where the nodes are highly mobile or node failure rate is high (Ratnasamy *et al.* 2003). This problem can be tackled by having multiple copies of the services available within the network. The proposed design recommends multiple copies of services as greater the number of copies available, more reliable the system becomes.

This is a trade off because redundancy is being recommended in order to make the system robust and reliable. The number of copies depends on the network size as well as the percentage of node availability. In a network where there is a risk of a high percentage of nodes becoming unavailable, more copies of services will be required.

TinyDB (Madden *et al.* 2005) and DSware (Li *et al.* 2004) also use the data-centric approach. However, TinyDB uses a tree type structure that is very simple to implement and, as already explained in chapter 3, raises scalability issues and also TinyDB cannot answer queries having spatial and temporal entities. TinyDB also does not consider the robustness problem mentioned above, namely in case of highly mobile nodes or higher node failure rate. DSware on the other hand does tackle this issue by having more copies of data stored at one time but it updates or synchronizes the copies only when network traffic is low which might result in loss of data. If the copy having the latest data is unavailable before updating the other copies that data will be lost forever. In addition both of these solutions store data temporarily or for short periods of time, whereas the proposed framework is storing the data for a lifetime of the network which makes spatial and temporal queries possible in addition to saving resources spent on sending all the data to the sink. The proposed design also has multiple copies of data but all these copies are being updated simultaneously instead of waiting for low network traffic. Again there is a trade-off in order to ensure availability of updated data at all times. The service-oriented architecture makes the proposed system more scalable and robust. Having multiple copies of services available throughout the network ensures other services can takeover easily in case of a service failure and also more copies of services could be added without affecting the network

The proposed framework can be implemented both as a whole and in parts. The reason for this is to keep the framework adaptable and flexible depending on the application requirements and resources available as mentioned in the requirements section above. There is a trade-off between efficiency and flexibility because middleware specially developed for specific applications would be more efficient. Adaptability is important to integrate and bind sensors into heterogeneous environments in which they have to talk to each other across the network or where many applications are sharing the same network.

In summary, middleware for sensor networks should be event based as this best matches the characteristics of WSN (Römer *et al.* 2002) because their primary aim is to sense events. In addition to being event-based, the middleware should also be able to handle queries that applications might require and also enable reuse of the sensors for different applications. As data will be stored on the network, other applications can access this data by query. The systems exclusively developed for specific applications will be more efficient in performance but are restricted to only a few applications whereas the framework proposed in this thesis will be more adaptable and flexible enabling it to work with many applications. The advantage of the proposed middleware is that application developers will not have to worry about the type of middleware to use each time they write a new application.

The background chapter explained that in event-based systems there are no provisions for making queries in emergency situations or investigating interesting events directly or more closely. One novelty in MidWSeN lies in the fact that it not only allows users to register an interest in an event but also query the network whenever required. Furthermore users can also add context to the data by defining their requirements all in the same framework.

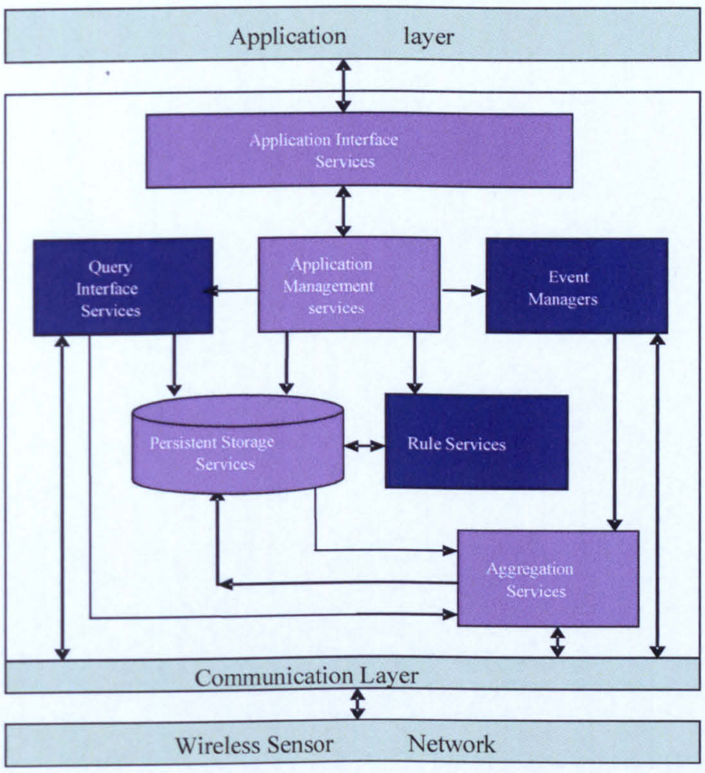


Figure 4-1 Middleware Architecture of MidWSeN

Figure 4-1 gives an overview of all the services presented in the proposed framework and the relationship between these services. At the top of the diagram is the application layer which represents the actual application. The communication layer represents the routing mechanism and other techniques used to facilitate communication among the nodes within the network and with the base station. The framework is divided into two different types of services; core and optional services. The core services are necessary for the complete functionality of the framework and will always be available. The Application Interface (AIS), Application Management (AMS), Persistent Storage (PSS) and Aggregation Service (AS) are the core services. The rest of the services, for example, Query Interface (QIS) and Event Manager (EM) are optional services. The optional services may not be required at all times and therefore not be available under all circumstances. Optional means that either one of the two services the Query Interface Service (QIS) or the Event Manager (EM) can be used or at least one must be present otherwise the system will not function. Rule Service is completely optional, however, and its presence is dependent entirely on network requirements. The long rectangle boxes are the actors with which the proposed middleware interacts, for example, the sensor networks and communication layer which consists of routing, etc. The arrows represent the interaction and direction of data flow between two services. The reason for having two types of services (core/optional) is to make the framework adaptable to changing environments as has been mentioned in the requirements. This flexibility could be twofold, either the applications do not require a certain service or the resources available cannot support one or more of these services in a particular environment or given time.

Flexibility is important in WSN environments because it may be a pure sensor network working outdoors in a remote area without any kind of human intervention which means limited resources like battery power, small memory or bandwidth. Alternately the environment might be heterogeneous with powerful nodes, for example, pocket computers or laptops having more resources as part of the network. Human intervention may also be possible which means that batteries can be changed whenever possible or more nodes may be added to the network frequently. The scale and density of the network also has an affect because in a large or dense network there is much better utilisation of resources by distribution of sensing and processing tasks. Some of the sensors are sent to sleep while the rest of the sensors continue sensing until their batteries are depleted and then the sleeping nodes can take over when ever required. Depending on the scale of the network multiple copies of services could be available.

4.4 Core Services

This section introduces the core services of the proposed framework MidWSeN. These services are important because without them the framework will not be able to function. The availability of these services is vital at all times. All these services except for the application interface service will reside within the sensor network on the sensor nodes. These services might be on different nodes but it is also possible that some of them might be on the same node. It really depends on the capacity of the nodes. There might be more than one copy of these services residing on the network. This would be decided based on the rate of node availability and the size of the network. If the node availability rate is low more copies of these services will be required. Similarly, for a large network more copies will be required. This phenomenon will be explained in the next chapter, as well as in the Performance Evaluation and Implementation chapter in some detail. The rest of the section explains the functionality of these services individually.

4.4.1 Application Interface Service

The application interface service (AIS) provides the communication link between the application and the middleware by converting the format to/from the application and middleware. This service may reside on a PC or it may be a web application connected to the sensor network. This depends entirely on the application and is implementation dependent.

The AIS will receive new requests from users or applications and forward them to the appropriate service in the WSN. If applications want to register new event listeners the AIS would forward such requests to the Application Management Service (AMS), which in turn would be forwarded to the Event Manager (EM). If the event occurs the EM will notify applications through AMS and AIS. The AIS will also direct queries to the AMS in case of any historical information from Persistent Storage Service (PSS). The AMS will direct queries to the PSS in order to get results from it and pass them back to the registered application through the AIS. Alternatively if applications want any real time information from the nodes directly, queries would be forwarded to the Query Interface Service (QIS).

The proposed middleware framework does not limit the user to any specific language. The user will be able to use different types of query format that can capture their requirements and a parser can extract the required information to pass the extracted parameters to the AIS in the message payload, which in turn will send it to AMS. Whatever query or event format the application use can be described as a struct, for example, in the header file and passed on to the method *SendMessage*( ) which passes it on to the middleware. The middleware basically gets a list of parameters and some meta data explaining that data. This would depend entirely on the application and may vary from application to application.

Until now Middleware systems have limited the user to SQL or SQL-like languages or some specific languages (Bonnet et al. 2000; Jiao et al. 2005; Madden et al. 2002). Such binding restricts the user to a language which may not capture the users requirements, therefore, the AIS does not restrict the user to a specific type of query language. Tools such as PS/SQL are available which have the ability to convert code developed in them to directly to C language code, which can be compiled and downloaded easily to the motes or sensors in its binary format. There is a potential danger, however, that code converted directly from a higher language to C might create some sort of infinite loop or loops which might go on undetected for a long time utilizing precious resources thus spending a lot of energy. Therefore, the proposed system gives freedom to the application developer to use any descriptive language that fulfils their requirements. However, middleware will extract the parameters and pass it on instead of directly feeding the code because of the catch mentioned above in directly feeding the code to the sensors. So it is safer to change it into a standard format rather than doing something that might give unexpected results.

The AIS has the following methods

- int SendMessage(int destadd, int msglength, struct Message *msg); this method sends the parameter list of the query or event to the AMS. This method gets the parameters list after the parser extracts the parameters from the query or event and forwards it to the Application Management Service. For example, in case of MidWSen the struct would be as follows:

  ```
  struct Message {
    struct  *msg_payload;
    uint  *msg_priority;
    uint  *msg_querytype;
    }
  ```

- int *DiscoverService*( ); this method discovers the appropriate service in this case it will always be the AMS. This method return 1 if an acknowledgement is received otherwise 0.
- int *RecieveData*(int data): this method receives data from the AMS.
- int *RecieveMessage*( ): this method receives message from the AMS

4.4.2 Application Management Service

The application management service (AMS) sort different requests from different applications. The novelty of AMS lies within the fact that it optimizes the selection of services within the sensor networks. It has been mentioned before that in-network processing reduces energy consumption considerably. The Application Management Service (AMS) will reside on sensor nodes within the sensor network. This important service works as a gateway or link between the application residing on the sink, which might be a more capable device such as a PC, and other services within the sensor network. All queries to sensors or applications registering new event listeners will be directed through this service to the correct destination. Figure 4-1 shows that this service communicates with the Query Interface Service (QIS), Event Manager (EM), Rule Service (RS) and the Persistent Storage Service (PSS) in addition to the AIS.

Service optimization is done through exchange of messages between the AMS and different services, such as the QIS or EM, which may be in XML or TinyML (Ota *et al.* 2003) or SensorML (VAST 2007) format. The operating system usually provides the mechanism for passing messages to the sensor nodes so this is an implementation dependant process. In order to successfully pass messages between a source and destination the following elements are considered vital. The first element is explicitly specifying the source and destination of the message in this case the address of the sender node and the receiving node. The next important factor is how to provide buffers to the incoming messages and also when the storage associated with the source message can be reused, which, in WSN is very constrained. The last element is the processing of incoming messages. For example the Active Message (AM) model in TinyOS, names a handler that will be invoked and process the message on the recipient nodes. TinyOS message buffers own the send buffer after accepting the send command until the send is complete which is notified by a send done event thus following a strict ownership protocol. The storage management on the recipient node is handled dynamically by TinyOS. When a message arrives, it fills any available buffer and the Active Message layer of TinyOS will decode the handler type and send it to the intended destination. The *SendMessage*() of the AMS will get the parameters list from AIS and forward it to either the Query Interface Service or Persistent Storage or Event Manager.

The information required for service optimization will be provided by the sensors in the acknowledgement message sent in response to the service discovery message broadcast by the AMS. These messages will have the required information; for example, in the case of service selection, they are distance of the service from the event area or the shortest possible route, hop count, resources (such as, memory or energy) and location, which will help the AMS to select the appropriate service. When the AIS sends a query it also informs the AMS whether it is an event or a query and also in case of a query whether it is a historical query or a real time query. The information is passed on in the message as separate fields showing the required parameters. It will send a new event listener request from the AIS to the event manager (EM). In case of a query the AMS will decide whether to request data from PSS by looking at the message and decide by the *querytype,* which will specify if it is a historical query and send the query to PSS. The AMS will also retrieve the results back from the PSS and forward it to the AIS.

The AMS also has another important function. The application might have a set of rules, which are important for the normal functioning of the system; these rules will reside in the Rule Service (RS) which is explained in the next section. If a single rule or set of rules fire then the RS will inform the AMS which will in turn decide if a vital violation of the rules has occurred, for example, if it is an emergency. This information will be provided by the developer when specifying the rules that indicate what action has to taken if a certain rule or set of rules are violated. The AMS will inform the user through the AIS and start querying for further information on the applications request. The user will be able to query the sensors directly for further information through the QIS. The whole process is shown in detail in Figure 4-2.

The AMS has the following methods.

- int *SendMessage* (int destadd, int msglength, struct Message *msg); this method gets the parameters list from AIS and forwards it to either the Query Interface Service or Persistent Storage or Event Manager and will return a value of 1 if the whole message was successfully sent over the network otherwise it will return 0.
- int *DiscoverService*( ); this method discovers the appropriate service by broadcasting a message to the network depending on the return value of method

Service or Persistent Storage or Event Manager and will return a value of 1 if the whole message was successfully sent over the network otherwise it will return 0.

- int *DiscoverService*( ); this method discovers the appropriate service by broadcasting a message to the network depending on the return value of method Decide( ) and would return 1 if an acknowledgement is received otherwise 0. It does not accept any parameters.
- int *GetData*(int data); this method gets data from the QIS or PSS returns 1 upon successful completion of the process otherwise will return 0.
- uint *Notify*(struct Event theEvent); this method notifies the AIS in case an event happens or a rule is violated will return a value of 1 in case of an acknowledgement otherwise 0. Event could be defined in C as a struct as mentioned above. For example:

```
struct Event {
        long time;          /* When this event occurs. */
        char type;          /* What type of event. */
}
```

- int *SendData*(uint data); this method sends the data which is of type unsigned integer back to the AIS.
- int *Decide*(uint querytype); this method decides based on querytype where to send parameter list either the QIS or PSS or EM. If the querytype is equal to 0, for example, it means that it is of type event and it will return 0, if it is a query then it will return the value of 1 and in case of historical value it will return 2.
- uint *SendQuery*(int destadd, Message *msg); this method forwards the query to the QIS.
- uint *RegisterEvent*(int destadd, Message *msg); this method forwards the event to the EM.

4.4.3 Persistent Storage Service

The persistent storage service (PSS) first introduced in (Javed *et al.* 2005) stores data provided by the nodes for the lifetime of the network. This service resides within the network. The reason for this storage is twofold; first the data can be used for further processing and second being the data could be utilized to make the system self-adaptive through past information. This service would record all the data given by the sensing nodes for the lifetime of the WSN with additional user defined context wrapped around it. A detailed account of this service is given in chapter 5.

4.4.4 Aggregation Service

In sensor networks many sensors will be sending readings of equivalent data, which implies redundant data. Due to the limited memory and energy constraints it is difficult to store so much data; therefore this service will aggregate data coming either from the EM or directly from the nodes. The novelty of this service is that it will aggregate the data with context provided by the user and as well as save memory by reducing the volume of the data being stored. This will help the user to query data with context. The aggregation service (AS) (Javed et al. 2007a)can be used at both the node level and at the data storage service level depending upon the specific application. As already mentioned in requirement analysis section, in-network processing will be used for this service.
The Aggregation Service provides the standard *Max, Min, Sum, Count* and *Ave* functions. The developer can choose the method or methods best suited to the application. Further, these methods can be adapted if required. Acknowledging the fact that aggregation could be application dependent another method called *MyAgg* is being added to the service. In this method the application developer can write his own specific type of aggregation, which might be specifically required by the application other than the ones provided. This adds flexibility to the middleware and provides freedom to the application of choosing its own aggregation service. This service is explained in greater detail in Chapter 6.

4.5 Optional Services

This section explains the optional services used in the proposed framework. The user can reduce the size of the middleware by choosing between the Event Manager (EM) and the Query Interface Service (QIS), depending on the requirements of the application or use both, subject to the availability of the resources within the network. At least one of the QIS or EM will always be made available. Alternatively, if the environment cannot support the complete framework than it is possible to reduce the size of the framework by removing some of these services without affecting the function of the system, for example, Rule Service (RS). Ideally the middleware will work with all of the services mentioned in the proposed framework but to add flexibility and make the framework adaptable to different environments some of the services have been kept optional. The rest of the section explains the optional services individually.

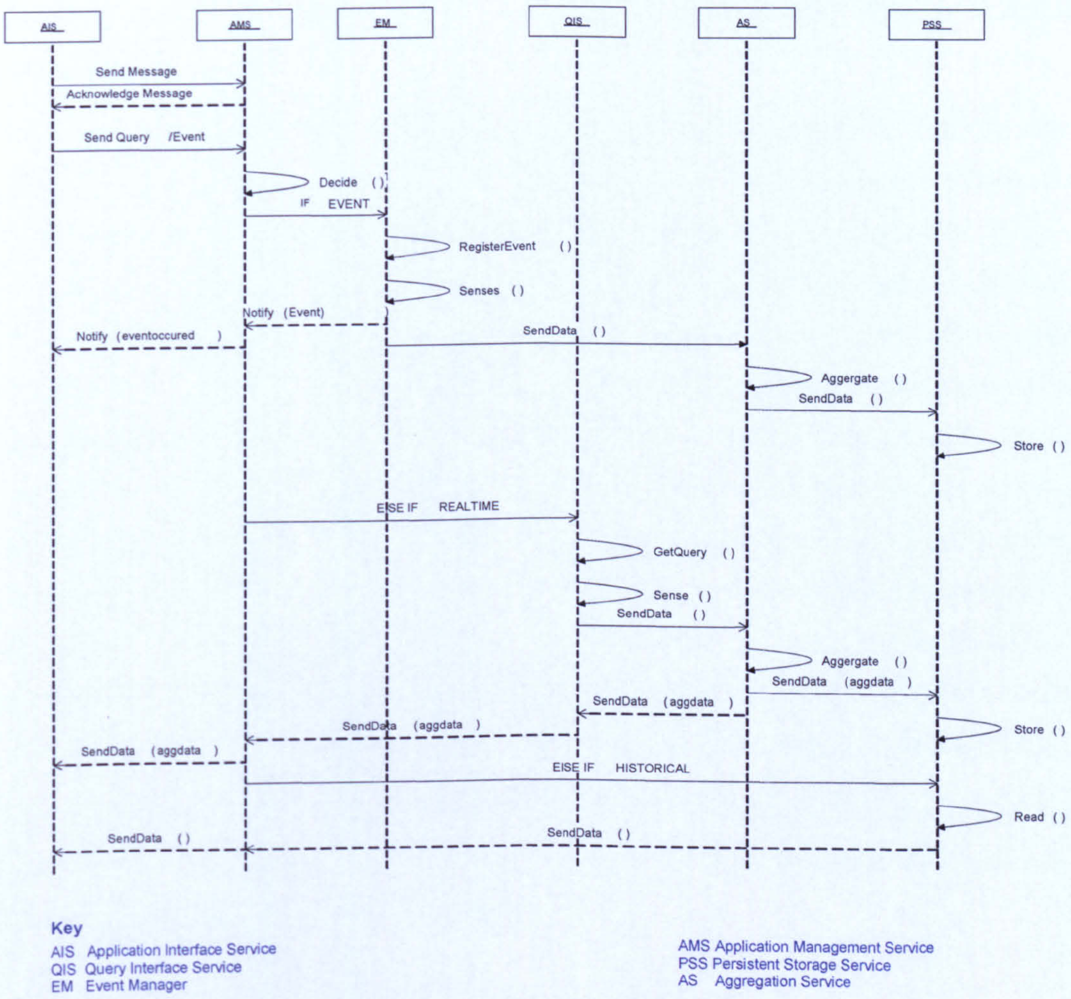


Figure 4-2 Sequence Diagram for MidWSeN framework

4.5.1 Rule Service

The Rule Service (RS) is used to detect abnormal behaviour in the environment. If there is unusual behaviour which the sensors can sense, for example, there is a sudden rise or fall in the temperature like it goes beyond 100 degrees Celsius or falls below 0 degree Celsius the RS will be able to detect such behaviour using the rules specified by the developer. These rules will be checked at a period specified by the developer. If these rules are violated the service will inform the user through the AMS. The user can investigate the situation further through the QIS by directly querying the sensors. Rules can be defined using any type of query language, for example, SQL or SQL-like languages.

Context awareness is another important aspect of the proposed system in order to make our system self-adaptive. We are using the RS and AS to add context to the data. The difference between the existing approach (Baldauf et al. 2007; Choi et al. 2005; Fahy et al. 2004) and proposed approach is that they are using a centralized approach in which there is a centralized server, which stores context data and serves information to client applications. The rest of the contents are distributed whereas the proposed solution is using a completely distributed approach, which means it will use more than one context server. The AMS and the PSS together act as a context server and the knowledgebase, while the RS supplies the rules. Thus, knowledge base and rules will be separate and the decision of user involvement is left to the developer as to who will make the changes in the rules, the user or the developer because functionality may vary from application to application. This is again different from the current approach, which makes the system more flexible because usually the middleware makes decisions about user involvement.

The rule service is included for completeness of the proposed framework; however, our main focus is on the other two core services, the Persistent Storage and Aggregation Services being the foundation of the framework.

4.5.2 Query Interface Service

The design goals mention that the proposed middleware should be generic and flexible and so an important requirement of the proposed framework is to allow the application to query the nodes directly, therefore, the Query Interface Service (QIS) is provided. This is an optional service, which means it may be used only if the application requires, for example, in an emergency situation and if the application needs to further investigate an interesting phenomenon. The QIS may be used to receive real time queries sent towards the nodes directly where the user wants additional information quickly from the sensors. This will not only reduce latency in real time situations because the data will be sent back directly to the application as soon as the sensors are able to provide the required information but also provide additional information which could be very useful in a critical situation. This service is provided within the network and the network must have enough resources to support it, therefore it has been kept as an optional service. The application can choose between the QIS or Event Manager (EM) depending on the requirements but it is important that at least one of the two services is always provided. A network with more resources or a large network that can pool more resources will be able to support both services (QIS and EM).

This service is an interface therefore; the developer will provide the application-dependent implementation. It has the following methods;

- int *GetQuery*(struct Message *msgpayload); this method gets the query or list of parameters sent by the AMS and sends it to the sensors.
- int *SendData*(int arrdata[], int arrSize); this method is used to send data to the aggregation service as the data comes in the form of streams from sensors therefore arrays are being used and also the size of the array is not known in advance therefore the size of array is passed as a parameter. This method will return 1 upon successful completion of sending data otherwise 0.
- int *SendAggData*(int data); this method sends the aggregated data back to the AMS. The aggregation service will aggregate the data sent by the sensors and return the aggregated data back to the QIS, which will send this aggregated data to the AMS.
- int *SenseStart*( ); this method tells the identified sensors to start sensing. No parameters are passed to this method it will start sensing and return the sensed data.

- int *SenseStop*( ); this method tell the sensors to stop sensing.

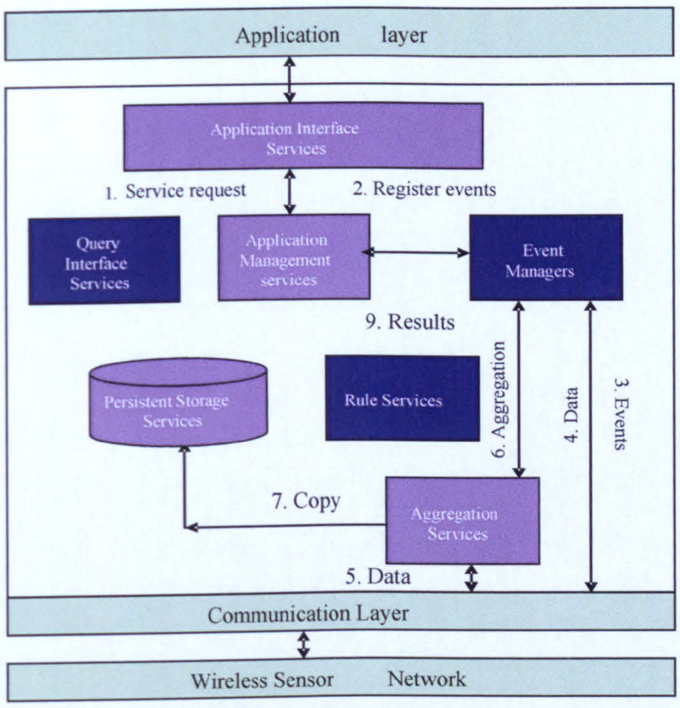


Figure 4-3 Event listening and pulling data from sensors

4.5.3 Event Manager

It has been mentioned in the requirements analysis that the system should be event based because such a model is closer to the characteristics of WSN. The Event Manager Service is used to register and control different event listeners from different applications. The EM will register new event listeners in order to monitor certain events. It will receive notification of events when they occur, sort them according to the registered event listeners and send it to the aggregation service. It will also notify the applications that registered the event-listeners through the AMS and the AIS. The EM is also an optional service and resides within the network. As mentioned in the previous section either of the two QIS or EM or both can be provided depending on resource availability and application specifications but at least one of the two services (QIS and EM) will always be available in the network. This service also works as an interface as such the implementation will be provided by the developer and have the methods outlined below.

This has already been mentioned in the AMS section that the most popular operating system TinyOS, which is an event-based operating environment automatically, provides a handler that will be invoked and process the message on the recipient nodes. In case of such an operating system the method *RegisterEvent*( ) will just forward the message to the AM layer of the operating system and the rest of the event handling will be done by the operating system itself. Most of the operating systems for sensor networks provide mechanism for event handling because it is very close to the nature of sensor networks. Otherwise if the operating systems do not handle events then the *RegisterEvent*( ) method will provide this mechanism.

- int *RegisterEvent*(struct Message *msg); this method will register any event listeners sent by the AMS and forward it to the sensors. This method will assign event handlers to individual events. Only one event handler can be active at a time.
- int *SendData*(int arrdata[], int arrSize); this method performs similarly as the method described in the QIS and sends the data coming from the sensors to the AS.
- uint *Notify*(struct Event theEvent); this method performs similarly as the method described in the AMS above and notifies the AMS in case an event occurs.
- int *SenseStart*( ); this method tells the identified sensors to start sensing and returns the data.
- int *SenseStop*( ); this method tells the identified sensors to stop sensing.

