An agent based layered framework to facilitate intelligent Wireless Sensor Networks

A dissertation submitted to the Department of Computer Science
at the

University of Cape Town

In fulfilment of the requirements for the degree of

Master of Science in Computer Science

By

Andre Scholtz

Supervised by
Dr Hanh Le

May 26, 2011

Abstract

Wireless Sensor Networks (WSNs) are networks of small, typically low-cost hardware devices which are able to sense various physical phenomenon in their surrounding environments. These simple nodes are also able to perform basic processing and wirelessly communicate with each other. The power of these networks arise from their ability to combine their many vantage points of the individual nodes and to work together. This allows for behaviour to emerge which is greater than the sum of the ability of all the nodes in the network. The complexity of these networks varies based on the application domain and the physical phenomenon being sensed. Although sensor networks are currently well understood and used in a number of real world applications, a number limitations still exit.

This research aims to overcome a number of issues faced by current WSNs, the largest of which is their monolithic or tightly coupled structure which result in static and application specific WSNs. We aim to overcome these issues by designing a dynamically reconfigurable system which is application neutral. The proposed system is also required to facilitate intelligence and be sufficiently efficient for low power sensor node hardware. Our approach involved designing a WSN framework utilizes a layered architecture consisting of a number of modular layers arranged in a stack, each layer performing operations of a greater level of abstraction than the layer below.

Intelligence is facilitated by utilizing intelligent software agents. Intelligent processing within each of the layers was achieved through Blackboard Systems, each consisting of Blackboard Agents. Mobile agents were also utilized to allow for application logic and processing to be distributed, rather than just the distribution of raw sensor data.

A proof of concept prototype which combines layered architecture and intelligent software agents was implemented. Evaluations were performed which investigated the ability for applications to be remotely deployed and reconfigured on a WSN using the prototype framework. A simple application which utilizes the framework and expresses intelligence behaviour was evaluated. System latency and overhead together with scalability were investigated. The findings of these evaluations suggest that proposed system satisfies all of the required research aims.

Acknowledgements

To my supervisors: Hanh Le and Anet Potgieter, without your knowledge and input, this work would not have been possible.

To my loving family: Gary, Carol and Sean, your constant support and motivation has kept me going though the ups and downs, of which there were many.

To my friends who stuck through it with me: Thank you all. Anton Eicher and Duncan McRoberts, your technical input really helped solve some of those terrible challenges. Gavin Marchbank for your constant pushing on those days where I needed it.

Thank you all!

Thank you to the Meraka Institue for providing funding for this research.

Contents

List of Figures

List of Tables

Acronyms

abDD	Agent-based Directed Diffusion
ACO	Ant Colony Optimization
API	Application Programming Interface
BDI	Belief-Desire-Intention
BDJI	Belief-Desire-Joint Intention
CSWSN	Client/Server based Wireless Sensor Network
DMS	Data Management System
DAI	Distributed Artificial Intelligence
GA	Genetic Algorithm
GPS	Global Positioning System
GDP	Gross Domestic Product
J2ME	Java 2 Platform, Micro Edition
J2SE	Java 2 Platform, Standard Edition
JVM	Java Virtual Machine
MANET	Mobile Ad-hoc Wireless Network
MA	Mobile Agent
MAWSN	Mobile Agent based Wireless Sensor Network
MAS	Multi-agent System
OTA	Over The Air
RFID	Radio-frequency Identification
S-R	Stimulus-Response
TSP	Travelling Salesman Problem
WPSN	Wireless Patient Sensor Network
WSN	Wireless Sensor Network

Chapter 1

Introduction

A WSN is a network consisting of a number of sensor nodes. Each of these nodes is a small, typically low-cost hardware device which is able to sense various physical phenomenon of its surrounding environment [1]. These nodes are able to wirelessly communicate with each other as well as perform basic processing and data storage [2]. Some sensor nodes are also able to act on their environment through various actuators. Positioning of the sensor nodes with in a network can be well defined and structured or possibly random. In some cases the nodes may be mobile, allowing for their position to change over time.

The power of these networks of simple nodes arises from their ability to operate together with each other and provide information from different vantage points. This allows for the WSN to perform tasks which are more complex than the sum of the ability of the nodes which comprise the network.

Given that a number of difference physical phenomenon can be sensed by various sensors, a large number of different WSN applications exists. The complexity of these system also range from the sensing of static objects and spaces to sensing of the dynamic interaction between objects within a space. The application domains in which the network operate can from simple environment sensing, such as period temperature reading, to complex military applications, such as intelligence mine fields which detect specific vehicles.

However, although sensor networks are currently well understood and utilized for real world applications, a number of limitations exist in the current systems.

The greatest of these challenges is that most WSN applications are developed either as single monolithic blocks or as a few tightly coupled modules. This lack of structure results in greater difficulty when designing, developing and testing the system. Monolithic systems also provide a poor means for handling complexity. A lack of reconfigurability and intelligence are also major issues faced by current WSN applications.

This chapter presents the research motivation first as a result of the WSN shortcomings, followed by objectives for this research. An overview of this thesis is then presented.

1.1 Research Motivation

The motivation behind this research is to overcome a number of challenges faced by current WSNs. In order to achieve this, a number of shortcomings were first identified. Details of these challenges are presented below:

Network lifetime of current sensors networks is often limited due to the small energy source of each of the sensor nodes. These energy sources are typically not replenishable. The way in which sensor data is processed and communicated dramatically impacts the way in which this energy is consumed. Reducing the energy consumption of the nodes directly improves the lifetime of the WSN.

Routing is a critical area of networking. This is a traditionally well researched and challenging area. Routing in WSNs is considered important due to its influence over the overall network performance as well as the network life time due to energy consumption.

Localization of nodes refers to the ability to determine the position of nodes within space and in relation to each other. This is important because the sensor data corresponds to physical area. This area also complements the routing within a sensor network as the position of nodes in relation to each other provides critical information regarding the cost of communication.

Besides those areas identified as shortcomings or challenges, a number of addition features where identified which could improve the performance of WSNs. Details of these additional features are provided below:

Self-organization is a concept which refers to the ability of individual nodes to arrange and organize themselves to form a cooperative unit. As the power of WSNs is mainly due to this ability of nodes to work together, self-organization is a critical area which can be improved on.

Reconfigurability of a system is critical to allow for changes to be made without having to redevelop and deploy the application. In a number of cases where changes are required, physical interfacing with sensor nodes is required in order to alter the application. This is not always possible.

Distribution of processing is an area of much complexity and greatly challenging. Centralized processing within such systems has negative impacts in a number of areas including processing time and energy consumption as a result of distant communication.

Intelligence is an abstract requirement which refers to the ability for an application to learn and improve its performance as experience is gained. A number of areas within a WSN application could benefit from intelligent processing and learning.

Security and Privacy are areas of growing interest. Both of these mechanism are required for WSN applications which handle sensitive data or require data integrity to be ensured. The largest challenge of this is the typically high performance overhead such mechanisms introduce.

1.2 Objectives

The problem with many exiting WSN systems is they address only a few of the above mentioned challenges. These systems are also not adaptable to new requirements. The objective of this thesis is to design and develop a WSN frame-

work which can provide structure and facilitate intelligence. Such a framework is expected to overcome or facilitate applications to overcome a number of the above mentioned shortcomings in current WSNs, depending on the need.

In order to achieve this large objective, the following smaller objectives need be satisfied. The proposed framework need be **application neutral**, allow for **reconfiguration** and **facilitate** intelligence. All of this need be achieved while not having a significantly negative impact on the performance of the WSN applications which utilize the proposed framework.

The contributions made by this research were to provide insight into the domain of WSNs and their use with Agent based systems and blackboard architecture. This work also provides an proof-of-concept prototype on where empirical research and experimentation can be performed.

1.3 Scope and Limitations

This research is focused on the design of WSN frame and the development of a proof of concept prototype.

This framework design acts as a foundation onto which various WSN applications can be developed, not as a standalone WSN application for various applications. Because of this, development and design focus was placed on the framework and not the applications. However, for testing and evaluation purposes, a number of WSN applications were developed to illustrate how the framework operates.

Due to the prototyping nature of this research a simpler development and evaluation platform was used than that used by real world WSN applications. Java based Sun SPOT was used as the platform due to its ease of use and availability of hardware.

1.4 Research Methodology

Experimental evaluation was utilized to determine the success of our proposed system. An empirical approach was taken whereby a proof of concept prototype of the proposed framework was designed and implemented. This prototype served as a test bed. A number of qualitative experiments were performed and the results captured. These results were then analysed to determine the degree to which the research objective had been satisfied.

1.5 Dissertation Outline

Details of each of the chapters within this dissertation are provided below:

Chapter 2: Background This chapter provides the necessary background information on which this research is based. The chapter will investigate Wireless Sensor Networks as well as Software agents. Each of these concepts will be discussed by explaining how they operate, some examples systems and finally the benefits and shortcomings of each. All the proceeding chapters draw upon the concepts and examples presented in this chapter.

Chapter 3: Research Aims and objectives In this chapter an explanation of the research problem, aim and objectives are presented. The research objective based on the aim will also be presented together with any assumptions or constraints on which the work is based. This chapter also provides justification as to why this research is important and why each of the objectives are required.

Chapter 4: Our Proposed System This chapter presents the details of the approach and method used in order to satisfy our research aim. Details of the technologies to be utilized and how they operate within the proposed system are presented. The design process and components of the proposed solution are presented. Finally details regarding the interaction between components is provided.

Chapter 5: Implementation and Application details In this chapter details of the implementation of the proposed system are provided. The platform on which the system was implemented is discussed. The implementation of the various system components are presented and technical details are provided. The final section of this chapter provides details regarding the application specific components.

Chapter 6: Evaluation The evaluation and testing of our proposed system is discussed in this chapter. The evaluation metrics for the experiments are first identified, as these determine if the earlier stated goals and objectives have been achieved. The experimentation procedures to determine the performance of the system across the evaluation metrics is then discussed in detail. This chapter concludes with a presentation of the final results of the various experiments.

Chapter 7: Conclusions In this chapter we conclude by presenting a brief summary of the report as well as presenting a final analysis of the evaluation results. A recap of the aim, motivations, objectives and approach of this research is presented. A final high level analysis of the system evaluation is then provided which discusses if the research problem has been satisfied. Possible future extensions and work based on this work is then presented.

Chapter 2

Background

In this chapter we outline the key topics and research areas related to this research. This chapter is divided into two sections. In the first section we discuss WSNs. We present the various categories of WSN based on what they sense and their application domain. We then look at example WSN systems as well as the potential shortcomings of these systems

The last section examines Software Agents as a means to facilitate intelligence in WSNs. We start by introducing the concept of agents and the various types of agents which exist. Systems consisting of multiple agents are detailed and examples of such systems are discussed. Examples of agent based WSNs are also detailed. We conclude by identifying some of the benefits of utilizing agent based systems as well as possible shortcomings.

2.1 Wireless Sensor Networks

In this section we introduce the concept of Wireless Sensor nodes and how these nodes function together to form a WSN. We will then look at the capabilities of such networks, how they are categorized as well as examine a number of example systems. Some of the problems and challenges faced by these networks will also be discussed.

Recent advancements in technology have enabled the development of small,

Figure 2.1: A MICA2 Wireless Sensor Node

low-cost hardware devices, known as Wireless Sensor Nodes, which are able to communicate wirelessly over short distances [1]. Besides wireless communication, these devices are also capable of basics data processing and storage [2]. The energy source of such a device is typically a battery. Figure 2.1 is a picture of a MICA2 sensor node manufactured by MEMSIC [3].

A WSN is a network consisting of a number of such sensor nodes which have been deployed within or close to a physical phenomenon which is to be sensed [1].

As individuals, the Wireless Sensor Nodes have very little capability, but when combining their many vantages points throughout the sensor field and working as an aggregate, they have substantial sensing and processing power [2]. Based on this ability to sense and process the sensor data, WSNs could provide an end user with intelligent results, thus allowing them to better understand the environment being sensed [1]. These networks could provide Ambient intelligence [4], where different devices gather and process information from different sources to control physical processes as well as interact with end users.

The positioning of these nodes in space could be predetermined or random [1]. Sensor nodes can also be mobile. This mobility could be out of the control of the node, or through the nodes self control [4].

A wide variety of sensors exist which are able to sense a large range of different physical and ambient conditions. These include the following [1]:

- temperature
- humidity
- movement
- lighting conditions
- pressure
- soil make up
- noise levels
- the presence or absence of certain kind of objects
- mechanical stress levels
- medical metrics

These networks can be homogeneous by containing a number of the same type of sensors, or heterogeneous by containing a number of different sensor types.

WSNs can also be viewed as a specialized form of Mobile Ad-hoc Wireless Network (MANET). Although MANETs and WSNs are very similar, there are a number of characteristics not shared by both networks. A comparison of these two types of networks is provided below [5]:

- A WSN can have several orders of magnitude more nodes than a MANET.
- WSN nodes are often more densely deployed.

- The topology of WSNs are more dynamic than MANETs. This is also compounded by sensor nodes being prone to failure.
- WSN nodes have no global identity number to uniquely identify them.
- The computational capabilities, memory and power are greatly restricted on WSN nodes.
- WSN nodes are often less mobile, due to the human involvement in MANETs.
- The aim of a WSN system is for nodes to work together to maximize system wide performance, whereas the aim of MANET nodes is to maximize individual performance.

In the subsection below, we present the various categorizations of WSNs

2.1.1 Categories of Wireless Sensor Network

Given the large number of different sensor types, WSNs can be developed for a number of different applications. WSN applications can be differentiated based on a number of different attributes. To provide context into WSNs and how they function, a number of different categorizations are discussed below.

Cruller [2] differentiates WSNs based on the phenomenon being sensed. These categories are:

Monitoring of space The task of such a network is to monitor and collect readings over time across a given space. This space would generally be complex enough to exhibit spacial variations in the attributes being monitored. Many of the initial WSN systems were of this type.

Due to the large amount of sensor data being handled, the greatest challenge for such a system is to provide only that data which is necessary at the required time. The data supplied should not simply be a constant stream of raw sensory reading from each of the sensor nodes. Because of this, basic data processing to reduce the size of the data should be performed by the nodes before it is sent to the sink node.

Monitoring of physical entities or objects This involves the monitoring of physical structures, such as machines, buildings or human beings.

Typically such monitoring is performed to ensure that the entity or object is behaving normally under the given conditions. This is based on the concept that when an object is presented with a given stimuli, a typical response can be expected. In such a network, information regarding whether or not the monitored object is behaving in the expected way is transmitted from the node providing the sensed data, known as the source, to the node to the sink node, where the data sensed and transmitted by the source need be sent.

Monitoring the interaction of physical entities or objects with each other and the encompassing space

Such monitoring is a combination of the monitoring of space and the monitoring of entities or objects. This is the most complex of the three WSN

applications, due to the highly dynamic environment as well as the large number variables involved.

Such a WSN monitors the environment as well as how the various entities react within it. As with space monitoring, sensor readings are taken and processed for various areas within the encompassing space. Sensors also monitor the objects within the space to determine their reactions to the other objects and the environment.

A second approach to categorizing WSNs is based on the application area. Five application areas have been identified [1]:

Military WSNs can form a critical part of the military command, control, communications, computing, intelligences, surveillance, reconnaissance and targeting systems.

Because of this, WSNs are considered important and they are used to provide a number of applications within this area.

Friendly forces, equipment and ammunition can be tracked and monitored by using sensors. Battlefield surveillance and reconnaissance of opposing forces can be performed by deploying networks to monitor critical areas. Targeting of weapons can be performed by incorporating WSNs into guidance systems. Sensors on the battlefield can be used to assess damage or detect the presence of nuclear, biological or chemical attack.

The recent increases in terrorism have also promoted the development of more technologically advanced methods to be used on the battlefield.

Environment Applications of this type include tracking of wildlife, environmental conditions or large-scale earth monitoring.

A number of example applications exist in this area. Forest fire detection where users are notified of the origin of a fire before it spreads. Biocomplexity mapping of the environment by monitoring the environment as well as its contained wildlife. The interaction of wildlife and other wildlife as well as the wildlife and its environment is all monitored. Flood detection and precision agriculture systems have also been developed for environment monitoring.

For such applications, WSNs provide an accuracy which is far greater than previous sensing methods, such as airborne or satellite sensors.

Health These applications for WSNs involve the area of medical health care and monitoring of patients. These applications need not be directly linked to the monitoring of patients, but rather aimed at improving health care in any way possible.

Telemonitoring of human physiological data is an application of this type where patients are monitored using sensor nodes attached to their body. These sensors provide the medical staff with vital patient information either by downloading the stored sensory data, or by remotely accessing the data.

Tracking and monitoring doctors and patients inside the hospital is an application which is not directly linked to the health of the patients, but

allows for hospital staff to easily locate doctors and patients within the hospital. Medication within the hospital can also be tracked in this way to prevent the prescription of the incorrect medication to patients.

Home Such WSNs reside in the households of the users. The aim of these applications is to provide more intelligent environments for the users.

This includes home automation where smart sensors and actuators are built into home appliances which can interact and communicate. Smart environments which learn to adapt to the users needs are another home application of WSNs.

Other commercial areas These applications are related to business, the management of product quality, manufacturing and inventory. The business environments of office buildings and factories are also covered by this category of application.

Another example of such an application is inventory management in factories and warehouses. By utilizing sensor networks on various inventory items, users can efficiently locate them within the building.

Other examples in this application area which do not include the work place or products are vehicle tracking and the detection of vehicle theft. These networks utilize sensors to determine the positions of the vehicles and transmitters to communicate the information to the user.

The third categorization scheme of WSNs is based on the interaction between the source and sink nodes [1, 4]. These categories are described below:

Continuous Sensing Data is collected by the source node or nodes and is continuously streamed to the sink node.

Periodic Measure Data is collected by the source node or nodes and is periodically transmitted to the sink node, rather than continuously streaming the data.

Event Detection Data is collected by the source or group of source nodes. This data is then processed by the network and communicated back to the sink only if a predefined even has taken place within the sensor field.

Event Identification Like event detection, source nodes only communicate with the sink when an event has taken place. However, unlike event detection, identification of an event involves monitoring a sensor field for a number of possible events. The sink is then notified as to which of the possible events has taken place. This will usually involve the collaboration of a number of source nodes.

Location sensing and tracking This too requires the collaboration of a number of sensor nodes within the network. These nodes work together to determine the position of an event within the sensor field. This would often require event detection and identification before the event location can be determined and tracked.

Edge Detection Edge detection of a given phenomenon can also be performed by combining detection, identification and locating of events. The area in

which the event is occurring can be clearly defined. Function approximation could be combined to predict event edges in areas which lack sensor data.

Local Control of Actuators WSNs can contain sensors as well as actuators. A subset of the network nodes could communicate to form a closed control loop where Sensor Nodes locally control actuators. This can be done without requiring the sensors and actuators to have communication with a base station or centralized controller node.

Function Approximation By processing the data output in a WSN, predictions of formulas can be made. These can be used to predict some conditions in those areas which are not being sensed.

In the following section we examine at a number of example WSNs and their applications.

2.1.2 Example Wireless Sensor Network Systems

In this subsection we examine a number of example WSN systems. For each of the examples, we discuss to which of the above mentioned categories the network belongs, how the network functions and the possible shortcomings of the system.

The first WSN system we discus is a localization system developed to determine the position of a shooter within an urban environment [6]. This is an example of a military application that monitors space.

The system determines the location of the shooter as well as the trajectory of the bullet by sensing the muzzle blast of the weapon and the shock-wave. Both acoustic and light sensors were used in this network which could be deployed randomly or in a predetermined pattern. Due to the possibility of random deployment, the sensors nodes were required to localize and organize communication by themselves.

According to the previously mentioned categories, the interaction pattern used within this network is that of location sensing and tracking. A sensor-fusion algorithm was used that relied on a centralized network node, which receives and processes all incoming sensory data.

An evaluation of the system found that 60 sensors disturbed in an 100×100 meter test area could determine the position of the shooter with a latency of 2 seconds. The accuracy of the system was approximately 1 meter, which meant that it could be determined from which window the shot was made. It is presumed that the accuracy of the system could be improved with improved self-localization of the sensor nodes or a greater density of sensor nodes.

The second example system we discuss is a WSN which monitors a physical entity. In this case the physical entity being monitored is a patient for medial application [7].

By monitoring patients and providing their medical staff with relevant real-time patient information, it can be ensured that the patients receive proper medical care. This is done by utilizing a number of different types of sensors to monitor the patients vital signs, creating a Wireless Patient Sensor Network (WPSN).

This is a heterogeneous system that operates through the use of two platforms: Lightweight sensor nodes and a heavy weight device such as desktop computers or smart phones.

The heavy weight devices are used to house an Agent based Data Management System (DMS). This is used to collect, integrate, analyse and present the sensed data to the medical staff. Lightweight software agents are used on the sensor nodes. These agents collect data from the various sensor nodes and send this data to the DMS. The topic of agents and agent based systems is covered in greater detail in 2.2.

Due to the use of agents to collect the sensor data, the interaction between the source nodes and the sink can vary. The agents can be configured to provide a continuous stream of sensor data or a periodic reporting of data. The flexibility of agents also allows for multiple applications to be performed in parallel on the same network, although evaluations found this to be much slower than performing a single application.

Another example which aims to provide the users with better data in order to improve decision making is that by Sikka [8]. The aim of this system is to provide farmers with better information about their farm, thus improving the way in which the farm is managed. This WSN application is considered an important commercial application, given the large contribution farming makes to the Gross Domestic Product (GDP) of developing countries. Various types of sensors were used on the farm land as well as the animals, making it a heterogeneous network which monitors the interaction of objects with each other and the encompassing space.

The system was developed as a test-bed, rather than a single application network. This allows for various interaction patterns to be utilized to handle the network data.

