Using Local and Global Knowledge in Wireless Sensor Networks

Christopher Gwilliams
Cardiff University

February 11, 2014

Contents

List of Figures

List of Tables

Abstract

Wireless sensor networks (WSNs) have advanced rapidly in recent years and the volume of raw data received at an endpoint can be huge. We believe that the use of local knowledge, acquired from sources such as the surrounding environment, users and previously sensed data, can improve the efficiency of a WSN and automate the classification of sensed data. We define local knowledge as knowledge about an area that has been gained through experience or experimentation. With this in mind, we have developed a three-tiered architecture for WSNs that uses differing knowledge-processing capabilities at each tier, called the Knowledge-based Hierarchical Architecture for Sensing (K-HAS). A novel aligning ontology has been created to support K-HAS, joining popular ontologies from the sensing and observation domains. We have shown that, as knowledge-processing capabilities are pushed further out into the network, the profit is increased; where the profit is defined as the value of the sensed data received by the end user.

Collaborating with Cardiff University School of Bioscience, we have deployed a variation of K-HAS in the Malaysian rainforest to capture images of endangered wildlife, as well as automate the collection and classification of these images. Technological limitations prevented a complete implementation of K-HAS and an amalgamation of tiers was made to create the Local knowledge Ontology-based Remote-sensing Informatics System (LORIS). A two week deployment in Malaysia proved that the architecture was viable and that, even using local knowledge at the endpoint of a WSN, improved the efficiency of the network. A simulation was implemented to model K-HAS and this proved that the network became more efficient as knowledge was pushed further out towards the edge, by allowing nodes to prioritise sensed data based on inferences about its content.

Chapter 1

Introduction

A wireless sensor network (WSN) consists of a collection of heterogeneous nodes with sensing and, typically, wireless capabilities. These sensing nodes can be extremely complex and powerful devices with the ability to sense multiple phenomena simultaneously, or they can be simple motes that have limited processing power and are tasked with sensing one thing in their environment.

Upon deployment, these nodes use their wireless capabilities to form links with their neighbours, where a neighbour is any node that is within transmission range. The way that nodes discover, and communicate with, their neighbours is defined by their routing protocol. Routing protocols vary based on the purpose of the WSN, the requirements of data transmission as well as the characteristics of the nodes. Communication between nodes is expensive and drains the available power faster than any other action that a node performs. For example, if a WSN is deployed in a building with consistent power available, then the routing protocol does not need to be adapted to ensure the nodes maximise their battery life by transmitting as little as possible. However, not every WSN has unlimited resources at their disposal and these protocols, as well as the underlying structure of the network, are used to ensure the network is able to perform well for as long as possible.

Each WSN is different and each will have different constraints, a WSN that monitors traffic along a busy road may experience memory limitations,

whereas a WSN that is deployed in the middle of a desert may experience power issues. Typically, however, all WSNs do the same thing: sense one or more characteristics of their environment and forward that data on to a specified endpoint.

1.1 The Local Knowledge Problem

The majority of WSNs do not know what data they are sensing, or have any knowledge of their environment. This means that, unless fixed by a routing protocol or human that deployed the network, data is delivered on a chronological basis and is then filtered at the base station, usually manually. Some WSNs store all of the data on the node and users of the network must use a 'pull' model to query for data from nodes, but this requires some technical knowledge and, while it does increase the battery life of the nodes, it is a manual process again.

The environment that a WSN is deployed is usually rich and the data sensed often contains patterns that can be used to improve the performance of the network. For example, if a node knows that it is has only been triggering between the hours of 6pm and 5am for the past few weeks, it can enter a deep sleep outside of those hours or use that time to transmit data it has been storing while it knows it will be inactive. Alternatively, this knowledge can be used to prioritise data throughout the network so that the most important data is received first, instead of the most recent. An example of this could two camera nodes deployed facing the entry and exit of a building, tasked with looking for intruders between 5pm and 8am. If the camera facing the exit is triggered at 5:01pm and the camera on the entrance is triggered at 5:05pm, then the knowledge that the security guard leaves through the exit between 5:01pm and 5:08 pm will allow the entrance camera to prioritise its capture as more important, as it is an irregular occurrence.