4.6 Summary

This chapter presents a novel middleware framework, MidWSeN, for WSN. Requirements for the framework and its overall architecture have also been discussed in this chapter. The requirements explain that the system should be adaptable in order to have a general middleware due to the unique characteristics of WSN being application specific. Different tasks should be performed in a distributed environment using a minimum of software and hardware in an energy efficient manner. The middleware should also be able to improve latency in real time situations and save energy and bandwidth by doing in-network processing. The middleware should be able to provide context aware historic data about previous events in aggregated form.

The research in this thesis has tried to address all these issues. It presents a novel framework called MidWSeN described above which combines three important features query, events and context-awareness in one. Existing frameworks have concentrated on one feature out of the three mentioned above. The individual services defined within the framework are also novel being designed differently and have added flexibility. The framework is flexible having core and optional services which means that if some of the services are not required then they can be omitted from the final configuration which will further reduce the size of the middleware, so the size of the middleware is also adaptable in case of constrained resources, such as, memory. Optional services mean that the user can select one service out of the two optional services either EM or QIS or both if required and enough resources are available but at least one of the two services will always be provided within the network. The RS is also optional and can be excluded if not required thus reducing the size of the middleware.

The Application Interface Service does not restrict users to any particular software format whereas other existing works restrict them to SQL or SQL-like languages. The AIS provides the communication link between the application and the middleware. The AMS sorts application requests and sends these queries forward to the required service. The PSS stores the data provided by the sensors for the lifetime of the network. The AS will aggregate data according to context identified by the user coming from the sensors to reduce the amount of data being stored. The originality of our aggregation service is that it aggregates the data with the context identified by the user. The aggregation service also provides aggregation at two different levels, the node level and the data-storage level, again dependant on application requirement. The RS will check the system against a given set of rules. Developers can also choose between the general rules provided by the framework or specify their own for the rule service, fulfilling specific requirements of the application. Context management is distributed as compared to the centralised approach in previously published work. The QIS will enable the application to query the sensors directly in a real time situation to improve latency. Applications can also query the PSS whenever required. The EM will register new event listeners to monitor any events happening in the lifetime of the network.

This chapter has explained the overall design of the middleware framework in some detail but as mentioned earlier two services in this framework the Persistent Storage Service and Aggregation Service need further explanation. These services are an important part of the foundation of the proposed framework therefore the next two chapters are dedicated to a more detailed discussion of them. The following chapter defines the PSS in greater detail and also shows how it interacts with other services. In addition, to further economize the use of memory a prioritization algorithm is also explained.

Chapter 5 MIDLLEWARE PERSISTENT STORAGE SERVICE

5.1 Introduction

It has been noted in the previous chapters that information gathering is a primary function of WSN. The approaches described in published literature to date simply sense data and send it to the base station for further processing (Ville *et al.* 2003) or only partially process data within the network (Li *et al.* 2004; Madden *et al.* 2005). It has also been noted previously that in-network processing utilizes less energy than transmitting all the data to the base station as transmission typically consumes much more energy than normal processing.

This chapter discusses the Persistent Storage Service (PSS) within the middleware framework (Javed et al. 2007b). This service records all data given by sensing nodes with a unique identification for the lifetime of the WSN with contextual information within the network. The novelty of the PSS lies in the fact that data is being stored for the lifetime of the network on sensors and by performing all or most of the processing within the network utilizing fewer resources in comparison to sending all the data to a powerful base station for processing. As mentioned in previous chapters and above this has not been done before in WSN because the sensors did not have sufficient memory. Improvements in technology have made it possible to increase and improve memory resources, while still very small; the ability of sensors to collaborate (Janakiram et al. 2005) and share resources with each other to perform a task also enables them to pool their memory together to store data by using different techniques, for example, peer-to-peer network techniques(Subramanian et al. 2005), data-centric storage (Ratnasamy et al. 2003) etc. In a distributed environment hundreds or even thousands of sensors can collaborate with each other to pool enough memory to store more data for longer periods of time. Looking at the history of the development of sensors it is evident that sensors have become more powerful with added ability of processing and memory rather than having just reception and transmission capabilities. With the advance of technology, for example, the new NAND flash memory technologies (Mathur et al. 2006), it can be predicted that Moores Law will prevail and processing power and other resources will continue to improve with the passage of time. This has already been mentioned in the introduction chapter that in the new imote2 sensors (Intel 2007) Intel has increased RAM to 32MB together with improved flash memory of 32MB bringing down the cost as well as improving efficiency due to the new NAND technology.

The storage capacity of sensors is limited and there might be a situation where new data needs to be stored but the required storage space is not available on the network. To handle such a situation a new prioritization algorithm for memory management that will only store higher priority data to accommodate important data is also being proposed and explained in this chapter.

The aim of the persistent storage service presented in this chapter is from the middleware perspective and not the database view. Therefore the service does not provide any database details such as the exact format of the data. Although suggestions have been made wherever necessary and those aspects which are important from the framework's point of view are also discussed in detail such as data centric storage but there are other details which the developer or user will have to provide. The intension is to collect different components already present, combine them and adapt them to the new framework requirements in a way which makes it novel, rather than building a completely new system from scratch.

The rest of this chapter explains the Persistent Storage Service in detail. Section 5.2 explains why it is necessary to save data on sensors for the proposed middleware. Section 5.3.1 explains the data-centric technique for storing data within the network. Subsection 5.3.2 explains other services used in conjunction with the PSS. Subsection 5.3.3explains the detailed design of the service. Section 5.4 gives a detailed case study. Section 5.5 introduces an algorithm to improve memory utilization by using the prioritization technique. The final section 5.6 gives a summary of the main details presented in this chapter.

5.2 Persistent Storage as a Middleware Service

This section explains the advantages of storing data within the network. Storing data within the network for the lifetime of the network means historical data is available. Storing data with additional contextual information means more meaningful data is available and allows the user to make spatial and temporal queries. This could help in many applications, for example, when tracking mobile targets (Abdelzaher et al. 2004; Zhang et al. 2003) historical data will show not only that the target is moving but also in which direction it is moving. In animal tracking (Sikka *et al.* 2006) data storage will help in identifying the route and this helps to predict the direction animals are taking. In addition, other services within the framework might also be required to be applied to this data, for example, Aggregation Service (AS) will help to aggregate redundant data thus saving memory by only storing aggregated data.

The interaction of the PSS with other services within the framework and processing the data on the sensors within the sensor network means only meaningful and complete information will be transmitted to the application. The provision of complete information leads to a reduction in the number of transmissions, which saves energy and other precious resources. Not only will information be complete but it will also improve delay in getting information. Sending raw data or partial results to the base station is time consuming because first all the data has to be stored in the base station and then analyzed after completion of data transmission. In MidSWeN, the availability of historical data and in-network processing means data will be analyzed quickly and a complete answer sent back, accelerating the whole process. Therefore storing and processing the data within the network will not only save resources but also improve latency. It has already been mentioned in the requirements analysis given in the previous chapter that latency is an issue in these kinds of networks especially in real-time applications.

Another advantage of storing data within the sensor network is that it can also develop implicit relationships. It should be noted here that developing implicit relationships outside the sensor network is also possible but all data needs to be transferred first to the base station, which will be time consuming and also resource intensive for the sensor networks. Factors which might not be directly associated with a particular event or may not have a direct affect on the event but are associated with some attribute of an event. For example, assuming scientists are monitoring the number of explosions of different intensity of a volcano during 24 hours. If they are also recording the time of day the explosions happened and it turns out that these explosions are mostly happening during daytime. The scientists might start thinking that the sun has something to do with it although there is no direct connection between the two them. They might then start querying past instances of such occurrences. Therefore storing the data will not only improve resource utilization but it will also improve the quality of information being transferred finally to the user by helping them to create implicit relationships.

The data stored on these nodes can also be utilized by different applications. For example, if the same temperature data is required by two different applications then this data can be shared by querying the PSS. Hence, this will not only help in sharing the data by multiple applications because the required data is already stored in the network, but also saving resources because the sensors need not sense or store the data again.

The above section discussed in detail the importance of having persistent storage within the middleware framework and the benefits of having historical data in WSN. The next section explains in detail how the data is being stored in the network along with the advantages and disadvantages of the data-centric design used in the proposed middleware. A solution to the difficulties faced in this type of design is also explained.

5.3 Persistent Storage

The above section explains the need for storing data within the network. This section explains how this data is going to be organized by the PSS from the sensing nodes with a unique identification for the lifetime of the WSN with user-defined context within the network. The PSS can be utilized by the applications to collect data in a specific region for a specified amount of time. This service is being used to improve the utilization of limited resources in sensor nodes. It has been noted above that transmission utilizes more resources than processing data near its source, therefore storing and processing data within the network will save resources by improving their utilization. Therefore data storage is an important feature of the MidSWeN.

The next subsection explains how data is stored within the network. It illustrates that data-centric storage which stores and processes data near its source is the best option because of its application knowledge. Further, the disadvantage of data-centric storage and its remedy is also discussed in the following section.

5.3.1 Data-Centric Storage

This section explains the answer to the next question that needs to be answered which is how all this data is going to be stored in the network. The sensor nodes might have different storage capacities and will likely all be limited. Nodes collaborating within a distributed network can solve this problem. The MidSWeN uses a data-centric (Ratnasamy *et al.* 2003) approach to storage. Data-centric technique processes and stores data near its source utilizing the storage capabilities of the neighboring sensor node, based on the assumption that the network will use data-centric design. This has been made clear in the background chapter that data-centric design is the best available choice because it has application knowledge which makes it more energy efficient (Intanagonwiwat *et al.* 2003).

Data-centric storage uses two primitives `Put(key, value)` and `Get(key)`. The `Put()` method stores data using a specific key which in data-centric storage is an attribute name, for example, temperature and `Get()` retrieves data from the key. These two primitives `Put()` and `Get()` are supported by a routing mechanism which uses low level naming and look up algorithms (Chord (Stoica et al. 2001), CAN (Ratnasamy et al. 2001), Tapestry (Zhao et al. 2004), etc.) which maps a hashing key to build a distributed hash table. When a node detects an event, which has been queried for, it sends the data from its sensor representing this event to the storage service. The storage service might store data on the same node if it has sufficient storage capacity or another neighboring node that has available memory capabilities. Data-centric storage allows data to be stored by name and both observations and events are named (Heidemann *et al.* 2001). The given event name is hashed into a key using lookup service algorithms (the peer to peer service algorithms are closer to sensor networks therefore are recommended such as Chord, Can, Tapestry) which should be a location within the sensor network and the Put(key, value) method stores the data at that particular location. Therefore, subsequent requests for data are sent to that particular node and the Get(key) retrieves the data from that location. Note that in data-centric storage the events are predefined by name, for example, temperature or pressure. This approach gives a name to each data item then uses communication abstractions or routing mechanisms that refer to those names rather than accessing nodes by network addresses. All data referring to the same general name is stored at the same sensor node or neighboring nodes near the source where the event was detected. Different nodes of the same type collaborate as a single logical node although physically they might be distributed along different nodes. For example, the temperature sensing nodes will sense and store the data on the same or neighboring nodes. Whenever there is a query about temperature it would be directed to the temperature sensors or neighboring nodes. The decision of storing the data on local node or neighboring nodes is taken on the requirements and the capability of the local node. Access to data is possible by building a distributed hash table (DHT) based on attribute names as already explained above. Moreover data which is queried very frequently can be kept in the cache memory of the base station which is assumed to be more capable in terms of resources such as memory, energy and processing power, to ensure quick processing which will further reduce the amount of energy consumed. As mentioned above Data-centric storage does require some sort of routing protocol that would support this kind of storage and take care of the communications between sensors. There are a number of data-centric routing protocols available which can handle named data (Intanagonwiwat *et al.* 2003).

Events may be of different types, such as, low level events which is usually referred to as observations in the sensor network literature and a combination of these low level events might lead to high level events, for example, individual animal sightings may be a low level event but a combination of these events might lead to a high level event of animal migration. There are a number of factors that might contribute to establishing a high level event such as location, time and number of sensors recordings of these sightings that might require interaction or collaboration among neighboring nodes or the whole network. There are different levels of interaction between nodes and data or information is exchanged at different levels. For a low level event the nodes might collaborate locally with neighboring nodes only but for a high level event the whole sensor network or large part of the network might collaborate or exchange information. Event detection (Kumar *et al.* 2005) for example may require processing of low level observations or results from several different sensors to produce high level events. For instance, a volcanic eruption is a high level event, which is a combination of other low level events, such as, increase in temperature, sound, pressure, humidity etc. This requires local collaboration between neighboring nodes only and uses local area dissemination techniques (Kang et al. 2006; Wang et al. 2004). In this form of collaboration information is accessed from neighboring nodes. In contrast there are techniques to access data both by nodes or users from anywhere within the sensor network, for example, comparing its temperature reading with average readings of the rest of the sensor network, so this requires wide area data dissemination techniques, for example, directed diffusion (Intanagonwiwat *et al.* 2003). There are several data dissemination methods available for both local collaboration and wide area data collaborative information processing. Some of them use data-centric techniques (Sylvia Ratnasamy et al. 2003) uses geographic routing technique GPSR and build a distributed hash table on top of this low level routing technique using peer to peer look up algorithms.

It is established that data-centric storage might not work in all situations (Ratnasamy *et al.* 2003), like environments with higher node failure rate or where the nodes are mobile. Therefore, the storage service replicates the data to many locations by multicasting (Sheth et al. 2003).Multicasting means that data would be sent to the intended recipients at the same time. Different copies are synchronized and updated at the same time using multicast. Multicasting do present some problems, for example, a computational overhead may be a burden on the resource constrained sensor networks or there may be scalability issues in large networks. Recent research (Sanchez et al. 2006; Wu et al. 2006) based on geographic locations (Sanchez et al. 2007), distributed routing tables and heuristics (Koutsonikolas et al. 2007) has managed to produce protocols which are light weight and energy efficient to overcome difficulties initially encountered by earlier protocols. In case of higher node destruction rate the number of copies of data present in the network can be increased to ensure availability of data to the applications. Multicasting of data to all copies ensures that multiple updated copies of the data are available for the lifetime of the network, which makes MidWSeN framework robust. However, a situation might arise where all the copies of data might not be updated, for example, due to traffic congestion in the network. The solution to this could be by resending the data if an acknowledgement is not received by the sending service which in this particular case is the aggregation service. Although resending data might prove expensive in terms of resources such as energy but it will ensure that all copies of data are updated and synchronized.

The above section provided the details of the data-centric design and the benefits of using this particular type of design, which is efficiency and consistency. This section also explains that the shortcoming of this service being not effective in certain circumstances can be tackled by having multiple copies of data. The next section gives details about how the PSS interacts with different services such as AMS within the framework.

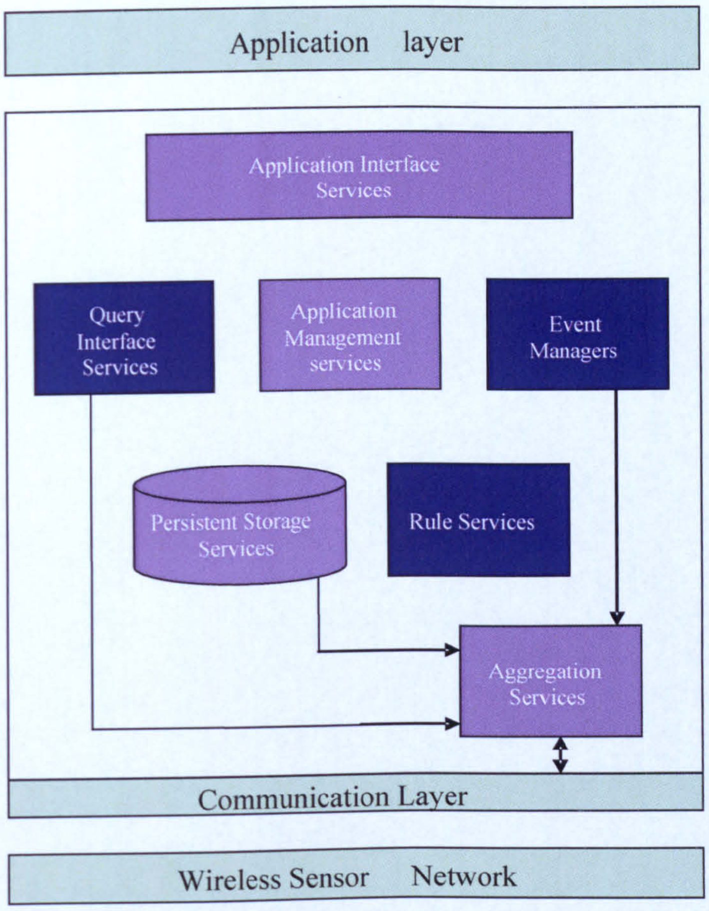


Figure 5-1 Interaction of other services with Persistent storage service

5.3.2 Interaction of other services with PSS

This section explains how the PSS will interact with the different services within the network. Figure 5-1 shows which services interact with the PSS. Data or services cannot be viewed in isolation. As already mentioned in section 2 there is a need to store the data on the sensors so that it can be processed further to turn it to more useful information. In-network processing coordinates sensors to interact with each other through exchange of data. Different phenomena will be monitored by the sensors in the environment, along with their required context. For meaningful interaction between the user and devices the system has to know the surrounding environment, for example, the geographic location or the time at which the events are occurring. The acquired data has to be viewed in its context in order to achieve correct assumptions. Context as mentioned in the background chapter can be defined as any information that can be used to describe the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.

For example, consider a scenario where scientists are monitoring a number of explosions of different intensity of a volcano. They are also storing the time and location of these explosions in order to establish whether they are towards the centre of the volcano or the sides. Now the sensors are sensing and storing the data with the required context. There is a series of explosions of different magnitude. Suppose the scientists want to query the system to know how many explosions were towards the centre of the volcano in a fixed time frame and compare this behavior to a similar behavior in the past two days to analyze whether the intensity of the explosions has increased or decreased towards the centre of the volcano in the past couple of days. This is a complex query that involves historical data. In order to answer the above query using existing event based techniques the scientists will be getting a considerable amount of data whenever an explosion occurs in an event-based network. In case of a query-based system they can also establish how many explosions occurred during the last *n* hours. However, this query would be sent repeatedly to get this data. This would utilize a considerable amount of resources, as the query needs to be circulated repeatedly within the network. Moreover, partial results will be communicated back to the base station repeatedly, which means more transmissions. Another disadvantage of this whole process is the delay because only after the complete results are communicated to the base station can the scientists look at the data and analyze the situation.

In MidWSeN framework this type of query is dealt with more efficiently requiring fewer resources. The data will be stored along with date, time and location, therefore the middleware will be able to compare the current results with the previous data date-wise and provide an answer. There would be no repetition of the query and no delay because all the required information is already stored within the system. The scientists will only get the results in the event of an increase or decrease in the intensity of the explosions. By looking at the preceding scenario it can be seen clearly that the proposed middleware performs more efficiently than the existing techniques.

The middleware already provides the date, time and location context. In addition the user can also add any context required by the application, for example, how much vibration it causes in the earth. This not only gives freedom to the developer to fulfill the application requirements but also helps in making the middleware adaptive.

Similarly, the RS also interacts with the PSS. Considering the above scenario scientists may be interested is whether these explosions cross a certain range of intensity or the number of explosions exceed a certain limit. It will be more efficient to keep this type of query, which requires regular monitoring of the system in the RS as a rule. These rules will constantly check the PSS as required by the user and inform the user whenever a rule is satisfied. As soon as one of the rules is satisfied the RS will immediately alert the application through the AMS, which will in turn send a message to the AIS. All processing will be done within the network and final result will be conveyed meaning fewer transmissions, which in turn mean a reduced amount of energy consumed. These rules can be defined using any query language.

5.3.3 Design

It has already been mentioned in section 5.3.1 that the MidWSeN framework uses data-centric storage. As already mentioned in the previous section there are several techniques available, for example, DCS (Ratnasamy *et al.* 2003) which could be used for MidWSeN as well. This type of data-centric storage is based on a distributed hash-table having two basic primitives `Put(Key, value)` and `Get(key)` where key is the name of the actual data, for example, *temp*. It also uses the GPSR (Karp *et al.* 2000) geographic routing algorithm for low-level routing and on top of that builds a distributed hash-table. MidWSeN is not restricting the user to any particular data-centric technique because a lot of research is still going on in this area and there might be more lightweight and energy efficient protocols introduced then the existing ones. Therefore, any data-centric technique which is efficient and lightweight is acceptable. Selection of a particular routing technique, however, depends upon a number of factors. First MidWSeN uses data-centric storage so the routing technique selected should be able to support this kind of storage. Second, Multicasting is being used in the framework proposed by research presented in this thesis; therefore, the routing protocol must support multicasting. Third, security might be an important issue for the applications using the network; therefore, security may also be a consideration in selecting a routing protocol. Finally, the size of the network may also be a consideration because for a large network more number of copies of data will be required hence for sending data to keep all the copies simultaneously would require a very efficient routing protocol. Also large network means more sensors can collaborate to pool more resources.

Similarly there is the aquisitional technique (Madden *et al.* 2005) which uses a relational database for storing data on the sensors. In a relational database the data generated in response to a query or event is stored in tuples, which can be considered as rows in a table. This data can be stored temporarily in a log for further processing while the tuples can be produced in sample intervals by the user that is a parameter of the query. The period of time between the start of each sample period is known as epoch in the literature which is an addition made by the above mentioned query technique. Here a routing technique will also have to be employed which is part of the communication layer shown in the Figure 5-1. The middleware requirement is only that it should be data-centric storage whichever technique used is not our concern because we do not want to limit the choice and also it does not affect our middleware. It is the same case with routing, for example in the case of DCS, which uses GPSR. Again the decision is based on the technique to the TinySQL(ref). In the absence of any appropriate service oriented software developed for sensor networks the services communicate through messages with each other which could be an XML message or TinyML (Ota *et al.* 2003) or SensorML (VAST 2007). The Application Interface Service sends a message to discover the Application Management Service.

Sensorid	hopcount	location	Memory	energy

Figure 5-2 Acknowledgment message sent by the service in response to the service discovery message

Payload	Priority	querytype

Figure 5-3 Message sent by the service to the acknowledging service

In response to the service discovery message the appropriate service sends an acknowledgement message as shown in Figure 5-2, which has the information about its energy and memory status, how much energy the node has and how much free memory is available. The hopcount shows the distance of the node from the sink. Location shows the geographic position of the node. In case more services are present in the network one which is closest to the event area having more resources would be selected on the basis of the hopcount, memory and energy will show the amount of memory and energy available of the responding node. This means the Service optimization will be performed by the AMS based on these three parameters first shortest path which is determined by the hopcount second is location of the service near the event area and third are resources which consist of the memory and energy of the node. The Location of the sensor could be determined by using any localization technique depending on the application requirements (C˘apkun et al. 2006; Lazos et al. 2005). This is also a novel contribution as no other work to the best of our knowledge has done service optimization within the network. This is usually done on the more powerful base station, which might be a PC or Laptop. The advantage of doing in-network processing is that it saves resources especially energy, which is very precious in WSN because communication costs more in terms of resources.

The AMS would forward the query or event listener to the appropriate service after deciding by looking at the message header if it is a query or an event listener. The message header will have a payload, which will consist of the event or query along with the context and the duration for which the query/event will remain in the network.

The priority field will show the priority assigned by the user to the data stored as result of the query or event. The priority field is explained in detail in section 5. In case of a query the AMS will further determine whether it is a real time query or historical query. The querytype have been added to specify the type of query in order to help the AMS distinguish between these different types of queries. We can categorize the queries into three different categories.

1) The first could be called '*historical query*', which would only involve historical data this query would be sent directly to the PSS by the AMS and the results would also be transmitted back through AMS. Figure 5-4 shows the process of a

historical query. The querytype for this, for example, can be '0' so that AMS knows it's a historical query.

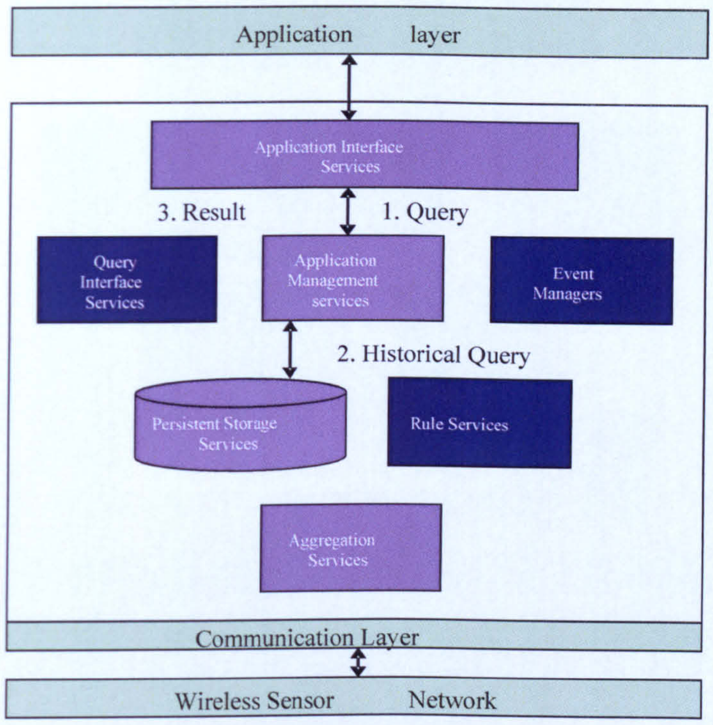


Figure 5-4 Sending historical query to Persistent Storage Service

2) The second type of query would be a *'real-time query'* which only involves current data. The AMS would send this query to the QIS and a copy of this data would be stored within the PSS. Figure 5-5 describes the process for a real time query being sent to the AMS, this service is distinguished by '1', for example.