A number of different sensors types were used in these networks, this included moisture sensors at different depths, weight sensors to determine the quantities of animal food and water, Radio-frequency Identification (RFID) tag readers, sensors for tracking animal movement as well as actuators to apply stimuli to the animals. A number of sensors, such as the tracking Global Positioning System (GPS) and actuators, were placed on the animals and were thus mobile. Many of the sensors, such as the RFID tag readers, were also energy intensive and therefore powered by solar panels.

In order to evaluate the system, various applications were implemented on the network: Monitoring of the farm land using the moisture sensors, animal tracking to determine animal kinship and movement and the use of actuators in an attempt to control the movement of various animals. These tests were performed over a period of 6 months, after which many of the nodes were still alive and self-powering.

The intention for this work is to later include more sensors of different types and to understand how actuators together with tracking sensors could be used for greater control over the farm animals.

An increasingly popular area of use for WSNs is monitoring of large areas or areas which are potentially dangerous to human beings. The system developed by Werner-Allen [9] to monitor active volcanoes satisfies both of these criteria. This system was designed to capture volcanic activity in the form of seismic

and infrasonic signals. Previously this would have been achieved by using large monitoring stations located around the active volcano, each of which would log the various sensory data to a hard disk. These hard disks would then have to be periodically collected by the researchers. Previous systems such as this also consumed more power than smaller sensor nodes and were far more cumbersome to deploy.

Deploying a WSN on an active volcano introduces a number of challenges for data storage, transmission and routing. These are due to high data rates, high data fidelity and sparse networks spread over a large physical area.

As the researchers were interested in specific events taking place, event detection was performed within the network. To ensure that all the required data was measured, before and after the event, a circular buffer was used to store approximately 20 minutes of sensor data. When an event was detected, this buffer of data would be transmitted to the sink nodes. This required multi-hop routing due to the sparse nature of the networks topology and large area of the sensor field. A local event detection system as well as a global event notification system ensure that data was only transmitted when a network wide event had taken place.

The second major challenge was the reliable transmission of data which was required due to the high fidelity of the data. A reliable data-collection protocol called fetch was used to ensure all lost or corrupted data was retransmitted to the sink.

The final challenge of routing was overcome by selecting routes based on link quality. Basic reconfiguration of this system was also possible with the system through a remote web interface. This allowed for basic system parameters to be remotely altered in real-time. There was an instance where a software component had failed which could not be remotely repaired. This required that researchers returned to the nodes to manually repair the fault.

Evaluations of this system found that the goals of providing sufficiently rich and accurate sensor data are being satisfied. However, in some cases small global events shortly prior to large global event would result in sensor data being transmitted to the sink. This transmission of data resulted in sensors not being able to log the larger event which was taking place while the nodes were transmitting their buffer of data relating to the smaller event. This could be viewed as a limitation on the centralized style of processing being used.

Sha [10] researched the concept a fire rescue WSN system. This research consisted of an initial investigation to identify the requirements and challenges of developing a system to assist fire rescue teams. The system involves monitoring the interaction of physical entities, in the form of fire-fighters and fire trucks, with each other and the encompassing space, the fire field. The application area for this research is that of health and safety. Continuous monitoring, event detection and tracking interaction patterns were all planned to be utilized for this complex system.

During a fire, an incident commander is in charge of monitoring the fire and the real-time assignment of fire-fighters. The proposed system, FireNet, aims to provide the incident commander with all the information they require. In order to satisfy this aim, a number of requirements were identified.

Accountability of fire-fighters must be had by having access to as much relevant information as possible. This includes static information such as their

name, ID, speciality and squads they are assigned and dynamic information, such as the time they have spent in the fire field, their real-time location, workload and their physical condition. Real-time monitoring related to the fire field is also required. This includes the humidity, temperature wind speed and density of smoke. Other critical events should also be monitored in real-time, such as the death of a fire fighter or a dramatic change in the fire field. Intelligent scheduling and resource allocation was also identified as a system requirement. Due to the large amounts of sensor data and the possible danger of this application, the users required an intelligent system to assist the decision making process. Web-enabled service and integration are the final requirements for the system. The WSN data is often required by staff at the fire station. Using a web-enabled system, real-time information related to the fire rescue could be provided remotely.

Based on these requirements, the FireNet architecture was proposed. This consisted of self-organizing heterogeneous sensors which are attached to fire-fighters and their vehicles. Heavy weight laptop computers equipped with GPS will be housed in the vehicles. These will act as sink nodes and landmarks to allow for localization of the other sensors. The nodes attached to the fire-fighters will also be able to facilitate multihop routing of data within the network

A number of challenges were identified which need be overcome before the system could be implemented. These include the requirement of real-time self-organization by nodes in harsh environmental conditions, fault tolerant routing to provide the required accuracy and service differentiation of data based on differing priority.

Based on previous WSN systems, such as those mentioned above, a number of shortcomings and challenges have been identified. These are outlined in the following section.

2.1.3 Shortcomings of current Wireless Sensor Networks

Current WSN systems are faced with a number of challenges and shortcomings. Many of these have been pointed out in the example systems listed in the section 2.1.2. Those shortcomings considered most important have been listed below with possible solutions.

Network lifetime is a challenge faced by WSNs due to the limited power source used by the sensor nodes. In most cases the sensors are battery powered or equipped with hardware to allow them to scavenge power from their environment, such as solar panels [8]. When these nodes exhaust their temporary power source, it is usually not possible to replace them. This limits the life time of the sensor network as a whole and makes power management an important challenge. Power is consumed by the sensor nodes through sensing, processing and communication [1].

The power consumed by sensing varies dramatically based on the type of sensor. To optimize the use of power by sensing, nodes spend a large portion of time in a standby mode, which utilizes very little power. They only wake for short periods during which they sense data. Processing of data is usually the cheapest process. Because of this it is more efficient to

process data before transmitting it to other nodes, rather than just transmitting the raw data [2]. Communication is generally the most expensive process. This can influence the lifetime of the individual nodes as well as the network as a whole. There is often a trade off between the Quality of Service provided by the network and the network life time [4]. Many applications aim for the distribution of communication load to increase the lifetime of the WSN [5]. It must also be noted that a number of network lifetime metrics exist. These include the time for the first node to fail, time until there is an area of the sensor field which is not covered by the network and time until the network is partitioned.

Routing refers to the process of determining the path through which data must travel as well as the forwarding of data in order to get to a given destination from its source or current location. This is a traditional, well research challenge found in large number of networking areas.

In this section we focus on the concept of routing as well as routing for WSNs [4]. A number of different routing techniques exist for WSN applications [11]. Typically multihop routing is used in such networks. This is required when non-neighbouring nodes must communicate, forcing data to be forwarded by intermediate nodes between the source and destination. Routing tables are required which store the next hop neighbours for each destination as well as the associated cost of that route. This cost is used to determine which next hop is utilized. The routing algorithm and protocol populate the routing table with this information. This cost metric can vary based on a number of different link parameters, such as available bandwidth, expected Quality of Service and latency.

Routing protocols can be proactive or reactive. Proactive protocols populate the routing tables based on the initial state of a network. This information is determined before it is requested. Reactive protocols provide the required routing information after it is requested, looking at the state of the network at the time it is requested. Typically proactive routing allows for routing information to be more readily available, however unlike reactive routing, this information could be out of date.

Wireless Sensor Networks face a number of routing challenges which are not faced by wired networks. Due to the distributed nature of WSNs, a low overhead distributed routing algorithm is required. Multipath routing is also important due to the dynamic nature of WSNs.

A traffic dependent adaptive routing algorithm [12] has been proposed which aims to provide a distributed solution which can overcome a number of these challenges. This system employed mean delay as the cost metric. Evaluation results suggest good performance for small sparsely connected networks.

Addressing of nodes within the network is another major challenge due to Sensor Nodes not having meaningful unique identifiers. Data centric approaches, where target nodes are based on the data they provide, or geographic based addressing must be used. WSNs do have an advantage over wired networks in that multicasting of data to multiple hosts can be done at minimal addition cost. This is due to wireless broadcasts arriving

at all nodes within the broadcast range and not just the node which is being addressed, as can be the case in wired networks.

In this work our focus is primarily on agent based routing. This is discussed in more detail in section 2.2.3.

Localization refers to the problem of determining the position of wireless sensor nodes in physical space.

Knowing the position of nodes is vital for targeting nodes based on their physical locations and determining the physical location of events which have taken place within the sensor field. Localization can also be utilized by routing algorithms to determine the shortest physical paths within the networks. GPS is an example method used for localization however it is quite expensive in terms of hardware resources and money. Indoor use of GPS is also not possible.

To prevent negatively impacting the network lifetime, localization algorithms are required to be distributed, as well as having a low overhead. A number of solutions have been proposed which utilize beacons to allow sensor nodes to determine their location. A beacon refers to a sensor node which has a known location. Other nodes in the network can determine their location relative to these beacons based on the signal strength of the link between the node and beacon [13]. These approaches are not optimal due to the inaccuracies of range measurement. Ramadurai [14] proposed a probabilistic position estimation algorithm which considers these measurement inaccuracies.

Self-organization refers to the concept by which global order can arise from local interactions. More specifically that a collection of individual units can co-ordinate together to form a coherent unit which adapts to achieve a goal more easily over time [15]. These individual units are able to achieve this without any centralized controller.

Such self-organizing behaviour is critical in WSNs due to their distributed and dynamic nature. The nodes are also prone to failure. Sensor nodes must have the ability to organize themselves with only their local information and no assistance from a centralized controller. Due to random deployment, WSNs are required to identify their network neighbours as well as routing tables. In some cases, nodes are also required to determine their position in physical space. These requirements are particularly challenging in the typically dynamic environments of WSNs [13].

In order for self-organization to be achieved, nodes must have the ability to communicate efficiently with each other, determine the performance of the network and work together to increase this performance. The overall performance of the system in a specific environment should improve as the system gains experience and the individual nodes organize themselves more efficiently.

Reconfigurability refers to the ability of a system to be reconfigured or altered without requiring redeployment. A number of current WSN systems allow for limited reconfiguration of system parameters such as sampling rate. Larger changes require that software be entirely redeployed on the nodes.

A lack of remote operations also requires users to physically connect to the sensor nodes to make any changes.

The ability for systems to be reconfigurable is important due to the dynamic nature of the network, as well as a possible change in network requirements. This is also true for scalability where systems require reconfiguration due to the addition or removal of sensors.

Achieving reconfigurability in previous systems was challenging due to the monolithic style in which systems are developed. These systems were intended for single applications. System components in such systems are often tightly coupled to optimize the memory usage and processing performance. This lack of good structure makes it very difficult to alter applications.

Distribution of processing is a challenge related to the way in which sensor data is processed within the WSN.

Currently many systems rely on network nodes to collect sensor data, which is then transmitted to a single centralized sink, where it is processed and presented to the user or transmitted back to network actuators to affect the WSN environment. This is referred to as centralized processing.

As mentioned in the above point regarding network life, the transmitting of raw data before being processed can be an energy hungry operation. This also results in a heavy overhead when combining the many vantage points of the different sensors, as required for applications such as event identification and tracking. This is due to the possible situation where a pair of sensor and actuator nodes in near proximity are forced to communicate via a centralized sink which may be many network hops away. This prevents the use of closed control loops within the network, where data is processed locally and directly influence actuators.

By distributing the processing within the network, the limited ability and resources of each node can be combined to create a more substation system with greater ability.

Because of these many factors, the processing of data should be decentralized among the various sensors within the network.

Intelligence is a broad term referring to an ability to reason and plan. Intelligent systems are also expected to learn from experience and improve their performance over time by learning.

In terms of WSNs, incorporating intelligence can optimize a number of areas. More relevant sensor data could be provided by learning from previous similar requests. Routing algorithms can also incorporate intelligence to learn and make predictions based on trends in the network state. Event detection and identification can improve accuracy and efficiency by learning from previously identified events.

Security and Privacy in WSNs has recently increased in interest due to the increased use of sensor networks.

This is specifically true for military applications, where integrity of data is essential [6]. It must be ensured that the data within such networks

can not be easily acquired or manipulated by unauthorized users. The injection of false or unauthorized data into the network must also be prevented. Commercial applications also require security and privacy, as these systems could handle confidential client information such as credit card details.

The inclusion of security and privacy measures within a Sensor Network is challenging due to the increased overhead in both processing and communication. The processing required for encryption can be very intensive on the lightweight sensor nodes. The size of the encrypted data is also larger than that of unencrypted data. This increases the bandwidth requirements of the network.

2.2 Software Agents

In this section we introduce the concept of agents. We will start by looking at what agents are and their defining characteristics. We will then look at various types of agents, how agents can be grouped to create multi-agent systems and the design of these systems. We will finish by briefly describing some example applications of such agent based systems and finally outline some of the shortcomings of current systems.

An agent can be described as anything that perceives its environment through sensors and acts upon that environment through effectors [16]. This description covers a number of different entities in a number of different fields. In this research we are interested in software agents, more specifically intelligent software agents.

A software agent consists of two parts, namely the agent architecture and agent program [16]. The agent architecture hosts the agent program, provides the program with sensory information regarding the environment and allows the agent program to act on the environment. Agent programs accept the percept information and respond by providing the architecture with actions to perform. The goal of the agent program is to create a mapping between possible percept sequences and actions to produce the best possible outcome. This mapping between percept sequences and actions is determined in a number of different ways, based on the agent requirements and environment.

Software agent programs have a number of typical requirements based on the agent application and type of agent system. Each of these desired characteristics are detailed below:

Rationality is a key component of almost all software agents. The aim of a rational agent is to act on the environment in such a way as to result in the most success [16, 17].This success measure varies based on the agents' application. A rational agent's program will attempt to determine the ideal mapping for all possible percepts.

Autonomy is highly a desirable characteristic to provide software agents with intelligence. This refers to the ability to gain experience and adapt based on knowledge of previous events [16]. An autonomous agents behaviour is determined by both its built-in knowledge as well its own experience, allowing it to operate in a wide variety of environments given sufficient

time to adapt. A number of different machine learning methods can be implemented to achieve this.

Mobility refers to the ability of independently traversing a network and executing on different hosts along the route [17]. This is only required in network based systems where agents can exist on a number of hosts. Mobile agents are discussed in more detail in section 2.2.1

Communication and Collaboration are essential requirements of systems consisting of multiple agents. In systems where multiple agents are attempting to work as a unit, agents are required to communicate and collaborate with each other. This prevents any competing actions from taking place. This requirement is discussed in more detail in section 2.2.2.

Because of the extremely strong coupling between agents and their environment, it is important that an agent be designed for its environment [16]. Very simple agents can express complex behaviour when they interact with sufficiently rich environments for which they have been designed [17]. An environment can be described in the following ways [16, 17]:

Accessible versus inaccessible refers to which aspects of the environment are accessible to the agents' sensors. If the agent is able to sense the complete state of the environment, it is said to be accessible.

Predictability of the environment from the point of view of the agent. If the next state of the environment is entirely determined by the current state and the actions selected by the agents, the environment is said to be deterministic. The environment could be stochastic in that actions selected by the agents together with some random element determine the next state. Chaotic environments are also possible which lack order and predictability entirely.

Episodic versus non-episodic refers to how the environment changes over time. Episodic environments are composed on a number of independent temporally separated episodes which can be acted upon without any concern for previous independent episodes. Non-episodic environments have states which change based on all past states.

Static versus dynamic environments change at a different frequency. Dynamic environments can possible change state while the agent is deliberating over what actions to take. This requires agents to constantly observe the environment even while determining their next action. Static environments remain constant while agents determine their action.

Discrete versus continuous refers to the types of percepts and actions which the environment provides and actions which the agent can perform on the environment. If the number of percepts and actions is a limited number and clearly defined, the environment is considered discrete.

Multi-agent environments are also possible where a number of agents exist in the same environment at the same time. This is only a concern where agents are collaborating to complete a task as a group.

Given the above descriptions of various agent and environment characteristics, a number of different types of agents exist. These are described in the following section.

2.2.1 Types of agents

A number of different types of agents exist. These differ in the way that their agent program determines a mapping between percept sequences and actions. Agents can also differ depending if they remain static at a single host or traverse between multiple hosts.

In the sections below we look at various types of agents. The distinction between these various agent types is based on how they determine what actions should be taken for given percept sequences.

Reflex agents are the simplest form of agent program [16]. These agents, also known as Stimulus-Response (S-R) agents, have no internal state and react to immediate stimuli from their environment [18]. These agent programs utilize a condition-action production system to determine the action to be taken. This works by determining a condition based on the percepts provided, which can be preprocessed from raw data. The correct action is then identified from the condition-action production system. Complex S-R agents also allow for production systems to be called as an action from another production system [18].

A major shortcoming of simple S-R agents is that decisions can be made based only on current percepts. No knowledge of previous states is used. In order to overcome this, agents must keep an internal state of the environment. Keeping an internal state of the environment is essential for distinguishing between world states which generate the same percepts, however require different actions to be performed [16]. For example an image processing agent for detecting motion based on a video stream as its environment. This agent would require knowledge of the previous state of the environment in order to determine if any changes have taken place in the current state.

In order for an internal world state to be maintained, two kinds of knowledge are required: Information about how the environment changes independently of the agents' actions and how the agents' action could effect the environment. Both of these are heavily dependent on the predictability of the environment. If the environment is sufficiently predictable and not entirely accessible, the state of unseen parts could be determined [16]. This could be done by interpreting current and past percepts and knowing how the environment would have evolved with and without the agent taking action.

Both of above mentioned reflex agents lack autonomy as all decisions are based on their static condition-action rules. Hard coded condition-action rules also make it difficult for application changes to be made. The agent types discussed below aim to overcome these issues.

Goal-based agents utilize goals, which describe desirable environment states [16]. By knowing the current state of the environment and predicting how

the environment evolves, goal-based agents can determine which action to take in order to achieve its desired environmental state. Determining which action to take is a trivial problem when the environment state is a single step away from the desired state. This is not the case when multiple steps are required. Searching and planning are required to overcome this challenge.

A major benefit of such agents is that their goals can be altered without requiring major design changes, unlike reflex agents which require their condition-action rules to be altered to alter their behaviour. Goal-based agents also have a drawback in that the actions chosen are not always the most optimal. These agents are concerned only with the first selected action which moves them closer to achieving the goal.

Utility-based agents select actions to perform by utilizing a general measure, which provides a better comparison between possible environment states.

A utility function is used by these agents to map the environment state onto a value which describes the degree to which the state is closer to satisfying the goal. This allows for rational decisions to be made when a number of possible actions result in an environment state closer to the desired goal state.

Another benefit of the utility function is that it allows for rational decisions to be made for conflicting goals. For example safety and speed for an automated vehicle. The utility function will make a trade off by assigning different weightings to the goals. This will also take place for events with a degree of uncertainty. The utility function will make a trade off between the success of the event with the importance of the goal.

The agent types mentioned above have also been extended to form more complex agent architectures. An example of this is a type of rational agent referred to as a **Belief-Desire-Intention (BDI) agent** [19]. The name refers to the information, motivations and deliberative states of the agent, which determine its behaviour.

Beliefs refer to the environment state knowledge which the agent modelled internally. Desires represent desirable states or situations which the agent would like to accomplish, as well as the priorities of these states. Intentions refer to the desires which the agent has selected based on the state of the environment and what actions can be performed at that time. This selection decision is based on the condition at that time, which can possibly change. To moderate the frequency at which the intentions are decided on, a new decision is made only when significant changes take place. This prevents unnecessary altering of intentions due to minor changes, resulting in the intentions never being fulfilled.

Besides categorizing agents based on methods used by their agent programs to map percepts to actions, agents can also be categorized based on their mobility.

Mobile Agents are those agents which are able to independently move between and execute on different connected hosts in a networked environment. This is in contrast to **Static Agents** which lack mobility and are limited to a single host.

This form of processing is based on a remote programming paradigm where application code is distributed to remote hosts and executed [17]. This is unlike remote procedure calling where static application code housed on a network host is remotely invoked.

The anatomy of a mobile agent typically consists of application code together and persistent memory. Application code performs the required processing on the data provided by the host. The results of this processing is stored in the persistent memory.

Utilizing mobile agents in a network environment provides a number of benefits over using static agents which request remote data. Mobile agents allow for ongoing processing to take place without requiring constant communication. For example an agent can be sent to the host containing the data, and return only when processing is complete, rather than constantly requesting data from the host. This also benefits the amount of network overhead required because agents are often smaller in size than the total data to be processed.

Mobile agents do however face a challenge in that a hosting platform is required. The hosting platform must provide a number of requirements [20]. These include providing the run-time environment in which the agents can operate. The hosting environment must also provide a means of transporting agents to other hosts and negotiating the exchange of agents with these hosts. Persistence is another key requirement. This allows for hosts to freeze an executing agent before it is sent to another host. On arrival at a host, the hosting platform must thaw the agent to return it to its previous state where it can continue execution.

In the following section we discuss how multiple agents can be joined to compose a multi-agent system.

2.2.2 Multi-agent Systems

A Multi-agent System (MAS) is a network of loosely coupled agents which interact to solve problems too great to be solved by the individual agents within the network [21]. By operating as a single large unit, the capabilities of each of the many individual agents in the MAS are combined to form a powerful unit.

According to Sycara [21], a MAS can be characterized by:

- Each agent having incomplete information and capabilities for solving the problem
- No global control over the system
- Decentralized data
- Asynchronous communications

Each of the agents in the MAS is functionally specific to specialize in solving a particular problem aspect. This modular design provides a strong tool for handling complexity and extendibility [21, 17]. The structure of a MAS is also able to change and evolve based on the requirements. Specialized agents can also group to create subsets within the MAS.