This knowledge can be categorised into *local* and *global*. Local knowledge (LK) is the knowledge of an area that has been gained through experience or experimentation and global knowledge (GK) is knowledge that is generally available to everybody. An example of this is someone who has been tasked

with deploying a WSN in the Amazon rainforest would use readily available sources, such as the Internet or prior research, to determine the humidity and weather patterns in order to use a node that could withstand such conditions. This would be classed as GK. However, a native to the Amazon may know that three of the locations that the nodes are to be deployed in are flooded for two weeks of the year, rendering their readings useless for that time period and increasing their risk of failure. This is LK, as it cannot be gained without experiencing the flooding in that area, or experimenting with water levels.

We believe that the use of this knowledge can increase the efficiency of the network, as well as prioritise sensed data by its value instead of the time it was recorded. To show this, we have developed a network architecture for WSNs that utilises knowledge from the data it senses, as well as its deployed environment. It is called the Knowledge-based Hierarchical Architecture for Sensing (K-HAS) and this thesis will show how K-HAS addresses the problem of delivering the most important data first and improving the overall efficiency of the network.

1.2 Motivation

Throughout this thesis, we focus on a scenario motivated by our collaboration with Cardiff University School of Biosciences, who run a research centre in the Malaysian rainforest, in Sabah, known as Danau Girang (DG) . Located on the banks of the Kinabatangan river, DG has been running for more than six years and holds Masters, PhD and Undergraduate students from around the world, studying the ecology and biodiversity of the unique region.

The reason that the rainforest that DG is set in is so unique is that the area was heavily logged until the late 1970s and now serves as a corridor, between large Palm Oil plantations, connecting two separate rainforest lots. The area is now secondary rainforest (rainforest that has grown since being destroyed) and is experiencing a large variety of wildlife using the area as a habitat, or as a path. Some of this wildlife is unique to this area of the world and DG has had sightings of animals that have not been seen in many years.

There is a variety of research projects currently underway in the field

centre, looking into fish population, crocodile attacks, hornbill habitats or the movement patterns of small mammals. One project that has been running almost since DG opened, is the *corridor monitoring programme*, a programme that consists of dozens of wildlife cameras deployed in various areas around DG and triggered whenever an animal triggers a break in their infrared (IR) sensor.

The Kinabatangan is a very humid place, with thick forest, making it very difficult to walk through and even more difficult for hardware to survive the conditions. Cameras are placed along the river and up to 1km into the forest, recording triggers onto SD cards. These SD cards are collected and stored at the field centre, where the images are manually collated and processed. The cameras are designed to have a battery life of three months but, due to the humidity, a battery life of 3 weeks is more realistic. In 2010, twenty cameras were deployed and half of them were inspected every two weeks, on a rotating basis. In that time, each cameras can record more than a thousand pictures and the dynamic nature of the rainforest, such as the sun through leaves, falling trees and reflections in the water can cause the camera to trigger when an animal has not walked past; we call these *false triggers*. False triggers can make up to 70% of the images on an SD card and each of these must be manually processed.

We have used this scenario to test our hypothesis and implement a WSN that automates the collection, transmission, processing and storage of images, using LK to classify the data and prioritise the flow of information through the network, making more efficient use of the limited power and bandwidth available.

1.3 Research Contributions

Here we outline the main reason contributions explained in this thesis, we believe that two contributions has been made and our experimental results serve to support these contributions. Our primary contribution is that we propose a novel tiered network architecture, K-HAS, that utilises the local knowledge of its surrounding environment, users and previously sensed data

to process observations within the network and prioritise the data according to the inferred classification.

To formally define the structure of K-HAS and the data standard used, we created a novel aligning ontology that combined ontologies for sensor networks and scientific observations. This modular ontology can be used as a whole or in parts for WSNs that use some, or all, of K-HAS' architecture. The extensibility of the ontology allows parts to be removed and replaced where necessary.

Knowledge is pushed out to the edge of the network to allow the nodes that capture observations to prioritise the data based on its content. The knowledge processing capabilities increase with each tier as the data moves toward the centre of the network, allowing more detailed inferences to be made and data to be given a higher priority.This smart utilisation of bandwidth allows data to be delivered in an order that is more useful than chronological, delivering the most valuable observations first and using human feedback to learn how important these observations were.

In addition, K-HAS uses a feedback loop to dynamically update the knowledge base on every node throughout the deployment so it is able to react to changes within the data recorded, and the network, in near real-time.

Experimental results from the Malaysia rainforest showed that current technology is not yet at the point where K-HAS could be deployed and left without human intervention for months at a time. A modified K-HAS system LORIS (Local knowledge Remote-sensing Ontology-based Informatics System) combined tiers that use knowledge-processing at the centre of the network, with commercially available, wireless cameras. Which showed that knowledge-processing automates the handling of sensed data when it is received and can be used to infer patterns for future observations.