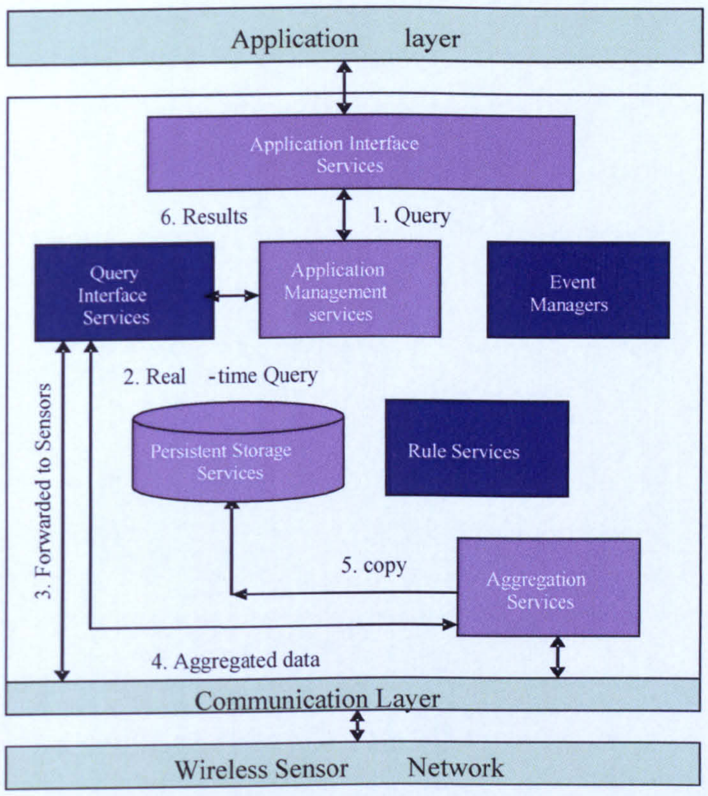


Figure 5-5 Sending Real-time query to PSS

3) The third category would be an event that could be distinguished by the AMS '2'. The AMS will send it to the EM and the process is described in Figure 5-6.

Figure 5-6 explains the process of an event and how it listens and pulls data from the sensors. Data will be generated following an event or when a real time query is sent to the network. In both cases data will be stored in the PSS.

The AMS will decide after testing the querytype if the query involves historical data or real-time data and the query would then be forwarded to the appropriate service or to the EM in case of an event. The results would again be transmitted through the AMS to the user and the PSS. The Aggregation service also interacts with the PSS directly. It will be explained in detail in the next chapter.

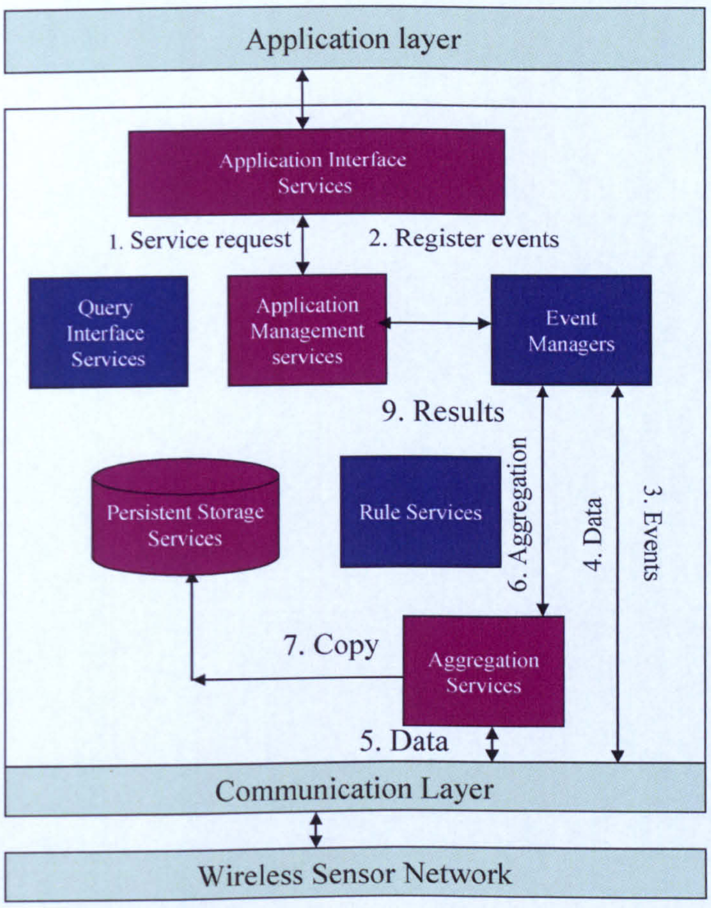


Figure 5-6 Event Listening and pulling data from sensors

The PSS has the following methods;

- int *Read*(int key); when the AMS sends a request for information this method reads information from the PSS.
- void *Store*(int key, int data); when data is sent to the PSS this method writes the information coming from the AS to the PSS.
- int *SendData*(int data); this method will send the aggregated data back to the AMS.
- bool *MemoryCheck*( ); this method would check the memory periodically and return a Boolean value of either true or false. If it returns false it means that the memory is lower than a certain threshold and therefore, the priority algorithm needs to be run. No parameters are passed to this method.

- void *Priority*(struct Message *msgpriority): this method will run the priority algorithm described in detail in section 5 of this chapter.

This section has presented the design of the persistent storage in detail and how it interacts with other services. The next section presents a case study that illustrates how the persistent storage works in a real time scenario.

5.4 Case Study

In this section we are presenting a case study to show how the Persistent Storage Service works. For example, an event is registered which counts the number of vehicles on the road so that in future the road is adapted to changing traffic conditions. After a while the traffic conditions change and sensors indicate more traffic on the road. The authorities now want to investigate the average frequency of different type of vehicles that uses the road in question. For example, for a certain period of time the heavy traffic load was below 10% trucks per 10 kilometres now due to the opening of a factory and some warehouses the average increases from 10% to 25% trucks per 10 kilometres that is the number is more than double; the authorities also find that light vehicles usage has generally increased but the percentage is still less than the percentage of heavy vehicles. This helps them conclude that the road is used more for heavy vehicles rather than light vehicles and the road has to be adjusted to these conditions. Now the event will be sent through the AIS, which will send a message to the network. For example, an XML message has shown below which is looking for the Application Management Service. The corresponding service will respond to the message by sending an acknowledgement message as shown in Table 1 Acknowledgment message sent by the AMS, for example, given below.

sensorid	hopcount	location	memory	energy
01	3	14,18	640	2

Table 1 Acknowledgment message sent by the AMS

The sensorid is of course the identification of the sensor. The hopcount determines how near or farther is the responding sensor to the one initiating the request of service. *Location* shows the relative geographical location of the responding sensor depending on the localization technique being used. Table 3 gives value of the *X* and *Y* coordinates, which is just an example there might be localization techniques that use other types of coordinates such as three-dimensional or latitude or longitude. Usually at the time of deployment the sensors determine their relative geographic location with respect to their neighbours as well as their respective geographic positions. The middleware framework is not binding the developer to a single type of localization technique (Rudafshani et al. 2007) because some sensors might be equipped with special hardware, such as, GPS and some sensors might not be equipped with any kind of special equipment. The only requirement is the use of such a technique which ever is used does not affect the overall system. The memory would give the amount of available memory on the sensor in KB. The energy depicts level of energy of the sensor. This has been mentioned in the previous section that the AMS decides which service to correspond in case of multiple services of the same type using information from the acknowledgement message.

payload	priority	querytype
Parameter list + context	1	0

Table 2 Message sent by the AIS to the AMS

Table 2 Message sent by the AIS to the AMS gives an example of the message sent by the AIS to the AMS. The payload consists of the parameter list from the query/event and the context provided by the user. Here *querytype* is equal to '1' that means it is an event. Priority is equal to '1' that means high. The field *priority* is being explained in detail in the next section.

In the example mentioned above the event will be as follows

Event

```
On EVENT vehicle(loc)
COUNT(lightvehicle), AVE(lightvehicle)
SAMPLE PERIOD 10s FOR 1hr
```

A similar event can be registered for heavy vehicles. In the event of either a heavy vehicle or light vehicle passing by, the sensors will record the event and the data would be forwarded to the aggregation service and user will be notified. After being aggregated the data would be forwarded to the PSS. The user subsequently poses a historical query the data is already stored in the persistent storage. For example

Query

```
SELECT light vehicle, heavy vehicle
FROM sensorstable
HAVING AVE(lightvehicle) < AVE(heavyvehicle)
```

As the persistent storage has saved the average it will compare the two averages and send the result. The value will be communicated back to the AMS which will forward it to the application, for example, the event given above means that each light vehicle and heavy vehicle sensor sends its readings every 10 s for 1 hour. Results will start flowing to the node having the QIS. Results may be logged at the sensing node and sent to the QIS after being processed and aggregated which will forward it to the AIS through AMS. The same data will also be sent and stored in the PSS.

The above section explains how different services such as AMS interact with the PSS. It further gives a detailed example of how the data would be stored, queried, or events registered and data retrieved. The next section explains how prioritization helps to save memory within the network.

5.5 Prioritization

This section explains prioritization and how is it being used in the proposed middleware to save memory. The middleware is storing data for the lifetime of the network and not temporarily as other works do. This will help in creating implicit relations as well as allowing data to be viewed in historical context. The biggest challenge that arises in this situation is how to store data under the constraints of limited memory. Also, even if the data is stored in the available memory what will happen if more memory is required or more data needs to be stored? To solve this problem a new prioritization algorithm for memory management is being proposed in this thesis. This has already been mentioned in the background chapter that prioritization techniques are being used in network communication but it is being used for the first time in memory management in WSN. In network communication packets are assigned priority and when the communication channels are busy or in case of contention only high priority packets are sent to avoid congestion. In memory management priority is assigned to data being stored in the memory and when ever more space is required data with lowest priority are deleted to create space for more or higher priority data. The next section explains in detail how this prioritization technique works to help provide space in memory management in the proposed middleware.

5.5.1 Prioritization Algorithm

This section explains the prioritization process in detail and how the algorithm works. The prioritization algorithm not only creates space in memory for more data to be stored but it also utilizes memory efficiently by only storing the most relevant data. Storing data in aggregated form means that no data will be lost and it will still be available for future use. The application will assign priorities to different data it is required to store when it registers an event or send a query. All the data related to that particular query or event would be assigned the same priority. For example, in fire detection application temperature data is being stored. The application has already saved a rule in the RS that when temperature is greater than a certain limit the application must be informed. This would be done through the AMS. The temperature exceeds the limit and there is a possibility of a fire. When this particular event occurs, the rule fires and the RS notify the AMS, which in turn notifies the application through the AIS. The application starts querying the network through the QIS so that an action can be taken. A lot of data might be generated and stored in this scenario. After the event has passed all the data stored might not be of importance only the fact that there was a fire alarm at such time and location remains of interest. The application may assign a lower priority to those details and whenever more memory is required all the data stored relating to the event and queries send can be aggregated to make space for more important data. The point to be noted here is that data is not been deleted altogether. So the data is there in some form and can be utilized whenever required for future need. This is also a novel contribution because other similar works suggest deleting the data as it gets older it is referred to as graceful degradation (Mathur *et al.* 2006).

The proposed priority algorithm will periodically check the memory, and if the free memory space falls below certain limit the priority algorithm will call the aggregation function, which will aggregate the data with the lowest priority. The algorithm is checking the memory by counting the number of items stored because the data is usually being stored in a linear form on the sensors and the pointer always points to the memory location in which the last data item was stored. After determining the memory size and deciding the limit of the memory, data would be assigned *n* priority levels, which could have *n* possible cases. First the algorithm will check for data at the lowest priority level and aggregate this data. A threshold is selected for the memory and whenever the memory reaches that threshold the algorithm will be applied. After checking the memory again if the memory space is above the required threshold then the process will stop but if still more memory is required then the (lowest -1) priority data will be checked and aggregated. The process will continue until the amount of free memory goes above the required minimum. Figure 5-7 gives the algorithm explained above. In case only data of the highest priority is stored in the network and even then the need arises for more memory it will inform the application and ask which data is to be aggregated. The aggregation service will also give a choice to the developer as to what kind of aggregation the application requires Sum, Ave, Max, Min, Count or MyAgg.

```
m_size= mem_check( );
p= lowest – priority; // highest value
while (p>=1)
/*This will check memory against its limit */
  { if(m_size >= m_limit)
/* Aggregation service will aggregate all the data d for priority  p */
       { call Aggregaton_Service(d, p);
          p=p-1;
          }
   }
```

Figure 5-7 Prioritization Algorithm for PSS

The choice of assigning priority is left to the user because it totally depends upon the application, which can be done by using the *priority* field mentioned in figure 2 and table 2 in the previous two sections respectively. Some applications might want to prioritize their data according to the events because some events are more interesting than others. Others might want to prioritize their data according to the frequency of the queries. If the frequency of queries is high for some data then the higher its priority goes. Some other applications might be interested in a certain range of data, such as whether the temperature is too low in a certain range or too high in another. These are only some of the possibilities there could really be endless depending on the applications and their specific requirements therefore it is best to let the application developer decide what best suits a particular application. If we simply delete the data in using FIFO that might not be the best solution because, as mentioned above, different applications might have different requirements. Furthermore the data is not totally being removed from the memory in case it might be required in the future as a reference but rather it is kept in an aggregated form.

5.6 Summary

In this chapter a novel middleware Persistent Storage Service was discussed in detail that stores data within the sensor network for the lifetime of the ad hoc wireless sensor networks (WSN). Previous works have used temporary storage but the proposed middleware implements persistent storage that provides increased functionality, which is one novel aspect of the work presented in this thesis. This would help in saving precious resources e.g. battery life and bandwidth by only transmitting processed data which means less data would be sent to the users. Storing the data within the network serves two purposes. First, it would help in further processing of the data, for example, to create implicit relationships between data and second it saves energy because transmission consumes more energy. The PSS not only provides data consistency at all times by having multiple synchronized copies but also the facility to add context. The copies will be synchronized by sending data at the same time to all copies within the network after adding application-context through the aggregation service.

This chapter also discusses the prioritization algorithm for storing data within Wireless Sensor Networks (WSN), which is also a novel contribution. Due to limited memory in

WSN it is not possible at all times to accommodate all important data to be stored within the network therefore different data can be assigned priorities according to applications requirements by the application. Data with the lowest priority will be aggregated to create space if additional memory is required to store more data. This will help in storing relevant data within the network for long periods of time without losing any important data.

The next chapter explains the aggregation service in detail, which is also an important part of the middleware framework MidWSeN. The aggregation service plays a vital role in condensing the data and making it possible to filter redundant data thus saving energy and storage space or memory. In addition it aggregates data with context thus producing context-enhanced data which is stored in the PSS. This gives a context-enhanced data, which is another novel contribution of the framework presented in this thesis.

Chapter 6 MIDDLEWARE AGGREGATION SERVICE

6.1 Introduction

The last chapter introduced the middleware Persistent Storage Service in detail. The focus of attention in this chapter is the aggregation service of the middleware framework MidWSeN. It has already been mentioned in the background chapter and system requirements in chapter 4 that aggregation is vital for WSN because it helps in saving energy by reducing the amount of data being stored and transmitted.

The purpose of the research presented in this chapter is not to create an aggregation service from scratch but to reuse the existing work with a different approach to draw maximum benefit. There are two main novel aspects of the aggregation service used in the MidWSeN. The first is in respect of the framework in general and the second relates to its use as an individual service in particular. The fact that aggregation is being used to save memory as well as other resources is novel by using it as a service in the middleware framework. The aggregation service not only saves energy by reducing the amount of data being transmitted across the network but also saves memory within the network because data is being stored within the network.

Another vital aspect of the MidWSeN framework is that data should be stored with contextual information therefore as an individual service aggregation is being done with respect to context. In other words it is a context-enhanced service which is the other novel aspect of the aggregation service. It aggregates the data according to the contextual information provided by the application which is later stored in the Persistent Storage along with the context. Due to its importance in the overall working of the framework the aggregation service forms one of the core services.

The snooping technique(Madden et al. 2002) is also discussed using new scenarios in this chapter. In snooping sensors listens to the communications of other sensors, this helps in conserving energy by reducing communications which proves more costly in terms of resources especially energy. Using Snooping technique in MidWSeN not only reduces communication cost in terms of resources but it has been tested in calculating averages which in principle was not used before because of the expected error margin. The research presented in this chapter identifies certain scenarios which could prove beneficial in some scenarios.

The rest of the chapter explains the aggregation service and its novel aspects in detail. Section 2 explains why there is a need for the aggregation service within MidWSeN and how it will help in providing a better service than existing aggregation services. Section 3 gives more detail about the aggregation service. Subsection 3.1 explains why the snooping technique is being used, its advantages and circumstances where this technique should be avoided. Subsection 3.2 explains the aggregation services interaction with other services within the framework. A case study is given in section 4. Finally section 5 gives a summary of the whole chapter.

6.2 Aggregation as a middleware Service

This section explains the reason for keeping the aggregation service within MidWSeN. Broadly there are three different levels where aggregation could be handled. These are levels are

i) Application Level

ii) Middleware

iii) Network Layer

The argument for application level aggregation is that because every application requires an aggregation service tailored specifically to its needs or in other words aggregation is application specific in WSN, it should be left to the application. On the other hand, a counter argument for a lower level aggregation service is that since it is so vital for every application it could also be left to the communication protocol. In this type of aggregation the data is aggregated as it moves along the network.

The problem with the first approach is that it burdens the developer as they will have to fully understand how the specific operating system (OS) and hardware interacts and other lower level details. Further it reduces the portability of the application because the code has to be changed every time there is a change in the hardware or environment as it might be written for a specific hardware configuration. This not only raises the maintenance cost of the application but also makes application development very cumbersome.

On the other hand most communication protocols (Heinzelman et al. 2000; Lindsey et al. 2003; Manjeshwar et al. 2001) also provide aggregation. Communication is a lower level operation, hidden from the application and hence does not have any knowledge of user requirements making it hard for applications to make any changes to customize it to application needs.

The design goal as mentioned in the Introduction chapter of this research is to make it possible for different applications to use this middleware; therefore aggregation service is kept as a core service in the middleware. It will not only relieve the application developer from the burden of the underlying embedded OS and hardware (Heidemann *et al.* 2001) but also make the program portable and easy to maintain in the face of any changes in the physical layer. On the other hand the developer can still have the freedom to tailor the service to their specific needs.

Only middleware can bridge lower level operations and high level applications in order to facilitate better understanding of the application requirements. Although aggregation is vital in most cases in WSN, a different scenario may be that some applications might not want to use aggregation at all because the sensors are sending different type of data for example video/audio type data (Dasgupta *et al.* 2003). Therefore it is important to keep the aggregation service in the middleware so this service is aware of the user requirements as well as take care of the lower level operations. The developer just specifies his requirements and does not have to worry about the lower level details.

This section has explained in detail that the aggregation service has been kept as a middleware service to bridge the gap between lower level operations and high level application. This gives the developer freedom to customize it to application requirements without needing to consider lower level details. It should be noted here that one of the characteristics of sensor networks is being application specific therefore it is important to allow the application to customize the service to its specific requirements. The next section explains the Aggregation Service (AS) in detail how will it aggregate the data and also interact with other services of the proposed framework.

6.3 Aggregation service

This section discusses the middleware approach to the aggregation service. By reducing the data or filtering redundant and unnecessary data it not only saves energy but also memory in the proposed middleware which gives a different perspective to this service. The work presented in this thesis is significantly different from the existing works in many respects.

i. First the whole concept is to save the data within the network and send only the processed data to the user on query or by registering an event because communication is more costly than processing. Therefore aggregation is not only being utilized to save energy but also memory.

ii. Second, data is being aggregated according to context specified by the user. For example, if time is important for a specific application then the data would be aggregated according to a specific time. The aggregated data will then be stored in the Persistent Storage along with the context. This helps to supply context enhanced data to the application.

iii. Third we are not binding the user to any particular topology or data structure, for example, a tree type structure and it can therefore be used in any environment.

iv. We are using a service-oriented architecture therefore it is possible to have several copies of the same service which means the system can work in more than one way. The same application can have different types of aggregation in one network at different places, whereas multiple applications can also use this service which makes the system more robust.

v. This service also provides an opportunity to the developer to adapt it to application specific requirements by providing a method called *MyAgg*. This not only helps in customizing the service to a particular application but due to the existence of multiple copies of this service also provides the opportunity of writing different aggregation methods depending on application requirements.

vi. Another way of customizing the service is to add application required context. This context can be added when the application is registering an event or posing a query. The data will be aggregated and stored along with the context in the persistent storage.

vii. Having multiple copies of this service makes it more robust and due to it is loosely coupled architecture new services can be introduced easily into the system making it more scalable.

The importance of the AS in the framework is demonstrated by the fact that it is being used by three different services the QIS, PSS, and EM. Data coming both from the Event Manager and directly from the nodes is aggregated and sent to the PSS. The PSS utilizes the AS in its prioritization algorithm which has already been explained in chapter 5.5. Figure 6-1 shows the interaction of different services with the AS.

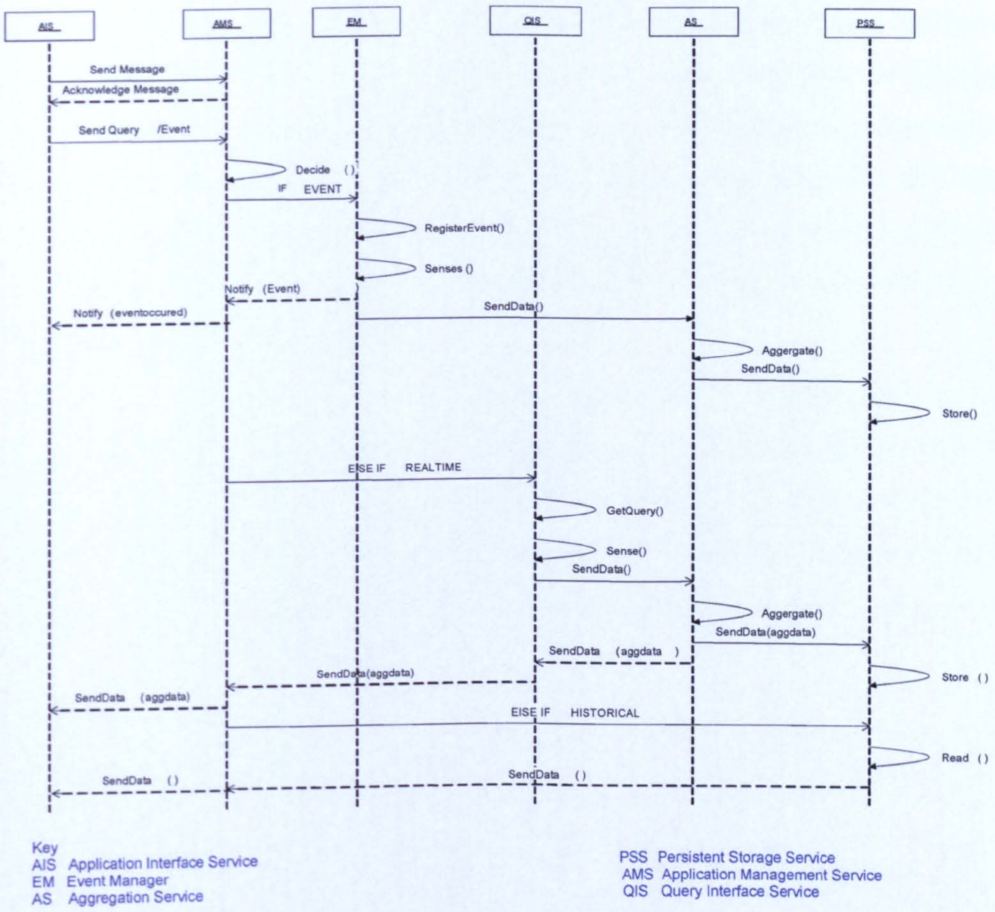


Figure 6-1 Showing Sequence of events for Aggregation process

6.3.1 Design

This service uses in-network processing which can reduce the amount of data being finally transmitted to the user in a power efficient manner. The novelty of this work lies in the approach of the aggregation service which uses context to aggregate data to have context enhanced data. Novelty also lies in the way the aggregation service is being used within the framework and its structure, rather than inventing a new technique. This service does not bind the user to any particular protocols or data structure thus making it flexible. Snooping technique (Madden *et al.* 2002) is being used to save energy, and snooping has been further tested in new scenarios like calculating average, changing topologies and different radio ranges.

The aggregation service will provide the standard *Max, Min, Sum, Count* and *Ave* functions. In addition, there is another method provided called MyAgg. In MyAgg method the application developer can write his own specific type of aggregation which might be exclusively required by the application other than the ones provided. This will add flexibility to the middleware and provide freedom to the application of customizing it to application needs being an important characteristic of WSN. This service will reside within the network on the nodes.

These methods have been defined as an interface which will be implemented by the application developer giving an opportunity to the developer to customize the service to it own specific needs further it will also give him the freedom to write his own specific aggregation technique apart from the standard ones provided. This also helps in utilizing the same service for different applications by simply calling the service when ever required.

```
int max(int data)
int min(int data)
float ave(int data)
int count(int data)
int sum(int data)
float MyAgg(int data)
```

The aggregation service further provides SendData ( ) method which sends the data to the PSS. It should also be noted here that as we assume to be having more than one copy of data therefore this SendData method will send data to all the copies present in the system at the same time. The knowledge of the number of copies data and exactly where these are located in the sensor network will be provided by the data centric storage technique which uses a distributed hash table to keep track of the data stored within the sensor network.

Another method called Snoop( ) is also provided in order to further economize on resources, this method is explained in detail in the next section. The aggregation service can be called at any level the application wants to aggregate data. This service can be used at both the node level and at the data storage service level depending upon the specific application. There is no recommendation of any particular type of routing protocol. Any routing protocol could be used as far as it can carry a query to all the required nodes or sensors and can route them back to the point where it is intended to reach in the system.

Whenever the aggregation service is called it will check the network for a node or group of nodes that provides the service and then utilize it to provide the necessary functions. This service is not binding the user to any particular protocol or data structure which is its main advantage and makes it feasible in any environment and also gives the developer freedom of choice.

6.3.2 Adding Context

One of the design goals mentioned in the introduction chapter was to provide context along with the data because it makes data more meaningful. Adding context to data is challenging in many ways. Various applications have different requirements one cannot generalize or name one or few contexts to fulfill all application requirements. Therefore to conclude; every application should provide its own requirements instead of binding or limiting the application to a set criteria. However, there might be some context for example time, date and location which would be useful to many applications. So this kind of context can be provided to all applications. Therefore to combine both the options discussed above the framework provides the time, date, and location option as well, gives opportunity to the application to provide its own option. Data is being aggregated using context as a criterion. For example, time is important for an application, enabling data having the same time stamp to be aggregated. Another example is location, where data coming from sensors having the same geographic location will be aggregated together and sent to the storage service. The advantage of this context enhanced aggregation is that data can be stored with context and later on queried using the context. Spatial and temporal queries are possible and implicit relations can also be determined using context. Therefore it is an important and interesting aspect of the Aggregation Service and of the overall framework.

The second challenge which needs to be addressed is where to add this context or at which stage context should be added to data. The obvious choice is before storing data;

the context can be wrapped or added to the data and then data can be stored along with this context. In sensor networks that may not be easy as the context data also needs to be collected along with other data, but if they are collected at two different stages it means more energy and resources are being utilized. So the best option is to collect context data along with other data, for example, if time and location is important then sensors should send their respective data along with a timestamp and its geographic location. This ensures that data and context data will come in the same packet meaning fewer transmissions. Other resources, for example, bandwidth can also be saved. Therefore the data along with context is collected by the EM or QIS and sent to the Aggregation Service. Here the data is aggregated according to the context and then this data along with the context is sent to the PSS and stored there with context. The user can then query the PSS using the contextual data.