Besides solving problems which are too large for single agents to solve, MASs provide a number of advantages for complex systems [21]. MASs allow for existing legacy agent systems to function together with new agents by forming a single MAS. Problems that can naturally be regarded as a society autonomously

interacting components or distributed collaborative problems can be effectively handled using MASs [21, 17]. Solutions which require the use of spatially distributed information sources or expertise are also well suited to MASs. MASs can also improve performance in terms of computational efficiency, reliability, robustness, maintainability, responsiveness flexibility and reuse.

MASs produce their behaviour through interactions which take place between the simple agents and the interactions between the agents and their environments. According to [17] these interactions develop due to actions having consequences which influence future behaviours of other actions. The global behaviour of the MAS is determined by the means in which the contained agents interact by communicating, collaborating and cooperating.

The most powerful aspect of a MAS is that its global behaviour can be greater than the sum of the individual capabilities and knowledge of each of the agents. This phenomena is referred to as emergence, and is considered by some as a core concept behind self-organization [22]. Emergence provides a great deal of power at an expense. Details regarding these short comings are provided in section 2.2.5.

Communication between agents requires that both agents have shared knowledge regarding the syntax, semantics and the means of exchanging messages [17]. These requirements can be easily achieved in homogeneous environments, but extremely challenging when agents are heterogeneous. Direct communication between agents is usually performed through well defined agent languages. Besides direct communication, messages between agents can also be communicated indirectly through stigmergy. Stigmergy is a method inspired by nature, by which communication takes place indirectly using the environment [23]. For example, communicating with someone who is following you by leaving bread crumbs for them to follow. This form of communication is particularly well suited for agents due to their strong bond with the environment in which they exist. A popular algorithm which utilizes the concept of stigmergy is Ant Colony Optimization (ACO) [23]. These systems mimic the way in which ants communicate through pheromone trails which are left by agents while searching for food. These trails act as messages left in the environment to let other ant know where the food is. These systems also express strong emergence in the way simple reactive ants are able to express complex behaviour such as determining optimal paths between the food source and their nest. An explanation of how ACO has been implemented and used for networking is presented in 2.2.3.

Blackboard Systems is another example of how stigmergy is used to facilitate communication between agents [24]. A blackboard system is based on the metaphor of group of specialists seated around a large blackboard. A problem which requires the specific skills of a number of specialists is presented on this blackboard. Each of the specialists observe the blackboard, waiting to apply their knowledge by recording their contribution on the blackboard. This turn taking process of incrementally solving the problem continues until the problem has been solved. In this metaphor, each of the specialists represent an agent and the blackboard represents the environment. Blackboard systems were designed as a means of dealing with ill-defined, complex applications [24].

Together with the communication between agents, the way in which agents collaborate and cooperate greatly influence the behaviour of the MAS. Controlling this collaboration is particularly challenging given the distributed nature of a MAS. Each of the agents within the MAS acts in a locally optimal way, which

is also required to to be mutually beneficial and globally optimal for the MAS. This act of agents grouping together to form communities which cooperate to achieve the goals of the individuals and of the system without a centralized controller is referred to as Distributed Artificial Intelligence (DAI) [25].

Stand alone agent architectures have been extended for DAI systems. An example of such a system is Belief-Desire-Joint Intention (BDJI) [25] which is an extension of the BDI agent discussed in 2.2.1.

Defining the intentions of individual agents in cases where global behaviour emerges is possible, however in most cases this is insufficient for group behaviour to be optimal. This is because joint actions are more than just the sum of the individual parts and there is a fundamental difference between individuals and groups. To overcome this, joint intentions are used which define the group commitment to be performed by the group as a unit while in a shared mental state. This shared mental state includes the common goal for the MAS, the want to collaborate, a common means of reaching their objective, acknowledgement that various other agents are related to the joint goal, a criteria for tracking the rationality of their actions and behaviour rules which define their local behaviour and behaviour to other agents.

Developing joint intention and shared mental state between the agents is a complex problem which requires communication between agents as well negotiation or persuasion. It must also be noted that a joint intention is the goal pursued by a group of agents as a unit and not the same single goal held by a number of agents in a MAS. For example a joint intention may require a number of unique intentions by the agents in that MAS.

The organization of a MAS largely impacts its global behaviour. This defines the structure of the MAS, patterns of information and control relations as well as the distribution of problem solving capabilities among agents [17, 21]. A number of possible approaches exist for organizing agents, a non-exhaustive list of these approaches is discussed in more detail below.

Hierarchy organizations exists whereby superior agents have authority for decision making and control over its subordinate agents. These subordinate agents can be superiors of agents at a lower levels. Interaction exists by orders being passed down to subordinates which can satisfy the requests or further delegate the requirements.

Organizations have also been designed which mimic social communities. These include community of experts, scientific community and Democracies. **Community of experts** is a flat organization typically used in blackboard systems. Each of the agents acts as a specialist in a particular part of the problem being solved. The agents then iteratively collaborate to solve the problem by each solving a subset of the problem. A **scientific community** operates by having agents construct full solutions locally which are then peer reviewed. Other agents can test, challenge or refine the solutions proposed. Democracies operate by having agents negotiate on solutions by voting.

Market based organizations operate by having agents which bid for tasks and resources. Interaction with the agents takes place through a price variable which values the service.

Evolution is based on the survival of the fittest agents in the group which then reproduce to form agents with traits from both parents. Evolution is an indirect organisational strategy in that it attempts to produce agents which are

best suited for the system requirements and remove those agents which are not performing. Besides individual agents within the MAS evolving, grouping of agents within the system can also evolve to produce better global performance overtime.

Subsumption is an agent organization and architecture by Brooks [26] which consists of a number of vertical layers which provide increasing levels of competence through more abstract goals. Each of the layers operates as an agent which perceives the environment and produces actions based on its perceptions. The higher levels of the system subsume the lower levels by suppressing their outputs. This interaction allows for lower levels to provide a reflex system which is active while attempting behaviours defined by higher layers. This provides a robust, bottom-up based design. The complexity of the behaviours produced by the system can also be increased by adding more complex layers which define more abstract goals. An example of a system which utilizes subsumption is detailed in section 2.2.3.

A number of example agent based systems which employ the before mentioned technologies are presented in the next section.

2.2.3 Examples of Agents based systems

In this subsection we examine a number of agent based systems. We start by looking at examples of control systems in the area of robotics followed by an example of an agent based WSN system. We discuss how agents are utilized within these systems as well as possible strengths or shortcomings of systems.

The first Agent based system we discuss is one of Brooks [27] layered control systems for mobile robots.

Tom and Jerry were two identical robots developed with the aim of demonstrating how a simple subsumption program can operate on basic hardware and produce complex behaviour.

The hardware for each Tom and Jerry consisted of a motorized toy car together with a single programmable logic chip and 4 one-bit infra-red proximity sensors. Three of the sensors were mounted to the front of the car, one facing straight and the other two facing slightly outwards. The fourth sensor was mounted to the rear of the toy car. Each of the sensors were tuned to trigger at a specific distance, with the central forward facing sensor set to detect objects much closer than the other sensors.

The subsumption program to control each of the robots consisted of 3 layers:

The **lowest layer** provided obstacle avoidance. This is performed by determining the direction of the obstacles using the two further reaching front sensors, and manoeuvring in the other direction. If an obstacle is detected by the short range front sensor, a halt reflex is triggered.

The **second layer** provides behaviour for random wandering. This operates by generating a random direction in which the robot is urged to move. This direction is periodically regenerated at an interval of approximately 10 seconds. When an obstacle avoidance reflex is triggered by the lower layer, the wandering direction is coupled with the avoidance direction. The halt reflex is uninhibited by this layer.

The **highest layer** creates following behaviour by detecting movement using input from the 3 front mounted sensors. When movement is detected the wandering behaviour of the lower layer is suppressed. The robot is attracted to move in the direction of the moving object. The lowest level reflexes prevent the robot from colliding with any objects, including the object it is chasing.

These robots demonstrate that a simple multi-agent system can produce seemingly intelligent behaviour. Complex interactions between the non-communicating robots emerges even though the robots do not explicitly know about each other or communicate directly. These robots also suggested that intelligent behaviour can be expressed without requiring any internal world model of their environment [28].

As with the robotic control systems, intelligent software agents have been utilized to overcome a number of challenges faced by current WSN systems. These include efficient handling, aggregation and routing of sensor data within a network. A number of example agent based WSN systems are detailed below.

Chen [29] proposed an architecture referred to as Mobile Agent based Wireless Sensor Network (MAWSN). The system aims to reduce network overhead through the reduction and aggregation of sensor data, thus decreasing power consumption. A reduction in the consumption of power intern extends the lifetime of the network.

The approach proposed involves MAs which are dispatched by the network sink to target multiple source nodes within an area of interest. The MA then traverses the source nodes collecting sensor data. This dispatching of MAs from the source is based on the assumptions that the sink knows which source nodes are of interest, the most optimal order in which to visit the source nodes and the routing required by the MA to reach the nodes in the required order.

The reduction and aggregation of data by the MAs takes place on three levels: In Node level by dynamically deploying application code on the MA. This allows for application to be transmitted to the source where the data is, rather than having to move the raw data to the sink to be processed. This application code processes the data and reduces its size. Deploying application code also allows for network application to change dynamically without requiring a redeployment of the network.

The second level is Task Level which eliminates spatial redundancy through data aggregation. This exploits the correlation between node position and similarity of data, in which closely positioned nodes are likely to return similar results. Because of this, data aggregation and fusion can be performed by the MA which reduces the total size of data being returned to the sink.

The final level is the Combined Task Level, in which data from multiple tasks is concatenated by the MA. This approaches groups the results from a number of tasks into a single large MA, rather than utilizing a number of smaller MAs. This reduces the network overhead at the expense of latency.

An evaluation of the system was performed by comparing simulation results of MAWSN and Client/Server based Wireless Sensor Network (CSWSN). The performance metrics used for the evaluation were energy consumption, average end-to-end delay, the trade off between energy usage and delay and packet delivery ratio. The number of targeted source nodes, reduction ration, size of processing code and size of sensed data were used as simulation parameters

The evaluations suggest that for the same number of target source nodes, MAWSN performs better than CSWSN in terms of energy consumption, packet delivery ratio and trade off between energy usage and delay. The average end-to-end delay of CSWSN was greater than MAWSN with less than 10 targeted source nodes as well as when the processing code of the MA was greater than 1.5 KB. In general the findings suggest that the performance benefit of MAWSN increases with the number of targeted source nodes.

However, the simulation parameters mentioned above were not based on real world WSN applications and only a single parameter was altered at a time. Because of this, the results presented may not be a true reflection of the systems performance in real world WSNs.

Agilla [30] is another example of a MA based system for WSNs. Agilla is a mobile agent based middleware designed for adaptive applications in WSNs.

The middleware facilitates the user of mobile agents which are able to traverse the network. The agents are able to migrate between nodes or clone themselves to create a duplicate instance on a neighbouring node. This migration can be weak, in which only the application code migrates, or strong, in which the application code as well as the current state of execution are migrating. A number of MAs can exist on the same host.

Communication between agents is facilitated through tuple space and an acquaintance list. Tuple space is a shared memory on each host which is accessible by local and remote agents. This allows for inter-agent communication in the same way as a Blackboard System mentioned above. The acquaintance list contains a list of all direct neighbours. Location based addressing is used to uniquely identify hosts, which together with the acquaintance list allow for complex multi-hop application to exist. An example of an Agilla application is fire tracking [31]. The fire tracker agent operates by discovering and forming a perimeter around the fire by simply checking which hosts border a fire, in which case they are marked in the tuple space. The agent is then cloned to neighbouring hosts to continue identifying and marking hosts bordering on the fire. This work proves that complex behaviour can arise from simple agents working as a unit.

Agent-based Directed Diffusion (abDD) is an approach proposed by Malik [32] to extend the lifetime of a WSN. This system proposes the utilization of mobile agents for Directed Diffusion.

The system aims to overcome the problem of over utilizing optimal low power consummation paths, resulting in the paths being energy depleted too quickly. Directed Diffusion is different from other systems in that it ensures that data routing is spread over the network as uniformly as possible.

The approach proposed involves initial low rate data flooding and gradual reinforcement of better paths. The scheme takes into consideration both the routing cost and remaining energy of a possible next hop. The mobile agents are then routed to a neighbouring node based on the ratio of remaining energy over the routing cost. The proposed system consists of both Stationery and Mobile Agents.

A stationery agent exists in every sensor node, which monitors all the activities for the nodes and updates it knowledge accordingly. It also maintains its own battery level, the battery level for all immediate neighbours and the

route cost to its neighbour. The interest cache stores data such as time the interest arrive and the gradient. A rule-based system governs the interest cache by determining how values are changed when events such as the arrival of an MA or the setting of a gradient occurs.

Mobile agents are autonomous agents which are able to migrate through the WSN with application specific code. They also have the ability to store aggregated sensory data which is collected at each of the sensor nodes. The abDD approach has three phases: Identifying the source nodes that satisfy the interest and building cost tables, distributing application code to the source nodes and Aggregating data among source nodes while returning it back to the sink.

An evaluation of the system was performed by comparing abDD with directed diffusion. The experiment results suggest that in terms of power consumption, the directed diffusion performed better during the initial stage of the experiment, but the proposed abDD performed better over time, with an increasing improvement over time. However, the results did not discuss the possible negative affect on the latency of the system given the power consumption trade off. The details regarding the nature of the sensor data and the application code of the MA were also not discussed. These could heavily influence the results of these experiments.

Another system aimed at prolonging the lifespan of WSNs through optimal path routing of Mobile Agents is proposed by Shakshuki [33].

This system aims to target source nodes based on interest queries rather than unique identifier or geographic location. The system must identify which source nodes to be targeted and determine the optimal path for a MA to traverse the network in order to collect and aggregate sensor data. This is achieved through a novel process which includes the use of Dijkstra's algorithm and Genetic Algorithms. The first step taken is the assignment of a task to the WSN sink by the application. Based on the application requirements, the sink then broadcasts an interest query to potential source nodes within the network. If a source node is able to satisfy the received interest query, it declares itself to the sink. This is done by sending exploratory data of its neighbouring nodes, which includes information regarding the link quality to neighbouring nodes.

The sink uses this exploratory information to determine a source-visiting sequence. The exploratory data is first converted into a complete weighted graph. This weighted graph contains all of the logical and physical links between all nodes with the associated link costs. Dijkstra's algorithm is then used to determine the shortest path between each of the nodes based on the line weight. As the shortest paths between all the pairs of nodes have to be calculated, Floyd Warshall's all pairs shortest path algorithm may be a better choice. This graph is then treated as a Travelling Salesman Problem (TSP) and a Genetic Algorithm (GA) is then used to approximate an optimal visiting sequence for the MA. The fitness function of the GA takes into consideration the total link cost for the suggested path.

Due to the dynamic nature of the WSN topology, the visiting sequence need be recalculated at a regular interval. The quality of the results depends on this interval. A trade-off exists where a smaller interval produces a better result however requires greater processing.

After the approximate optimal path has been determined, the furthest node

from the sink is selected as the first to be visited by the MA. This is based on the assumption that last node to be visited by the MA would be closest and this would mean the shortest trip for the MA containing all of the aggregated data. As the MA traverses each of the nodes in the source-visiting list, the sensor data is processed locally and then aggregated. After reaching and processing the final node in the list, the MA returns to the sink with the aggregated data.

Evaluations were performed to compare the proposed system to Directed Diffusion. Simulations were run with varying size WSNs using a random rectangular sensing field. Comparisons were made on the energy consumptions, latency and amounts of data being transported by through the network. When comparing energy consumption, the new approach performs 100% better than Directed Diffusion. In terms of latency, the new approach performs 100% better for less than 50 nodes and 200% better with an 80 node network.

The new approach also has substantially less data reported to the sink. This also lowers the energy requirements to transmit the data back to the sink.

A possible issue with this approach is the assumption that the sink has sufficient resources for the required GA computation. The TSP is known to be a NP-complete problem which requires a large amount of processing to solve. Such a problem also become exponentially more difficult as the number of nodes increases.

Besides WSNs, Agents have also been used in Mobile Ad-hoc networks. Baras [5] proposes a novel approach for routing in these dynamic environments through the use of swarm intelligence-inspired algorithms. The approach utilizes ant-like agents together with the inherent broadcast capability of wireless networks to discover and maintain paths in the network.

The aim of this work is to utilize ant-like agents which traverse the network to populate routing tables for optimal paths. This approach mimics the behaviour and stigmergic communication of real-world ants.

The algorithm operates by utilizing routing tables at each of the nodes which represent the probability of a data packet reaching a given destination via a given next hop. These routing tables are initialized using "Hello" packets which are broadcast by the sensor nodes to neighbouring nodes. Discovery information regarding neighbouring nodes is then used to populate the routing table entries with a probability of $1/n$ for each of the n possible next hops for the given destination.

Two types of MA are then used to reinforce the optimal paths in the network: Forward Ants and Backward Ants.

Forward Ants contain the addresses of the source and destination nodes, a sequence number, hop count and a stack. The stack contains the address of nodes which the forward ants have visited as well as the time of the visit. When a nodes routing table does not have an entry of a path to a required destination to which data must be sent, a forward ant is created. The node pushes its own address onto the forward ants stack together with the time it was created. This happens periodically until a route is identified.

When a node receives a forward ant, if the destination is not that of the receiving agent, it pushes its address and time onto the stack and the hop count is decreased by 1. Each of the forward ants in the network are identified by its source address and sequence number. If a node receives a forward ant with a sequence number less than a highest sequence of previous forward ants from

the same source, the ant is dropped. This eliminates any duplicate forward ants which have taken less optimal routes. Looping agents which visit the same node twice are also dropped.

When the forward ant arrives at its indented destination, the node extracts all the data is and the ant is deleted. This information is then used to create a Backward Ant. Backward ants use the information gathered by the forward ants to retrace the path in reverse and update the routing tables of the visited nodes.

A **Backward Ant** contains addresses of the destination and source nodes, which were the source and destination address of the forward agents respectively. It also contains the hop count and stack of the forward ant.

The backward ant travels in a unicast fashion back to the source. The ant is forwarded to each of the nodes on the stack, where a single address is popped off the stack. The backward ant then uses the address and timing information to update the routing table on the node. If routing table entries exist for the given destination, they are updated to increase the goodness of the route used by the ants.

Given these routing tables, data can then routed based on the highest hop probability next hop or routed probabilistically based on the next hop probabilities to prevent over utilization of the paths.

An assumption made by this system is that nodes can be uniquely addressed.

Based on previous Agent based systems, a number of benefits as well as shortcomings of such systems were identified. These are outlined in the sections below.

2.2.4 Benefits of Software agents

Intelligent Software agents can provide a number benefits over traditional modular or monolithic systems. Details of these benefits are provided below:

Emergence is the most powerful feature provided by software agents. This refers to the way in which the global behaviour of an agent based system can be greater than the sum of the individual capabilities and knowledge of each of the agents. This is unlike most other software system where the system where the performance of the entire system can be viewed as the sum of the parts.

This complex behaviour results from the interactions between potentially simple agents as well as their interactions with their environment. Such interactions can often result in intelligent behaviour as well as self-organisation.

Emergence does come at the expense of negatively impacting the effort required to design an agent based system. This is discussed in following section.

Scalability refers to the ability of a system to expand or contract with changing system requirements. Agent based systems allow for modifications to be made to the system through the addition or removal of agents to and from the system. New agents can be included to improve the performance of the system or old agents can be replaced with new and improved agents.

The loose coupling between agents allows for changes to be made to single agents without effecting other agents in the system. This allows for easy

scalability and no need for system components to be redeveloped due to system scaling, as can be the case with traditional tightly coupled modular systems.

Modularity of a large problem into a number of smaller problems is a common mechanism for dealing with complexity. Agent based systems allows for each part of the large problem to be treated individually, as a number of simpler problems. Software agents can be developed to each handle these smaller problems, distributing the larger complex problem between each other.

Besides distributing the problem between a number of agents within a single host, groups of agents can be distributed across multiple communicating hosts.

This modular approach is well suited to problems where a number of agents can work in parallel to solve a larger problem, such as network routing across a number of hosts.

These benefits suggest that software agents are suited for WSN frameworks and application.

2.2.5 Shortcomings of Software Agents

As well as the benefits of utilizing software agents provide, a number of shortcomings or challenges exist. Details of these are provided below:

System Design is the largest challenge when creating an Agent based system. This refers to the decomposition and allocation of work between agents for the problem being solved. The design of the system will determine what actions various agents will perform as well as how they will interact.

Two alternative design methodologies exist for creating agent based systems: top-down and bottom-up [34].

With a top-down design process, the global behaviour of the system is specified assuming that each component has global knowledge of the entire system. The solution is then decentralized with the addition communication constraints to replace perfect global knowledge. Top-down design is based on the hypothesis that the system can be decomposed into observation, state estimation and control in the following way. Information agents gather and distribute environmental information. This environmental information is then collected by modelling agents which produce an internal world model of the observed environment. Planning agents then use this internal world model together with possible action and control models and the current goals and intentions of the system to select which action to take.

In the bottom-up design process, the design starts by determining the requirements and capabilities of individual components. The global system behaviour emerges from the interaction among agents and between their environment. The individual agents are simple and do not rely on abstract representations, planning or higher order symbolic reasoning.

Both methodologies allow for simulation based analysis to determine global behaviour. In a top-down approach it is possible to determine set bounds on the systems behaviour. However, this is not possible for bottom-up design. Due to emergent behaviour, only the average system behaviour of bottom-up design can be determined. Because of this unpredictability, attempts must be made to better understand and control the interaction between these simple agents in order to better utilize their complex behaviour.

This is essential for developing system where simple agents need be added or removed from the overall system. Emergent behaviour is greater than the sum of the parts, making it difficult to predict the true results of removing or adding a single agent to an existing multi-agent system. This is also true for the removal of unwanted global behaviour. The link between the global behaviour and simple agents must be understood before the behaviour can be removed.