We model the ideal implementation of K-HAS in a simulation environment to prove that knowledge-processing capabilities utilised in all tiers of the network, right out to the edge, makes a WSN more efficient and ensures that important sensed data is delivered first rather than chronologically.

1.4 Thesis Structure

The rest of this thesis is structured as follows. Chapter 2 provides some background on wireless sensor networks and the use of knowledge. Chapter 3 explains technical decisions we made and the findings when running experiments in the Malaysian rainforest. Chapter 4 introduces the K-HAS architecture we have developed and explains the purpose of each tier. Chapter 5 details the ontology we have developed to support K-HAS and shows how current ontologies do not sufficiently cover all of the concepts involved with a *scientific observation*. Chapter 6 details our implementation in the Malaysian Rainforest and the changes we had to make in order for it to be feasible. Chapter 7 highlights the technical limitations of implementing K-HAS today and shows our simulations of it running as it was designed. Chapter 8 then concludes this thesis and summarises our contributions and findings, as well as highlighting work that could be undertaken to take this project further.

Chapter 2

Background

Using knowledge in a WSN is related to existing research into sensor networks that utilise context-awareness in order to improve their efficiency or adapt their sampling rate.

This chapter is split into the following sections. Section **??** outlines the issues surrounding WSN design and deployment. Section **??** details relevant existing routing protocols for sensor networks. Section **??** highlights popular sensor middleware in use today. Section **??** shows some examples of existing WSNs that are related to our motivating scenario. Section **??** introduces research into local and global knowledge and Section **??** shows some related work into WSNs that utilise knowledge or context to prioritise data and/or improve efficiency.

2.1 Wireless Sensor Network Issues

WSNs have been used in a number of domains, for a range of different purposes, from habitat monitoring [**?**] to military purposes [**?**] and healthcare [**?**]. While these applications are vastly different, the technology behind each is very similar. Each requires the use of nodes with sensors attached and each node requires a power source and storage devices.

According to [**?**], there are eight factors that affect the design of sensor networks, but we focus on a subset that are the most relevant to our research

problem. The following points must be considered when designing a WSN:

2.1.1 Fault Tolerance

WSNs typically contain a large number of nodes and each can fail for various reasons, from a lack of power, filling its storage capacity, to factors of the environment causing the hardware to fail. While the sensor nodes that are used in WSNs typically consist of the same platform, the variation between each deployment means that the device itself must be adapted to its environment, [**?**] used a custom protective casing for their nodes so that they were able to survive being in the open while ensuring that the transmission range was not affected.

2.1.2 Hardware Constraints

A sensor node typically consists of: a platform that contains the memory and processing power, a sensor (or sensors) and a transceiver that uses a wireless standard, such as Wi-Fi or Zigbee. Cost and size are the most common barriers to entry when designing a WSN. [**?**] mentions that the expectation of a sensor node is a matchbox-sized form factor. While the research is over ten years old, the original focus on sensor node was *smart dust* [**?**], small, inexpensive, disposable nodes that can transmit until their power reserve is depleted, [**?**] mentions that it is a requirement for the nodes to cost less than USD10. In [**?**], it is noted that, a decade on from the first WSN papers, smart dust has not been realised and the focus has instead been on larger, more powerful nodes that have reduced in cost and grown in power.

The Gartner Hype Cycle for 2013 [**?**] shows that smart dust is still in early innovation stages and may not be fully commercialised for another ten years. To counter this, research has been focussed on using software solutions to maximise the battery life in these more powerful, more expensive nodes, accompanies with the use of renewable energy sources.

2.1.3 Energy Constraints

The majority of sensor nodes do not have access to a constant power supply and must run on a battery that is, generally, a similar size to the node itself; or smaller. This means that the nodes must be as efficient as possible, knowing when to transmit data and when to sleep. The lifetime of a sensor network is extremely dependent on the battery life of each node and, unlike other mobile devices, they cannot typically be recharged [**?**]. Much work has been done on power efficient routing protocols, as well as the control of which attached devices are active [**?**, **?**, **?**].

The limited resources on the nodes mean that the sensing devices, and transceivers, attached must consume as little power as possible. Some routing protocols implement turning off wireless radios and scheduling a wakeup across the network [**?**], but the cost of turning off a device can waste just as much energy as leaving it on and sampling at a lower rate; if not more [**?**].

The use of energy in a node is dependent on how active the sensor(s) are, how much it transmits and receives, the transmission medium used as well as the environment it is in.