The next section explains the snooping technique which has been proven to be very economical in terms of resources such as energy. This technique works very efficiently with singular aggregates, for example, max/min. This technique further makes the AS discussed in this chapter more efficient in terms of saving resources. Snooping has been further tested in new scenarios like average for which it was not considered before due to expected error margin. The research presented in the next section reveals, however, that it is possible to use snooping in calculating averages as well in circumstances where only trends are being watched or accuracy is not an issue.

6.3.3 Snooping Technique

This section gives details about the eavesdropping or snooping technique (Madden *et al.* 2002) which is being used at the sensor level. As already mentioned by using this technique will further economize resources. Snooping technique is tested in new scenarios, for example, calculating averages and found it to be useful in some scenarios where accuracy is not an issue. In snooping the sensors listen to the communications of other sensors. This saves energy as listening consumes very little or negligible amounts of energy. We are aware of the fact that eavesdropping can also be a security issue because in applications like battlefield surveillance where the sensors might be used in enemy lines or other such applications the developer might not want to use this technique at all. Also in some applications the data might be encrypted so nodes might not be able to listen to other node's communications. As mentioned earlier, if a particular application does not require this service or a method it can avoid or override it, the applications with high security risk can choose not to call this service or override the method which adds flexibility to the service.

This technique is being used in new scenarios like calculating average, which has not been tried before. Previously TAG (Madden *et al.* 2002) proved that this technique is less costly in terms of energy consumption in comparison to other techniques in calculating a maximum or minimum value. In the snooping technique not all sensors send their data. Considering the particular case of calculating a maximum, sensors will listen to the communications of the other sensors and if the value of this other sensor is greater than the snooping sensor than it will not send its value. Since most of the sensors are snooping therefore sensors detecting the same type of data will not transmit their data. This means fewer transmissions which means less energy consumed as well as less congestion. However, this technique was only thought to be useful in single aggregates, for example, max or min and the effects of different parameters such as topology have not previously been tested were also not tested before.

The tests are conducted to see if changing parameters, for example, the topology or event range has any effect on the performance. The motivation was to see the effects of changing different parameters on the middleware framework. The results show that different parameters do have an effect either in terms of participating nodes or in terms of energy. These simulations results give a better insight to the developers and they would be well aware of the implications of changing different parameters. The results of the tests are discussed in detail in the next chapter.

The tests have explored how to calculate an average value with this technique. In case of an average it was assumed by TAG that the results will not be accurate. The question that needed an answer is that whether calculating average with this technique is a complete waste of resources and therefore should not be attempted at all under any circumstances. The simulation result given in the implementation chapter confirms their assumption but the results show that the error margin is less than an acceptable limit indicated by the simulation test which is 1.5 %. Now this means that it is not a complete waste of resources to try to calculate an average with this technique and it could be used in some applications where accuracy is not a major issue. For example, in the case of global warming in which scientists' only need to know the trend, whether the temperature is rising or otherwise and in other similar scenarios where the need is only to know the rising or falling of certain trends. Therefore it can be assumed safely that the snooping technique may be used in calculating average in some scenarios but not under all circumstances particularly where more accuracy is required because of the error margin involved. The next section explains how other services within the framework interact with AS.

6.3.4 Interaction of Other Services with AS

It was mentioned at the start of the section that the AS can be called at any level where the application wants to aggregate the data. Aggregation is an important service and therefore this section explains how it interacts with most of the other services within the framework. It will aggregate the data coming either from the Event Manager (EM) or directly from the nodes and send it to the Query Interface Service (QIS). The EM registers event listeners posted by the users on the sensors through the *RegisterEvent* method mentioned in chapter 4. Whenever an event occurs the sensors will sense and send data to the EM where the event listener is stored. The EM will then call the AS which will not only aggregate, but also add context to data and send it to the Persistent Storage Service (PSS) which stores the data.

This is the overall process and the exact procedure is explained in the next section in detail. As far as context is being added it depends both on the application as well as the aggregation. For example, if geographic location is to be added as context then it is important that aggregation is done in a way that location can be added as context. The context and aggregation type would be specified in the event/query and done accordingly.

In the case of an emergency or real time situation when the data is required quickly the user will send a query. This query will be sent through the Application Interface Service (AIS) which will forward it to the Application Management service (AMS). The AMS will forward this query to Query Interface Service (QIS) after testing that it does not involve historical data. The QIS will transmit this query to the network. In order to get the results quickly back to the base station the data will be sent back directly to the QIS instead of the PSS. Although a copy of the data would be saved at the PSS. The data has to be aggregated first so this could be done at the sensor level or after the data has been collected from the sensors at the QIS. At the sensor level the eavesdropping or snooping technique (Madden *et al.* 2002) is being used to further economize the resources as already been explained in detail in the previous sub-section.

After aggregation and compilation of the result the QIS will send this result back to the base station through the AMS. Again, the AMS will forward it to the AIS from where the user will get the answer for the submitted query. The above process not only reduces the amount of data but also ensures there is no delay in the delivery of information to the user to improve latency which is an important issue in emergencies or real-time situations. This is done by getting the results directly from the sensors and sending them back to the user whereas normally in this middleware data is being stored in the PSS and the user can query the PSS.

The whole process explained above can be seen in the Figure 6-2. Aggregation is also used in another way to save memory in the prioritization algorithm. We have already mentioned in the previous chapter that in this technique the user assigns priority to data and if the network runs out of memory the data stored in the PSS will be aggregated according to the priorities already assigned by the user at the time of sending query or registering an event. The data with the lowest priority will be aggregated. The above mentioned process not only filters data in order to store only relevant data but also creates more space in the memory for more data.

This section gave an over all idea of how aggregation interacts with other services being an important service within the framework. In the next section a case study is presented to show how the aggregation service will actually work with respect to other services within the framework.

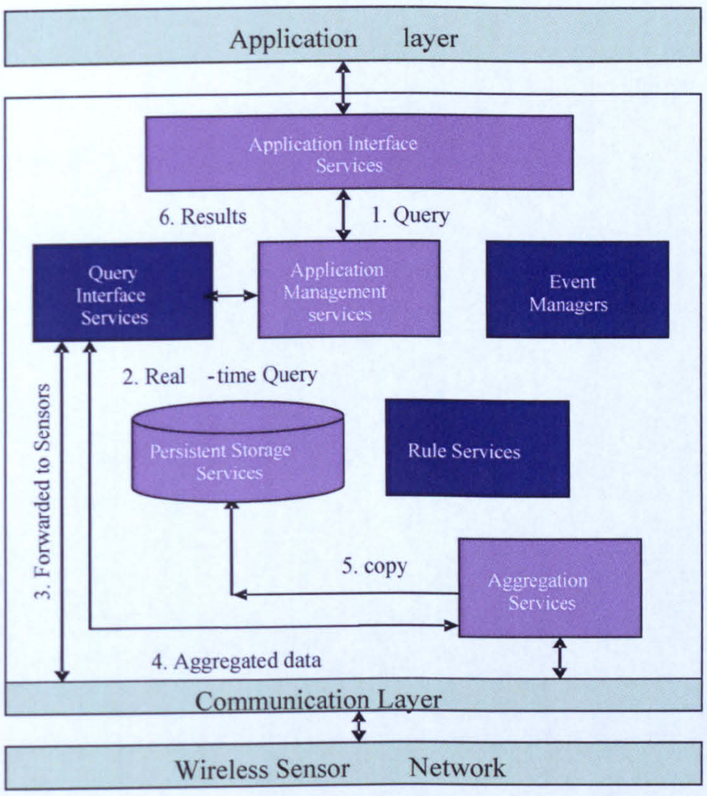


Figure 6-2 Interaction of other services with Aggregation service

6.4 Case Study

This case study represents how the aggregation works for different applications. It also demonstrates the interaction of different services with the aggregation service. The aggregation service handles three types of data:

- Real-time data generated by a real-time query or a registered event.
- Data aggregated at different levels level.
- Aggregating historical data stored in the PSS.

The same example which was demonstrated in the last chapter will be illustrated here. In that example an event for counting number of light and heavy vehicles on a road is being registered in order to adapt the road to changing traffic conditions. Here, as already mentioned in the previous chapter an average of different types of vehicles is being calculated that uses the road in question. An event is being registered as follows

Register Event

```
On EVENT vehicle(loc)
COUNT(lightvehicle), COUNT(heavyvehicle)
SAMPLE PERIOD 1 min FOR 1 hr
```

In the above event the number of light and heavy vehicles is being counted and is example of the first type of aggregation mentioned above. This event will be registered in the Event Manager. The process of finding a particular service as explained in the previous chapter is through exchange of messages. The AMS sends a message to find the EM and forwards the event after receiving an acknowledgement message from the service. When the sensors sense data it will send the results to the EM. Here there are three possibilities the first possibility is that no event occurs during the said time period so nothing will be sent. The second possibility is that only one event occurs in which case after a minute passes that data would be passed on to the EM. The third possibility is that more than one event occurs during the allotted time period in which case the sensors would store the data temporarily on the sensors and send all the events occurring during that time. Here there could be another possibility, the sensor could aggregate the data and send the aggregated result to the EM. This means aggregation is also possible at the sensor level. Looking at the third case the data is only being aggregated once at the service level because it is storing the data on the sensor and after the specified sample period finishes it will aggregate the data send it forward but it will consume more energy and bandwidth. In the second case less energy would be consumed although the aggregation is being done twice because each sensor will aggregate the data at sensor level and then the data from different sensors will be combined and aggregated at the service level. Aggregation at the sensor level consumes negligible amounts of energy and the sample period could also be increased which would mean fewer transmissions. It is possible to have an aggregation service on every node in the sensor network but not necessary. The count() function is given below to demonstrate how the data can be counted. The size of the array *data* would be equal to the size of the buffer used to store the data.

```
int count (int data[]) {
```

```
counter=counter+data[];
return counter
}
```

Another example of a query could be a real time-query to calculating an average.

```
SELECT sensid, light vehicle

SAMPLE PERIOD 10s FOR 1hr

HAVING ave
```

Here the data will be collected for one hour every 10s and then the average would be calculated. If the application wants to add context, for example, to collect the sample for a particular time or location or day it can be done by adding the context. In the following example time is being added as context.

```
SELECT sensid, light vehicle

WHERE Time = '02/00/00'

SAMPLE PERIOD 10s FOR 1hr

HAVING ave
```

It has been mentioned in the previous chapter that if the Persistent Storage Service does not have enough space to store more data then the Prioritization algorithm will be applied in order to create more space in the memory. The algorithm checks for a particular memory threshold and if the available free memory becomes less than the threshold value the Aggregation Service aggregates the data stored in the PSS having the lowest priority. The AS and the PSS can be on the same node but if they are on different nodes then data would be sent to the nearest AS which would aggregate the data and sent it back to the PSS in an aggregated form. This again involves exchange of messages. The PSS will send a service discovery message and the AS which will send an acknowledgement message in return. The PSS will then send the data with lowest priority to the AS which will aggregate it and send the aggregate value back to the PSS to be stored.

If the developer wants to write their own aggregation algorithm rather than using the standard ones provided then the method MyAgg( ) can be used. It could be used for any of the three types of aggregation discussed above. The developer can customize the aggregation service to application requirements by implementing the customize interface. The process is explained in the implementation chapter in detail.

6.5 Summary

In this chapter the aggregation service has been presented in detail. It further discusses how using in-network aggregation can reduce energy consumption and other resources utilized. This particular service achieves the primary goal of this research's middleware framework design by not restricting the user to any particular environment hence making it flexible. As mentioned earlier, MidWSeN combines three important aspects of a WSN; events, queries and context awareness. The most important aspect of this aggregation service is that it uses context to aggregate data then sends this data to the PSS to be stored along with the context which is a novel contribution. For example, if geographic location is important for an application then data coming from sensors having the same location will be aggregated. More than one context can also be combined to aggregate the data, examples being time and location or time, date and location.

This service is also responsible for sending data to different copies of the PSS or information can be provided by the particular data-centric storage technique being used by the network. The middleware, however, does not recommend or limit the network to a particular technique and assumes that there is more than one copy of data being stored in the network and the location of these copies is provided to the middleware.

Use of the snooping technique further improves resource utilization especially energy consumption which is very important in WSN environments. The effects of different parameters like distance, topology and network size using this technique were found to be significant. This technique was also found useful in calculating averages in scenarios where accuracy is not an issue.

This aggregation service provides the standard aggregates, for example, max(), min(), ave(), sum() and count() as well as *MyAgg()* which provides an opportunity for the developer to customize it to applications requirements. In addition there is a SendData() and Snoop() method available. This chapter provides a case study which explains how the aggregation service handles different types of data, for example, real time data and data stored in PSS.

The next chapter presents the evaluation, testing and implementation results of the framework. It evaluates the framework against its own requirements and compares it to contemporary works. It also provides the results, graphs and calculations done to highlight different novel aspects or proof of concepts.

Chapter 7PERFORMANCE EVALUATION AND IMPLEMENTATION

7.1 Introduction

The previous two chapters gave details of the design of individual services in the framework MidWSeN. This chapter gives the implementation and evaluation details of the framework. A detailed evaluation of the results obtained through simulations and implementation are also given in this chapter.

The overall plan was to demonstrate that the research presented in this thesis is feasible. In some cases, for example, the snooping technique the goal was to improve the performance of the framework so its impact with respect to different parameters is being tested. In others the objective is to demonstrate the idea and show how it works, for example, the priority algorithm. In order to show that the main services of the framework can work together in a pure sensor environment implementation details are given using the Crossbow motes (Crossbow Technology Inc. 2007) and TinyOS (TinyOS 2007). The reason for this is to illustrate the important and novel aspects of the framework. Simulations are being used to evaluate performance of individual components using different parameters for comparison. Other programming tools, for example, prototypes or algorithms have been used to show exactly how the sequence of events develop or demonstrate an idea. The objectives of our evaluation is

- To show the performance of individual components of the framework
- To demonstrate the working of services within the framework.
- To illustrate the feasibility of the ideas presented in the research.
- To compare our framework with other existing works.

7.2 Implementation

The purpose of the implementation explained in this section is to test the framework. The framework is designed to work coherently in a distributed environment. The implementation is done in parts to show the operation of different services. To build a pure sensor network environment TinyOS (Levis 2006) and nesC (Gay *et al.* 2003) are used to implement the framework on xbow telosb (Crossbow Technology Inc. 2007) motes. NesC is a new programming language specifically designed for embedded systems which uses component-based programming paradigm and follows the C structure. TinyOS is an operating system specially developed for sensors and is written in the nesC language. Radio communication in TinyOS follows the Active Message (AM) model in which each packet on the network specifies a Handler ID which is just like a port number in a message header. For communication between a PC and the mote a serial port is used. Java and javax.comm is used to send and receive data to and from the motes.

In the first part of the implementation an event is registered by using RegisterEvent( ) to get data in the EM and stored it on a mote through Store( ) method of the PSS. A java interface is used to connect the sink to the telosb mote using TinyOS (this interface comes as part of TinyOS1.10). The sink in this case is a PC. The component SensePhotoM of TinyOS was used in this case because the data sensed was light. The event was registered in the Event Manager. After sensing the data it is stored on the same mote in the Persistent Storage Service in this case the *Logger* is used for which Figure 7-1 and Figure 7-2 illustrates the code. The EEPROM or flash memory could both be used because both act like a persistent memory and will retain the data even if the mote crashes or is rebooted. The flash memory, however, is a better choice because it will have more space, 1MB in the case of mica2 motes and 32MB in the case of imotes. On the other hand EEPROM has less memory and some memory is set aside to store operating system and hardware interfaces as well, therefore it might not have a lot of memory to store data. In this case the EEPROM is used.

Then some queries are sent from the PC to get data using the SendMsg and ReceiveMsg components of the TinyOS. A simple multihop routing protocol to communicate between the motes is used which also comes with the TinyOS setup. By conducting this simple test four of the framework services, the Persistent Storage, the Application Interface Service, the Application Management Service and the Event Manager, can be demonstrated. The first query requests some data which is already stored in the persistent service. The AIS using SendMsg sends a historical query to AMS which looks at the querytype flag used as an attribute in SendMsg and decides historical data is required. The AMS sends the query directly to the PSS and retrieves data using GetData( ) and sends it back through the same route to the PC. Two different motes and a PC were used to show all three services are stored on different nodes. In these tests the queries were transmitted and historical data retrieved successfully. This test also proves that the services discover each other despite being on different nodes and, communicate and perform a task collectively. All these are simple tests just to illustrate that different services can talk to each and collaborate in a distributive environment to perform its task.

```
task void writeTask() {
    char* ptr;
    atomic {
        ptr =
(char*)buffePtr[currentBuffer];
        currentBuffer ^=0x01;
        }
     call LogerWrite.append(ptr);
     }
```

Figure 7-1 Code to illustrate data being written to the Logger

```
asyn event result_t ADC.dataReady(uint16_t this_data) {
    atomic {
        int p = head;
        bufferPtr[currentBuffer][p] = this_data;
         head = (p+1);
         if (head == maxdata) head = 0;
         if (head == 0) postwriteTask();
      }
      display(this_data);
      return SUCCESS;
}
```

Figure 7-2 Code to illustrate data being saved to the mote

Data is being stored within the network on the nodes for the lifetime of the network so that applications can query historical data as well. Saving data for the lifetime of the network is challenging because of its limited memory. Once the data is saved within the network on the motes there might be a situation in which more data needs to be stored but not enough memory is available to store important data. In order to deal with such a situation a prioritization algorithm has been presented in chapter 5. The actual code for this algorithm is given in Figure 7-3.

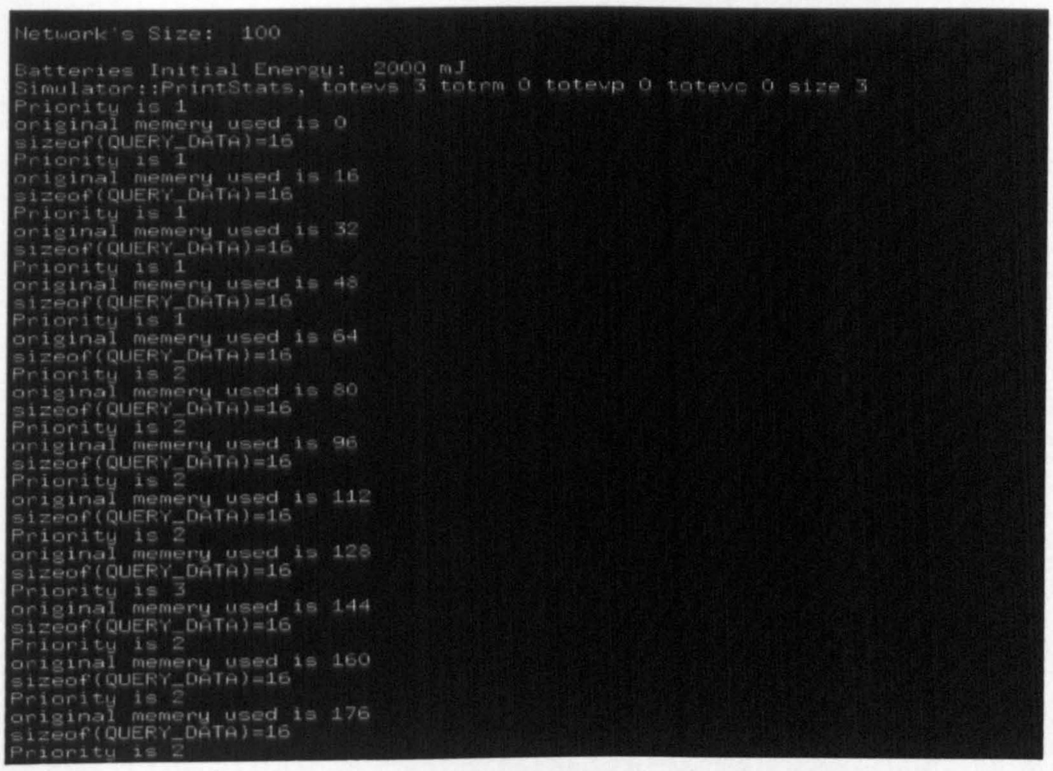
```
if (file_size>=file_limit) //check the limit condition
 {
   cout<<"the memory limit is exceeded we perform aggregation at level "<<p<<endl;
   //query_data_file.seekg (fstream::beg);
   query_data_file.open("query_data_file.txt", fstream::in | fstream::out | stream::binary);
   while ((!query_data_file.eof())&&(file_size>=file_limit)&&(counter<NO_NODE))
     {
       query_data_file.read((char *)(&a),sizeof(QUERY_DATA));
        cout<<"Priority "<<a.priority<<endl;
       if (a.priority==p)
         {
           cout<<"Priority found "<<a.priority<<endl;
           summ=summ+a.data;
           counter++;
           file_size=file_size - sizeof(QUERY_DATA);
         }
     } summ;
   double avrg= summ/counter;
   cout<<"we aggregated "<<counter<<" data at level "<<p<<endl;
   cout<<"memory size is "<<file_size<<endl;
   p--;  // We decrement the priority
   query_data_file.close();
 }
```

Figure 7-3 Code for Priority algorithm

The application assigns priority to data required to be stored while registering an event or sending a real time query. The system will keep on checking the system and when ever the memory limit goes down it will run the prioritization process. The data with the lowest priority will be aggregated thus creating more space without losing all data. This algorithm was also tested by simulation using GTNets simulator on a network size of 100 nodes.

The Georgia Tech Network simulator (GTNetS) (Riley 2003) used to simulate the algorithm is an event based simulator based on Linux operating system. The different wireless sensor network models such as radio propagation model, computation and communication energy dissipation models were implemented in this simulator. This simulator can be programmed using the C language. For simulations the radio energy model proposed in (Heinzelman et al. 2002) is used because it is a popular model in WSNs. The settings used are the radio electronics energy $_{elec}E$ is set to 50 nano Joule per bit. The radio transmitter energy for distances less than $d_{crossover}$, $l_{friss\text{-}amp}$ is set to 10 pica Joule per bit per m2, and the radio transmitter energy for distances greater of equal to $d_{crossover}$, $l_{ray\text{-}two\text{-}amp}$ is set to 0.0013 pica Joule per bit per m4. The assumption is that sensor nodes use the IEEE 802.15.4 (Howit et al. 2003) standard, therefore, the radio range is set to 15 meters and the distance threshold, for which the free space model is used, $d_{crossover}$ to 10 meters. The initial energy for each sensor node is set to 2 Joule.

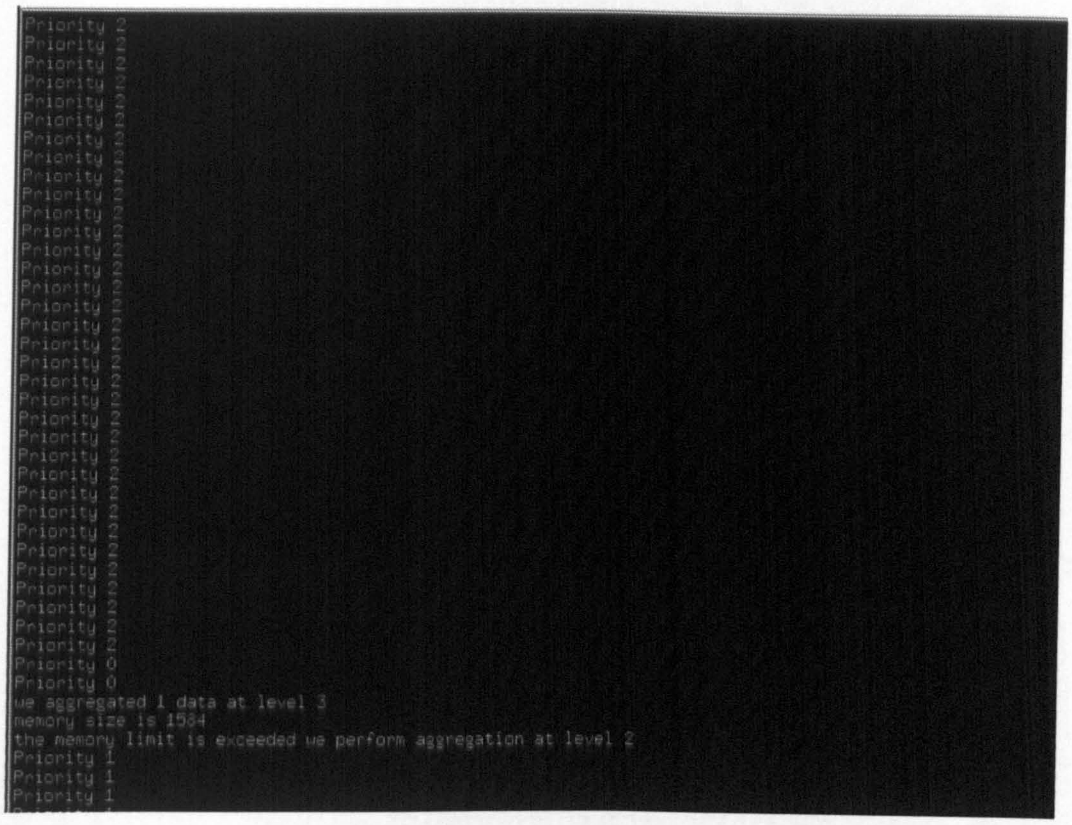
Priority was assigned to data queried with the highest priority being '1' and lowest being '3' as is evident in the Figure 7-4. Memory is being checked and as soon as it exceeds a certain point data at the lowest level is being aggregated which in this case is 3. If still there is not enough memory than data having priority 2 will be aggregated which is also shown in Figure 7-5.



```
Network's Size:  100

Batteries Initial Energy:  2000 mJ
Simulator::PrintStats, totevs 3 totrm 0 totevp 0 toteve 0 size 3
Priority is 1
original memery used is 0
sizeof(QUERY_DATA)=16
Priority is 1
original memery used is 16
sizeof(QUERY_DATA)=16
Priority is 1
original memery used is 32
sizeof(QUERY_DATA)=16
Priority is 1
original memery used is 48
sizeof(QUERY_DATA)=16
Priority is 1
original memery used is 64
sizeof(QUERY_DATA)=16
Priority is 2
original memery used is 80
sizeof(QUERY_DATA)=16
Priority is 2
original memery used is 96
sizeof(QUERY_DATA)=16
Priority is 2
original memery used is 112
sizeof(QUERY_DATA)=16
Priority is 2
original memery used is 128
sizeof(QUERY_DATA)=16
Priority is 3
original memery used is 144
sizeof(QUERY_DATA)=16
Priority is 2
original memery used is 160
sizeof(QUERY_DATA)=16
Priority is 2
original memery used is 176
sizeof(QUERY_DATA)=16
Priority is 2
```