In this work we are concerned with the latter of the two design methodologies.

Agent Mobility is another challenge faced by mobile agent systems. In order for agents to traverse the network environment, a number of requirements must be met by the hosting nodes. Firstly an hosting node is required to provide the execution environment for a mobile agent when it is processing data on the given host. After execution, the agent must halt execution or store its state in a persistent way such that it can migrate to a new host. The agent host must then transmit the halted mobile agent to the new host. During this transmission, the integrity of the mobile agents execution code and data must be maintained as there is a great chance that the agent will be unable to execute correctly if the code has been corrupted. Hosting nodes must also have the ability to receive mobile agents from other hosts. These newly received agents must then be unhalted and return to their state prior to migrating. Recent advances in network technologies have also introduced the need for security thus the encryption of mobile agents before traversing the network. In such a case the hosting nodes must have the ability to encrypt and decrypt the MA before and after migration.

In the next chapter we present the aim of this research. A number of research objectives are then presented which need be satisfied in order to satisfy the research aim.

Chapter 3

Research Aim and Objectives

In this chapter we present the aim of our research. A number of objectives based on this aim are then presented and motivated. This chapter also aims to justify the importance of the research aim as well as present any assumptions or constraints on which this work is based.

Wireless Sensor Networks is a growing area of interest with an increasing number of applications in many different fields. Current WSN applications face a number of challenges, some of which are presented in section 2.1.3. The greatest of these challenges is that most WSN applications are developed either as single monolithic blocks or as a few tightly coupled modules. This lack of structure results in greater difficulty when designing, developing and testing the system. Monolithic systems also provide a poor means for handling complexity. A lack of reconfigurability and intelligence are also major issues faced by current WSN applications.

Due to this, the aim of this research is to design and develop a WSN framework to provide structure for communication efficiency and facilitate intelligence in WSN systems and their applications. From this aim, the following research objectives were identified:

3.1 Application Neutrality

A large number of possible WSN applications exist in a number of unique areas. These applications vary in how the environmental data is sensed, processed and communicated to other nodes in the network. Due to this diversity, it is not possible for a single WSN application to be developed which satisfies all possible application requirements. However, it is possible to create a framework which is compatible with all application types. Such a framework needs to contain those aspects which are common to all WSN applications as well as allow for the inclusion of application specific components for various applications. Providing a non-application specific framework presents a number of positives. These include wider use of the framework for more applications, potentially providing a WSN application standard. Application neutrality also allows for the WSNs' application type to be altered without requiring changes to the framework.

Because of this, the framework need be compatible with a number of WSN application types.

3.2 Reconfigurability

A number of situations exist in which reconfiguration or alteration of a WSN application is required or would be beneficial. Unforeseen application requirements at the time of deployment together with the dynamic nature of sensor environment can lead to a change in applications requirements. Reconfiguration could also be utilized to replace old or poor performing WSN application methods and feature with newly developed and improved methods.

In most of the current WSNs, the ability to reconfigure applications is extremely limited or in some cases not possible. In many cases, only simple remote reconfigurations can be performed to alter simple system parameters. Larger application changes often require manual redeployment of the entire application and cannot be performed remotely.

Given this, an objective of our framework is that remote reconfiguration of WSN applications must be possible without redeployment of the entire application. It should also be possible for additional modules or components to be remotely added. This goal is tightly coupled with application neutrality as reconfiguration should be possible across application types which using the same framework.

3.3 Facilitating Intelligence

Intelligence in terms of WSN applications refers to the manner in which sensor data is gathered, processed, communicated and presented by the WSN application. In order for an application to be considered intelligent, it need learn from its experiences and improve its performance overtime.

The intelligent gathering of sensor data involves identifying and utilizing the most optimal method of collecting data based on the required frequency and accuracy of the application. The data should then be processed as efficiently as possible based on the type of sensor data and the available resources. Communication of data within the network also considers the data and available resources together with the systems communication requirements. Presenting data in an intelligent way provides the user with only data which is relevant to their task as well as providing this data when it is needed.

A key attribute of an intelligent system is the ability to learn. This allows for the system to gain experience based on previous actions and the resulting reactions. The performance of an intelligent system therefore improves overtime. Because of this, the inclusion of intelligence within WSN applications can result in improved performance as well as better utilization of available resources. An intelligent WSN should therefore display intelligence by improving the quality of sensory data provided, either through removing unnecessary data or including more relevant data, improving the processing time or improving the performance of network tasks, such as routing.

3.4 Efficiency

The final critical requirement of the framework is that of efficiency. Frameworks typically result in additional system overhead at the expense of additional structure. Latency can also be increased due to the additional framework components. In smaller applications, this trade off is too great. However in larger applications which require structure and reconfiguration, a framework may be greatly beneficial.

Efficiency and overhead is considered a critical requirement due to the extremely limited resources available on WSN devices.

Because of this, research must be done to identify the expected efficiency of the WSN framework and various applications. This allows for identifying those application types which are better suited for using within the framework, rather than being stand-alone applications. This should also provide details regarding the scalability of the system.

In the following chapter we present our proposed approach to satisfying the above mentioned research aim and required objectives.

Chapter 4

Agent based layered framework

In this chapter we present the approach used to satisfy our research aim. We will start by identifying the technologies used to develop our proposed solution. Details of how these technologies were utilized and how they operate within the solution are then presented. We conclude this chapter by detailing the interaction between the various components in the proposed system.

In order to satisfy the research aims presented in chapter 3, we propose a novel WSN framework which utilized layered architecture together with intelligent agents. This is an extension of our early research [35]

A layered architecture will be utilized to provide the control system structure for each of the WSN nodes. Intelligent agent will be housed within these layers to provide intelligence. Each of these technologies are discussed in more detail below.

4.1 Layered Architecture

A layered system is based on the layered architectural pattern. This pattern is used in a context where a large system requires decomposition. Decomposition is considered better practice than implementing a protocol as a single monolithic block [36].

The layered Architectural pattern operates by decomposing a large system into a number of loosely coupled components across multiple levels of abstraction. This allows for conceptually different issues to be handled independently in different layers.

These components are then arranged vertically in layers such that layer n provides services to layer $n + 1$ and delegates tasks to layer $n - 1$. Often a request from layer n can result in a number of requests in layer $n - 1$. This is due to higher layers being more abstracted and utilizing a number of primitive services in the layers below [36]. In general the highest layer in the stack is the most abstracted, providing overall function of the system.

The processing of data within such a system is performed by passing the data between the layers in a top-down or bottom-up direction. Each layer acts as a subsystem which performs a specific operation on the data it receives. A

typical flow consists of requests being passed from a high level down to a lower level. The higher level requests a service from the level below, which in turn can request services from the level below it. Communication in the opposite direction takes place using events or notifications which are sent to the layers above. The layers are only aware of directly neighbouring layers. A black-box approach is also taken when specifying interfaces between layers. This ensures that no reliance exists between the layers and they are loosely coupled [36].

The layered architecture pattern has been used in a number of real world applications. This includes multi-tier Client-Server applications. These systems are decomposed to separate the presentation of data from the retrieval and processing. A two-tier Client-Server consists of a data source and presentation, where as the three-tier contains a data source, domain logic and presentation layer. The domain logic tier processes the data prior to presenting it to the user. Depending on the complexity of the data, this approach may be extended to include a number of logic layers. Another popular example of a more complex multi-layered system is the TCP/IP protocol, which consists of 5 layers to handle network data from hardware to presentation [37].

The decomposition of a large system into a number of smaller subsystems provides a number of benefits [36, 38]. A major benefit is the reduction of complexity by allowing for various layers to be independently developed and tested, given clearly defined interfaces. This also allows for dependencies to be kept local. Standardization of layers interfaces can also facilitate the reuse of layers in different systems. The largest benefit is that loosely coupled layers allow for modification to be made to the system at a later stage [39]. This later modification of the stack allows for improved technologies and layers to replace deprecated layers within the stack. New layers can also be added later to accommodate for new system requirements.

However, layered architectures express a number of negative aspects. The initial design and decomposition of the large system into a number of decoupled layers is often more complex than designing a large monolithic system. Determining the level of granularity for each layer also presents a great challenge [36]. The greatest of the issues is that of the resource overhead required for a layered system. The performance of such a system can be negatively impacted due to the multiple layers being loosely coupled and having to communicate with each other [38].

Layered architecture overcomes the issues of reconfiguration and structure for the system, however the facilitation of intelligence is still required. This is incorporated through the utilization of multi-agent systems within the layers, to perform processing on the sensor data. The multi-agent approach taken is that of a Blackboard System (Section 2.2.2) within each of the layers of the stack. Details of the interactions between stack layers and their contained blackboard systems will be presented in detail in section 4.4.3.

By utilizing a layered architecture, structure is incorporated is added to our proposed system. Issues of reconfiguration are addressed by the modular and loosely coupled nature of the layers. Application Neutrality is also provided as the layered framework acts as only a container for the application logic of the system. No application processing is directly performed by the architecture. In order for processing to take place, application logic must be housed within each of the layers. This processing is provided by agent based systems.

4.2 Intelligent Agents

In order to provide for a layered system to provide intelligent behaviour, intelligent processing needs to take place within each of the layers of the system. To achieve this, Agent based systems were identified as a possible solution. Details of such systems are provided in section 2.2.

Blackboard systems were identified as a feasible solution to providing intelligent processing. Blackboard systems will exist within each of the system layers and process data arriving at that layer.

The motivations for selecting intelligent agents and blackboard systems in particular is that they are well suited for ill defined problems such as those faced by dynamic WSN applications. Blackboard systems also allow for easy reconfiguration by removing or adding new blackboard agents.

4.3 Design process

The design process followed for the proposed system was to identify the systems requirements based on a typical WSN application. This was done to identify what functionality the framework would have to accommodate. The typical WSN used was a general application able to execute mobile application agents. These MAs contain the application specific logic which is executed by the WSN nodes.

A modularized layered version of the application was then designed. This layered version of the application was created though a number of design iterations. The design iterations alternated between a bottom-up and top-down design processes. This was performed as a means of satisfying the systems requirements while still allowing the emergence of complex behaviours. The bottom-up design process focused on constructing the design from a number of incrementally complex layers, starting with a base hardware layer. This approach facilitated the possibility of emergent behaviour. The top-down design iterations decomposed the system, with the final aim in mind, into a number of modular layers. The highest layer was designed followed by the design of lower layers which provide incrementally more primitive services to the layers above.

The processing performed by each layers would be further decomposed to form a multi-agent system, through a similar design process.

Initial designs were based on existing layered systems for networking. For example the TCP/IP stack. The resulting layered WSNs application is presented in detail in the next section.

4.3.1 Typical Wireless Sensor Network Application

The application designed to be representative of a typical WSNs application consisted of 5 layers: Logical Link Layer, Network Layer, Transport Layer, Wrapper Layer and Execution Layer. Each of the layers host and execute various agents. Figure 4.1 depicts the layered system. The middle column of the figure depicts the proposed layered structure and agents. The left column depicts the inter-layer communication units, and the internal state and functions of each layer is given in the right column.

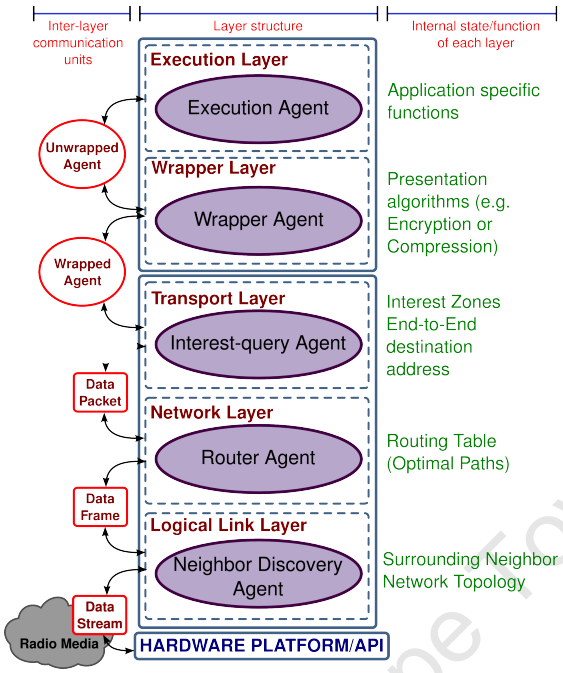

Figure 4.1: A Layered Wireless Sensor Network application

It must be noted that the number of layers can vary between the nodes of the same WSNs depending on the capability or requirements for each individual node. An example of this would be the exclusion of Wrapper and Execution layers from those nodes which mobile agents will not be executing on.

Each of the 5 stack layers mentioned above are discussed in detail below. The included figures represent the expected input, the agents within and the output for each of the layers.

Execution Layer

The function of the execution layer is to execute the mobile application agent as illustrated in figure 4.2. This layer accepts a mobile application agent as input which is then housed by the execution agent. The execution agent provides the run-time environment for the mobile application agent. It also ensures that the mobile application agent is provided with the required sensor and node specific data.

After the mobile application agent has completed its execution, the execution agent shuts it down. The output from this layer is a mobile application agent with aggregated data. Because this is the highest level of the system, no services are provided to higher layers.

This layer interfaces with the Wrapper Layer directly below it. The wrapper layer below provides an unwrapped mobile application agent.

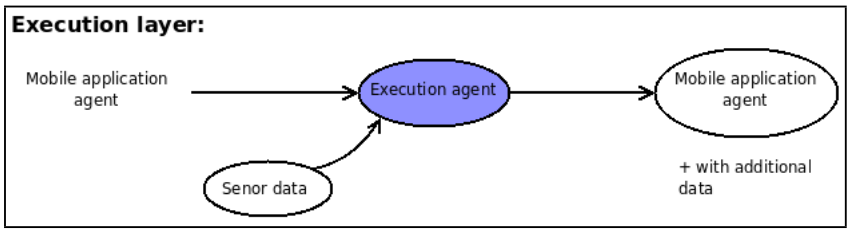

Figure 4.2: Execution Layer

Wrapper Layer

The function of this layer is to unwrap the incoming mobile agent packets. These packets are referred to as mobile agent packets or wrapped mobile agents.

The mobile agent packets which are transmitted through out the network are wrapped and require unwrapping before execution can take place. The wrapping could consist of compression and/or encryption. Figure 4.3 depicts this process.

In order to unwrap the mobile agent packet, a wrapper agent must determine how it is wrapped. After this the wrapper agents send the packet to various other agents in this layer for unwrapping. These other agents could consist of decompression as well as decryption.

Interaction between this layer and the layer above takes place by providing the above layer with a mobile application agent.

The input the layer receives from lower levels consists of mobile agent packets. At this stage, the packet has been routed to the correct node in the network. This layer can consist of possible sub-systems, namely decompression and encryption agents.

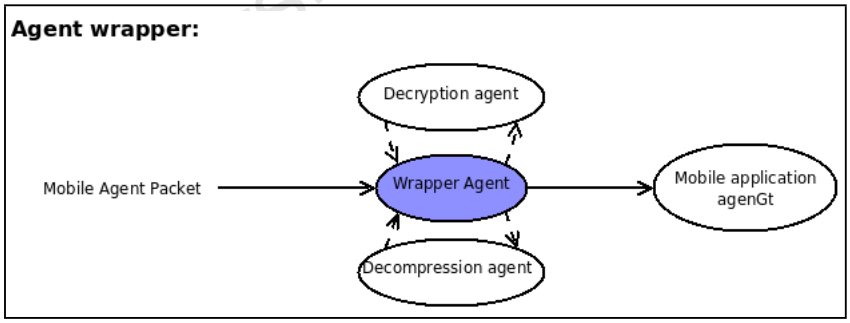

Figure 4.3: Wrapper layer

Transport Layer

The function of the Transport Layer is to ensure that the mobile application agents are executed on the correct network nodes. The layer inputs a mobile agent packet, which has an associate interest query. This interest query is examined by the interest query agent which determines if the hosting network node can satisfy the query. If the interest can not be satisfied, the agent packet

is not process. If the interest query can be satisfied, the mobile agent packet is en-queued to be processed by the layer above. Figure 4.4 illustrates the abstract of this process.

The output provided by this layer is a queue of all agent packets which are required to be executed on the host node.

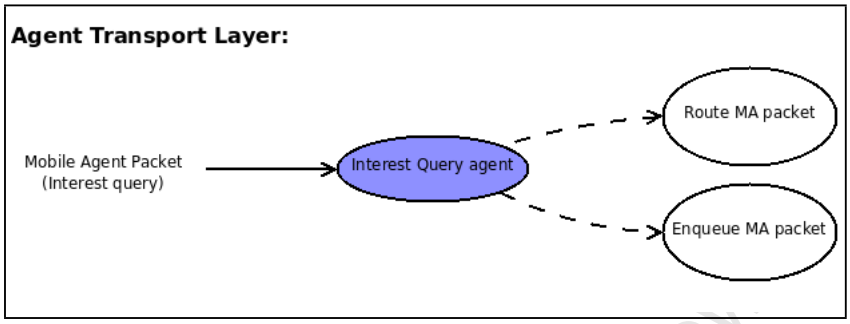

Figure 4.4: Transport Layer

Network Layer

The aim of the network layer is to produce optimal routing path information for the network topology. The network layer accepts information from its surrounding nodes and determines the optimal paths between various nodes. A router agent inputs the neighbourhood information, such as the distance estimates and quality of service of the links, location information and neighbouring node id's. This information is processed by the router agent, together with a multi-agent based reinforcement learning system, to determine the optimal network paths. This multi-agent subsystem works constantly to update and improve the routing information for the network. The input received from the lower levels consists of neighbourhood information, such as neighbouring node id's, distance estimates, link quality of service, and possibly location data if anchor nodes or GPS data is available. The output provided by this layer consists of routing tables containing optimal paths for the network. The interaction of these components is presented in figure 4.5.

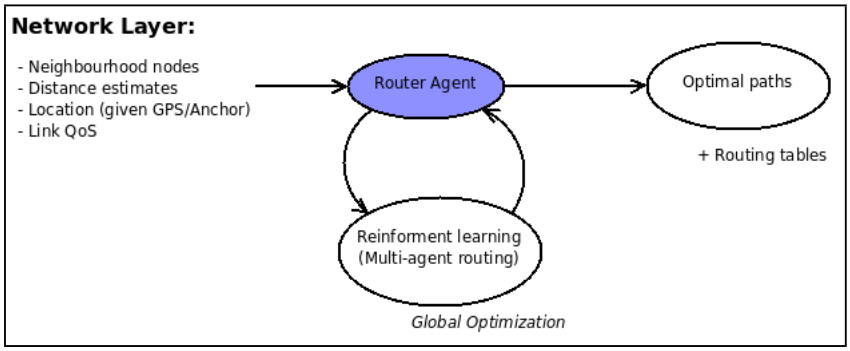

Figure 4.5: Network Layer

Logical Link Control Layer

The Logical Link Control Layer is the lowest level of the system. It acts as an interface between the Network Layer and hardware Application Programming Interface (API).

The function of this layer is to provide information on the immediate surroundings and local topology of a network node. This information includes the id's of the surrounding nodes, distance estimates, link quality, quality of service and possibly positional information.

Due to this being the lowest level of the system, no information is provided as input from other layers. The information is provided by radio emission, residential energy levels and query packets passed to a neighbourhood discovery agent. This agent then determines a possible local topology as well as location estimates. This is shown in figure 4.6.

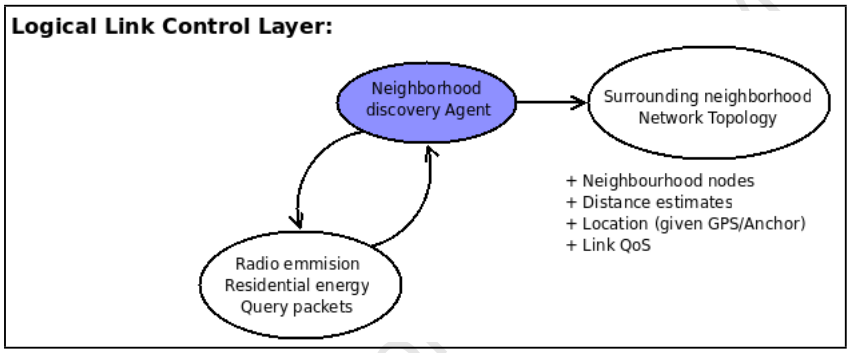

Figure 4.6: Logical Link Control Layer

Based on the layers of this typical WSNs application, the components and functional requirements of the framework were identified. These are discussed in detail in the next section.

4.4 Generalized System

In order to develop a WSNs application framework which satisfies the aims mentioned in section 3, two types of system components are required. Architectural components, which house the application components and Agent components, which facilitate intelligence within the system.

The relationship between these components is illustrated in figure 4.7.

Each of the system components is described below followed by the relationship between these components.

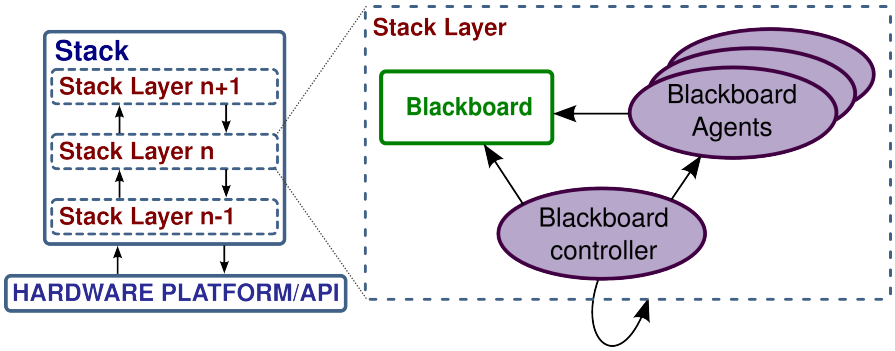

Figure 4.7: System Components

4.4.1 Architectural Components

These components act as the foundation for the layered architecture in which the WSNs application components are housed. Structure and modularity is provided through these components and no application related processing is performed. These components consist of a protocol stack which contains a number of linked protocol layers, as seen on the left side of figure 4.7. Each of these components are detailed below.