2.1.4 Transmission Medium

Common transmission media, such as Wi-Fi, are viable solutions in WSNs when a high data rate is required and power is readily available. However, research has shown that Wi-Fi is extremely power-hungry and [**?**] shows that Wi-Fi consumes almost 9 times more energy, while transmitting, than other standards, such as Zigbee. Bluetooth is a more power efficient standard that is becoming increasingly popular for sensing devices that are [art of the *Quantified Self* movement [**?**], with wearable devices that report measurements, such as heart rate, steps taken and calories burned. With the advent of the new low-power Bluetooh 4.0 , also known as Bluetooth Low Energy (BLE), this standard is supposed to allow months of continuous use on a coin-cell battery [**?**]. However, the range is limited to 100m and, using the same frequency, as Wi-Fi (2.4GHz) means that it is as susceptible to path loss and reduced transfer rates. [**?**] shows that a 2.4GHz Wi-Fi antenna is

capable of transmitting up to 350m, while a considerably lower frequency of 41MHz was able to achieve links of 10km. The use of 2.4GHz frequencies in wet conditions have been shown to reduce the performance by up to 28New low-power, low-frequency standards have emerged in recent years and allow for a considerably longer range and increased battery life, at the cost of transmission speeds. Digmish is an example of this and, while it can achieve 250kb/s using 2.4GHz, it has much slower speeds of 125kbps when using the 900MHz spectrum. However, it does offer a range of, up to, 64Km [**?**].

2.1.5 Environment

The environment that a node is deployed in can have a great impact on almost all aspects of a WSN, such as range and battery life. Harsh environments that are not easily accessible make it difficult to place nodes and protected environments may limit where nodes can be placed. In [**?**], nodes were deployed within glaciers and had to survive extreme temperatures, lasting without human intervention, for months at a time. [**?**] attached collars to Zebras that had to withstand high speed movement, dust and high temperatures. The deployment of any node requires extensive research as to the environment that it will be deployed in and adjustments must be made to ensure it is able to survive for extended periods without continued maintenance. Section **??** also shows that environment does not simply affect the hardware, but humidity can reduce the transmission range significantly, as well as moisture collecting on wireless antenna can reduce the range for days at a time.

2.2 Routing Protocols

Routing protocols specify how nodes in a WSN are organised, as well as how they transmit data throughout the network. In [**?**], the more popular routing protocols are surveyed and split into three of the main identified categories: data-centric, hierarchical and location-based. We use the aforementioned categories, as well as flat, to highlight some of the key protocols that are

relevant to our work. The protocols have the task of ensuring that a network is performing at its best, providing the best lifetime and ensuring reliable and consistent delivery of data. This must reduce *flooding*, where nodes send every message to every link, aside from themselves, effectively flooding the network with unnecessary messages, and find a way to deliver data to the endpoint using the most efficient path possible.

2.2.1 Flat

Initially, this was the most common structure for a WSN, dozens of nodes spread out over a geographical area, with one or more neighbours, sending observations to a single endpoint.

MCFA

The Minimum Cost Forwarding Algorithm (MCFA) is an flat routing protocol that works by assigning costs to each node, based on how many hops they are from the base station [**?**].

Each node has a path-estimate of the cost of transmission from itself to the base station. The base station sends out a broadcast message and it is received by all nodes in range. The message contains a cost from the base station (initially zero) while every node has their cost set to infinity. If the cost in the message, plus the link it was received, is less than the current cost. If yes, the estimate is update on both the node and the message; the message is then passed on to other nodes in range.

This approach allows for dynamic reconfiguration of the network, as well as a reduced overhead due to not having to maintain a global routing table on each node. The assumption with MCFA, however, is that the direction of routing is always towards a fixed endpoint.

2.2.2 Data-centric

Data-centric routing protocols are not like traditional WSNs where nodes are given addresses; they use a method that involves the advertising, or querying,

of the data that has been sensed and those with the relevant data can respond to the request.

SPIN

Sensor Protocols for Information via Negotation is one of the first data-centric protocols and attempts to address the issue of flooding the network whenever new data is sensed by addressing the data through metadata [**?**]. SPIN works on three messages passed between nodes:

1. ADV - A message sent by a node when it has sensed new data, advertising what it has recorded.
2. REQ - Sent by nodes that received an ADV to request the data.
3. DATA - Message containing the sensed data.

When a node has sensed data, it sends an ADV message to all nodes within range. If any of those nodes are interested in the data, then they respond with a REQ message, at which point the DATA message is sent to nodes that responded.