Figure 7-4 Priority assigned to data by application

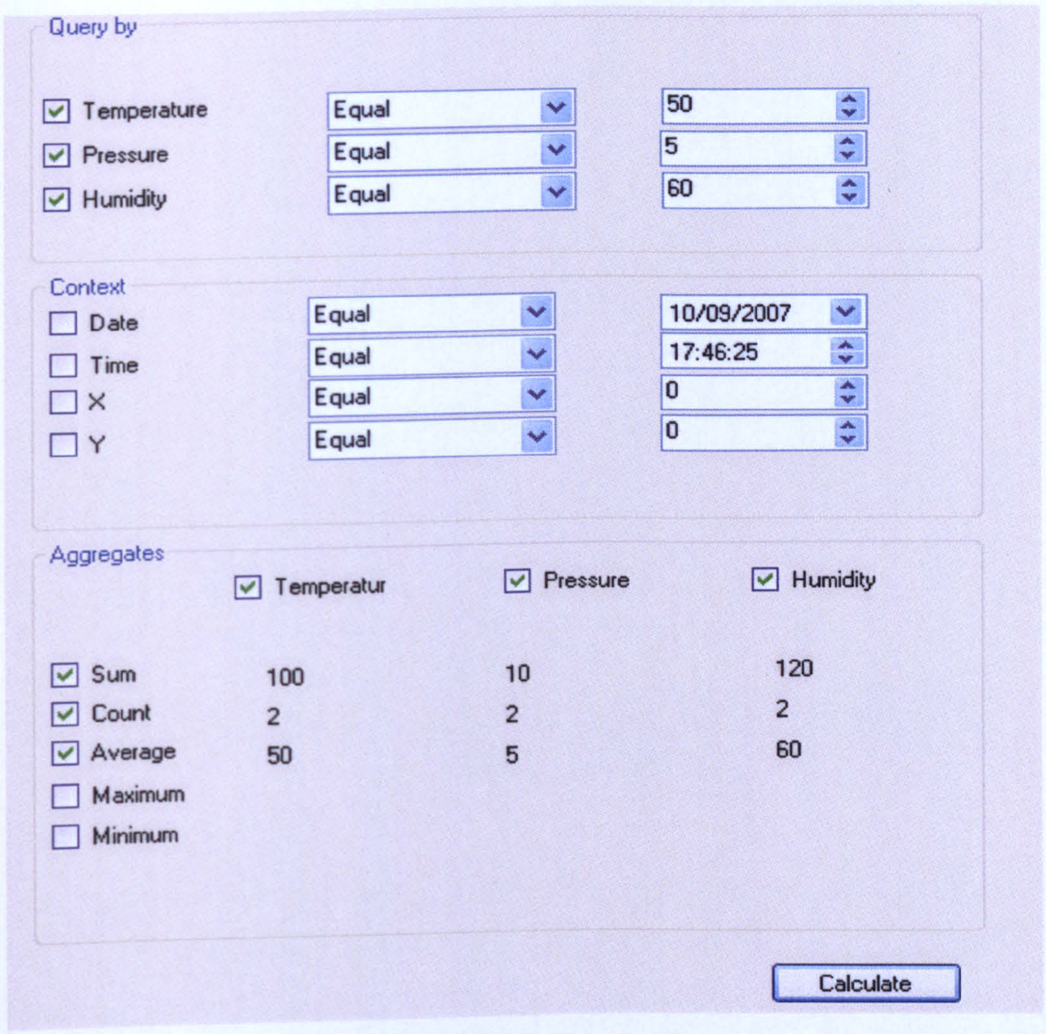


```
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 2
Priority 0
Priority 0
we aggregated 1 data at level 3
memory size is 1584
the memory limit is exceeded we perform aggregation at level 2
Priority 1
Priority 1
Priority 1
```

Figure 7-5 Aggregating data according to priority

Aggregation Service

This section gives the implementation details of aggregation service. This service has already been tested in the simulations using the prioritization algorithm. The results of the prioritization technique illustrated very clearly that it not only reduces the amount of energy consumed but also saves memory in order to accommodate more data. Here the objective is to demonstrate the basic idea of how to aggregate data using context and is just an example. It was explained in chapter 6 that in order to add context to data the application can specify a context. The data collected by the event manager (EM) or query interface service (QIS) will be aggregated according to that context. Finally the data is stored in the persistent storage (PSS) to be queried by the user along with the context. It has been implemented in C# in order to demonstrate proof of concept. Figure 7-6 shows simple aggregation without context where as Figure 7-7 shows aggregation with context which is date.


Query by

[✓] Temperature	Equal	50
[✓] Pressure	Equal	5
[✓] Humidity	Equal	60

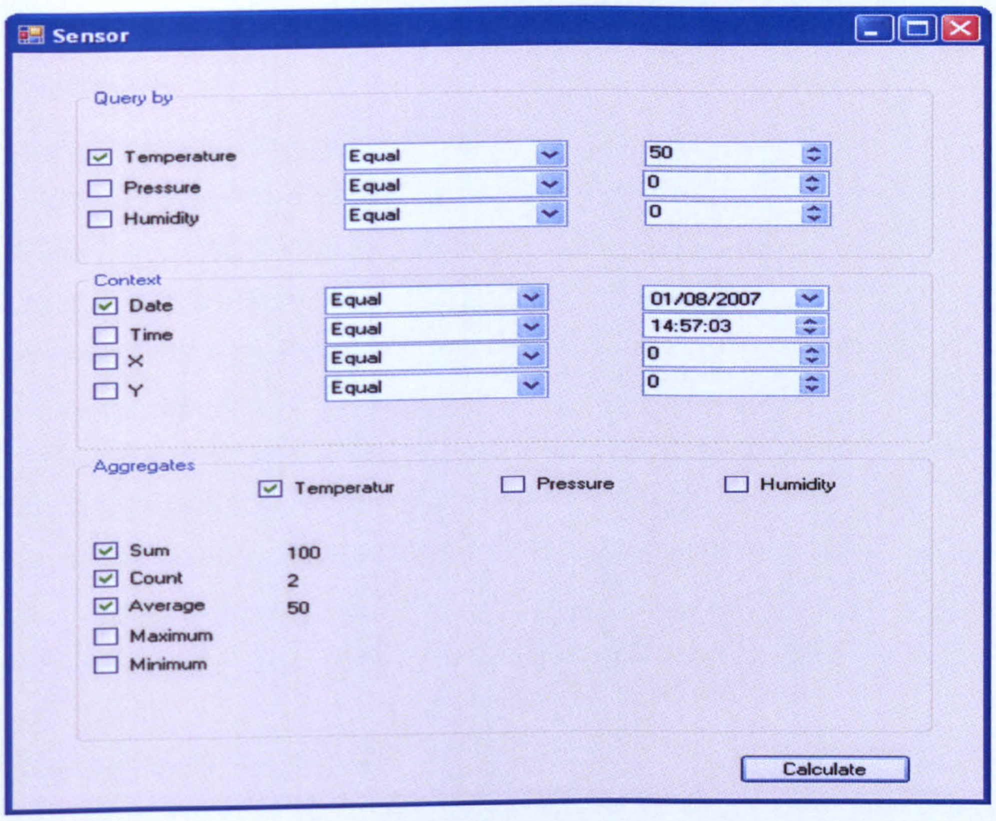
Context

[ ] Date	Equal	10/09/2007
[ ] Time	Equal	17:46:25
[ ] X	Equal	0
[ ] Y	Equal	0

Aggregates

	[✓] Temperatur	[✓] Pressure	[✓] Humidity
[✓] Sum	100	10	120
[✓] Count	2	2	2
[✓] Average	50	5	60
[ ] Maximum			
[ ] Minimum			

Figure 7-6 Query without context



Query by

☑ Temperature	Equal	50
☐ Pressure	Equal	0
☐ Humidity	Equal	0

Context

☑ Date	Equal	01/08/2007
☐ Time	Equal	14:57:03
☐ X	Equal	0
☐ Y	Equal	0

Aggregates

☑ Temperatur ☐ Pressure ☐ Humidity

☑ Sum	100
☑ Count	2
☑ Average	50
☐ Maximum	
☐ Minimum	

Figure 7-7 Query with date context

7.3 Evaluation

This section provides and evaluates the results against project requirements given in chapter 4.2. Mathematical analysis is also being used to validate the results. One of these requirements states that data is to be stored within the network for longer period of time. Chapter 5 explains in detail the design and working of our Persistent Storage Service. In order to see how much memory will be consumed to store a certain amount of data within the network and also find how much energy is consumed by WSN having limited memory and stringent resources. We used simulations to store events within the network to see how much memory would be consumed to store ten thousand events in a network of one thousand nodes. We only generated a single type of event and the radio range used was 10 meters. The parameters used are for comparison with other existing works. Generating 10,000 events with a length of 4 bytes each resulted in memory usage of 40 kb consuming 1147 milli joules of energy. The fact that 10,000 events with a length of 4 bytes each only consumed 40 kb is an indication that in reality it might be feasible to store data on the sensors and the figure 1147 milli joules indicates it consumes very low energy as well.

Another important requirement of the project also mentioned in chapter 4 is that the middleware should be robust in case of high node failure. This has also been noted in chapter 4 and 5 that in order to overcome the issue of robustness in data-centric design in case of high node failure rate or mobility more copies of data are required. So the next question needed to be addressed is how many copies of data would be sufficient to ensure there is at least one copy of data always available within the network. To seek an answer to this question the probability for a possible event is calculated which could result in unavailability of all copies of storage nodes. The probability of all storage nodes being unavailable can be calculated given the percentage of nodes made unavailable is known. We used Hyper-geometric distribution (Rohatgi *et al.* 2001) to calculate the probability of all data storing nodes being unavailable or having run out of batteries. The formula being used for the distribution is $\frac{\binom{D}{k}\binom{N-D}{n-k}}{\binom{N}{n}}$.

The parameters used are $N \in 0, 1, 2, 3, \ldots\ldots;$

$D \in 0, 1, 2, 3, \ldots\ldots, N;$

$n \in 0, 1, 2, 3, \ldots\ldots, N;$

$k \in 0, 1, 2, 3, \ldots\ldots, n;$

Here N is our total number of nodes within the network; D is the total number of storage nodes, n is the total number of nodes unavailable and k is the number of unavailable storage nodes.

If we assume that D and k are always the same, then we take $\binom{D}{k} = 1$

So we can have the formula $\frac{\binom{N-D}{n-k}}{\binom{N}{n}}$.

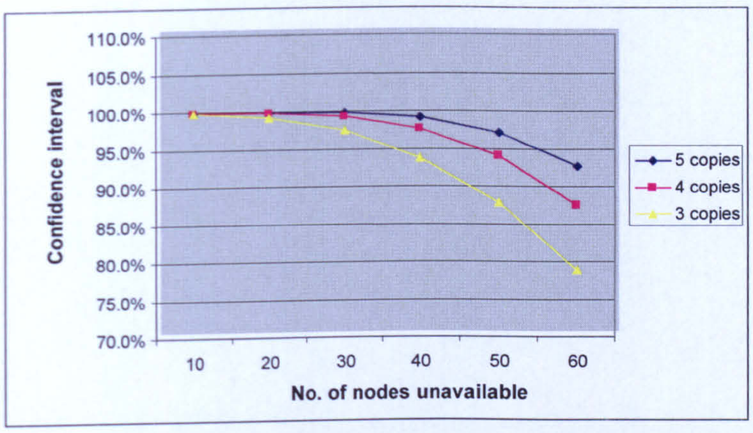
Now applying the above formula to an example we assume $N = 100$, $D = 5$, $n = 40$ and

$$k = 5 \quad \text{we get} \quad \frac{\binom{95}{35}}{\binom{100}{40}} = \frac{\frac{95!}{35!60!}}{\frac{100!}{40!60!}} = 0.00873993...$$

Total No. Of Nodes	No. of copies	No. of Nodes unavailable	Confidence interval
100	3	10	99.9%
100	3	20	99.2%
100	3	30	97.4%
100	3	40	93.8%
100	3	50	87.8%
100	3	60	78.8%

Table 3 Showing confidence interval for 3 copies of data in a sensor network

This means there is only a probability of 0.00873 that all or any of the storage nodes would be unavailable which even if magnified in reality could still be considered as encouraging. In percentage terms it will come out as .0873% with a confidence factor of 99.1% which is negligible. In other words if we have 5 copies of data it will give us a confidence of 99.1% that all 5 copies of the data could survive. Using the above formula we tried to calculate the confidence factor for different number of copies as well as different number of nodes unavailable. Table 3 gives complete details. In graph 1 the yellow line represents data given in Table 3 which clearly shows that if percentage of the nodes unavailable is high, more copies of data would be required. For example, with 60% of node unavailability which is a very high rate and might not be always the case in reality, the confidence factor for 3 and 4 copies (refer to Table 3 and 4) is less than 90%. Less than 90% is not a desirable requirement where as for 5 copies it is above 90% (Table 5) which is our aim. This means the number of copies required depends on the percentage of nodes unavailable.



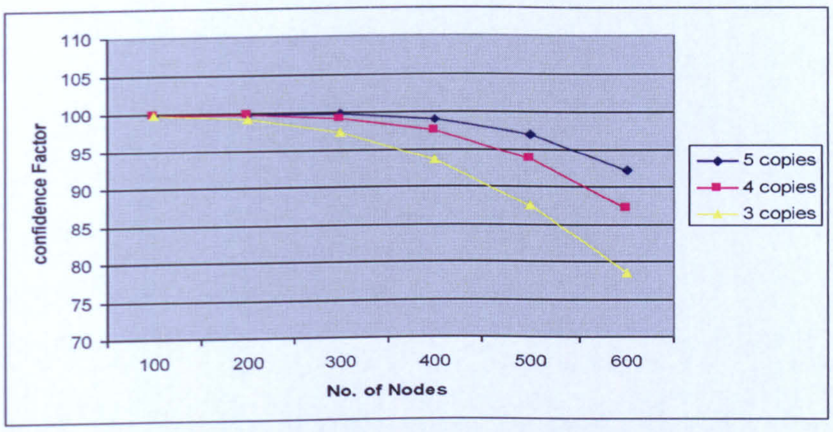
Graph 1 No. of nodes unavailable per 100 nodes

Total No. of Nodes	No. of storage copies	Total No. of nodes unavailable	Confidence Interval
100	4	10	99.9%
100	4	20	99.8%
100	4	30	99.3%
100	4	40	97.6%
100	4	50	94.1%
100	4	60	87.5%

Table 4 Showing confidence interval for 4 copies in a sensor network

Total No. of Nodes	No. of storage copies	Total No of nodes unavailable	Confidence Interval
100	5	10	99.9%
100	5	20	99.9%
100	5	30	99.8%
100	5	40	99.1%
100	5	50	97.1%
100	5	60	92.7%

Table 5 Showing confidence interval for 5 copies in a sensor network



Graph 2 No. of nodes unavailable per 1000 nodes

The next step was to check if increasing the number of nodes would have any affect on the number of copies required in a network but as graph 2 shows it had a minimal affect as the number of nodes are increased from one hundred to one thousand. Graph 2 follows the same pattern as graph 1 therefore it could be safely concluded that increasing the number of nodes does not affect the number of copies required. The above set of simulations is a positive indication to say it is feasible to save data within the network for longer period of time. The other point worth noting is that multiple copies of data will make the system robust and reliable because one copy of data will always be available, however, the number of copies was found to be dependant on percentage of nodes made unavailable. This will help the designer of the network not only to know in advance how many copies of data would be required but also how many copies of all the services described in our framework.

I) Simulation for Aggregation using Snooping Technique

In the second set of simulations the snooping technique is tested. This technique has already been described in detail in section 5.3.1. Previously TAG (Madden *et al.* 2002) proved that this technique is less costly in terms of energy consumption in comparison to other techniques. However, they did not test the affect of different parameters on the performance of the snooping technique. The objective of these tests is to see if changing the network parameters, for example, topology or event range has any effect on the performance, in other words, to see the effects of the snooping technique on the middleware. Although theoretically it seems that changing the parameters should not have an effect on energy consumption. Another assumption TAG made about snooping was that it is only effective in single aggregates, such as, max/min but is not effective in averages because of the expected error margin. The snooping technique is also tested for average to calculate the error margin to see if it is not feasible at all to use this technique for average or is it possible is some cases. Again the Georgia Tech Network Simulator (GTNetS) (Riley 2003) is being used to conduct these tests.

There were two objectives that were achieved from these tests. The first objective was to find out how much this technique is effective in reducing the energy consumed in case of calculating a maximum value and also to see the effects of changing network size, event range and topology in terms of saving energy. It has been established before in this thesis that network size, event range and topology may change frequently in WSN therefore it is important to know the affect of these parameters on the performance of this technique. The second objective was to check the snooping technique's effectiveness in case of average and the accuracy of its result because it has not been tested previously for average assuming that the results will always have an error margin.

No. of nodes deployed	No of Snooping Nodes	Percentage
900	53	5.88%
400	57	14.25%
196	62	31.63%

Table 6 Uniform topology 250x250 grid

No. of Nodes deployed	Snooping Nodes
900	20/2.22%
400	20
196	20

Table 7 Uniform topology 200x200 grid

A. Uniform Topology

Keeping the topology uniform several tests are conducted. The first test shows if the percentage of snooping nodes increases or decreases by keeping the network size constant and reducing the number of nodes deployed. Using 10 meters distance between the nodes and network size of 250 x 250 the results were obtained as shown in Table 6.

It can be seen from Table 6 that reducing the number of nodes has a positive affect because fewer nodes are sending the data thus consuming less energy. In the second round of tests the network size is reduced to 200 x 200. As can be seen from the results shown in Table 7 the number of sending nodes reduces considerably and remains constant. This means snooping technique works more efficiently for small sized networks.

To study the effect of distance on energy consumption we increased the distance between the nodes. As we can see from Table 8 there is no significant effect on the percentage of sending nodes but has considerable affect on the total energy level. This is so because by increasing the distance more nodes become involved and thus increases energy level. So distance does not have a substantial affect in terms of the percentage of participant nodes but has considerable affect on energy consumed.

Event Range	Sender Nodes	Percentage	Energy Consumed
10m	15	40%	0.444mj
20m	21	52%	2.44mj
30m	37	56%	4.66mj
40m	61	55%	7.55mj
50m	101	52%	11.55mj
60m	137	51%	15.77mj
70m	185	52%	21.55mj

Table 8 Effects of Distance on Snooping

Next step is to calculate average using the snooping technique having uniform topology. The results are calculated by giving the nodes a criterion. If the values of the nodes were within a certain range of the actual average, only then the nodes would send their values. The worse case scenario is also calculated mathematically and the simulation results back it up as shown by graph 3 and 4. For example, if one node is sending the value 10 and all the other nodes sense a value of 11. Keeping the error margin <= 1 the value of 11 does not fall into the error margin and hence other nodes do not send their values.

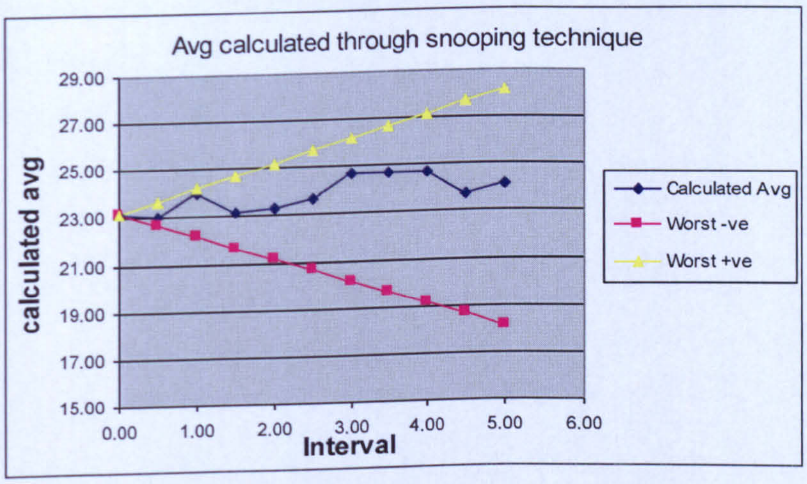
Mathematically if e is the error, a is the actual average and the number of nodes is represented by n. We can say $((a \pm e) + n^*a)/n+1 \rightarrow a$ the actual value returned is $a \pm e$.

So by applying the above formula to our example we can say

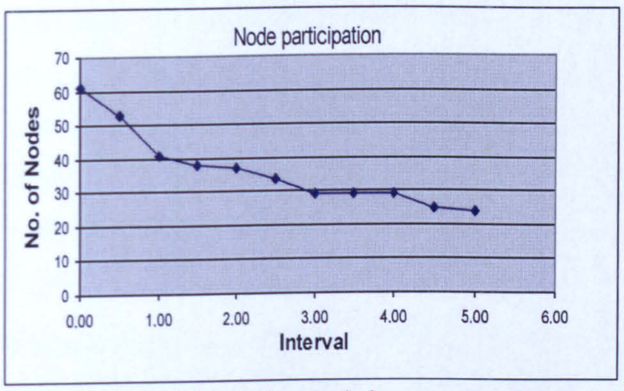
$(10+ (n^*11))/(n+1) = (10/(n+1)) + ((n^*11)/(n+1))$

$\rightarrow 0+ 11n/n+1 = 11(n/n+1) \rightarrow 11$

Using this above formula worse case positive and negative are calculated which gives the following graph. It can be seen by looking at graph 3 that the simulation results are between the worst case positive and negative. It can also be deduced from this graph that as the interval is small the average is the same and almost all the nodes are participating but as the interval increases the results gets worse and again improves as the interval gets larger. The node participation graph 4 shows that the number of nodes keeps decreasing, however, this means less energy consumed. Hence, it can be concluded that the error margin falls within the range of ±1.5 %.



Graph 3 Worse case scenario in calculating average with snooping



Graph 4. Nodes participation in calculating average using snooping

B. Random Topology

The previous tests were done with uniform topology so the next step was to change the topology. The results shown in Table 9 reveals that although the percentage of sender nodes decreases in the Random topology the energy consumed increases this may be because being a dense network, more nodes are involved.

It is apparent from the simulation results shown above that changing the topology does have considerable affect on the participant nodes and thus the energy consumed is increased. Overall, the snooping technique does have a positive affect by reducing the number of nodes and thus reducing the energy consumed.

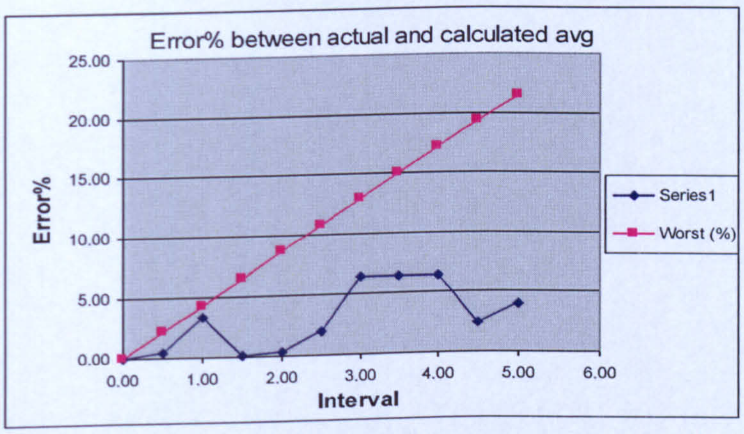
Event Range (mtr.)	Senders Nodes	Percentage	Energy Consumed (mj)
10	29	20	7.50
20	135	15	32.69
30	306	16	77.19
40	504	14	109.53
50	803	13	170.00

Table 9 Random topology

C. Discussion

Looking at the above simulations it can be concluded that the snooping technique works under most circumstances for calculating a maximum or minimum value. However, network size, distance, and topology do have a considerable effect on the energy consumed which as mentioned earlier is vital in WSN. Reducing event range and network size improves energy consumed as is evident from Table 6 and Table 7. Changing the topology from a uniform to a random topology also increases the energy consumed as shown in table 7. This may be due to the fact that in a Random topology the network is densely populated and therefore more nodes are involved in sending data hence consuming more energy.

In the case of calculating an average the results show that if the interval is large the error keeps decreasing. This is a positive sign in the sense that the error margin is under 1.5 % as can be seen in Graph 4. Which means it could be used in some applications where accuracy is not very important, for example, in the case of global warming in which the scientists only need to know whether the temperature is rising or otherwise and in other similar scenarios where the requirement is only to know whether the trends are going up or down. Therefore, it can be safely assumed that the snooping technique can also be used in calculating average in some scenarios but not under all circumstances.



Graph 4 Error Margin for calculating average using snooping worse case scenario

Looking at the above discussion it can also be safely said that this technique will have a positive effect on the working of the middleware and the aggregation service by reducing the energy consumed by the framework. Snooping can safely be used under most circumstances and relied upon to save energy. This fulfils the requirement for the project to be energy efficient as already mentioned in chapter 4.

7.4 Project Evaluation

In this section the MidWSeN is being evaluated against the project requirements discussed in chapter 4. The first requirement is to keep the framework adaptable and flexible to accommodate as many situations as possible or as many applications as possible. In order to achieve this service-oriented architecture is selected so the middleware could be deployed in any environment without knowing the underlying platforms or architecture. The framework is adaptable and flexible in more ways than one. The use of optional and core services makes the framework to adapt to different environments and application requirements. The optional services will not be deployed if the applications do not require these services. For example, if an application is event-based than it will not require the QIS, where as, if it is query-based than it may not require the EM. It could also use both functionalities at the same time if enough network resources are available and application requires it. Therefore, adaptability has been kept as an individual requirement. The size of the framework could also be reduced by selecting only one of the optional services if enough resources are not available to support the framework.

Further, there could be any number of copies of the services depending on applications and network requirements. For example, as already mentioned in the simulation section above, in a large network having a high unavailability rate of nodes more copies of the services working could be made available within the network.

Our services also have flexibility and adaptability individually. For example, as mentioned in the description of the aggregation service, the use of *MyAgg* method in which the application can specify its own particular aggregation method. In the data adaptation service the application can specify any particular context which fulfils its specific needs. Also the user is allowed to use any particular protocol or data structure which makes it feasible in many environments.

The second requirement is to use in-network processing because it reduces the number of messages being communicated over the network thus saving precious resources like energy and bandwidth. The choice of data-centric design helps in providing in-network processing because it processes the data near the source. For example, the temperature sensors will sense and store the data on the same sensor or neighbouring sensors does reducing the amount of resources being used.