Protocol Stack

The protocol stack is the largest component of the framework. It acts as container in which all other system components are housed and controlled. A single instance of the stack resides within each of the hosting sensors nodes of the WSN. However, the contents of the stack may vary between different nodes within the same network as these contents depend on the requirements of that node. Nodes which just act as communication channels will have small stacks than nodes which process sensor data.

The purpose of the protocol stack is to provide protocol layers with a container in which they can operate and communicate. In order to achieve this, the stack handles the dynamic addition and removal of protocol layers. The linking between layers to facilitate communication is also handled by the stack. As the stack acts only as a container, no application processing is performed.

Protocol Layer

The Protocol layers are the building block of the layered architecture. A number of protocol layer instances exist within a single protocol stack. This number can vary between stacks in the same network. Each layer, as with the protocol stack, act as a container for other components and does not perform any application processing.

The purposed of the protocol layers is to provide a logical container for processing components. These layers in turn work together to perform the required processing. Each of the layers within the stack contains processing components to perform tasks at different level of abstraction, depending on the layers position within the stack. The protocol layer position at the top of the

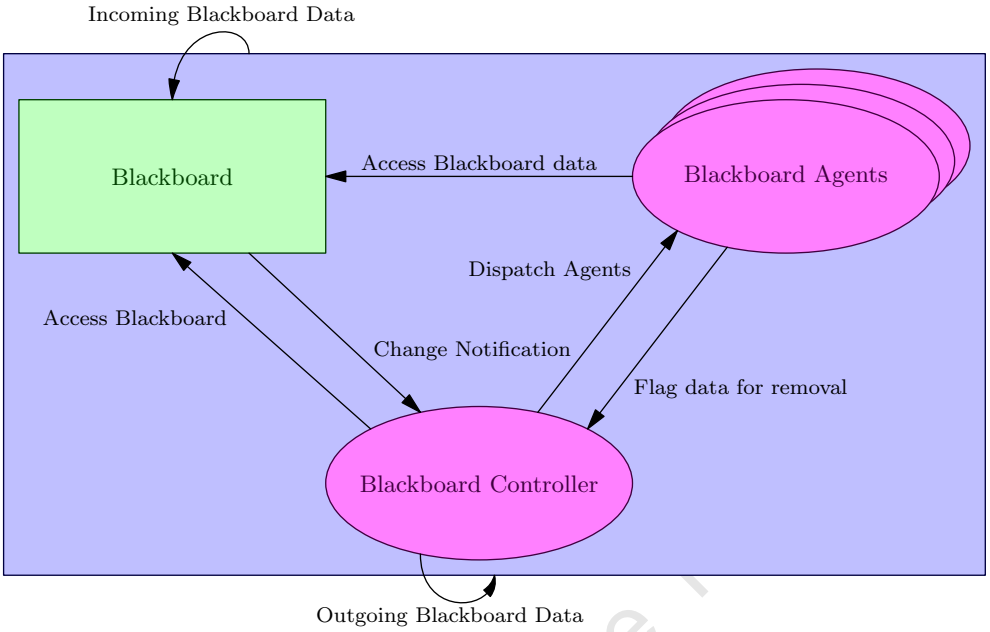

Figure 4.8: Agent component interaction

protocol stack contains components to perform the overall system goal, which is the most abstract task. The layers below provide the required services to the layers above. This decomposition created by the layers allows for logical grouping of components.

4.4.2 Agent Components

These components provide the processing mechanisms for the WSN application. They are housed within each of the Protocol layers of the framework. Unlike the before mentioned architectural components, Agent Components operate on or process the sensor data. These Agent components are based on a Blackboard system as mentioned in Section 2.2.2. For each framework instance, a single blackboard system exists within each of the protocol layers.

The Blackboard System is composed of three components: A Blackboard, Blackboard Agents and a Blackboard controller. This is relationship of these components is depicted in figure 4.8 and explained in detail in section 4.4.3. Each of the components is detailed below:

Blackboard

The blackboard can be viewed as the agent environment which exists within each of the blackboard systems. This environment is observed by and provides a work area for the various blackboard agents.

New unprocessed data which requires processing as well as partially processed data which requires further processing are both contained within the blackboard.This data could require processing by a single agent or a number

of agents. The blackboard is not application or data specific, allowing various types of data to be contained for various different application requirements.

The role of the blackboard within the system is that of a shared data structure which provides a number of blackboard agents with access to unprocessed data. Data within the blackboard can be accessed, modified and removed. Other information regarding the contents of the blackboard, such as how many data entries exist and if the contents of the blackboard have recently been modified, is also provided.

The processing of data within the blackboard is performed by a number of Blackboard Agent. Details of this are provide in the section below.

Blackboard Agent

The Blackboard Agent is a specialized form of agent which acts similarly to a S-R agent. It observes the Blackboard for a stimulus. If this stimulus is perceived, the data is processed by the Blackboard Agent.

A number of Blackboard Agents can exist within a single Blackboard System, creating a MAS. All the Blackboard Agents within the MAS share a common Blackboard.

Each of the Blackboard Agents within the system provide an application specific function. Processing of the Blackboard data begins by each of the agents inspecting the data elements within the Blackboard. If the Blackboard Agent identifies the data element as being of interest, the data element is processed by the agent. This interest is based on its specific application function of the agent. If the data element is not identified as being of interest, it is left unprocessed. This method of agent based data processing was designed as a means of dealing with ill-defined and complex situations [24].

In terms of their logic, each of the Blackboard Agents has only two core methods. The first is an execution condition method, which defines the stimulus to trigger the agents response. The second is an execution operation method, which defines the agents response to the data. This is utilized to perform the processing on data which has satisfied the execution condition. This simple design and logic makes for the simple and efficient implementation of such agents.

In order to prevent strong coupling between the various Blackboard Agents, no direct lines of communication exist between the various Blackboard Agents. All inter-agent communication is performed through stigmergy, using their shared environment.

To allow for control of the interactions between the Blackboard Agents and Blackboard, a controller is required. This controller, referred to as a Blackboard Controller, is disused in detail in the following section.

Blackboard Controller

The Blackboard Controller is an agent which controls the interactions between the Blackboard Agents and the Blackboard. This is the most complex of the agent based components.

Like the Blackboard, only a single Blackboard Controller exists within each Blackboard System. However, it can not be modified and is not application specific. This allows for the same controller to be used for all applications.

The responsibility of the Blackboard Controller is the management and control of Blackboard Agents within the Blackboard system. This involves handling new incoming data which is added to the Blackboard, controlling the processing of the data and removing processed elements from the blackboard.

New incoming data is added directly to the Blackboard by the component which contains the Blackboard System. The Blackboard controller is then notified that the contents of the Blackboard have been modified. This change notification triggers the Blackboard to start processing the Blackboards contents.

Processing of the data is performed by the Controller dispatching the Blackboard Agents to inspect and possibly operate on the Blackboard data. The controller iterates through each of the Blackboard elements and each of the Blackboard agents. The index of each Blackboard Element is passed to each of the Blackboard Agents, one at a time. This ensures that each element is investigated and potentially operated on by each each of the Blackboard Agents. The controller is passed the result of this method. If the execution condition is true, the Blackboard Controller instructs the agent to operate on the given Blackboard element. This execution process can also result in the Blackboard element being flagged as *ready for delete* or *processing complete*. This flagging allows the Blackboard Controller to prevent unnecessary or conflicting processing to be performed by the Blackboard Agents. Blackboard elements which could not be processed by any of the agents are marked as processing complete as none of the Agents have the ability to operate on them.

After all of the Blackboard Elements have been checked and possibly processed by all of the Blackboard Agents, those elements which were flagged as being ready for delete are removed.

Those elements which have been flagged as having processing completed are also removed from the Blackboard. These elements could be passed to another Blackboard System for processing.

The final action performed by the Controller is to check if the contents of the Blackboard has changed during or as a result of the processing. If the contents have changed since the beginning of the processing, the process is started again. This ensures that all the data is processed as much as possible by the Blackboard agents.

As described above, architecture components provide system structure and agent components provide the processing and application logic. This allows for multiple Blackboard Systems, of different levels of abstraction, to exist within multiple Protocol Layers.

In the following section we explain the interaction between the Protocol Stack and its contained Blackboard Systems.

4.4.3 Component Interaction

The interaction between architectural and agent components consists of data entering the stack, where it is then passed between a number of Protocol Layers containing Blackboard Systems. These Blackboard Systems perform the required processing on the incoming data. Details regarding data entering the stack, the handling of the data and how the components are reconfigured is provided below.

Data provided to the Protocol Stack consists of raw sensor data gather locally or Blackboard data from other sensor nodes. This data is gathered by a Hardware layer. The data is then passed to the lowest Stack Layer where it is processed by the contained agents. After the processing is complete, the data is passed by the Blackboard Controller to the next layer in the stack.

This process of the layer accepting data, the contained Blackboard System processing the data and the data being passed to the next layer, is performed a number of times within the stack. Data typically percolates up the stack to the highest stack layer, being processed by each of the layers within the stack. At this point the direction of the data is changed to allow for the data to head back down through the layers again. A flag variable indicates the direction in which the packet is moving within the stack. This direction determines where a Blackboard Controller will send the Blackboard element as well as what processing is performed within the blackboard. In some cases the data is not required to reach the top of the stack, in which case it can be turned around or deleted by the Blackboard.

As we are proposing a distributed system, communication of data between nodes is required. In order to facilitate this, communication between various protocol stacks on various nodes must be possible. This is made possible by including a node address or interest definition to describe the node on which the Blackboard element is required reside. This address or interest description are defined and modified by the Blackboard Systems. An example of this is a routing table Blackboard Agent. Transmission of elements is performed by passing data down the stack to the hardware layer.

A key requirement of the proposed framework is reconfiguration. Reconfiguration can be performed on an architectural level by modifying the stack layers or on an agent level by modifying the blackboard agents. As agents are contained within the layers, modification of the stack layer impacts the agents which it contains. Reconfiguration is controlled using a Base Station node. This is a specialized form of node which allows for direct interfacing with other devices, such as desktop computers.

Additional Blackboard agents can be added to protocol layers on remote nodes by sending admin flagged Protocol Packets from the Base Station node which contain the agents code. This is done by creating a specialized protocol packet. The payload of this packet is set to contain the application code of the blackboard agent. This packet is then flagged as an admin packet and the addresses of the remote well as the layer address are set. Processing of the packet is performed by adding the protocol packet to the top of the base station's stack, where it is passed down the stack and processed. The packet is then sent via the base stations hardware layer to the required remote sensor node. When the packet arrives at the required node, it is handled in the same way as other protocol packets, by percolating up the layers of the stack. At the top of the stack the packet is handled by an admin Blackboard agent. This blackboard agent identifies such packets based on their admin flag. After new packets have been identified as admin packets, the admin Blackboard Agent creates a new Blackboard agent from the contained application code and adds the new agent to the required layer.

As the addition and removal of Protocol Layers does not require any new application logic to be included to the Protocol Stack, they do not require admin packets to be created. These operations are performed by Mobile Agents which

are sent to the remote sensor nodes. The MAs contain application code which is executed by Execution Blackboard agents in the highest layer of the remote stacks. The execution of the code on the remote stack allows new layers to be created as well as existing layers and Blackboard agents to be removed from the stack. These MAs are also used for the remote execution of application code on remote nodes.

Implementation details of these components is provided in the next chapter.

Chapter 5

System implementation

In this chapter we detail the implementation of the proposed system. The platform on which the system is implemented is discussed and details provide. The Implementation details of the various components is then presented. These include the framework components which were implemented. The final section of this chapter provides details of the implementation of the application specific components.

5.1 Platform

The hardware platform selected for this work is the Sun Small Programmable Object Technology, referred to a Sun SPOT [40]. Sun SPOTs are a WSN platform which follows the 802.15.4 standard. An image of a Sun SPOT node can be seen in figure 5.1. These nodes contain a 180 MHz 32 bit processor together with 512K RAM and 4 megabyte flash storage. Communication between nodes is achieved through a 2.4 GHz IEEE 802.15.4 radio. These nodes contain a wide range of sensors including a three-axis accelerometer, temperature sensor, light sensor and 2 momentary switches. A number of hardware inputs allow for additional sensors to be added to the node. 8 tricolour LEDs positioned on the front of the node can also be actuated. The nodes are powered by an on board rechargeable battery [41].

The nodes do not have any operating system and use Java 2 Platform, Micro Edition (J2ME) running on a Squawk Java Virtual Machine (JVM). The Sun SPOT code and the Squawk JVM on which it runs are open source.

The reason this platform was chosen is the platform, being Java based, is well suite for development which is more rapid and for proof-of-concept purposes. The hardware was also readily available at the time that this research was conducted. Deployment of code to the Sun SPOTs was performed using Over The Air (OTA) deployment. Details of the deployment steps are provided in appendix A.

Sun SPOT API 5.0 was used for the development of the system.

Figure 5.1: A Java Sun SPOT sensor node

5.2 Implementation details

In this section we detail how the Java Sun SPOT platform was used to implement a proof-of-concept prototype of the system proposed in chapter 4. This system was implemented to allow for empirical evaluations and experimentation to be performed in order to evaluate its scalability and performance. The stack components are detailed first, followed by details regarding the software agents.

5.2.1 Stack Components

A class diagram illustrating the functions performed by each of the stack stack components is show in figure 5.2.

The largest component of the stack based system is the protocol stack itself. The protocol stack contains all the other stack components, besides the hardware layer. The Protocol stack is implemented to contain a dynamic list of all contained protocol layers. Methods exist within the protocol stack to allow for layers to be added to or removed from this list of layers. Each layer is identified using its position with the list of layers as a key. This position can change if new layers are added or old layer removed. Retrieval of information regarding the various layers within the stack are provided by the protocol stack. Addition of blackboard agents to layers within the stack, as well as the addition of protocol packets to be processed by the stack is achieved using the protocol stack. The final methods provided by the protocol stack are those to pair the protocol stack with the hardware layer, in order to provide access to the underlying hardware, sensors and actuators.

Protocol Layers are linked together to form the internals of the protocol stack. To achieve this, each protocol layer contains a handle to the layer above and below itself. If other protocol layers do not exist, the handle refers back to itself. Methods are provided to link protocol layers to those layer above and below, these methods are called by the protocol stack when new layers are added. The handles are also rearranged in the event of a layer removal. Each layer has access to its parent protocol stack in order to provide its contained blackboard components access. This access is required for agents to gather information

Figure 5.2: Stack Components class diagram

or manipulate the protocol stack. Besides forming a chain of protocol layers, the layers each act as the container for a blackboard systems. Details of these components are provided in the subsection below.

The unit of data which is passed between protocol stacks and the protocol layers within protocol stacks is the protocol packet. The packets contain network information such as the destination, next hop and port to use for the next hop, all of which have mutator methods. Layer communication information is also contained for traversing within a single protocol stack. This consists of a flag to indicate if the packet is moving up or down the stack, a flag to indicate if the packet have been process and a flag to indicate if the packet is ready for deletion. These variables allow for the packet to be processed and handled by the blackboard components. Payload information within the packet consists of a generic payload, a flag to indicate if the payload is for administrative purposes and an index indicating to which protocol layer the administrative task must be fulfilled. All of these variables are mutable through methods provided by the protocol packet. Methods also exist to read and write persistent memory which is stored between layers as well as different network nodes. Details of how this data is serialized is provided below.

The final high level stack component is the hardware layer. This component is based on a hardware interface in order to allow for the stack to potentially operate on a number of platforms other than the Java Sun SPOT. The hardware layer is the only hardware specific component of the system and provides methods to access the underlying hardware of the sensor nodes. These methods include accessing sensor data as well as actuating any possible actuators, such as LEDs on the Sun SPOTs. Sending and receiving of data between network nodes is achieved through network sender and listener classes, which are controlled by the hardware layer. With the Sun SPOTs, sending of data is achieved through radiogram connections. Data is serialized to form byte arrays and then sent to to nodes with the given address, or broadcast. Receiving of data is achieved through a listener thread which listens for datagrams on a radiogram connection. When data is received by the listener thread, the controlling hardware layer is notified, and the new data is passed to the protocol stack where it is processed.

Due to restrictions with the version of J2ME used by the Sun SPOTs, object serialization is not immediately possible. In order to incorporate the feature, KSN serialization [42] was included to the hardware layer. This library allows for Java style serialization to be performed on the Sun SPOTs platform.

5.2.2 Agents

As mentioned in chapter 4, all processing of data is performed by the software agents. A class diagram depicting the functions of each of these components as well as their relationships with each other is shown in figure 5.3.

The first agent discussed is the **blackboard agent**. These agents exist within the protocol layers of the system and form part of the blackboard systems contained within each of the protocol layers. Details of the interaction of these components presented in section 4.4.3. These blackboard agents are implemented to contain a handle to their blackboard, blackboard controller and a list of all other blackboard agents within the protocol layer. Each blackboard agent contains a method to determine if the agent execution condition has been

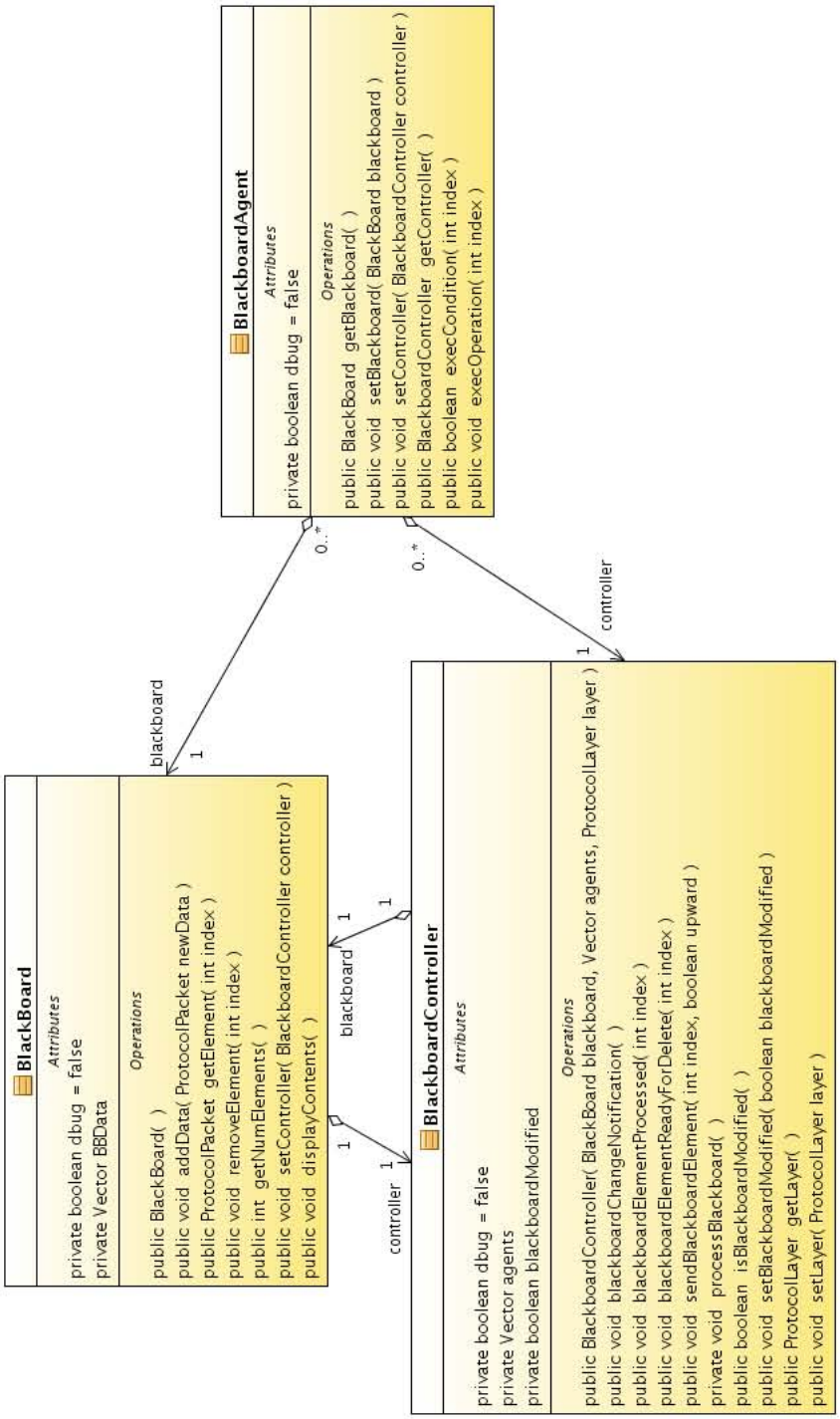

Figure 5.3: Agent Components class diagram

satisfied as well as the execution operation which is performed on the blackboard data.

Another restriction faced with J2ME is that of object reflection. This prevents Java objects from being remotely deployed and instantiated on sensor nodes which have not been compiled with the class definition of the object. In order to overcome this, **FScript blackboard agents** were created as extensions of the blackboard agent. These agents contain scripted logic which is executed by an interpreter within the protocol layer. FScript blackboard agents contain all blackboard agent components with the addition of a string of code and an FScript executor specific to the blackboard agents instructions. This code contains the methods to determine if the blackboard agents execution condition has been met and what operation should be performed on the given protocol packet.

The **FScript executer** class is an extension of a third party class, call FScriptME [43]. FScriptME is a scripting language is a lightweight language and interpreter specifically designed for J2ME environments. The extensions made for our implementation allowed for persistent memory to be used by the scripts. This is achieved by including an object stack which can be read from and written to by the scripts and is serialized when the scripts container object is serialized.