SPIN eliminates the need for a global view of the network topology, as nodes only need to know their single hop neighbours. However, SPIN does not guarantee equal diffusion of data throughout the network as a node may be interested in the data sensed at the other edge of the network, with only nodes that are not interested in between. This would mean that those nodes would not request the data or pass it on.

SPIN-IT

An extension to SPIN, SPIN-IT uses a slightly different approach to receiving data and is developed solely for the transfer of images [**?**].

Nodes use the existing message structure of SPIN, but REQ messages are used as queries, sent to all nodes in transmission range. The receiving nodes keep these requests and generate a new REQ message, thus allowing nodes to store temporal paths. When a REQ reaches a node that has the

desired data, it responds with a ROUTE-REPLY message. This message is used because images are large and resource-constrained WSNs would have a much shorter lifetime if a lot of unnecessary transmissions were made. The ROUTE-REPLY is used in case multiple nodes, in range of the requesting node, have the data it has requested and it can then choose the optimal route. As each node keeps a history of REQ messages, these can be used to trace the requested data back through the network, to the originating node, without the overhead of maintaining a global routing table.

COUGAR

A slightly different data-centric approach is the proposed COUGAR protocol, viewing the network as a distributed database. While similar to SPIN due to the fact that it does not forward data as soon as it is sensed, COUGAR uses a query language that abstracts the underlying network structure and uses that query to generate a plan that utilises in-network processing to provide an answer [**?**].

Within the network, a *leader* is selected and this node is used to aggregate the data from nodes that were able to fulfil all, or some, of the query. At risk of failure, each query should result in a *leader* being dynamically selected and it must have sufficient resources to be able to satisfy the request. This protocol was only proposed, and much of the technical detail has yet to be completed, but the concept of treating the network as a distributed database is a novel idea and this is one of the first protocols to suggest the use of a query language that could be used by people without specific domain knowledge.

2.2.3 Hierarchical

Hierarchical networks are WSNs that contain nodes of different classes; nodes at the end of the network are typically clustered into groups and served by a gateway node. This gateway could be in charge of aggregating the data, processing the data, or simply forwarding it to an endpoint. Clusters of nodes allow the network to be spread out over a wider geographical area and gateway nodes can use a different transmission method to provide long

distance links to the base station. Gateway nodes serving a cluster of nodes means that the network can scale easily as well, simply by adding a new cluster to the network.

TEEN

The Threshold sensitive Energy Efficient sensor Network (TEEN) protocol is designed for reactive sensor networks, networks that require instant reactions to changes sensed in their environment. TEEN recognises that transmission is the most power hungry action for a node so each node is coded with a hard and soft threshold. The hard threshold is a value that makes nodes transmit the reading to their cluster head. Similarly, the soft threshold is a small change in the value of the sensed attribute that causes further transmissions.

During the initialisation of the network, the base station sends information about the thresholds and sensing attributes to all cluster heads in the network; the cluster heads then forward this on to all nodes in their cluster. When a node senses data over the hard threshold, it transmits to the cluster and only transmits again when new sensed values are greater than the hard threshold and the difference between the current sensed value and the previous is greater than the soft threshold [**?**].

Clusters are assigned for a period of time and then new clusters are selected by the base station, at which point new attributes and thresholds are broadcast to all nodes. This kind of protocol allows the network to be dynamic after deployment and allows user input based on the data that has been sensed in the previous cluster times.

For example, a network could be tasked with sensing humidity in a rainforest but the thresholds have been set such that nodes are transmitting readings that are not of interest. A user can change these thresholds and they will be pushed out to the nodes at the time that the next clusters are chosen, without any need to visit the node or configure them individually.

2.2.4 Location-based

Instead of using the physical addresses of nodes, or the data they store, location-based protocols are based on the region that nodes are deployed in. **FLESH OUT**

Span

Span is a protocol where nodes are selected as *coordinators* based on their positions. A node can decide to be a coordinator based on the amount of energy it has and the number of neighbouring nodes it would benefit if they were able to use it as a bridge [**?**].

An example of this would be node B placed between node A and C. C and A are unable to communicate directly so, when node B wakes up, it decides whether it should become a coordinator. It knows that it has sufficient energy levels and it can provide connectivity for a previously disconnected area of the network, so it chooses to become a coordinator, staying awake and routing sensed data to other coordinators, which form the backbone of the network.

Results showed that using Span, in a system that transmits using 802.11, provides an network lifetime increase of more than a factor of 2 over networks that just use the 802.11 protocol.