In-network processing is being used in the aggregation service and the persistent storage. Aggregation is important as mentioned in detail in chapter 6 and it has therefore been kept as a requirement. By aggregating data within the network, data communicated over the network is reduced. Filtering redundant data by different techniques like snooping also helps in reducing redundancy. The persistent storage communicates data only when a query is sent so this also considerably reduce the amount of data communicated over the network. Persistent storage is also another key requirement because it helps in providing historical as well as helps in answering spatial and temporal queries by adding context. Adding context is an important requirement and the Aggregation Service provides this functionality by adding time, date or location as context which could help to answer spatial and temporal queries. Furthermore, the framework allows the application to add its own context to the data. This is an important contribution of our framework because it not only helps in economizing the resources but also makes the data more meaningful.

Another important requirement is robustness as node failure rate may be high. Our middleware keeps many copies of the services and data in order to keep the system robust. The next requirement is of minimum hardware and software required to deploy the framework. The framework is lightweight and does not require any special hardware apart from sensors or a PC. In total there are seven services out of which one lies on a more resourceful network and on a powerful node such as a computer, the Application Interface Service, where as the remaining services may reside on the sensor nodes. The AMS, PSS and AS are the core services, and only one can be selected from the remaining three optional services, so effectively only four services may be residing and are used on the sensor network to get the full functionality of the framework. The reduction of the size of the framework not only makes it flexible but also light weight. Individually also the services are not heavy in terms of storage or programming overhead which is evident from the design and implementation details. This framework is designed for distributed operations and the services will be on different nodes which cooperate with each other to perform a certain task

So in conclusion it can be said that the framework is adaptable and flexible enough to be useful in many environments and thus fulfils all the project requirements. Having said it should not be assumed that the framework is complete in every respect. There are certain aspects that are important for WSN which has not been included. For example, security, this is very important, but not part of the framework. Generally, security is important for a network because the network has to be protected from unauthorised access. If a node in sensor networks is compromised then it can compromise the security of the whole network. Therefore it is not only important to protect the network from malicious attacks but it is equally important to identify malicious nodes and recover the network. Specifically, with respect to the framework presented in this thesis, data is being stored on the nodes and of course it is important to protect data so that only authorised personnel can access the data.

The other important factor which is very vital for sensor networks is self-management. Although the Application Management Service is present the framework which does perform certain functions, for example, discovering other services but there are other aspects that could further develop the framework. Other issues could be having knowledge of the different copies of data and directing the query to that exact location of the data or directing it to the nearest copy of the data. Then there is also the issue of managing different copies of services that might be used by different applications. The framework might be required to include another service or add some more functionality to the existing services in order to collect more information about the network itself and share it within the network or with the applications.

The reason for not including these important aspects was to first fulfil the basic requirements and that is what exactly have been done in the framework. A basic platform is given on which one can build a whole new structure by adding other important services or functionality such as security. Therefore, these functionalities are important but outside the scope of the study for the time being and forms a part of the future works.

Another issue that needs clarification is that although the data is being stored within the network on the nodes, how or what data structures should be used are not being suggested because it is being looked at from the middleware point of view and not from database point of view therefore this also remains as an open research issue. However, since data-centric design is being used therefore it has to conform to this particular design.

In a heterogeneous environment other optional or temporary services may also be included which can be made available temporarily if the network can support it and taken off when ever the resources becomes scarce or they are not required. This could be done particularly in indoor applications where a device doing specialized service is added whenever required. The next section gives a comparison with the existing works in order to highlight the distinguishing features of the work presented in this thesis. It illustrates the fact that although there may be specialised works which are more efficient, the framework presented in this thesis is novel because it provides a more general and holistic approach.

7.5 Comparison with Related Works

The main objective of this research is to provide a comprehensive middleware framework that could incorporate most major if not all the main features. The proposed framework also utilizes the work done by other research teams. Therefore this framework shares many similarities with existing projects. There are of course many distinguishing features of this framework which have already been explained in detail in previous chapters. This section is specifically dedicated to comparing other significant research projects with the proposed framework.

Information gathering is a primary function of WSN. Therefore data storage is also an important feature of the middleware. In this regard TinyDB (Madden *et al.* 2005) and DSWare (Li *et al.* 2004) can be considered to be most relevant to the work presented in this thesis. Both use in-network processing and data-centric storage which have already been defined in chapter 4 and 5 respectively. Although the significant difference is that their approach is a pure database approach where as the approach adopted in this thesis is to concentrate on other aspects, for example, events and context as well in addition to provide services such as aggregation.

TinyDB (Madden *et al.* 2005) uses a database approach and tree topology. This approach views WSN as a single distributed entity, which could be queried using SQL-like queries to assign sensing tasks by the user (Bonnet *et al.* 2000; Levis *et al.* 2002; Madden *et al.*

2005). For example, in TinyDB it is difficult to handle complex sensing tasks like building a relationship between sequences of events in certain geographical areas. In the MidWSeN framework it is possible to build such a relationship using the persistent storage because the data can be stored with context. TinyDB is limited because it only considers real-time queries whereas the middleware framework considers not only real-time queries but also historical queries. Another important aspect of our work is context which means that user can send context- enhanced queries. Hence the TinyDB approach though easy to use, suffers from some scalability issues as well as from limited expressiveness. MisWSeN is a middleware approach and therefore different from TinyDB. The framework is not advocating any specific topology or data structure which makes MidWSeN framework more flexible.

DSWare (Li *et al.* 2004) has attempted to include in-network processing of compound events but in a restricted manner. Their approach uses data-centric storage with a slight modification. They replicate the data to several physical locations to improve the robustness of the system. Different copies of data are synchronised when the system workload is low. This means data will not be synchronised at all times. If, for example, before updating other copies the node holding the data becomes unavailable then that data will be lost completely. They have also included some real-time scheduling mechanisms providing flexibility at the cost of complex application development. They are using SQL-queries sacrificing more memory and processing capabilities for flexibility and expressiveness. In comparison the MidWSeN framework also advocates more than one copy of data which is the persistent storage but it updates the copies all at the same time which ensures all copies of data will have the latest or updated information and no data will be lost in case of destruction of a node.

This section is only going to discuss the works using in-network aggregation because the proposed service is also processing the data within the network. In Tiny Aggregation (TAG) (Madden *et al.* 2002) suggest a service using database query language technique of selection and in-network aggregation. The query is broadcast to the network and data collected using a tree structure only combining relevant data and sending it to the base station as the data is being passed up the tree from child nodes to parent. TAG does not propose any particular routing protocol and only concentrates on the aggregation. It restricts the user to a particular data structure which is easy to use but does not cover all types of queries, for example, temporal or spatial. Another important work is Directed Diffusion (Intanagonwiwat *et al.* 2003), a communication paradigm which diffuses an interest by flooding the network and routes the data back by setting gradients on the most reliable paths. Travelling data is aggregated on its way back through different nodes. Hence, they provide the routing mechanism too along with aggregation. Again aggregation is being combined with routing which is a lower level operation. Both the works limit the user to SQL or SQL- like languages. The proposed MidWSeN framework does not limit the user to any particular topology or data structure. Again both TAG and Directed Diffusion concentrate only on aggregation and routing whereas the MidWSeN framework provides other functionalities as well. Therefore both of the above mentioned works might be more efficient in what they propose but are very limited in terms of scope and functionality.

The above mentioned works are also limited in their context. They only take into account a pure sensor environment whereas in reality it might be a heterogeneous environment. If we are to embed sensors in our daily life then they might be assisting the more traditional computers with more resources and power. In other words sensors can be in different types of environment which varies from very constrained ones to not so constrain. The proposed work is significantly different from the above mentioned works in many respects. First the whole concept is to save the data within the network and send only the processed data to the user on query because communication is more costly than processing. Second the framework is not restricting the user to any particular query language. The user can query the network using any type of suitable software. Finally no recommendations are being enforced for any particular topology or data structure hence, making it flexible to use in different environments, that is, it could easily be adapted to a very constrained environment and visa versa.

7.6 Summary

This chapter provided an evaluation of MidWSeN framework proposed in this thesis. For evaluation purposes different mathematical calculations and experiments are conducted to prove the design and concepts presented in this thesis. The results, graphs and tables are provided in order to confirm the functionality of the framework and give proof of concept.

The framework is evaluated against the requirements outlined in chapter 4 to conclude that the framework have been successful in reflecting those requirements in the design as well as the implementation. The performance of the framework indicated by the results provides a novel contribution to the area of WSN by providing a foundation to a holistic approach of combining query, events and context. It also provides a critique of research work presented in this thesis by taking an impartial view of the research done and further research questions raised during the course of this research.

Finally the chapter provides a comparison with the existing work and concluding that although the work may not be as efficient as compared to other related works being dedicated to a single aspect but it certainly has a broader and flexible approach. It is not limited in scope and provides a more comprehensive solution to the problems discussed in the early chapters. The next chapter provides conclusions and future research areas are indicated which came up during the course of the existing work.

Chapter 8 CONCLUSIONS AND FUTURE WORK

8.1 Introduction

Sensors have revolutionized the world of information by adding physical dimension to the data. Scientists can foresee the future as a Smart and integrated world. In such an environment every object on our bodies and environment would be able to connect and communicate with us and each other. With the decrease in the cost of sensors and improvements in sensor technology, soon every company will consider it important to have their own sensor network as they have numerous applications. This can lead to a Sensornet which can replace the Internet. The technological advances and the tremendous potential these small devices offer will eventually lead us to this vision of a Smart world.

Wireless Sensor Networks are challenging because of their very limited resources. Taking a short tour of history in the Introduction revealed that a lot of different types of hardware and technologies are working together which means operating in a heterogeneous environment. Therefore to handle heterogeneity and its related issues another layer of abstraction called middleware is required. Existing middleware techniques for wired or traditional networks, however, were found not competent enough to be applied to WSN because they are resource intensive. Wireless sensor networks have unique characteristics because of their very limited resources and requirement to operate without human intervention in harsh terrains. Therefore a need for new middleware strategies was found to be necessary in order to cope with the unique characteristics of the WSN. The middleware systems developed so far have been restricted because they are application specific in order to make them efficient and cannot be reutilized for different applications. As research has proved, in-network processing and storage is more cost effective in terms of resources, therefore processing and saving data on sensors for longer periods within the network proves more beneficial in terms of resources. The main objective of this research was to create MidWSeN, a flexible middleware framework that would work with different applications to acquire context-aware data.

There has been a considerable amount of work done in this area and sensor networks have come along way since its initial stages. Early sensors were very basic and could only sense and transmit now they can not only do this but also receive process and store data. Similarly each network was dedicated to a single application or single type of application but now researchers are exploring the avenues of using the network for multiple applications. The focus of this thesis was to build a flexible middleware framework that can facilitate as many applications as possible within the limited resources available. This would pave the way to sensor reusability and enable the sensors used by different applications to talk to each other and create a well integrated environment. The Middleware Wireless Sensor Network (MidWSeN) framework has tried to achieve this goal by combining different services. This framework combines queries, events and context to enable applications to extract the required data or information in a flexible and not so limited manner.

The rest of the chapter gives a summary of the whole thesis and the contributions made in this research. This work can be the foundation of other issues related to the area and some of the future work is also identified within this context.

8.2 Thesis Outline

This section gives a brief summary of the whole work presented in this thesis. The subsections are divided according to the topics discussed in the different chapters.

8.2.1 Introduction

The first chapter underlined the need for having a flexible middleware for WSN. It briefly narrated the history of different technologies starting from the advent of networks. It briefly defined area of wireless sensor networks and challenges related to this field. It identifies the need for a middleware which can be utilized by multiple types of applications. It also described the design goals and novel contributions of the work.

8.2.2 Background

Chapter 2 and 3 gave background information about sensors and an overview of the existing significant works done in WSN in some detail. Chapter 2 discussed wireless networks in general and WSN in detail. It described the characteristics, design principles and applications of sensors. It also gave a brief description of the hardware, technologies and software related to WSN. Some of the important and currently active research areas in wireless sensor networks have also been mentioned to give an overview of research activities going on in the field of WSN.

Chapter 3 gave a comparison between existing WSN middleware and also explained the reasons for traditional network middleware not being applicable to sensor networks. It discussed the existing works and their drawbacks in terms of flexibility, ease of use and efficiency. Further it elaborated on the point of a new approach to the problem of limited scope of existing middleware in WSN.

8.2.3 Design of a New Middleware Framework MidWSeN

Chapters 4, 5 and 6 discussed the overall design of the framework as well as the design of individual and in particular its important the Persistent Storage and Aggregation Services. Chapter 4 outlined a new set of requirements which are considered beneficial towards devising a better and flexible middleware. This chapter also presented a novel middleware framework design for WSN which combines three important features query, events and context. It also presents an overview of the novel core and optional services included in the framework which makes it flexible and their individual contribution to the overall structure.

The purpose of Chapter 5 was to highlight the novel Persistent Storage Service in greater detail. This service is used to store data within the sensor network on the sensors for longer period of time. This chapter explained the benefits of storing and processing data on the nodes and also provides the mechanism of the process done within the framework. It presented a novel algorithm for prioritization being used to provide memory for important data in case of memory shortage. It also explained how other services within the framework interact with this service in order to provide an efficient system.

Chapter 6 explained another novel service the Aggregation Service in detail. The aggregation service is not only used to save memory but also add context to data in addition to the traditional way of compressing or filtering data and saving resources which is a novel in WSN. This service aggregates data using context. The chapter also explained the necessity to keep aggregation as a middleware service and how it functions within the framework to provide efficient use of resources.

8.2.4 Evaluation and Results

Chapter 7 provided the evaluation of the framework and the implementation details. The main objective of that chapter was to show the performance of individual components of the framework and to demonstrate the working of the services within the framework. That chapter illustrated the feasibility of the work by presenting results. TinyOS and nesC was used to implement the Persistent Storage to show that it is possible to store data on the sensors. Simulations were used to store 10000 data elements and calculated the memory and energy to show the feasibility of storing a large amount of data and found the results encouraging. The Prioritization algorithm was also implemented using GTNetS simulator to show that it reduces the amount of data considerably making space for more data to be stored. The Aggregation Service was also implemented to demonstrate how data can be aggregated using context provided by the user. The work was evaluated against the requirements given in chapter 4 and found that the framework have been able to fulfil these requirements. A comparison of the framework against other related works is also presented in detail. These works may be more efficient in certain ways but their approach was found to be very limited and inflexible. In conclusion one can say that the framework has been able to achieve the design goals and MidWSeN provides a much broader and flexible approach compared to the works previously done in WSN.

8.3 Contributions

To broaden the base of the middleware called MidWSeN have not focused on a single aspect of WSN. MidWSeN has been designed to be flexible and adaptable to accommodate many applications. A number of contributions are made in this thesis which is highlighted in this section. The contributions provided in this thesis are detailed as follows.

- A deep insight to the field of middleware for sensor networks were achieved by conducting an in depth study of the area which resulted in the production of a new set of requirements. These requirements try to encompass the problems or shortcomings felt in the existing works. The main problem with other works is that they focus on a single aspect and are very limited in their approach. These requirements became the basis of establishing the design of a new and better middleware framework for WSN.

- The first contribution is the design of a new, light and energy efficient middleware framework (Javed *et al.* 2005) called MidWSeN which combine queries and events. In addition it also provides mechanisms to provide context enhanced data to the user. This has already been mentioned that the problem with other works is their limited approach where as the approach of the research presented in this thesis is to broaden the base by encapsulating different aspects in the same framework. This framework can be used for both queries as well event based applications and is flexible to be customised to application specific requirements. The framework has core and optional services which can be adjusted according to the requirements of the network or applications using the network. The framework not only combines most important aspects mentioned above but it is also scalable and robust. It has the ability to work in normal as well as emergency situations providing mechanisms to improve latency in case of real-time situations. The framework does not bind the user to any particular environment by not specifying any particular protocols or data structure. It can work equally well in constrained as well as heterogeneous environment. Using service–oriented architecture helps in deploying the middleware in almost every environment without worrying about the underlying platform.
    - The framework consists of some core and also optional services. The optional services are used to adjust to applications requirements as well as available resources making the framework flexible. The services can be customised individually to application requirements keeping in view the application specific nature of sensor networks. There are multiple copies of each service available decided according to network size and node availability rate to make the system robust. Scalability is achieved by

making it easy to introduce new services having service-oriented architecture.

- Persistent Storage Service (Javed et al. 2007b) is one of the novel core services which allow the data to be stored for longer period time on the nodes within a sensor network. With the advancement of technology processing and memory capabilities are increasing therefore making it possible to store and process the data on the sensor nodes. In addition being a distributed environment the sensors are capable of collaborating to perform a task. Storing data on the sensor nodes helps in providing historical data as well as helps in developing explicit or implicit relationships between data by providing processed and context enhanced data to the user. Storing and processing data within the network also saves precious resources such as bandwidth and energy by reducing the number of transmissions to the base station because more transmissions mean more resources consumed. The Persistent Storage Service, with the help of other services within the framework, provides mechanisms to store data on the sensor nodes for longer period of time and retrieve context enhanced data.
    - WSNs have very limited resources and storing data within the network might result in utilization of all the memory available in the network. A novel Priority algorithm was presented to manage memory in case such a situation arises where more memory is required to store new in coming data. This algorithm aggregates the data of lowest priority provided by the user to create space for more important data whenever the system crosses a certain memory threshold. Data is prioritized by the user according to the application requirements. This helps in ensuring the availability of memory for more important data and also manages in keeping the data in aggregated form rather than deleting it totally. The advantage of keeping data in aggregated form is that the data will always be available for future use if required which is the novel aspect of Prioritization algorithm.
- Aggregation Service (Javed et al. 2007a) is another novel core service in the framework which not only aggregates data according to context but also adds this

context to the data to be stored in the PSS. Thus this service is being used to provide context-enhanced data to the user. Aggregation also helps in saving memory by compressing and filtering the data. This service has the flexibility of allowing the application to customize the service to its specific requirements by having their own aggregation methods other than the standard ones provided as well as adding their own specific context to the data.

- Application Management Service (AMS) is also a novel core service provided to help manage other services. This service residing on nodes within the sensor network provides optimization by processing required information and deciding the optimal path to other services. The AMS communicates and links other services within the framework to the base station. This service also helps to inform the user if an emergency situation occurs with the help of the Rule Service if a rule or set of rule fires.
- The ideas presented in the thesis have also been evaluated by simulations and calculations. The implementation of the framework is a contribution which provides better utilisation of the resources in sensor network.

8.4 Future Work

Many new research questions have been raised during the course of this research which can form the basis of further research.

8.4.1 Security

Security is an important issue and an active research area in WSN. It is not only interesting because of the stringent resources of these types of networks but also due to the fact of its distributed nature. The general issues have already been mentioned in chapter 2. This chapter is looking at it specifically from the framework point view. In an environment where several applications might share or reutilize sensor nodes security is of prime importance. Another context to the framework is that data is being stored on the nodes; therefore it is absolutely vital to protect this data from malicious nodes or applications which can misuse this information.

It will be interesting to know how adding security features will affect the framework. It might be in the form of another service which can provide a general policy to help protect the network, for example, by using key management techniques (Kifayat *et al.* 2007) with provision for individual applications to provide an extra layer of protection for their own specific data. Encryption/decryption techniques can be used to communicate and store data. If data is send in an encrypted form than it will affect one of the technique used in the framework, snooping. It has already been mentioned in the section describing the snooping technique that if there are security issues than the developer can override the snooping method. It will require encryption/decryption algorithms to be applied to data communicated over the network which will affect the entire framework because six out of seven services exchange data.

If the data is stored in an encrypted form than significant changes are required in the AMS, QIS and PSS while querying data especially historic data. Data cannot be queried if stored in encrypted form so data will have to be decrypted before querying data. Encryption/decryption algorithm will have to be employed to encrypt/decrypt data every time stored data is queried which might prove costly in terms of resources, especially energy. To store data in encrypted form will also prove costly in terms of storage because encrypted data occupies more storage space than data stored in unencrypted form, in which case storing data in encrypted form might not prove feasible. In a distributed network each node is important because compromising a single node can affect the whole network. So it is important to provide some sort of security. In such a case the best option is to provide security at different layers, such as using encryption/decryption technique at communication level, in order to make the framework more resilient to enemy attacks. A combination of different security techniques together with the applications ability to provide its own security will make the security of the network less vulnerable to security threats.

There are other security issues such as to identify malicious nodes or nodes which are compromised. Security is important and it will be a challenging to introduce security to this framework. It might require significant changes to the framework to provide a light weight security mechanism.

8.4.2 Self-Managing Sensor Networks

One of the characteristics of sensor networks is that they are self-managing without human intervention because they may be deployed in difficult terrains inaccessible to humans. Therefore Self-management (Yu *et al.* 2007) is another active research area which needs to address a number of issues. Some of the major and general issues are already mentioned in chapter 2. Other issues that came up during the course of this research are how to efficiently gather more information about the network which could help assess in managing the network efficiently. This is called self-awareness which requires information about the environment so that further action can be taken to self-optimize the system. For example, it can gather information about the network to decide how many backup copies of data are required to ensure the availability of data to the user or data should be retrieved from the nearest copy of data in order to improve latency and save resources. In sensor networks energy is the most important resource to ensure longer lifetime of the network. How to collect this information, which criteria is to be used to enforce such policies and where will it be decided are all open research issues which needs to be addressed.

Another interesting issue is self-organization for distributed coordination. To formulate a policy or policies for locally managing different applications to use the same network, share data and other resources by reutilizing nodes. This would require real time scheduling policies to ensure smooth working of the network. For example, multiple applications are using the same sensor network one application requires dense deployment of nodes, so more nodes are being utilized but the next application utilizes lesser amount of nodes to be active, therefore rest of the nodes are automatically sent to sleep mode. The overall framework would be affected by introducing such policies especially the AMS will be significantly changed.

To keep collecting information about the energy level of network and enforcing local policies to extend the lifetime of the network is vital. The framework already has an Application Management Service which can be used to collect information about the network and ensure that local policies for management are enforced. Therefore self-management policies can easily be incorporated into this framework.

8.4.3 Other Research Issues

Another important issue that needs to be addressed is how to evenly distribute the different copies of data and services within the network. A mathematical formula has been provided to try to find how many copies of data are required to ensure there always be at least one copy of data available at all times within the sensor network. The same formula could be used for determining copies of services but how to distribute these copies of services evenly or in a specific pattern within the sensor network is also an issue. This could depend on the usage of the network and type of applications that are using this network.

The framework is using a service-oriented architecture it will be interesting to see how other temporary services can join and leave the network at will. These temporary services can join the network, offer some specialized service for a specified period of time and leave the network. Policies have to be devised to give permission for these services to enter the network without affecting the existing services; this is another challenging research issue.

8.5 Final Remarks

Wireless sensor networks are challenging because of the very limited resources. Middleware is important in WSN because of the heterogeneous environment and the operating systems are still in the development stage as this field matures from the experimental to the practical stage. Therefore there is a need for a middleware which can bridge the gaps left unfulfilled. Now there are examples of several applications where sensors are being used in the real life application. These experiences of real world applications have a great impact on the research because it opens a whole new era of modification and new requirements. As more real world applications employ sensors more emphasis will be laid on the fact that data is to be stored and processed on the sensors to reduce the burden on resources transmitting raw data or semi-processed data as well as make it more adaptable to diverse applications and sensor reuse. Technology is improving at rapid speed to catch up with the new requirements. The trend has already started, as mentioned in several places in this thesis, as big companies like Intel and Sun Microsystems are providing more powerful sensor technologies. Therefore it is really important to start thinking about placing sensors in a heterogeneous environment and develop ideas and software to be more adaptable than specific. Although there is always the trade-off between efficiency and flexibility but it is a trade- off that will pay higher dividends in future.

The research presented in this thesis is an attempt towards this goal. The vision is to make sensor networks more adaptable and flexible to be able to work with diverse applications in heterogeneous environments. Sensors or networks can communicate to other sensors when they come in contact and exchange information and services with out worrying about the underlying hardware and software. The middleware framework MidWSeN is an attempt to realize this vision.

References

Abdelzaher, T., B. Blum, Q. Cao, D. Evans, J. George, S. George, T. He, L. Luo, S. Son, R. Stoleru, J. Stankovic and A. Wood (2004). EnviroTrack: Towards an Environmental Computing Paradigm for Distributed Sensor Networks. 24th International Conference on Distributed Computing Systems, Tokyo, Japan, 582-589.

Abowd, G. D., C. G. Atkeson, S. L. Jason Hong, R. Kooper and M. Pinkerton (1997). "Cyberguide: A mobile context-aware tour guide." ACM Wireless Networks 3(5): 421-433.

Abowd, G. D., A. K. Dey, N. Davies, M. Smith, Peter J. Brown and P. Steggles (1999). Towards a Better Understanding of Context and Context-Awareness. Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing, Karlsruhe, Germany, Springer-Verlag, 304-307.

Agapioy, G., D. Vali, Charalambakis, P. Georgiadis, E. Plakidis, P. Rprris, K. Ioannou and A.Garmpis (2006). "Experimental performance evaluation of the emerging WiMAX technnology." WSEAS Transactions on information science and applictions 3(2): 322-327.

Akkaya, K. and M. Younis (2005). "A Survey on Routing Protocols for wireless Sensor Networks." Elsevier Ad Hoc Network Journal 3(3): 325-349.

Akyildiz, I. F., W. Su, Y. Sankarasubramaniam and E. Cayirci (2002). "Wireless sensor networks: a survey." Computer Networks **38**: 393-422.

Al-Karaki, J. N. and A. E. Kamal (2004). "Routing techniques in wireless sensor networks: A survey." IEEE Wireless Communications **11**(6): 6-28.

Baldauf, M., S. Dustdar and F. Rosen-berg (2007). "A Survey on Context-Aware Systems." International Journal of Ad Hoc and Ubiquitous Computing, 2(4): 263-277.

Bhatti, S., J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson and R. Han (2005). "MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms." ACM/Kluwer Mobile Networks & Applications , Special Issue on Wireless Sensor Networks **10**(4): 563-579.

Bluetooth. (2007). "Bluetooth." Retrieved 4 Oct, 2007, from http://www.bluetooth.com/bluetooth/.

Bonnet, P., J. E. Gehrke and P. Seshadri (2000). "Querying the Physical World." IEEE Personal Communications 7(5): 10-15.

Boulis, A., C. C. Han and M. B. Srivastava (2003). Design and Implementation of a Framework for Programmable and Efficient Sensor Networks. MobiSys 2003, San Franscisco, USA, 187-200.

Boulis, A., C.-C. Han, R. Shea and M. B. Srivastava (2007). "SensorWare: Programming sensor networks beyond code update and querying." Pervasive and Mobile Computing 3(4): 386–412.