Besides the blackboard agents and FScript blackboard agents, the blackboard system also consists of a blackboard and a blackboard controller. All the of these components are housed within each of the above detailed protocol layer.

The **blackboard** is implemented as a shared vector of data which multiple blackboard agents can read from and write to. Synchronized methods are provided to allow for protocol packets to be accessed or added to the blackboard. Protocol packets within the blackboard system are addressed using their index within the blackboard. The blackboard also provides methods to get the number of elements currently in the blackboard as well as display the blackboard contents. Each blackboard has a handle to its controller in order to notify it about changes which have been made to the blackboards contents.

Blackboard Controllers are the most complex of the blackboard components. They are implemented to have pointers to the protocol layer in which the controller is housed, the blackboard which is being observed and the list of blackboard agents which it is controlling. A flag variable is also maintained to indicate when the contents of the blackboard have been modified. This allows for processing to be performed until the contents reach a stable state, which is usually when all the contained protocol packets have been removed. This change flag is controlled using a callback which is called by the processing blackboard agents. The processing blackboard agents are also able to call methods to flag a specific blackboard element for deletion or to indicate that processing has been completed.

Processing the blackboard is controlled by this controller. The contents of the blackboard are iterated through. For each element, the blackboard agents are iterated through. Each blackboard element which is not processed is then passed to each of the blackboard agents to determine if that agent is able to operate on the given blackboard element. If it is possible, the execution operation

of the agent is called. After the processing, the set processed flag for each of the elements which were not modified by any blackboard agents are set. Each of the blackboard elements are then iterated through and their ready for delete flag is checked, if it is, the element is removed. A final iteration of the blackboard elements is performed to move those elements which have been processed to the next protocol layer in the stack. This is handled by the controller, and the direction in which the packet moves is based on the protocol packets direction flag.

Mobile Agents, like Blackboard agents, are mobile with persistent state and thus extend the KSN Serialisable interface[42]. Mobile Agents are similar to blackboard agents except they are executed by execution agents, which are detailed in the section below. These agents contain methods which are called by execution agents on remote Sun SPOT nodes as well as methods for basestation execution. In order to access the sensor nodes hardware, these agents have pointers to the hardware of the node as well as the parent stack in which they are housed. As with the FScript blackboard agents, a version of the mobile agent was created to allow for FScript code to be contained. The FScript mobile agent contains FScript code as well as an FScript executor which interprets instructions specific to the mobile agents. Both the FScript interpreter for the mobile agents and blackboard agents extend a parent FScript executor. The reason for the separation is that the number of instructions included in the FScript executor's negatively impacts the processing time of the FScript code. A list of the functions provided in each of the 2 FScript executor classes, as well as their parent class, are provided in appendix B.

5.3 Application Components

Using the framework components, a proof-of-concept WSN application was created. This application consists of both pure Java coded blackboard agents as well as FScript blackboard agents. The pure Java blackboard forms the basis of the stack framework as they are expected to be common for most WSN applications.

The lowest layer of the application consists of **Logical Link Control layer**. This layer was implemented to perform the functions of sending outgoing network data via the hardware layer as well as neighbour discovery.

The **Sender agent** contained an execution condition of operating on protocol packets which are travelling in a downward direction. This was determined by examining the *isUpward* flag of each packet. If this condition was met, the execution operation was executed which sends the packet out. This method set the *isUpward* flag to true to ensure the packet travels in the correct direction when arriving at the receiving stack. The hardware was then instructed to send a copy of the protocol packet. After the packet is sent, the blackboard controller is notified to mark the packet for deletion and the blackboard is notified of a change in its contents.

The **neighbourhood discovery agent** is more complex than many of the other blackboard agents. This is due to it containing a thread. This blackboard agent operates by periodically broadcasting hello packets. Hello packets are

implemented using the KSN Serialisable interface, allowing for persist data to be contained which can be transmitted between nodes. The Hello Packet class contains a sender address and a flag to indicate if the packet is a reply. Methods are provided to access and mutate these variables. When the neighbourhood discovery agent is constructed, a hello packet sender thread is started. This thread creates protocol packets with destination and next hop address set to broadcast. Hello Packets are then created and set as the payload for the newly created protocol packets. These packets are then added to the blackboard within the protocol layer, where the Sender agent will send them out to the network. The rate at which the packets are sent is determined by the broadcast interval variable of the agent. As with all blackboard agents, the neighbourhood discovery agent also contain a execution condition and operation methods. The execution condition of the blackboard agent is that protocol packets which are moving in an upward direction are found and contain payloads of the class type hello packet. The execution operation first determines if the packet received is a reply to a hello packet. If the message is not a reply, the packet need be returned to the sender to inform it that this node is within range. The packets next hop and destination addresses are first set to the sender address of the hello packet. The hello packet payload reply flag is then set to true and the sender address is set to address of the current node. Finally the packets direction is set to false and the direction of the packet is changed to down.

If the protocol packet operated on in the execution operation method is a reply flagged message, the list of discovered neighbours could require modification. A dynamic list of discovered neighbours is maintained by the agent. If a reply is received which contains a hello packet with a sender address not found in the list of neighbours, the new address is added to the list. A second list of timeout values is also contained which counts down each time a hello packet is broadcast. Each item in the time out list is associated with a node in the discovered list. When a reply is received, the count down entry associated with the sender address of the reply is then restored to a discovery timeout value. When a time out value reaches 0, both the time out and its associated node are removed from the lists. These counting down when hello packets are sent and resetting when replies are received allows for the agent to discover when previously discovered nodes are no longer replying. Reply packets are flagged for deletion as the last step.

The final operation notifies the blackboard controller of the modifications so processing can be performed. Both the broadcast interval and discovery timeout can be set based on the requirements of the application.

The highest protocol layer in the implemented application is the **execution layer**. The layer is responsible for the execution of mobile agents as well as administrative tasks such as adding new blackboard agents to the protocol stack.

The **execution blackboard agent** is responsible for the execution of MAs. These can be both FScript MAs or pure Java MAs. The execution condition method of this agent checks if the protocol packet being investigated is flagged as moving up the stack and the admin payload flag is false. The execution operation method starts by setting the parent stack and hardware layer variables of the contained mobile agent to reference the stack in which they are contained and the hardware layer of the node. The *operate* method of the MA

contained within the protocol packet is then called. This is called through an MA interface shared by both FScript MAs and pure Java MAs to allow for polymorphism. In the case of an FScript MA, the FScript executor is called given the *operate* method function within the contained FScript code. After the processing is completed, the direction of the protocol packet is set to be downward, the blackboard is notified of the modification and the element is set to be processed so that the controller can move the protocol packet down the stack again.

The **Admin agent** serves the single purpose of adding blackboard agents to a given protocol layer within the protocol stack. The execution condition method of this agent is similar to the execution agent, except the admin flag of the examined protocol packet may be set to true. This indicates that a protocol packet contains an FScript blackboard which need be added to the stack. The execution operation method of this agent simply adds the payload of the protocol packet, which is a FScript blackboard agent, to the given protocol layer. The address of the layer to which the blackboard agent need be added is specified by the stack address of the protocol packet.

A **routing layer** was implemented and positioned between the above mentioned layer. This layer contains FScript Blackboard Agents. These agent contain the execution condition and execution operation methods as the Java blackboard agents.

A **sentinel agent** was implement in order to prevent incoming protocol packets from unintentionally being executed on the incorrect nodes. The execution condition method evaluated the direction of the packet and the destination which the packet is to reach. If the packet is found to be going up or has a destination address different to that of the host, the execution condition is met, and the method signals the blackboard agent interface which informs the blackboard controller. The execution operation method sets the packets direction to down and calls the processing complete method.

The second FScript blackboard agent in the routing layer is the routing layer. As the implemented system serves to be a prototype, a simple static routing table was implemented in this agent rather than a dynamic table. The execution condition method of the routing agent returns true if the direction of the protocol packet is down. Routing data is stored in persistent memory which is wrapped in a routing table class. This class stores a list of next hop and route costs for each known node in the network. Addition, removals and modifications of routing entries are handled by the routing table class. Routing table entries were hard-coded in the FScript routing agent. The next hop of the protocol being operated on was then set based on the destination of the protocol packet and the lowest cost next hop.

As previously mentioned, FScript MAs were also implemented to perform application tasks. These MAs are executed by the execution blackboard agent in the execution layer. Each of these scripts contained a remote execution method which was called by the execution agents housed on remote Sun SPOT nodes, and basestation execution methods, which were called by execution agents house on base station nodes. A number of these MAs were created. These include

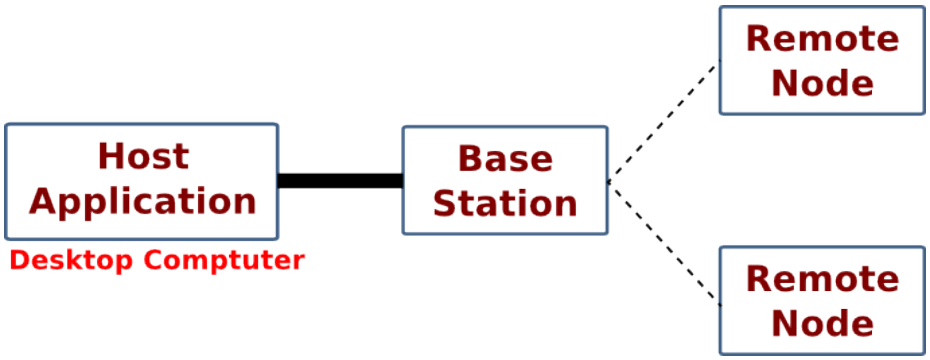

Figure 5.4: Host Application interface with remote nodes

agents which actuate the LEDs on the nodes based on the perceived temperature or light intensity. Administrative agents were also created in order to capture information about a remote stack, after which it would return to the basestation and display the information gathered. Addition and removal of layers and removal of blackboard agents were each handled by FScript MAs. These applications were loaded from file and deployed via a host application executing on a desktop computer. A basestation Sun SPOT provided an interface between the remote nodes and the host application. The relationship between the host application, basestation and remote nodes are shown in figure 5.4.

Java Sun SPOT host applications allow for desktop applications to interact with other remote Sun SPOTs via a basestation node. A host application was implemented to contain the same protocol stack and layer structure as the remote nodes. The only difference between the remote and host application was that the host application was utilized for remotely deploying protocol packets to various sensor nodes. Command line arguments passed to the host application in order to specify which mobile agents need be loaded. Nested command line arguments allowed for the mobile agent loaded by the host application to be passed arguments such as address and stack address. Examples of these deployments and arguments are given in Appendix A.

Chapter 6

Evaluation

In this chapter we present details of the evaluations which were performed on the proposed WSN Stack framework. We start by presenting the overall aim of the evaluations and reason for performing them. Details of the evaluation platform which was used are then presented. The final section of this chapter details each of the experiments which were performed, including the aim of the experiment, the methodology employed as well as the raw results.

A discussion of the results and possible conclusions are presented in chapter 7

6.1 Aim

The motivation for evaluating the proposed system is to empirically determine the degree to which the research aim and goals have been satisfied.

The research aim of this work is to design, develop and implement a WSN framework to provide structure and facilitate intelligence in WSN systems and their applications. This aim consists of 4 research objectives, namely: application neutrality, reconfigurability, facilitation of intelligence and efficiency. In order to determine the success of the overall research aim, the success of each of the contained research objectives must be determined. This was determined by conducting a number of experiments.

6.2 Evaluation Platforms

An initial feasibility prototype of the blackboard and stack systems were developed to run on a desktop computer using Java 2 Platform, Standard Edition (J2SE). This provided insight into areas of the system which were previously not considered, without requiring any complex implementation. No further detail of this are provided as it only served as an initial feasibility prototype for the stack based approach and no experimentation was conducted using it.

The same implementation and Java Sun SPOT platform presented in chapter 5 was used for all the experiments.

6.3 Experiments

Five sets of experiments were conducted to determine the success of proposed system. These were aimed at determining the success of the various research goals. All of the experiments were conduced in an in-doors laboratory environment. The range between the various nodes during the experiments was 2 meters. For each set of experiments we present the aim, method, metrics and results.

6.3.1 Remote Deployment

Aim

The aim of this experiment is to determine the effort required to remotely deploy a WSN application to a sensor node. We define remote deployment as a deployment where the sensor node receiving the deployed application or component is not directly connected via a physical medium to the host computer which is deploying the application or component.

In the experiment we use time as a metric to describe the effort required. We observe the amount to time taken to remotely deploy an application. Both the initial effort required for deploying an instance of the framework to the remote node, as well as the deployment of application components which utilizes the framework are looked at.

Methodology

In order to evaluate the amount time taken to remotely deploy a WSN application, a number of remote deployments were performed and the deployment times observed.

We define deployment time as the amount of time taken for the application or component to be sent from a host application on a desktop computer to the remote Sun SPOT via a Sun SPOT base-station 5.4. This time consists of the start-up time of the host application, network transmission time between the base-station and remote Sun SPOT and the time taken for the Sun SPOT to instantiate the received data. Time measurements began when the user initiated the deployment of the application or component and ended when the application or component was initiated and began running on the remote Sun SPOT. The deployment times for a number of different components and applications were observed. Each component or application was deployed 5 times in order to gain a reliable measurement. Details of the deployments are provided below:

The first deployment, #1 in table 6.1, involved deploying an instance of the Stack Framework to a single remote Sun SPOT node. The remote Sun SPOT contained no custom application software. Due to this, the Java Sun SPOT OTA download was used to remotely deploy the Stack Framework. Details of the OTA deployment process are provided in appendix A.

After deploying the framework, a number of application components were remotely deployed to the remote Sun SPOT. Each of the application components were deployed separately and their deployment times measured. In order to deploy application components to the remote Stack, a host application containing an instance of a Stack Framework is executed on the host computer. This host application is executed for each of the deployments. This processes

and sends the components from the host computer to the remote Sun SPOT via the base-station. The start-up and execution time of the host application are included in the deployment times.

The second deployment involved the deployment of a MA to add a new a Protocol Layer to the remote stack, #2 in table 6.1. This MA was executed by the Execution Agent within the Execution Layer of the remote Sun SPOTs stack. The execution performed by the MA creates a new protocol layer of the given name. After creating the new layer, the MA flags itself for deletion.

A third deployment, #3 in table 6.1 was performed to add a Blackboard Agent to the newly created Protocol Layer. This was performed by executing the host application and passing it a Blackboard Agent script together with a flag to indicate that an administration protocol packet need be created. These administration packets are handled by the Admin Blackboard agent of the Stack Framework. The deployed Blackboard Agent contained simple logic to delete all incoming protocol packets which are destined for other sensor nodes. A check is performed to check if the Protocol Packets destination address is the same as that of the node, if the addresses are different, the Protocol Packet is deleted. This Blackboard Agent is referred to as a sentinel agent.

The final deployment was a MA which contained application logic to be executed on the remote spot. Details shown as #4 in table 6.1. Unlike the previous deployments, this did not modify or contribute to the Protocol Stack. This deployment is performed in the same was as the second deployment of an MA to add a new Protocol Layer. However, no additional parameters were passed to the Host Application. The MA deployed contained the simple application logic to toggle a number of LEDs of the remote Sun SPOT. The number of LEDs was dependent on the amount of light perceived by the sensor node..

In order to provide a comparison, a simple standalone application was developed and deployed to a remote Sun SPOT. The details are presented as #5 in table 6.1 . This stand alone application does not utilize the Stack Framework and was deployed to a Sun SPOT which contained no other software. The logic of this application was similar to that of the MA mentioned above, LEDs on the sensor nodes were toggled based on the intensity of the light being sensed.

The application logic of this application was the same as the above mentioned MA which toggles a number of LEDs based on the perceived light intensity.

Details of all the deployment times for each of the deployments are presented in the results section below.

Results

The deployment times for each of the above mentioned experiments are presented in table 6.1. These results suggest that the time taken for the initial deployment of the stack framework, #1, is greater than that of the standalone application, #5. However, the deployment times of the various framework components is less than that of the standalone application. These results suggest that a trade off exists been the deployment time of the framework and its potential benefits over stand alone applications.

The deployment times of the various stack components in table 6.1 is also shown to have little variance, only a 1.1017 second difference between all 3 deployments.

#	Component	Deployment Time (seconds)	Std. Deviation
1	Framework	441.1936	0.8855
2	Protocol Layer	7.3577	0.995
3	Blackboard Agent	6.2560	0.0714
4	Mobile Agent	6.4435	0.0489
5	Standalone application	10.0922	0.0614

Table 6.1: Time taken to remotely deploy framework and application components

6.3.2 Remote Reconfiguration

Aim

The aim of this experiment is to determine the effort required to remotely reconfigure a WSN application. We define remote reconfiguration as the ability to reconfigure the software on a sensor node which is not directly connected via a physical medium to the host computer from which the reconfiguration is being performed. Reconfiguration involves the addition or removal of software components on the remote sensor node. These components can form part of the WSN application or the framework on which the application operate.

As with the previously detailed remote deployment experiments, time is used as a metric to describe the effort required. We observe the amount of time taken to perform a number of remote reconfigurations, including removing and adding Blackboard Agents and Protocol Layers.

Methodology

We define reconfiguration time as the amount of time taken for a stack framework components or application to be modified by a host application running on a desktop computer. This host computer communicates with the remote Sun SPOT via a Sun SPOT base-station. Modification consists of the addition or removal of framework components. This reconfiguration process is similar to the previously mentioned remote deployment process, thus measured in a similar way.

Reconfiguration time consists of the start-up time of the host application, network transmission time between the base-station and remote Sun SPOT and the time taken for the reconfiguration to be completed. Time measurements began when the experimenter initiated the deployment of the application or component and ended when the reconfiguration has been performed.

The reconfiguration times for a number of different reconfiguration were observed. Each of these reconfigurations were performed 5 times in order to gain an reliable measurement.

This experiment began with a remote Sun SPOT in the same state as the end of the previous experiment. The remote Sun SPOT stack thus consisted of the default Logical Link Control Layer and the Execution Layer, as well as a test layer which contained a simple sentinel blackboard agent. Details of the reconfigurations performed on this framework are provided below:

The first reconfiguration involved removing the sentinel blackboard agent from the test layer on the remote Sun SPOT, #1 in table 6.2. Removing of components from the stack framework is performed by an *Agent remover* MA which is executed by the execution agent of the remote Sun SPOTs stack. The Agent remover MA accepts a Sun SPOT address, layer index and agent index. When the MA is executed, the addressed agent is removed. After removing the agent, the MA flags itself for deletion.

The second reconfiguration, #2 in table 6.2, removes the test layer from the remote Sun SPOT stack. The removal of layers is performed in the same way as the removal agents, however the Layer remover MA does not require an agent index.

After removing the sentinel blackboard agent and the layer in which it was contained, the remote stack is reconfigured to include a new layer, #3 in table 6.2. This layer, the routing layer, is created in the same way as the layer deployment in the remote deployment experiment. Addition of layers is performed by deploying an MA containing the name of the new layer to add and the layer index to which the blackboard agent must be added. The execution performed by the MA creates a new protocol layer of the given name. After creating the new layer, the MA flags itself for deletion.

A fourth reconfiguration was performed to add a Routing blackboard agent to the remote stack. This was performed by executing the host application and passing it a Blackboard Agent script together with a flag to indicate that an administration protocol packet need be created. A layer index is also included. This administrative packet is handled by the admin agent on the remote Sun SPOT, which instantiates the new Blackboard Agent in the protocol layer of the given index.

The final stack reconfiguration performed was the addition of a sentinel blackboard agent to the routing layer, #5 in table 6.2. Redeployment of this agent is the same as the deployment performed in the previous experiment. This reconfiguration was performed to identify the effort of adding multiple blackboard agents to the same layer.

A Mobile agent was also reconfigured and transmitted to a remote Sun SPOT. The MA is a modified version of the MA to toggle the LEDs on the Sun SPOT based on perceived light intensity. This MA toggled the LEDs based on the sensed temperature. This MA was deployed in the same way as the deployment mentioned in the section 6.3.1. Details of the deployment are shown as #6 in table 6.2.

In order to provide a comparison between the reconfiguration effort of the protocol stack and Mobile agent with the effort of reconfiguring a standalone application, a modification was made to the standalone application mentioned in the remote deployment evaluation. The new standalone application is similar to the previously mentioned version, however the logic was changed to be the same as the above mentioned MA, #7 in table 6.2. Rather than toggling the LEDs based on the perceived light intensity, LEDs were enabled based on the sensed temperature. As this was a standalone application which was modified, the source files were required to be recompiled and entirely redeployed.

Details of the reconfiguration times for the experiments are presented below.

#	Component	Reconfiguration Time (seconds)	Std. Deviation
1	Remove Blackboard Agent	6.8321	0.05
2	Remove Protocol Layer	12.9834	14.4974
3	Add Protocol Layer	13.0683	14.3405
4	Add Blackboard Agent	13.8796	14.8954
5	Add Blackboard Agent	12.9995	14.3550
6	Reconfigured Mobile Agent	13.1561	14.3455
7	Reconfigured Standalone application	20.7398	16.7726

Table 6.2: Times taken to remotely reconfigure the framework components and application

Results

The reconfiguration times for each of the above mentioned experiments are presented in table 6.2. These results suggest that the effort required to modify any of the framework components or applications is less than that of reconfiguring a standalone application, #7. However, the total time required to perform all the stack component reconfigurations is greater than that of reconfiguring a standalone application. These results suggest that our proposed system is more efficient than a stand alone application in situations where small scale reconfiguration is required.