Burkhardt, J., T. Schaeck, H. Henn, S. Hepper and K. Rindtorff (2002). Pervasive Computing: Technology and Architecture of Mobile Internet Applications, Pearson Education.ISBN: 0201722151

C˘apkun, S. and J.-P. Hubaux (2006). "Secure Positioning in Wireless Networks." IEEE Journal on SELECTED AREAS IN COMMUNICATIONS **24**(2).

Carovillano, B. (2003). "Radio Enthusiasts Celebrate Marconi's 1903 Breakthrough." VARBusiness 2003 issue of VARBusiness. Retrieved Jan 18, 2007, from http://www.crn.com/it-channel/18822423.

Cerpa, A., J. Elson, D. Estrin, L. Girod, M. Hamilton and J. Zhao (2001). Habitat Monitoring: Application Driver for Wireless Communications Technology. ACM SIGCOMM Workshop on Data Communications, Costa Rica, Latin America, 20-41.

Choi, J., D. Shin and D. Shin (2005). "Research and Implementation of the Context-Aware Middleware for Controlling Home Appliances." IEEE Transactions on Consumer Electronics **51**(1): 301-306.

Chong, C.-Y. and S. P. Kumar (2003). "Sensor Networks: Evolution, Opportunities, and Challenges." Proceedings of the IEEE **91**(8): 1247- 1256.

Cisco Systems, I. (2004). Introduction to LAN Protocols. Internetworking Technologies Handbook. Cisco. Indianapolis, Cisco Press. : 35-42

Conard, J., P. Dengler, B. Francis, J. Glynn, B. Harvey, B. Hollis, R. Ramachandran, J. Schenken, S. Short and C. Ullman (2001). Introducing .NET, Wrox Press;.SBN-10: 1861004893

ISBN-13: 978-1861004895

Copidate Technical Publicity. (2006). "Sensorland.Com." Retrieved December, 2006, from http://www.sensorland.com/HowPage059.html.

Costa, P., L. Mottola, A. L. Murphy and G. P. Picco (2006). TeenyLime: transiently shared tuple space middleware for wireless sensor networks. Proceedings of the internatiional workshop on Middleware for sensor networks, Melbourne, Australia, ACM, 43-48.

Coulson, G. (2004). What is Reflective Middleware. IEEE Distributed Systems Online. Coulson. **5**.

Crossbow Technology Inc. (2007). "Crossbow Wireless Sensor Networks ", 2007, from http://www.xbow.com/Home/HomePage.aspx.

Culler, D. E. and H. Mulder (2004) "Smart Sensors to Network the World." Scientific American Volume, **8** DOI:

Curino, C., M. Giani, M.Giorgetta, A.Giusti, A. L. Murphy and G. P. Picco (2005). TinyLIME: Bridging mobile and sensor networks through middlewares. 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom) IEEE Computer Society, 61-72.

DARPA. (2007). "DARPA." Retrieved 24 July, 2007, from http://www.darpa.mil/.

Dasgupta, K., K. Kalpakis and P. Namjoshi (2003). An efficient clustering-based heuristic for data gathering and aggregation in sensor networks. Wireless Communications and Networking, 2003. WCNC 2003., New Orleans, Louisians, USA, IEEE, !948-1953.

De, P., Y. Liu and S. Das (2006). Modeling node compromise spreading in wireless sensor networks using epidemic theor. IEEE International Symposium on on World of Wireless, Mobile and Multimedia Niagara-Falls, NY, IEEE Computer Society, 237-243

Demirkol, I., C. Ersoy and F. Alagöz (2006). Protocols for Wireless SensorNetworks: A Survey. IEEE Communications Magazine.

Du, W., J. Deng, Y. S. Han, S. Chen and P. K. Varshney (2004). A Key Management Scheme for Wireless Sensor Networks Using Deployment Knowledge. The 23rd Conference of the IEEE Communications Society, Hong Kong, IEEE Comunication Society, 597-607.

Dunkels, A., Thiemo Voigt, N. Bergman and M. Jönsson (2004). The Design and Implementation of an IP-based Sensor Network for Intrusion Monitoring. 2nd Swedish National Computer Networking Workshop, Karlstad, Sweden,

Dunkels, A., Björn Grönvall and T. Voigt. (2004a). Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. the First IEEE Workshop on Embedded Networked Sensors 2004 Tampa, Florida, USA,

Dunkels, A., Thiemo Voigt, N. Bergman and M. Jönsson (2004b). The Design and Implementation of an IP-based Sensor Network for Intrusion Monitoring. 2nd Swedish National Computer Networking Workshop, Karlstad, Sweden,

Eady, F. (2007). Hands-On ZigBee Implementing 802.15.4 with Microcontrollers NEWNES.13: 9780123708878

Estrin, D., R. Govindan, J. Heidemann and S. Kumar (1999). Next Century Challenges: Scalable Coordination in Sensor Networks. MOBICOM, Seattle, ACM press, 263-270.

Estrin, D., D. Culler, K. Pister and G. Sukhatme (2002). "Connecting the physical world with pervasive networks." Pervasive Computing, IEEE 1(1): 59- 69.

Eswaran, A., A. Rowe and R. Rajkumar (2005). Nano-RK: An Energy-Aware Resource-Centric Operating System for Sensor Networks. 26th IEEE Real-Time Systems Symposium, Miami, Florida, USA 256-265.

Fahy, P. and S. Clarke (2004). CASS - Middleware for Mobile Context-Aware Applications. The Second International conference on Mobile Systems,Applications and Services, Boston, Massachusetts, USA, ACM Press New York, NY, USA,

Fasolo, E., M. Rossi, J. Widmer and M. Zorzi (2007). "In-network aggregation techniques for wireless sensor networks: a survey." Wireless Communications, IEEE, IEEE Personal Communications 14(2): 70--87.

Feng, J., F. Koushanfar and M. Potkonjak. (2002). System-Architecture for Sensor Networks Issues, Alternatives, and Directions. IEEE International Conference on Computer Design, Freiburg, Germany, 226-231.

Ferrari, G. (2006). Ad Hoc Wireless Networks A Communication-Theoreteic Perspective John Wiley & Sons Inc.047009110x

Ferro, E. and F. Potorti (2005). "Bluetooth and Wi-Fi wireless protocols: a survey and a comparison " Wireless Communications 12(1): 12-26.

Fok, C.-L., G.-C. Roman and C. Lu (2005). Mobile Agent Middleware for Sensor Networks: An Application Case Study In Proceedings of the 4th International Conference on Information Processing in Sensor Networks, Los Angeles, California, 382-387.

Frodigh, M., S. Parkvall, C. Roobol, P. Johansson and P. Larsson (2001). "Future Generation Wireless Networks." IEEE Personal Communications 8(5): 10-17.

Gay, D., P. Levis, R. v. Behren, M. Welsh, E. Brewer and D. Culler. (2003). The nesCLanguage: A Holistic approach to Networked Embedded Systems. Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation California, USA, ACM press, 1 - 11

Gelenter, G. (1985). "Generative communication in Linda." ACM Computing Surveys 7 (1): 80-112.

Gellersen, H.-W., A. Schmidt and M. Beigl (2002). "Multi-Sensor Context-Awareness in Mobile Devices and Smart Artefacts." ACM journal Mobile Networks and Applications 7(5): 341-351.

Giordano, V., P. Ballal, F. Lewis, B. Turchiano and J. B. Zhang (2006). "Supervisory Control of Mobile Sensor Networks: Math Formulation, Simulation, and Implementation." IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS, **36**(4): 806-819.

Grandinetti, L., Ed. (2005). Grid Computing: The New Frontier of High Performance Computing. Avances in Parallel Computing Amsterdam, The Netherlands, ELSEVIER.

Griffiths, R. T. (2002). History of the Internet. Griffiths, Leiden University.

Griffiths, R. T. (2002). "History of the Internet." Retrieved 25 July, 2007, from http://www.let.leidenuniv.nl/history/ivh/frame_theorie.html.

Grosso, W. (2001). Java RMI, O'Rielly. 1-56592-452-5

GSM. (2007). "GSM World." Retrieved 25 July, 2007, from http://www.gsmworld.com.

Gu, T., H. K. Pun and D. Q. Zhang (2004). A Middleware for building context-aware mobile services. Vehicular Technology Conference, 2004., Milan, Italy, IEEE, 2656 - 2660.

Gummadi, R., X. Li, R. Govindan, C. Shahabi and W. Hong (2005). Energy-efficient Data Organization and Query Processing in Sensor Networks. 21st International Conference on Data Engineering, Tokyo, Japan, 157-158.

Hadim, S. and N. Mohamed (2006). "Middleware Challenges and Approaches for Wireless Sensor Networks." IEEE DS Online Middleware 7(3): 1-23.

Hadim, S. and N. Mohamed (2006). Middleware for Wireless Sensor Networks: A Survey. First International Conference on Communication System Software and Middleware, New Dehli, India, 1- 7.

Han, C.-C., R. K. Rengaswamy, R. Shea, E. Kohler and M. Srivastava (2005). SOS: A dynamic operating system for sensor networks. Proceedings of the Third International Conference on Mobile Systems, Applications, And Services, Seattle, Washington ACM Press New York, NY, USA 163-176.

Hauben, M. (2006). History of ARPANET Hauben. **2007**.

Heidemann, J., F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin and D. Ganesan (2001). Building Efficient Wireless Sensor Networks with Low-Level Naming. Symposium on Operating Systems Principles, Alberta, Canada, ACM, 146--159.

Heinzelman, W., A. Chandrakasan and H. Balakrishnan (2002). "An Application-Specific Protocol Architecture for Wireless Microsensor Networks." IEEE Transactions on Wireless Communications **1**(4): 660-670.

Heinzelman, W. B., A. L. Murphy, Hervaldo S. Carvalho and M. A. Perillo (2004). Middleware to Support Sensor Network Applications. IEEE Network Magazine Heinzelman. **18**: 6-14.

Heinzelman, W. R., A. Chandrakasan and H. Balakrishnan (2000). Energy-Efficient Communication Protocol for Wireless Microsensor Networks. 33rd Hawaii international Conference on System Sciences, Maui, Hawaii IEEE Computer Society, Washington, DC, . 8020.

Hewlett-Packard. (2007). "Hewlett-Packard." 2006, from http://www.hpl.hp.com/news/2006/jul-sept/memoryspot.html?jumpid=reg_R1002_USEN.

Horovitz, A. and J. Liberty (2007). Programming .NET 3.0: Rough Cuts Version, O'Reilly.10: 0-596-51039-X

Howit, I. and J. Gutierrez (2003). "IEEE 802.15.4 Low Rate-Wireless Personal Area Network Coexistence Issues " Wireless Communications and Networking 3: 1481-1486.

Hu, L. and D. Evans (2004). Localization for Mobile Sensor Networks. Tenth Annual International Conference on Mobile Computing and Networking (, Philadelphia, PA, USA, ACM Press, 45-57.

Huebscher, M. C. and J. A. McCann (2004). Adaptive middleware for Context Aware Applications in Smarthomes. 2nd Workshop on Middleware for Pervasive and AdHoc, Toronto, Canada, ACM, 111-116.

Hwang, J., D. H. C. Du and E. Kusmierek (2004). Energy Efficient Organization of Mobile Sensor Networks. 2004 International Conference on Parallel Processing Workshops. , Montreal, Canada, 84- 91.

Intanagonwiwat, C., R. Govindan, D. Estrin, J. Heidemann and F. Silva (2003). "Directed diffusion for wireless sensor networking." IEEE/ACM Transactions on Networking (TON) 11(1): 2-16.

Intel. (2007). "Intel Mote 2." Retrieved Sept., 2007, from http://www.intel.com/research/downloads/imote_overview.pdf.

Janakiram, D., A. V. U. Phanikumar and A. M. Reddy-V (2005). Distributed Collaboration for Event Detection in Wireless Sensor Networks. the Proceedings of 3rd International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC05), Grenoble, France.,

Javed, H., M. Merabti and B. Askwith (2005). Persistent Storage service in Self Organizing Wireless Sensor Networks. The 6th Annual PostGraduate Symposium on the Convergence of Telecommunications, Networking & Broadcasting, Liverpool, UK, 441-446.

Javed, H., M. Merabti and R. Askwith (2007a). Aggregation Service in Middlware for Wireless Sensor Network. The convergence of Telecommunications, Networking & Broadcasting, Liverpool, UK, 21-27.

Javed, H., M. Merabti and R. J. Askwith (2007b). A Persistent Storage Middleware Service For Wireless Sensor Network. IEEE Wireless Communications and Networking Conference, Hong Kong, 2490-2494.

Jiao, B., S. H. Son and J. A. Stankovic (2005). GEM: Generic Event Service Middleware for Wireless Sensor Networks. Second International Workshop on Networked Sensing Systems, San Diego, California, USA,

Kang, H., X. Li and P. J. Moran (2006). Autonomic Sensor Networks: A New Paradigm for Collaborative Information Processing. 2nd IEEE International Symposium on

Dependable, Autonomic and Secure Indiana, USA, IEEE Computer Society, 258 - 268

Karlof, C. and D. Wagner (2003). "Secure routing in wireless sensor networks: Attacks and countermeasures." Elsevier Ad Hoc Network Journal 1: 293-315.

Karp, B. and H. T. Kung (2000). Greedy Perimeter Stateless Routing for Wireless Sensor Networks. Sixth Annual ACM/IEEE International conference on Mobile Coputing and Networking, Boston, MA, USA, 243-254.

Kifayat, K., M. Merabti, Q. Shi and D. Llewellyn-Jones (2007). "Application Independent Dynamic Group-Based Key Establishment for Large-scale Wireless Sensor Networks." China Communications 4(1): 14-17.

Kon, F., M. Rom'an, P. Liu, J. Mao, T. Yamane, L. C. Magalh˜aes and R. H. Campbell (2000). Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB. In Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing New York, USA, Springer-Verlag, 121–143.

Kon, F., Fábio Costa, Roy Campbell and G. Blair. (2002). "The Case for Reflective Middleware." Communications of the ACM 45(6): 33-38.

Korpipää, P., J. Mäntyjärvi, J. Kela, H. Keränen and E.-J. Malm (2003). "Managing Context Information in Mobile Devices." IEEE PERVASIVEcomputing 2(3): 42-51.

Koutsonikolas, D., S. Das, Y. C. Hu and I. Stojmenovic (2007). Hierarchical Geographic Multicast Routing for Wireless Sensor Networks. International Conference on Sensor Technologies and Applications Valencia, Spain, IEEE Computer Society Press, 347-354.

Kumar, A. V. U. P., A. M. R. V and D. Janakiram (2005). Distributed collaboration for event detection in wireless sensor networks. Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing, Grenoble, France, ACM Press, New York, NY, 1-8.

Lazos, L., S. Capkun and R. Poovendran (2005). ROPE: Robust Position Estimation in Wireless Sensor Networks. Fourth International Symposium on Information Processing in Sensor Networks California, USA, IEEE, 324 - 331.

Lee, W. L., A. Datta and R. Cardell-Oliver (2006). Network management in wireless sensor networks. Handbook of Mobile Ad Hoc and Pervasive Communications. Lee, American Scientific Publishers, USA.. 1

Levis, P. and D. Culler. (2002). Mate: A tiny Virtual Machine for Sensor Networks. Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, USA, 85-95.

Levis, P. (2006). "TinyOS Programming" Retrieved August, 2007.

Li, S., S. H. Son and J. A. Stankovic. (2004). "Event Detection Services Using Data Service Middleware in Distributed Sensor Networks." Telecommunication Systems 26(2-4): 351-368.

Li, X. and N. Santoro (2006). An Integrated Self-deployment and Coverage Maintenance Scheme for Mobile Sensor Networks. Mobile Ad-hoc and Sensor Networks, Hong Kong, China, Springer, 847-860.

Light, J. and B. Arunachalan (2006). Mobile middleware service architecture for EMS application. First International Conference on Communication System Software and Middleware, 1 - 5

Lim, A. (2001). "Distributed services for information dissemination in self-organizing sensor networks." Science Direct Journal of the Franklin Institute **338**(6): 707-727.

Lindsey, S. and C. Raghavendra (2003). "PEGASIS: Power Efficient Gathering in Sensor Information Systems." ACM SIGMOD Special section on sensor network technology and sensor data management **32**(4): 66-71.

Liu, D., P. Ning, S. Zhu and S. Jajodia (2005). Practical Broadcast Authentication in Sensor Networks. 2nd Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services San Diego, CA, USA, 118--129.

Liu, T. and M. Martonosi (2003). Impala: a middleware system for managing autonomic, parallel sensor systems ninth ACM SIGPLAN symposium on Principles and practice of parallel programing, San Diego, California, USA ACM Press, 107-118

Lorincz, K., D. Malan, T. R. F. Fulford-Jones, Alan Nawoj, Antony Clavel, Victor Shnayder, Geoff Mainland, S. Moulton and M. Welsh. (2004). "Sensor Networks for Emergency Response: Challenges and Opportunities." In IEEE Pervasive Computing, Special Issue on Pervasive Computing for First Response **3**(4): 16-23.

Madden, S., M. J. Franklin, J. M. Hellerstein and W. Hong. (2005). "TinyDB: an Acquisitional Query Processing System for Sensor Networks." ACM Transactions on Database Systems **Vol. 30**(No. 1): 122-173.

Madden, S. R., M. J. Franklin, J. M. Hellerstein and W. Hong. (2002). TAG: a Tiny Aggregation Service for Ad-Hoc Sensor Networks. 5th Annual Symposium on Operating Systems Design and Implementation, Boston, USA, *131-146*,.

Mainwaring, A., J. Polastre, R. Szewczyk, D. Culler and J. Anderson (2002). Wireless Sensor Networks for Habitat Monitoring. First ACM International Workshop on Wireless Sensor Networks and Applications, Atlanta, Georgia, USA, ACM, 88-97.

Malan, D., T. Fulford-Jones, M. Welsh and S. Moulton (2004). CodeBlue: An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care. International Workshop on Wearable and Implantable Body Sensor Networks, Imperial College London, United Kingdom,

Manjeshwar, A. and D. P. Agrawal (2001). TEEN : A Protocol for Enhanced Efficiency in Wireless Sensor Networks. 1st International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing, ,, San Francisco, CA, IEEE Computer Society, 30189a.

Marr6n, P. J., A. Lachenmann, D. Minder, M. Gauger, O. Saukh and K. Rotherme (2005). "Management and configuration issues for sensor networks." International Journal of Network Management(15): 235–253.

Mathur, G., P. Desnoyers, D. Ganesan and P. Shenoy (2006). UltraLow Power Data Storage for Sensor Networks. fifth international conference on Information processing in sensor networks, Nashville, Tennessee, USA., ACM Press, New York, NY, 374 - 381

Meguerdichian, S., S. Slijepcevic, V. Karayan and M. Potkonjak (2001). Localized Algorithms In Wireless Ad-Hoc Networks: Location Discovery And Sensor

Exposure. The ACM Symposium on Mobile Ad Hoc Networking & Computing, Long Beach, California, USA, ACM, 106-116.

Microsoft. (2007). ".NET Framework Developer Center ", 2007, from http://msdn2.microsoft.com/en-gb/netframework/aa663309.aspx.

Nachman, L., R. Kling, R. Adler, J. Huang and V. Hummel (2005). The Intel Mote platform: a bluetooth-based sensor network for industrial monitoring. . Fourth International Symposium on Information Processing in Sensor Networks UCLA, Los Angeles, California, USA, 437-442.

Naguib, H., G. Coulouris and S. Mitchell (2001). Middleware support for context-aware multimedia applications. Third International Working Conference on New Developments in Distributed Applications and Interoperable System, Krakow, Poland, Kluwer, B.V. Deventer, The Netherlands, The Netherlands, 9-22.

Naur, P., B. Randell and R. M. McClure (1968). Software Engineering. NATO SOFTWARE ENGINEERING CONFERENCE 1968, Garmisch, Germany,, 14.

Ni, L. M., Y. Zhu, J. Ma, M. Li, Q. Luo, Y. Liu, S. C. Cheung and Q. Yang (2005). Semantic Sensor Net: An Extensible Framework. 3rd International Conference onNetworking and Mobile Computing, Zhangjiajie, China, Springer 1144-1153

Nicopolitidis, P., M. S. Obaidat, G. I. Papadimtriou and A. S. Pomportsis (2003). Wireless Networks. West Sussex, John Wiely & Sons Ltd.470-84529-5

Oram, A. (2001). Peer-to-Peer: Harnessing the Power of Disruptive Technologies. Sebastopol, O'Reilly & Associates, Inc.ISBN 0-596-00110-X

Ota, N. and W. T. C. Kramer. (2003). "TinyML: Metadata for Wireless Networks." Retrieved 2006.

Ou, S., K. Yang and J. Zhang (2007). "An effective offloading middleware for pervasive services on mobile devices." Pervasive and Mobile Computing 3(4): 362–385.

Perrig, A., D. Wagner and J. Stankovic (2004). "Security in Wireless Sensor Networks." Communications of the ACM 47 (6): 53-57

Pietzuch, P. R. and J. Bacon (2002). Hermes: A Distributed Event-Based Middleware Architecture. 22nd International Conference on Distributed Computing Systems, Vienna, Austria, IEEE Computer Society 611 - 618

Pietzuch, P. R. and S. Bhola (2003). Congestion Control in a Reliable Scalable Message-Oriented Middleware. 4th International Conference on Middleware, Rio de Janeiro, Brazil, Springer, 202-221.

Qi, H., P. T. Kuruganti and Y. Xu (2002). "The Development of Localized Algorithms in Wireless Sensor Networks " Sensors 2: 286-293.

Ratnasamy, S., P. Francis, M. Handley, R. Karp and S. Schenker (2001). A scalable content-addressable network. ACM SIGCOMM, San Diego, USA, 161-172.

Ratnasamy, S., D. Estrin, R. Govindan, B. Karp, S. Shenker, L. Yin and F. Yu (2002). GHT: a geographic hash table for data-centric storage. Wireless Sensor Networks and Applications, Atlanta GA, ACM, 78-87.

Ratnasamy, S., Brad Karp, Scott Shenker, Deborah Estrin, Ramesh Govindan, Li Yin and F. Yu (2003). "Data-Centric Storage in Sensornets with GHT, a Geographic Hash Table." Mobile Networks and Applications 8(Special Issue on Algorithmic Solutions for Wireless, Mobile, Ad Hoc and Sensor Networks): 427-442.

Riley, G. F. (2003). The Georgia Tech Network Simulator. Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research Karlsruhe, Germany, ACM Press New York, NY, USA 5 - 12

Rohatgi, V. K. and A. k. M. E. Saleh (2001). An Introduction to Prbability And Statistics. New York, John Wiley & Sons, Inc.0-471-34846-5

Romer, K., F. Mattern, T. Dubendorfer and T. Schoch (2003). Smart Identification Frameworks for Ubiquitous Computing Applications. IEEE International Conference on Pervasive Computing and Communications, Forth Worth/Texas, 253-262.

Römer, K., O. Kasten and F. Mattern (2002). "Middleware Challenges for Wireless Sensor Networks." ACM Mobile Computing and Communication Review **6**(4): 59-61.

Rosenstein, A., J. Li and S. Y. Tong. (1997). "MASH: The Multicasting MASH: The Multicasting." ACM Computer Communication Review **27**(3): 5-13.

Rudafshani, M. and S. Datta (2007). Localization in Wireless Sensor Networks. 6th International Conference on Information Processing Sensor Networks, Massachusetts, USA, ACM, 51-60.

Sanchez, J. A., P. M. Ruiz and I. Stojmnenovic (2006). GMR: Geographic Multicast Routing for Wireless Sensor Networks. 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks, Reston, VA, USA, IEEE Communication Society, 20 - 29

Sanchez, J. A., P. M. Ruiz and I. Stojmenovic (2007). "Energy-efficient geographic multicast routing for Sensor and Actuator Networks " Sensor Networks Computer Communications **30**(13): 2519-2531.

Sang, Y. and H. Shen (2005). A scheme for testing privacy state in pervasive sensor networks. 19th International Conference on Advanced Information Networking and Applications, Tamkang University, Taiwan, IEEE.org, 644 - 648.

Satyanarayanan, M. and D. Narayanan (1999). Multi-fidelity algorithms for interactive mobile applications. 3rd international workshop on Discrete algorithms and methods for mobile computing and communications, Seattle, Washington, United States, ACM Press, 1-6.

Shen, C.-C., C. Srisathapornphat and C. Jaikaeo (2001). "Sensor information networking architecture and applications." IEEE Personal Communications **8**(4): 52-59.

Sheth, A., B. Shucker and R. Han (2003). VLM2: A Very Lightweight Mobile Multicast System For Wireless Sensor Networks. IEEE Wireless Communications and Networking Conference (WCNC), New Orleans, Louisiana, USA, 1936-1941.

Sikka, P., P. Corke, P. Valencia, C. Crossman, D. Swain and G. Bishop-Hurley (2006). Wireless adhoc sensor and actuator networks on the farm Proceedings of the fifth international conference on Information processing in sensor networks, Nashville, Tennessee, USA ACM Press, 492-499

Souto, E., G. Guimaraes, C. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz and J. Kelner (2006). "Mires: a publish/subscribe middleware for sensor networks." Personal and Ubiquitous Computing **10**(1): 37-44.

Stoica, I., R. Morris, D. Karger, M. F. Kaashoek and H. Balakrishnan (2001). Chord: A Scalable Peertopeer Lookup Service for Internet Applications, California, USA, ACM, Subramanian, R. and B. D. Goodman, Eds. (2005). Peer to Peer Computing: The Evolution of a Disruptive Technology Hershey, Pennsylvania, IGI Global.

Sun Microsystems. (1994). "Jini Network Technology." Retrieved 6th March, 2006, from http://www.sun.com/software/jini/.

Sun Microsystems. (2005). "Jxme." Retrieved 06 March, 2006, from http://jxme.jxta.org/.

Sylvia Ratnasamy, Brad Karp, Scott Shenker, Deborah Estrin, Ramesh Govindan, Li Yin and F. Yu (2003). "Data-Centric Storage in Sensornets with GHT, a Geographic Hash Table." Mobile Networks and Applications **8**(Special Issue on Algorithmic Solutions for Wireless, Mobile, Ad Hoc and Sensor Networks): 427-442.

Tavakoli, A., P. Dutta, J. Jeong, S. Kim, J. Ortiz, D. Culler, P. Levis and S. Shenker (2007). "A modular sensornet architecture: past, present, and future directions." SIGBED **4**(3): 49-54.

Thaddeus, R. F., Fulford-Jones, G.-Y. Wei and M. Welsh (2004). A Portable, Low-Power, Wireless Two-Lead EKG System. In Proceedings of the 26th IEEE EMBS Annual International Conference, San Francisco,

TinyOS. (2007). "TinyOS." Retrieved Aug, 2005, from http://www.tinyos.net/.

Tsuchiya, P. F. (1988). "The Landmark hierarchy: A new hierarchy for routing in very large networks." ACM Computer Communications Review **18**(4): 35-42.

Tubaishat and S. Madria (2003). "Sensor networks: an overview." IEEE Potentials **22**(2): 20- 23.

Varshney, U. and R. Vetter (2000). "Emerging Mobile and Wireless networks." Communications of the ACM **43**(6): 73-81.

VAST. (2007). "SensorML." Retrieved 10 Aug, 2006, from http://vast.nsstc.uah.edu/SensorML/.

Ville, L. S. and P. Dickman (2003). Garnet: A Middleware Architecture for Distributing Data Streams Originating in Wireless Sensor Networks. 23rd IEEE International Conference on Distributed Computer Systems Providence, Rode Island, USA, IEEE, 235-240.

Vinoski, S. (1997). CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments. IEEE Communications Magazine. Vinoski. **35**: 46-55.

Wang, H., G. Pottie, K. Yao and D. Estrin (2004). Entropybased sensor selection heuristic for target localization Information Processing and Sensor Networks Third International Symposium on Information Processing in Sensor Networks Berkeley, California, ACM, 36-45.

Wang, N., D. C. Schmidt and C. O'Ryan (2001). Overview of the CORBA component model. Component-based software engineering: putting the pieces together. Wang. Boston, MA, Addison-Wesley Longman Publishing Co., Inc.: 557-571

Werner-Allen, G., Konrad Lorincz, Matt Welsh, Omar Marcillo, Jeff Johnson, Mario Ruiz and Jonathan Lees (2006). "Deploying a Wireless Sensor Network on an Active Volcano." IEEE Internet Computing **10**(2): 18-25.

Wilson, B. J. (2002). JXTA. Indianapolis Indiana, New Riders.0-73571-234-4

Woo, A., S. Madden and R. Govindan (2004). "Networking Support for Query Processing in Sensor Networks." Communications of the ACM **47**(6): 47-52.

Wu, S. and K. S. Candan (2006). Multicasting GMP: Distributed Geographic Multicast Routing in Wireless Sensor Networks. 26th IEEE International Conference on Distributed Computing Systems, Lisboa, Potugal, IEEE Computer Society Press, 49 - 49

Yao, Y. and J. Gehrke (2003). Query Processing for Sensor Networks. First Biennial Conference on Innovative Data Systems Research, Asilomar, California,

Yu, M., H. Mokhtar and M.Merabti (2007). "A Self-Organized Middleware Architecture for Wireless Sensor Network Management." International Journal of Ad Hoc and Ubiquitous Computing(Wireless Sensor Networks,): in press.

Yu, Y., B. Krishnamachari and V. K. Prasanna (2004). "Issues in Designing Middleware for Wireless Sensor Networks." IEEE Network **18**(1): 15-21.

Zeidler, A. (2007). Event-based Middleware for Pervasive Computing- Foundations, Concepts, Design VDM Verlag Dr. Mueller e.K.10: 3836413094

Zhang, W. and G. Cao (2003). "Optimizing Tree Reconfiguration for Mobile Target Tracking in Sensor Networks." SIGMOBILE Mob. Comput. Commun. Rev. 7(3): 39-40.

Zhao, B. Y., L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph and J. Kubiatowicz (2004). "Tapestry: A Resilient Global-scale Overlay for Service Deployment " IEEE Journal on Selected Areas in Communications **22**(1): 41-53.

APPENDIX A

Use Case Diagrams

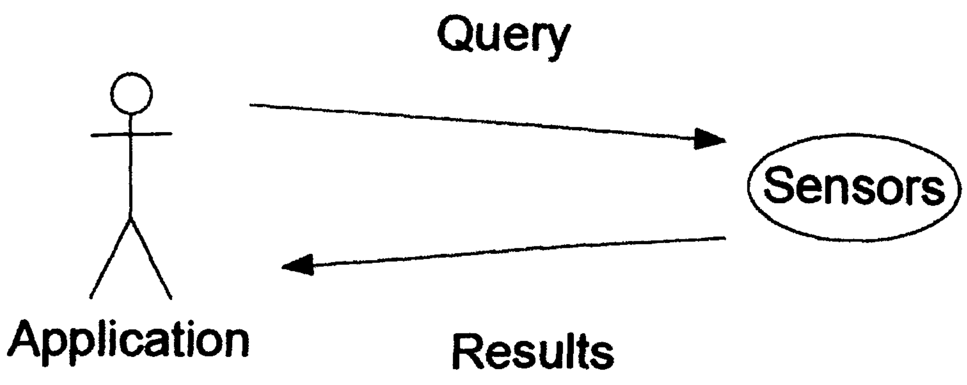


Diagram 1 Use case 1 application can query sensors

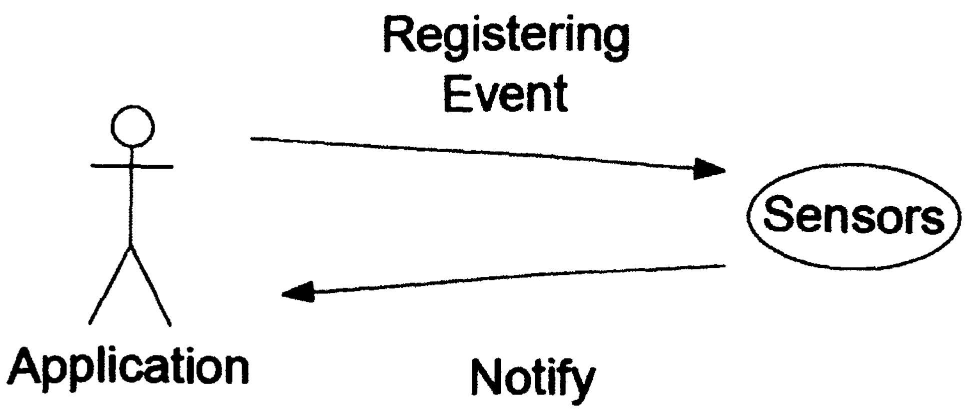
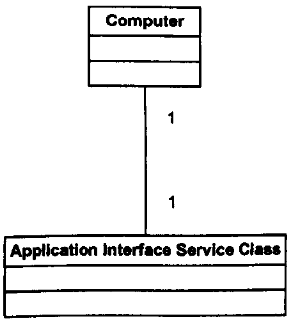


Diagram 2 application can register an event

Class Diagrams



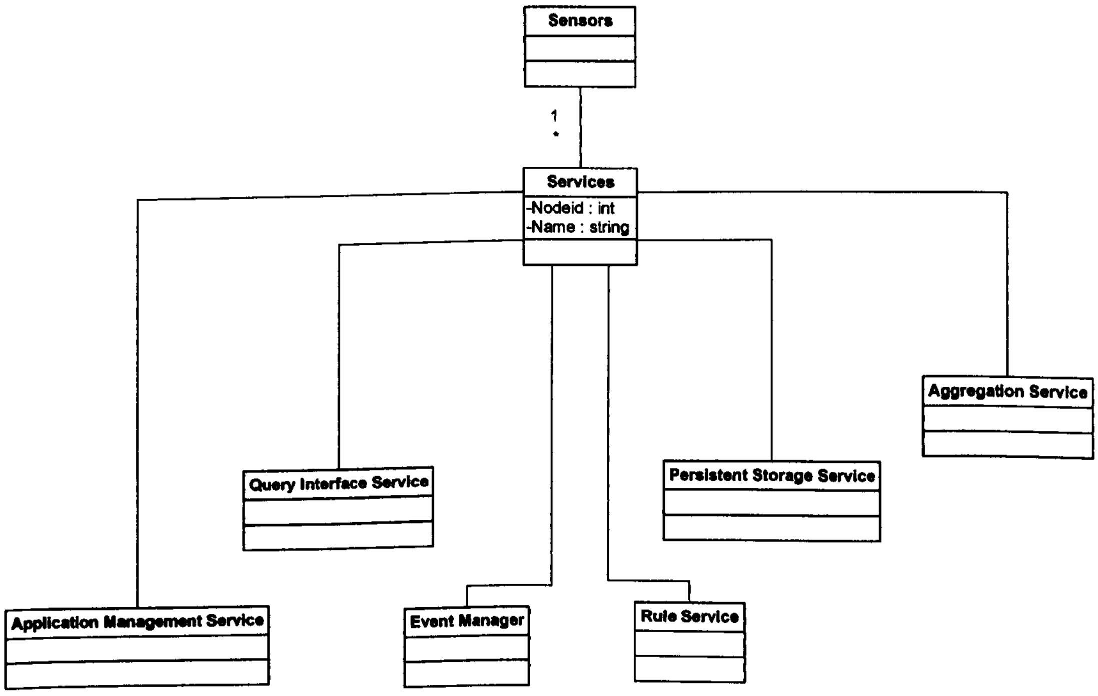


Diagram 3 shows relationship between services and devices

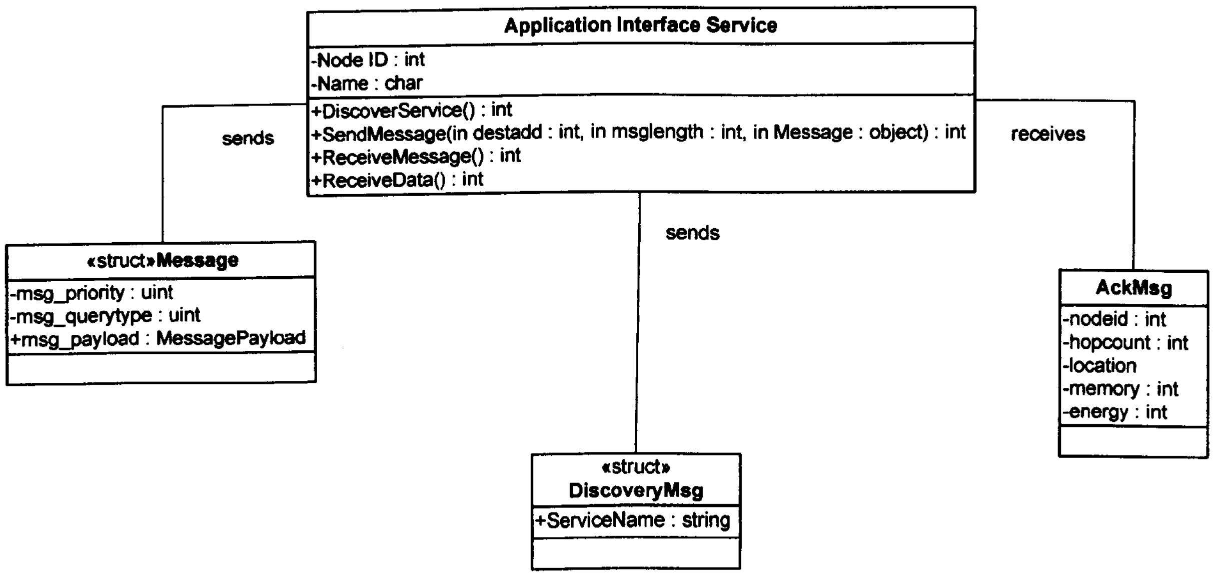


Diagram 4 shows the Application Interface Service

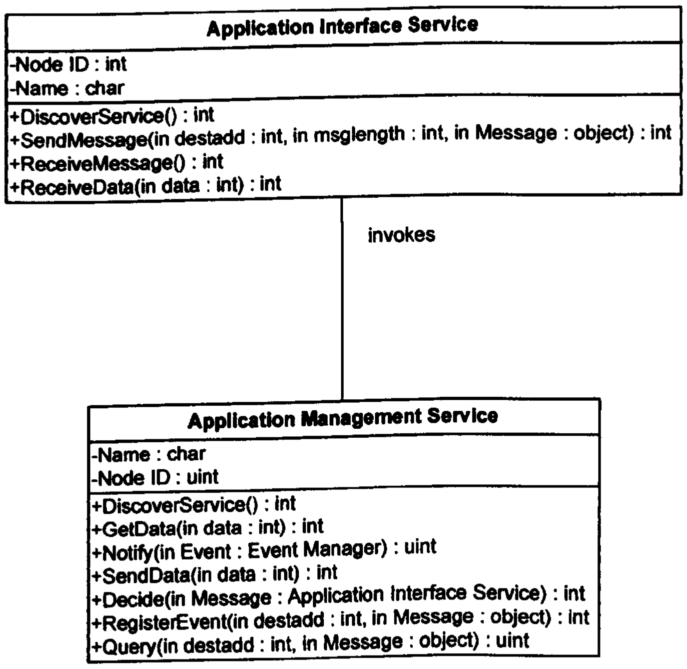

Application Interface Service
-Node ID : int
-Name : char
+DiscoverService() : int
+SendMessage(in destadd : int, in msglength : int, in Message : object) : int
+ReceiveMessage() : int
+ReceiveData(in data : int) : int
invokes
Application Management Service
-Name : char
-Node ID : uint
+DiscoverService() : int
+GetData(in data : int) : int
+Notify(in Event : Event Manager) : uint
+SendData(in data : int) : int
+Decide(in Message : Application Interface Service) : int
+RegisterEvent(in destadd : int, in Message : object) : int
+Query(in destadd : int, in Message : object) : uint

Diagram 5 shows the relationship between Application Interface Service and Application Management Service

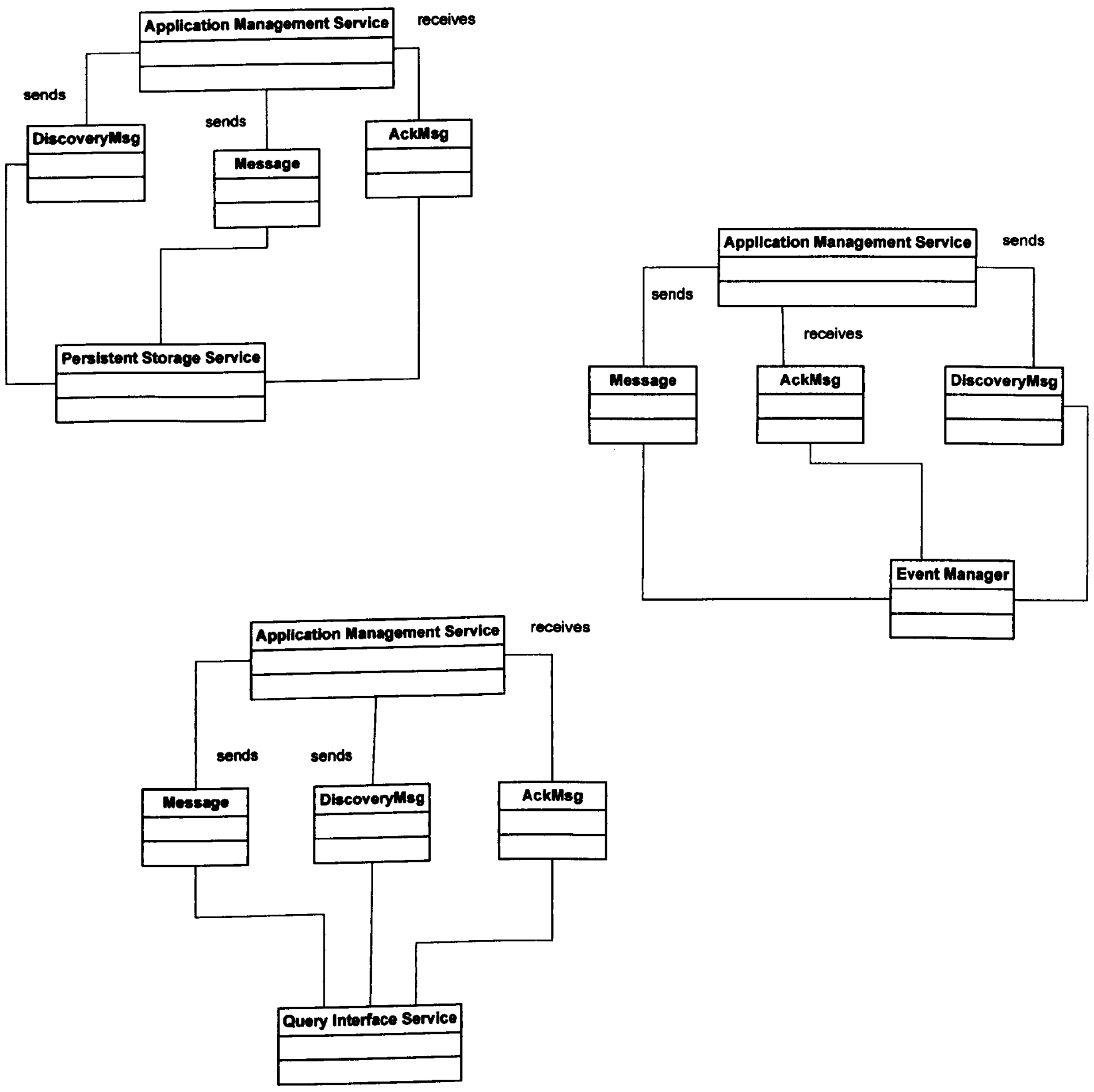


Diagram 6 shows relationship between the Application Management Service with Persistent Storage, Event Manager and Query Interface Services

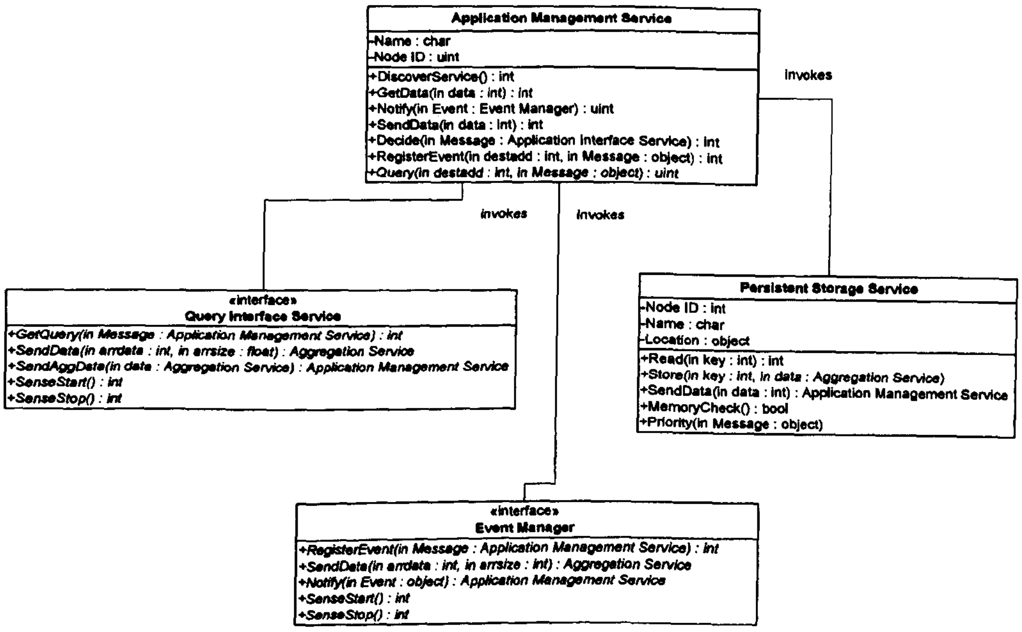

Application Management Service
-Name : char
-Node ID : uint
+DiscoverService() : int
+GetData(in data : int) : int
+Notify(in Event : Event Manager) : uint
+SendData(in data : int) : int
+Decide(in Message : Application Interface Service) : int
+RegisterEvent(in destadd : int, in Message : object) : int
+Query(in destadd : int, in Message : object) : uint
invokes
invokes
invokes
«interface»
Query Interface Service
+GetQuery(in Message : Application Management Service) : int
+SendData(in arrdata : int, in arrsize : float) : Aggregation Service
+SendAggData(in data : Aggregation Service) : Application Management Service
+SenseStart() : int
+SenseStop() : int
Persistent Storage Service
-Node ID : int
-Name : char
-Location : object
+Read(in key : int) : int
+Store(in key : int, in data : Aggregation Service)
+SendData(in data : int) : Application Management Service
+MemoryCheck() : bool
+Priority(in Message : object)
«interface»
Event Manager
+RegisterEvent(in Message : Application Management Service) : int
+SendData(in arrdata : int, in arrsize : int) : Aggregation Service
+Notify(in Event : object) : Application Management Service
+SenseStart() : int
+SenseStop() : int

Diagram 7 shows the Application Management Service, Persistent Storage, Event Manager and Query Interface Services

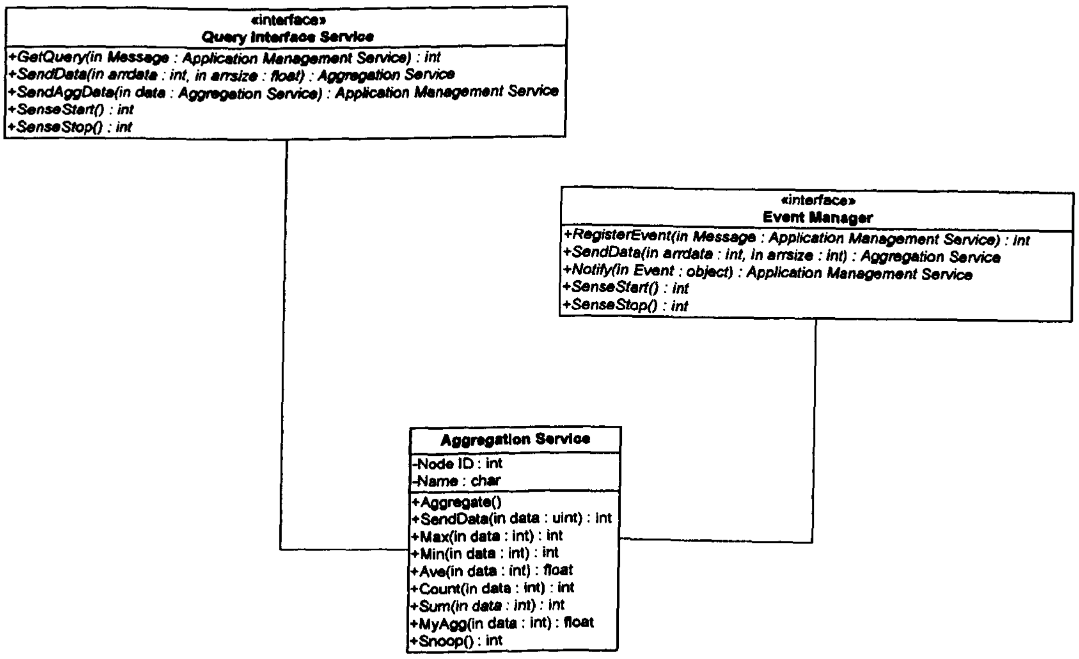

«interface»
Query Interface Service
+GetQuery(in Message : Application Management Service) : int
+SendData(in arrdata : int, in arrsize : float) : Aggregation Service
+SendAggData(in data : Aggregation Service) : Application Management Service
+SenseStart() : int
+SenseStop() : int
«interface»
Event Manager
+RegisterEvent(in Message : Application Management Service) : int
+SendData(in arrdata : int, in arrsize : int) : Aggregation Service
+Notify(in Event : object) : Application Management Service
+SenseStart() : int
+SenseStop() : int
Aggregation Service
-Node ID : int
-Name : char
+Aggregate()
+SendData(in data : uint) : int
+Max(in data : int) : int
+Min(in data : int) : int
+Ave(in data : int) : float
+Count(in data : int) : int
+Sum(in data : int) : int
+MyAgg(in data : int) : float
+Snoop() : int

Diagram 8 shows relationship between Query Interface, Event Manager and Aggregation Services

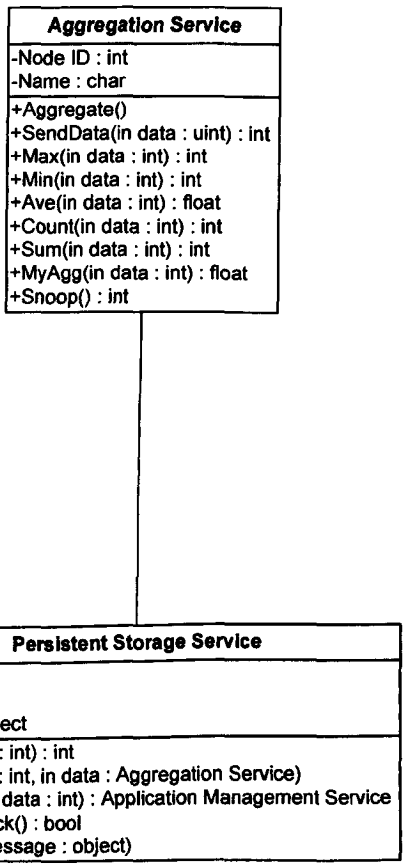

Aggregation Service
-Node ID : int
-Name : char
+Aggregate()
+SendData(in data : uint) : int
+Max(in data : int) : int
+Min(in data : int) : int
+Ave(in data : int) : float
+Count(in data : int) : int
+Sum(in data : int) : int
+MyAgg(in data : int) : float
+Snoop() : int
Persistent Storage Service
-Node ID : int
-Name : char
-Location : object
+Read(in key : int) : int
+Store(in key : int, in data : Aggregation Service)
+SendData(in data : int) : Application Management Service
+MemoryCheck() : bool
+Priority(in Message : object)

Diagram 9 shows relationship between Aggregation and Persistent Storage Service

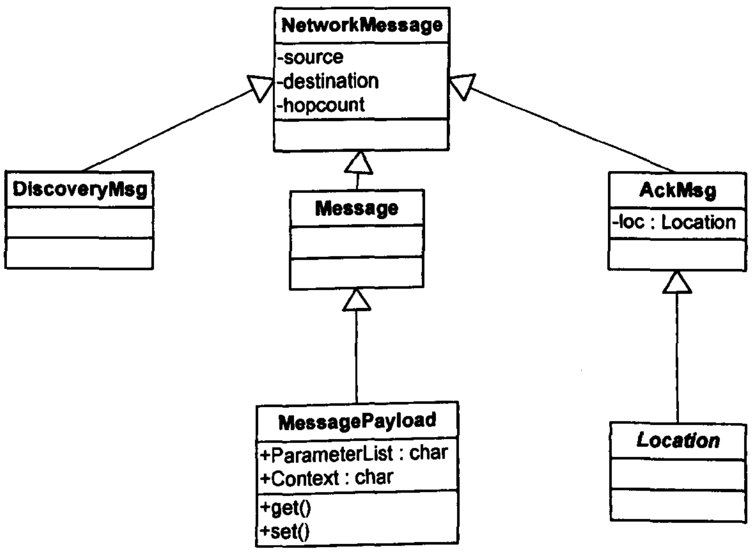


Diagram 10 shows generalisation between different Message classes