6.3.3 Intelligence

Aim

In this experiment we aim to determine the degree to which a stack based application, together with the proposed framework, can express intelligent behaviour. This relates directly to our research aim of facilitating intelligence. We operationalise intelligence as the ability to learn from previous actions and to improve performance over time. Learning from previous actions is performed by the evaluation of the actions outcome and comparing expected or previous results of actions performed. As the application is exposed to more data, the processing is expected to become increasing efficient and effective. Specific actions could also be identified as performing worse than expected, thus no longer being performed.

Methodology

In order to evaluate the degree to which intelligence has been facilitated, a WSN test application to display simple intelligence was implemented and deployed onto our framework implementation. The performance metric used is processing time. Processing time is defined as the time between the arrival of the protocol packet on the host node and the time when it leaves or is deleted from the host node.

The test application developed consists of a Protocol Layer, 5 Blackboard Agents and a single MA. The Protocol layer housed all 5 of the Blackboard Agents.

Identify Character	Frequency
A	15%
B	5%
C	25%
D	5%
E	50%

Table 6.3: Distribution of Mobile Agent Identity characters.

Each of the Mobile Agents contained an identifier in the form of a single character, which was pushed onto the persistent memory stack of the MA. This allowed for the Blackboard Agents to access the identifier. The role of each of the Blackboard Agents was to identify those incoming packets which contained a specific hard-coded identifier ($ID = \{A, B, C, D, E\}$). Each of the identifier values were checked for by a single one of the 5 Blackboard Agents.

If the incoming protocol packet contained a MA with the identity variable matching that of the Blackboard Agent, the protocol packet containing the incoming MA is flagged as processed. This mimics the expected behaviour in larger real world WSN applications where incoming packets would be inspected and processed by a number of Blackboard Agents within a single layer.

Besides the identifier value, the Mobile Agent, referred to a MAIntelligenceTest, contained basic application logic. This logic was executed by the execution layer of the Protocol Stack. The MA would display its identifier value as well as the current time in milliseconds. This allowed for processing times to be gathered. The MA would then fag itself for deletion by the Blackboard Controller.

Basic intelligence was implemented by allowing each of the Blackboard agents to reposition themselves within the protocol layer. The position of each of the Blackboard agents within the layers influences the amount of time taken for processing to be performed by the blackboard agent. After flagging the protocol packet as processed, each of the Blackboard Agent would swap positions with the Blackboard agent ahead of them in the list. This allowed for the more popular Blackboard Agents to progress to the front of the list. This is based on the assumption that no dependencies exist between the various Blackboard Agents.

The experiment was conducted by deploying the Protocol Layer and Blackboard Agents to a remote Sun SPOT Node via a Sun SPOT base-station. Mobile Agents ($n = 20$) were then passed identifiers via command line arguments, between A and E before being sent to the remote Sun SPOT. The distribution of identifier values which were used is presented in the table 6.3.

The time at which the MAs arrive at the remote Sun SPOT was displayed in milliseconds. This was used together with the time at which the packet was deleted to determine the processing time.

Results

Graph 6.1 illustrates the processing time for each of the MAs as well as the identifier value which was used. The average processing times indicate the

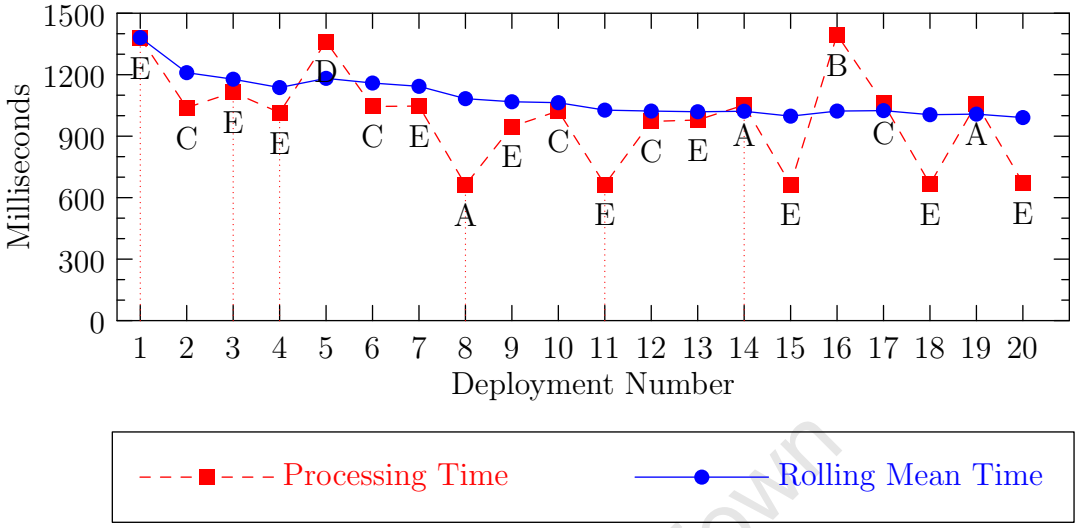

Figure 6.1: Line plot showing the processing times and Rolling Mean processing time of MAs. Mobile Agent identifiers are labelled at each data point. Deployment numbers are displayed along the X axis

change as more MAs are processed.

The individual processing times of each of the deployments fluctuate. Initial processing times of the MAs is not efficient given the distribution of MA identifiers. Over time the required processing times of the more frequent identifiers becomes less. This can be seen in the deployments 1, 3, 4 and 11. A MA with the identifier of E is deployed in all these deployments. Processing of the first MA is extremely poor. This processing time becomes increasingly better with the third and forth deployments. An optimal time is then reached in deployment 11. The opposite of this can be seen in deployments 8 and 14, both of which involve a MA with an infrequent ID of A. The processing time of deployment 8 is considered optimal, which is later not the case, as in deployment 14. This optimal processing time is due to processing being performed by the Blackboard situated first in the protocol layer, which then becomes less optimal as the Blackboard agents position is shifted further back.

Table 6.2 shows the processing time for idenfiers E as more deployments are made, while table 6.3 shows the times for identifiers A. These results suggest that the processing times of the more frequent identifier, E, improves overtime at the expense of the processing times of the less frequent identifier, A. These results that the system can learn from experiences, thus displays intelligence. The result can be scaled to larger packets of sensor data, allowing for more efficient processing of more frequent sensor data at the expense of less efficient processing of less frequent sensor data. This improves the overall processing performance.

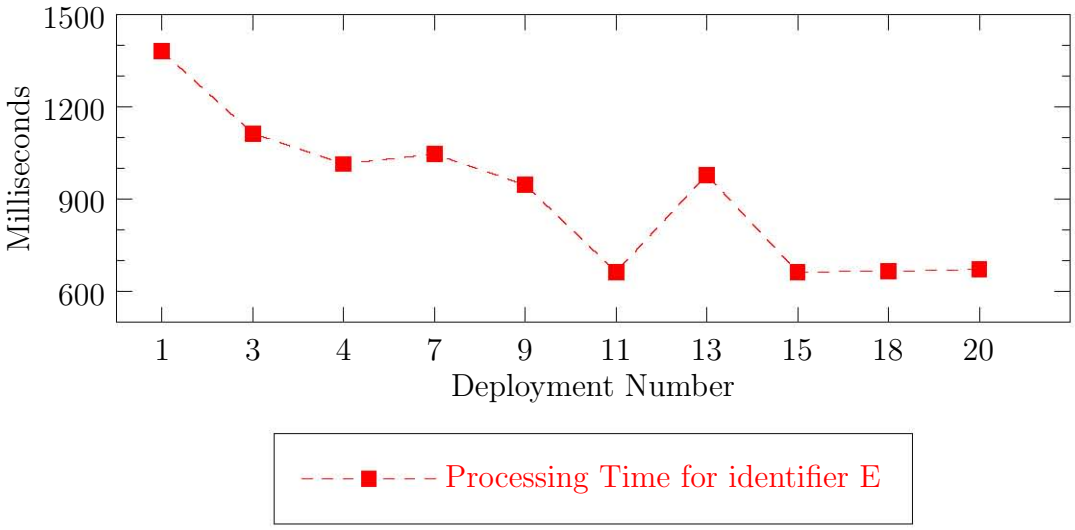

Figure 6.2: Line plot showing the processing times for those deployments with the most frequent identifier, E

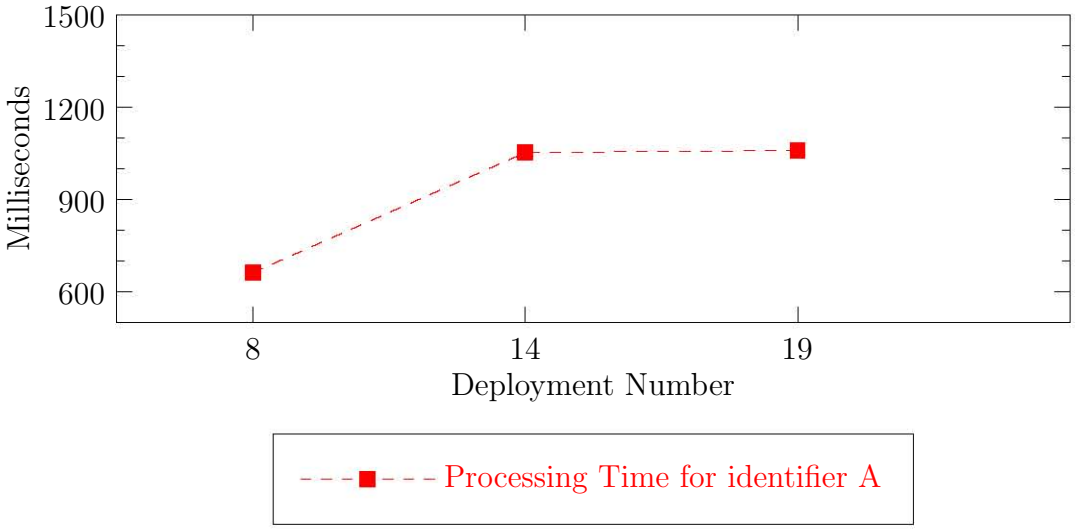

Figure 6.3: Line plot showing the processing times for those deployments with the least frequent identifier, A

6.3.4 Latency and Overhead

Aim

The aim of this experiment is to determine the systems overhead, in terms of both the memory usage as well as the systems latency. By determining the overhead of the system, we can identify if the system is sufficiently efficient and how overhead is effected by scaling through the addition of more components to the framework.

In order to understand the latency and overhead of the entire proposed system, experiments must be performed on both the stack framework components as well as the applications which utilize the framework. These experiments examined the amount of memory used by the various components as well processing time and communication latency of the components.

Methodology

In this section we describe the methodology utilized for determining the memory overhead, followed by the methodology used to determine the system latency.

The first set of experiments aimed to identify the memory overhead of the various stack and application components. In order to determine the memory usage for each of the components, a Java Sun SPOT skeleton application was created. This application is an empty Sun SPOT application which serves as a base to which various components can be added. The only logic and code contained in the skeleton application is that to display the total memory usage of the application. To this skeleton application, each of the stack framework components were added. After the inclusion of each of the components, the application was compiled, deployed and executed. The memory usage for each deployment was noted. These memory usage measurements were then subtracted from the memory usage measured from the previous deployment to provide memory overhead of the component which was added to the skeleton application. Details of which components were evaluated as well as the overhead for each of these components is provided in table 6.4.

The second set of experiments were conducted to determine the latency of the proposed system. 3 different latency measures were identified: within the protocol stack, between nodes and MA application processing latency.

Latency within the protocol stack is defined as the amount of time required for a protocol packet to traverse the protocol layers within the protocol stack, as well as the latency within the protocol layers. Latency within the protocol layers is dependent on the number of Blackboard agents within the given protocol layer. The time taken to traverse the stack thus depends on the number of protocol layers within the stack and the number of blackboard agents they each contain.

In order to measure the latency within the protocol stack, a protocol stack was deployed. This stack included the standard stack components as well as three addition protocol layers: Test, Empty and Routing Layer. This stack is illustrated in figure 6.4.

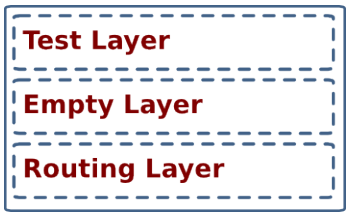

Figure 6.4: Latency and Overhead experiment stack

The Test layer was positioned highest of the 3 protocol layers in stack. This layer is the same as the layer included in the intelligence experiment previously detailed. It contained 5 blackboard agents which attempt to operate on incoming protocol packets which contain an MA with a specific identifier variable. The Empty protocol layer, positioned below the Test layer, contained no Blackboard Agents. The final addition to the stack was the Routing layer which contained routing and sentinel agents. Details of these agents are provided in section 5. A logger component was utilized to display verbose output as processing was performed within the protocol stack. Protocol Packets containing MAs were then passed to the remote Sun SPOT via the host application. These MAs, like those used for the intelligence experiment, contained identifier variables. 5 MAs, each with different identifier values, were passed. The output included the time at which various operations were performed as well as when protocol packets arrive and depart each layer. Details of the measured latency for the various layers are presented in table 6.5. These results are discussed in the section which follows.

Latency between nodes is defined as the amount of time taken to transfer data from one node to another node. This time does not include processing by either the sending or receiving nodes, only the time between sending a packet and receiving that packet. The time varies depending on the amount of data which is being transferred as well as the distance between the nodes.

In order to determine this time, a remote Sun SPOT application was created which contains a protocol stack with a single addition protocol layer. A blackboard agent which returns all protocol packets to the sender node was placed in this layer. A logger component on the remote Sun Spot as well as a host base-station application was set to display the time at which data is received by the node as well as when the data was sent. A number of protocol packets with varying payloads were sent from the base-station node to the remote node, which were in turn returned to the node from which it was originally sent. The difference between the sending and receiving times on the base-station was then subtracted from the difference between the receiving time and sending times logged by the remote Sun Spot. This provided the amount of time that the data was being transmitted in both directions. A one-way transmission latency between the nodes was then estimated by halving the time taken to transmit the data in both directions. Transmission latencies were measured for MAs which add protocol layers, MAs which remove blackboard agents and protocol layers, protocol packet which contain blackboard agents and Mobile Application agents. Details of the latency times for each of the MAs is presented in table 6.6

MA application processing latency refers to the time taken for a MA ap-

#	Component	Memory Overhead (bytes)
1	Sun SPOT skeleton application and JVM	62680
2	Hardware Layer	3080
3	Protocol Stack	96
4	LLC Layer	244
5	Neighbourhood discovery Blackboard Agent	1120
6	Sender Blackboard Agent	48
7	Execution Layer	244
8	Execution Blackboard Agent	48
9	Admin Blackboard Agent	48

Table 6.4: Memory overhead of framework components and application

plication to be executed by the execution layer of a receiving protocol stack. These MAs contain scripted application logic which is executed by the execution blackboard agent.

Experiments were conducted to determine this processing time by deploying 3 different MA Applications. The execution time of each MA was measured as the difference between the execution blackboard agent starting and ending the execution of the MAs contained code.

The first two MAs contained logic to gather sensor data on the node: the first is the *Light to LED* MA used in the remote deployment experiment, and the second the *Temperature to LED* MA used in the remote reconfiguration experiment. The final MA did not gather any sensor data. This MA gathered information regarding the current protocol stack configuration and displayed this information. Details of the processing times of each of the components are presented in table 6.7

Results

Memory overhead of the various components are presented in table 6.4. The Largest memory usage is that of the Sun SPOT skeleton application onto which the other application components are added. The memory requirements of the other components are considerably smaller, with the smallest being the Sender, Execution and Admin blackboard agents. The total memory usage is 67680 bytes.

In terms of latency, table 6.5 indicates the latency within the protocol stack. These times are based on the experiment where each of the layers have a number of different blackboard agents, each with varying complexity. The Test layer contained 5 agents with a latency of 579 milliseconds. The suggests that the average latency per blackboard agent, in this case, is 115.8 milliseconds. The empty layer was best performing layer with a latency of only 34.6 milliseconds. This is due to the layer containing no blackboard agents, and thus immediately passing the received protocol packet to the next layer. The Routing layer contained only 2 blackboard agents. These agents are more complex then those blackboard agents in the test layer due to the required routing information and processing. The blackboard agents in this layer contributed on average 182.6 milliseconds of latency. These results suggest that the expected latency within

#	Component	Latency (milliseconds)
1	Test Layer	579
2	Empty Layer	34.6
3	Routing Layer	365.2

Table 6.5: Latency within the protocol stack

#	Component	Latency (milliseconds)
1	Addition of a *Routing Layer*	282.5
2	Addition of a *Routing Blackboard Agent*	729.8
3	Addition of a *Sentinel Blackboard Agent*	194.5
4	MA to remove Blackboard Agent	223.3
5	MA to remove Protocol Layer	110.2
6	MA *Light To LED*	167.9

Table 6.6: Latency of different components and MAs between Sun SPOT nodes

a stack can vary depending on the number of blackboard agents within a layer as well as the complexity of the agents.

Similar finds can be suggested with the latency between sensor nodes, as showing in table 6.6. The addition of the Routing blackboard agent requires 729.8 milliseconds opposed to addition of a sentinel blackboard agent which requires only 194.5 milliseconds. These findings suggest that like latency within the stack, latency between nodes is heavy influenced by the complexity of the data being passed between nodes. The finding also suggest that transmitting of similar agents can result in dramatically different latencies. For example, the MA to remove a protocol layer required approximately twice as much time as an MA to remove a blackboard agent.

The final latency experiments results are presented in table 6.7. These results suggest that MAs with similar complexity and logic can have extremely different processing times. This is indicated by the difference in processing times between an MA to enable lights based on the light intensity it sense and an MA to enable lights based on the perceived temperature. This 35 fold difference in processing time is believed to result from the way in which light values on sensed by the Java Sun SPOT. The results do suggest that MA applications which are not required to sense using the sensor nodes sensors, perform far greater than those applications which do. Thus applications which access or manipulate data which resides on the sensor nodes perform quicker than applications which require sensor readings.

6.3.5 Scalability

Aim

The aim of this experiment is to determine the influence scaling of a WSN has on the proposed stack based system. Scaling of the network is performed through the addition and removal of sensor nodes. The first aim in this experiment is to

#	Mobile Agent application	Processing Time (Milliseconds)
1	Light to LED	11735
2	Temperature to LED	344
3	Protocol Stack Information	57

Table 6.7: Processing times of MA Applications by Execution Agent

identify which areas are influenced by altering the number of nodes within the network. This is followed by determining how the identified areas are effected.

Methodology

Two areas were identified as being influenced by scaling the sensor network: communication latency and processing and memory overhead of certain stack components.

Communication latency is influenced both if the number of nodes within the network increases, as well as if the physical area being sensed by a sensor nodes increases.

In terms of the latency of communication within a larger network, in many cases, multihop routing is required. Multiple hops increase the communication time as a single end-to-end message is composed of a number of intermediate transmissions. Processing time at each node influences the communication cost as processing needs to be performed by each of the intermediate nodes along the communication path. These times will vary based on the contents of the protocol packet being sent, the number of hops required and the processing time performed by each node along the path. Estimate can be made by multiplying the between node latency and processing time to route a packet to the next hope by the expected number of hops. Based on the previous experiments, assuming a routing layer was responsible for processing and an MA to enable lights based on perceived light was sent, each hope would require 533.1 milliseconds.

The second area influenced by scaling the WSN is the processing and memory overhead of certain stack components, specifically those components which keep track of which nodes are directly available for communication and how to route data to other nodes further away. These include neighbourhood discovery and routing agents, which are typically requirements in multihop networks. In this experiment a neighbourhood discovery agent was implemented and evaluated.

Details of the implementation of the neighbourhood discovery blackboard agent are provided in chapter 5. This agent is placed in the Logical Link Control Layer and not a scripted agent. The reason for this is that such an agent is typically required and does not require extensive reconfiguration. Parameters within this agent can however be modified to suite the requirements of the application.

The task of this blackboard agent is to discover the other nodes which are in direct communication range and maintain a list of these nodes, which would typically be passed to a routing agent. Discovery involves both finding of new nodes which are now in range, as well as identifying when previously in range nodes are no longer within range. These additions and removals of nodes could be the result of nodes awaking or sleeping as well as mobile sensor nodes which

are able to move in and out of range. This agent operates by broadcasting hello packets and waiting for a reply from any nodes which receive the broadcast. When a reply is received, that node is added to the list of neighbours. The nature of broadcasting allows for all nodes within range to be addressed with no greater cost than sending a message to an individual node.

Nodes which were previously found are considered to be lost if they no longer reply to a set number of hello packets. This set number of replies which can missed before a node is considered unavailable is referred to as timeout count.

Both the interval at which hello packets are broadcast and the timeout count can be configured to best suite the needs of the WSN application.

A set of experiments were run to identify the time taken for a node to discover its neighbouring nodes as well how long it takes to discover a previously discovered node is not longer in range or available .

In order to measure the time taken to discover neighbouring nodes, an experiment was conducted where a node was introduced to a small WSN containing 3 other nodes containing the same stack. All 3 nodes were in range and thus direct neighbours. Times were logged to determine how long it took for the newly introduced node to discover each of the 3 neighbouring nodes. 3 different broadcast delays were selected: 2, 5 and 7 seconds. 5 iterations of the experiment were performed with each time. Logging of time was performed by noting the time at which the node was started and the times at which various other nodes were added to its neighbourhood list. Results of the experiment are presented in graph 6.5.

A second set of experiments were performed to evaluate the time required for a previously discovered node to be removed from the neighbourhood list due to a timeout. This experiment was conducted using two nodes, both containing the same protocol stack configuration. After both nodes had discovered the other, one of the nodes were shutdown. The time taken for the other node to discover that its neighbour was no longer available was logged. This time was logged by displaying the time at which the one node had been shutdown and when the other node discovered it was no longer available. 3 different broadcast delays were used, as with the above mentioned discovery experiment. 3 different timeout counts were also used, ranging from 1 to 3. In order to gain a reliable measure, experiments with each of 9 combinations of the broadcast delays and timeout counts were conducted 5 times. Results of this experiment are presented in graph 6.6.

Results

Figure 6.5 depicts the time taken for a sensor node to discover up to 3 neighbouring nodes. 3 different broadcast intervals are represented. The time taken to discover the nodes remain uniform across all 3 delays with none performing exponentially better than any other. However, the discovery times are greater for the higher broadcast intervals. This suggests that a smaller broadcast interval provides faster discover than a larger broadcast interval. It is possible that this decreased time should come at the expense of power consumption, as more messages are being sent thus more power being used.

The standard deviation of the discovery times are also indicated in the graph. This appears to become greater for all 3 broadcast delays when discovering more neighbours. This effect is greater for larger broadcast intervals.

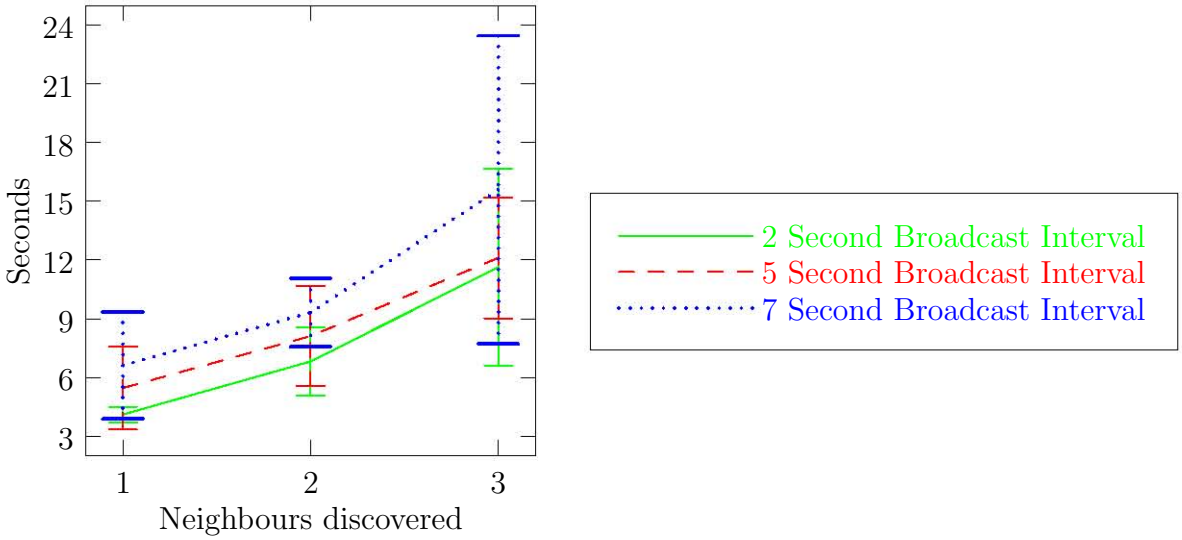

Figure 6.5: New neighbour discovery times

Figure 6.6 depicts the time taken for a sensor node to discover the loss of its only neighbouring node. Like the discovery of neighbours, times taken to discover a loss is proportional to the timeout count. The difference between discovering a loss of a neighbour with a broadcast delay of 2 and timeout counts of 1 is less than the same broadcast delay and a time out count of 3. Unlike the previous graph, the standard deviation of logged times remains more consistent across all samples.

These results also suggest that a smaller broadcast delay will result in a faster response to the removal of a node from the network, however this would be a trade off with power consumption. Lower timeout counts also appear to provide a great response time. This may however result in the false positives, resulting in the incorrect removal of a node form the neighbourhood list as the node only dropped a single hello packet which it is required to reply to.

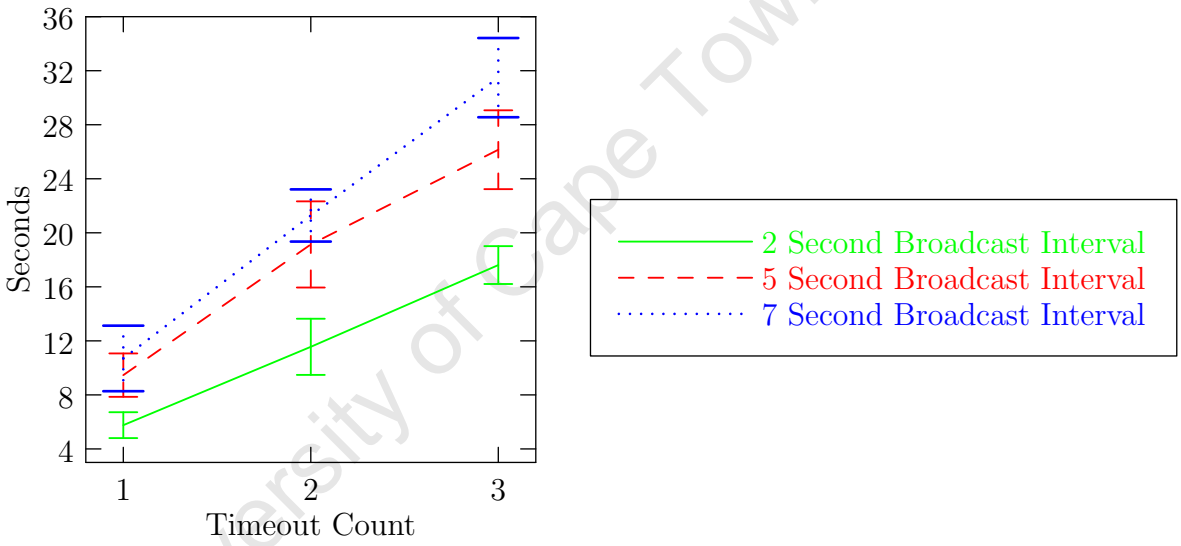

Figure 6.6: Lost neighbour discovery times

Chapter 7

Conclusion

This chapter concludes our dissertation. A summary of the work which was undertaken, and detailed in the body of this thesis, is presented. The evaluation findings of this work are then reflected on. Significant findings, and how these satisfy our research aims, are presented. A brief summary of the contributions made by this research is also provided. Possible future work is presented in the final section of this chapter.

7.1 Summary of work undertaken

Initial background research was undertaken in order to understand WSNs, their applications and the current design and development approaches. A number of example WSN applications in various application domains were studied. From this research a shortcomings of current WSN systems were identified. Network lifetime, routing, localization, self-organisation, reconfiguration, distribution of processing, intelligence and security were identified as challenges faced by current WSN systems.

As a potential solution to overcome some of the issues faced by current WSN systems, software agents were investigated. As with WSNs, the types of software agents were identified and a number of example systems studied. Intelligent agents were found to have the benefits of emergence as well as scalability, at the expense of design complexity.

Based on the background research and investigation into current WSN systems and applications, a research aim was determined. This aim was to design and develop a WSN framework which provides structure and facilitates intelligence. This aim was decomposed into a number of research objectives. These objectives were application neutrality, reconfigurability, facilitation of intelligence and efficiency.

In order to satisfy the research aim and its objectives, a proposed system design was created. This approach utilizes a combination of layered architecture and intelligence software agents. Layered architecture was selected in order to provide structure and modularity while software agents facilitated intelligence. Feasibility checks were performed by on the proposed solution by checking that

typical WSN application tasks could be performed by a layered agent-based system. An iterative design process was followed to provide greater refinement of the features required by the proposed system. Once the components had been finalized, the interaction between the components was formalized. This is essential due to ensure a protocol is followed by the framework and its components.

An implementation of the proposed framework design was then developed along with a prototype application which utilizes the framework. This application was developed to serve as a proof-of-concept and testbed.

The final stage of this research was the evaluation and experimentation of the implemented framework, and components and the prototype application. A number of evaluations were performed in order to determine the degree to which the proposed system satisfied the research aim and its objectives. A discussion of the evaluation findings is presented in the section below.

7.2 Evaluation findings

In the first 2 sets of experiments which were conducted, and initial deployment of the stack framework was required and deployed only once. Applications components were deployed onto this framework instance. Scripted application agents were also handled by the framework instance without any prior knowledge of the application. However these findings suggest that the proposed system compatible with a number of different applications, and is that application neutral. These finding do not suggest that all possible WSN applications have been catered for.

Framework and application components from the first experiment were removed and replaced with other components in the second experiment. These changes were possible without requiring the redeployment of the entire framework. Due to this, we conclude that the research goal of reconfiguration has been achieved.

Evaluations to determine if intelligence could be facilitated suggest that a framework applications performance improved over time with more exposure to data. This experiment indicated that processing times of a framework based application decreased overtime, thus gaining experience. This gaining of experience is considered to be intelligent behaviour and suggests that intelligent applications are facilitated by the proposed framework system.

Performance metrics were gathered during all the conducted experiments. These results suggest that the initial deployment of the proposed framework is less efficient than that of the deployment of a typical stand alone application on the same platform. However, such efficiency is traded for flexibility. The cost of altering and redeploying a standalone application is shown to be greater than the reconfiguration of framework components. Processing times for framework applications were also found to be marginally greater than the times of standalone applications. The findings suggest that the proposed framework is sufficiently efficient for applications were reconfiguration is highly probable.

However, in cases where reconfiguration is not required, greater efficiency can be achieved with standalone applications which perform the same function as the framework and application components.

Based on these findings, we proposed that the aim and goals of this research have been satisfied.

7.3 Contributions

This research has provided insight into the domain of WSNs, presenting a number challenges faced currently by these systems as well as agent based systems as a means of overcoming these challenges. A novel framework was created from this research which combines layered architecture and software agents in order to provide a controlled and structured emergence.

The evaluations conducted on the prototype of this system suggest that it is application neutral and reconfigurable. This allows for flexibility both in the initial application domain it is applied to, as well as allowing for changes to be made to the system. Intelligence is also considered present in the system as it is able to learn from its experience overtime, and thus perform more efficiently. All of these requirements are provided by the system while maintaining a sufficient level of efficiency to operate on the low powered sensor devices.

A finally noteworthy contribution of this work are the details and implementation provided to allow for a number of restrictions in J2ME platforms to be overcome in order to allow for remote deployment and execution of application code.

7.4 Future Work

As this research provide an initial proposal and prototype of a layered WSN framework, a number of areas could be researched and potentially improved upon in the future.

The first area which could be optimized is that of communication between nodes. Specifically error detection and correction between nodes as the current implementation does not incorporate such mechanisms.

Heterogeneous sensor networks could benefit from such a system as different devices within the same network are able to communicate. The only component of the system which is hardware dependant is the hardware layer. Other stack layers are device independent.

Future research could also be conducted both in the applications of the proposed framework, as well as implementations of the framework on other platforms. Applications and framework components could be developed to handle encryption and compression. It is expected that these components would be required for larger, commercial WSN systems. Implementations of the designed framework could also be developed for other WSN platforms as well none traditional networks. These include robotics system where robots act as sensor nodes in a network of other robots.

Bibliography

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

[2] D. Cruller, D. Estrin, and M. Srivastava. Overview of sensor networks. *Computer(Long Beach, CA)*, 37(8):41–49, 2004.

[3] MEMSIC inc. *MICA2 Datasheet*. MEMSIC inc., 2009.

[4] H. Karl and A. Willig. *Protocols and architectures for wireless sensor networks*. John Wiley & Sons Inc, 2005.

[5] J.S. Baras and H. Mehta. A probabilistic emergent routing algorithm for mobile ad hoc networks. *WiOpt03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, 2003.

[6] M. Maroti, G. Simon, A. Ledeczi, and J. Sztipanovits. Shooter localization in urban terrain. *COMPUTER*, pages 60–61, 2004.

[7] John Herbert, John O'Donoghue, Gao Ling, Kai Fei, and Chien-Liang Fok. Mobile agent architecture integration for a wireless sensor medical application. In *WI-IATW '06: Proceedings of the 2006 IEEE/WIC/ACM international conference on Web Intelligence and Intelligent Agent Technology*, pages 235–238, Washington, DC, USA, 2006. IEEE Computer Society.

[8] Pavan Sikka, Peter Corke, Philip Valencia, Christopher Crossman, Dave Swain, and Greg Bishop-Hurley. Wireless adhoc sensor and actuator networks on the farm. In *IPSN '06: Proceedings of the 5th international conference on Information processing in sensor networks*, pages 492–499, New York, NY, USA, 2006. ACM.

[9] G. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, and J. Lees. Deploying a wireless sensor network on an active volcano. *IEEE INTERNET COMPUTING*, pages 18–25, 2006.

[10] K. Sha, W. Shi, and O. Watkins. Using wireless sensor networks for fire rescue applications: Requirements and challenges. In *6th IEEE International Conference on Electro/Information Technology*, 2006.

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

[12] Piyush Gupta and P. R. Kumar. A system and traffic dependent adaptive routing algorithm for ad hoc networks. In *In Proceedings of the 36th IEEE Conference on Decision and Control*, pages 2375–2380, 1997.

[13] H. Le, D. Hoang, and R. Poliah. S-Web: an efficient and self-organizing Wireless Sensor Network Model. *Network-Based Information Systems*, pages 179–188, 2008.

[14] M.L. Sichitiu and V. Ramadurai. Localization of wireless sensor networks with a mobile beacon. *Proceedings of MASS*, pages 174–183, 2004.

[15] Travis C. Collier and Charles Taylor. Self-organization in sensor networks. *J. Parallel Distrib. Comput.*, 64(7):866–873, 2004.

[16] S.J. Russell, P. Norvig, J.F. Canny, J. Malik, and D.D. Edwards. *Artificial intelligence: a modern approach*. Prentice hall Englewood Cliffs, NJ, 1995.

[17] Vasant Honavar. Intelligent agents and multi agent systems. In *Proceedings of the Congress on Evolutionary Computation (CEC'99)*, Washington D.C., USA, July 6–9 1999.

[18] N.J. Nilsson. *Artificial intelligence: a new synthesis*. Morgan Kaufmann, 1998.

[19] A.S. Rao and M.P. Georgeff. Bdi agents: From theory to practice. *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319, 1995.

[20] T. Sundsted. An introduction to agents. *Java World*, pages 06–1998, 1998.

[21] K.P. Sycara. Multiagent systems. *AI Magazine*, 19(2):79–92, 1998.

[22] J. Fromm. Ten Questions about Emergence. *Arxiv preprint nlin.AO/0509049*, 2005.

[23] Ajith Abraham, Crina Grosan, and Vitorino Ramos. *Stigmergic Optimization Studies in Computational Intelligence*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[24] D.D. Corkill. Blackboard systems. *AI expert*, 6(9):40–47, 1991.

[25] N.R. Jennings. Specification and implementation of a belief-desire-joint-intention architecture for collaborative problem solving. *Journal of Intelligent and Cooperative Information Systems*, 2(3):289–318, 1993.

[26] R. Brooks. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of [legacy, pre-1988]*, 2(1):14–23, 1986.

[27] Rodney A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6(1&2):3–15, June 1990.

[28] R.A. Brooks. Intelligence without representation. *Artificial intelligence*, 47(1-3):139–159, 1991.

[29] M. Chen, T. Kwon, Y. Yuan, and V.C.M. Leung. Mobile agent based wireless sensor networks. *Journal of Computers*, 1(1):14–21, 2006.

[30] C.L. Fok, G.C. Roman, and C. Lu. Agilla: A mobile agent middleware for sensor networks. *Washington University in St. Louis, Technical Report*, 2006.

[31] C.L. Fok, G.C. Roman, and C. Lu. Mobile agent middleware for sensor networks: An application case study. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 382–387, 2005.

[32] H. Malik, E. Shakshuki, T. Dewolf, and M.K. Denko. Multi-agent system for directed diffusion in wireless sensor networks. In *Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on*, volume 2, 2007.

[33] Elhadi Shakshuki, Xinyu Xing, and Haroon Malik. Mobile agent for efficient routing among source nodes in wireless sensor networks. In *ICAS '07: Proceedings of the Third International Conference on Autonomic and Autonomous Systems*, page 39, Washington, DC, USA, 2007. IEEE Computer Society.

[34] V. Crespi, A. Galstyan, and K. Lerman. Top-down vs bottom-up methodologies in multi-agent system design. *Autonomous Robots*, 24(3):303–313, 2008.

[35] H. Le, A. Scholtz, and A. Potgieter. An Agent Based Layered Framework to Facilitate Intelligent Wireless Sensor Networks. In *Network-Based Information Systems, 2009. NBIS'09. International Conference on*, pages 136–141. IEEE, 2009.

[36] F. Buschmann. *Pattern-oriented software architecture: a system of patterns*. Wiley, 2002.

[37] W.R. Stevens. *TCP/IP Illustrated Vol. I: The Protocols*. Pearson Education India, 1994.

[38] M. Fowler. *Patterns of enterprise application architecture*. Addison-Wesley Professional, 2003.

[39] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-wesley Reading, MA, 1995.

[40] Sun Labs. *SunSPOT Owners Manual, Red release 5.0*. Sun Labs, 2009.

[41] Sun Labs. *Sun Small Programmable Object Technology Developer's Guide*. Sun Labs, 2009.

[42] S Meisinger S Kessler. Karlsruhe sensor networking project, February 2009.

[43] Murlen. Fscriptme, June 2009.

Appendix A

Framework and Component Deployment

Details of the OTA deployment process for the implemented framework as well as the various components are presented below:

A.1 Framework deployment

Performed using Sun SPOTs OTA deployment process

OTA Deploy
ant -DremoteId=*[node address]* deploy
OTA Run
ant -DremoteId=*[node address]* run

Table A.1: SunSPOT OTA deployment

A.2 Component deployment

Preformed using a basestation and host application

Remote Stack Information
ant host-run -Dmain.args=”-m MAgetStackInfo.fs *[address]*”
Add new protocol layer
ant host-run -Dmain.args=”-m MAaddLayer.fs *[address] [layer Label] [layer index]*”
Remove protocol layer
ant host-run -Dmain.args=”-m MAremoveLayer.fs *[address] [layer index]*”
Add blackboard agent
ant host-run -Dmain.args=”-b *[blackboard agent file] [address] [layer index]*”
Remove blackboard agent
ant host-run -Dmain.args=”-m MAremoveAgent.fs *[address] [layer index] [agent index]*”

Table A.2: SunSPOT component and Mobile Agent deployment

Appendix B

Sun SPOT FScript functions

Details of the various FScript functions which were implemented is proved below. The parent FScript executer instructions are detailed in table B.1. This is followed by the child classes for Blackboard agent and Mobile agent specific functions. In each table the function name, input, output and summary of what is performed are provided.

Function	Input	Output
println	(string) Message	
Displays the given string.		
writeObject	(object) Object to be pushed on	
pushes the object onto the persistent stack in order to maintain state		
readObject		(object) Object read
returns the object at the top of the persistent stack		
getSpotAddress		(string) Spots address
returns the address of the sunspot		

Table B.1: Instructions common to all Sun SPOT FScript Executers

Function	Input	Output
displayProtocolPacketDetails		
displays details of the protocol packet currently being inspected, including destination, next hop and direction		
getPacketDirection		(string) "UP" or "DOWN"
returns "UP" if the protocol packet is moving in an upward direction, or "DOWN" if moving down.		
setPacketDirection	(string) "UP" or "DOWN"	
sets the packet direction to "UP" or "DOWN"		
getPacketDestination		(string) address
Returns the final destination address of the protocol packet being inspected		
setPacketDestination	(string) address	
Sets the destination address of the packet being inspected		
getPacketNextHop		(string) address
Returns the next hop address of the protocol packet being inspected		
setPacketNextHop	(string) address	
Sets the destination address of the packet being inspected		
RTAddRoutingTableEntry	(string) Destination, (int) cost, (string) nexthop	
Adds a routing table entry given the destination, cost and next hop		
RTgetNextHop		(string) address
returns the next hop given the address of the destination		
RTgetCost		(int) cost
the cost of next hop to the given destination		
updateCost	(string) Destination, (int) cost, (string) nexthop	
updates the routing table entry with the given next hop and destination to have the updated cost		
processingComplete		
set the processed flag of the protocol packet being inspected to be true		
readyForDelete		
set the ready for deletion flag of the protocol packet being inspected to be true		
getBlackboardIndex		(int) position
returns the position of blackboard agent within the blackboard system		
setBlackboardIndex	(int) position	
sets the position of blackboard agent within the blackboard system		
isFSMA		(int) 0 or 1
Checks if the protocol packet being inspected contains an Fscript Mobile Agent.		
getPersistentStackSize		(int) stacksize
Returns the number of elements stored in the persistent stack		
peekPersistentStack		(object) Persistent Object
Peeks persistent object stack, returning the top most element without removing it from the stack.		

Table B.2: Instructions specific to Blackboard Agent Sun SPOT FScript Executers

Function	Input	Output
displayParentPacketDetails		
displays the details of the protocol packet in which the MA is contained. This includes the destination, next hop and direction		
getTemp		(int) Temp
Returns the perceived temperature in degrees Celsius		
getLight		(int) Light
Returns the perceived light intensity		
LEDsOn		
Sets all the LEDs to an on state		
LEDOn	(int) LED index	
Sets the LED at the given index to an on state		
displayCurrentTime		
Displays the current time in milliseconds		
logTime	(sting) log description	
Adds an entry to a logger. This displays the given description and the time at which it was logged		
getStackInfo		(string) Stack information
setDestination	(string) address	
Set destination of parent protocol packet		
addStackLayer	(string) layer name, (int) layer index	
Creates a new protocol layer of the given name at the given position in the stack		
removeStackLayer	(int) layer index	
Removes the stack layer at the given index from the protocol stack		
removeAgent	(int) layer index, (int) blackboard index	
Removes the blackboard agent in the given stack layer index with the given agent index		
readyForDelete		
flags the parent stack as being ready for deletion		

Table B.3: Instructions specific to Mobile Agent Sun SPOT FScript Executers