GEAR

The Geographic Energy-Aware Routing protocol (GEAR) is similar to SPAN in that it makes routing choices based on both energy-awareness and location. Each node maintains an *estimated cost* and a *learning cost* of forwarding a packet through its neighbours. The estimated cost is calculated using the distance to the packet destination and the energy remaining on the node whereas the learning cost is the estimated cost that takes holes in the network into consideration [**?**].

GEAR is designed to perform in two phases: forwarding a packet towards a region and disseminating a packet within a region. When sending a packet towards a destination, GEAR either sends a packet on to the node in range

that is closest to the destination or, if such a node does not exist, then a hole is identified. If a hole is identified then the node that minimises a cost is selected.

To disseminate a packet within a geographic area, uses algorithms based on the density of the network. Recursive geographic forwarding is typically used but this can result in an endless loop if the density of the network means that the region is unable to contact the destination. In that case, restrictive flooding is used.

Provide a concluding section for routing protocols here?

2.3 Sensor Middleware

Acting as a bridge between the hardware and the user, sensor middleware is software that abstracts the underlying network from the user and provides a means of accessing sensed data and administrating how the network performs. These middlewares must not be specific to a single network and provide support for as many different sensor nodes as possible. In [**?**], a middleware is said to provide standardised services to many applications and perform operations that make effective use of limited system resource.

In [**?**], a middleware should include four major components: programming abstraction, system services, runtime support and Quality of Service (QoS) mechanisms. In this section, we will discuss the challenges surrounding middlewares for WSNs and highlight some existing middleware that are particularly relevant to our research problem and/or motivating scenario.

2.3.1 Issues

WSNs present a range of new challenges to existing middleware, due to their resource constraints, deployment environments and more. However, there has been research into the key issues that must be addressed in order for middleware to be considered suitable. While there have been a number of surveys into these challenges [**?**, **?**, **?**], we will detail those that we believe to be most relevant to our work.

Resource Constraints

It is rare that nodes in a WSN would have a constant power source, unlimited memory and a casing that can survive a harsh environment without decaying. In order to ensure that the lifetime of nodes is mazimised, middleware needs to offer a power scheduling system that makes efficient use of the hardware on the node, typically turning off the radio at particular times.

Ideally, a middleware will be able to coordinate nodes through wireless communication, making efficient use of transmissions and dynamically modifying sleep schedules based on the power remaining.

Heterogeneity

Not every node in a network will have the same capabilities, manufacturer or hardware. WSN middleware needs to provide a standard interface to add data, regardless of the format it is recorded in. Some middlewares have been built for a specific set of hardware, however this homogeneity can provide increase the performance and efficiency of the network by only supporting one device.

Real-world Integration

WSNs are often tasked with recording phenomena that are time-crucial, so a sensor middleware should provide a real-time interface to the data that it has sensed. Ideally this data would be available outside of the network, though the use of an API.

Quality of Service

This issue is perhaps the most complex as QoS could apply to almost all aspects of the networks, such as efficiently using bandwidth, 100% uptime for nodes, guaranteed packet delivery or access to data stores. Some of these requirements are managed by the implementation of the routiong protocol, the middleware should be able to monitor deployed nodes and report on their current status, as well as identify failures.

2.3.2 Existing Middlewares

In this section, we identify existing middlewares, explain how they address the issues highlighted in Section **??** and highlight how they relate to our research. While there are a lot of existing middlewares, our research did not show any that directly utilised the environment to make informed classifications. We did, however, find some applications that utilise context and rules to administrate the network.

FACTS

One such example is the FACTS middleware, an approach that uses a fact repository to coordinate nodes. Rules can then be implemented to process sensed data and fired when certain conditions are met [**?**]. More traditional sensor middlewares control the network and manage sensed data, this rule based approach allows for more flexibility, where rules can control the transmissions and process the data upon receipt.

Figure **??** shows the FACTS architecture, with the middleware holding the rulesets and a dstributed fact repository. Data within the network is stored as facts, providing a standard data format throughout the network and hardware abstraction. When new facts are received, ususally because of new sensing data, the rule engine checks to determine whether any rules should be fired. The ruleset definition language (RDL) is introduced here and each ruleset contains a group of relevant rules. Each rule is given a priority so that, if more than one rule is triggered by a fact, then the higher priority rules are fired first.

While FACTS itself does not utilise any local knowledge, the repository is used as a source for all previously sensed data and would prove as an excellent source of knowledge to assist with the classification of future readings. Also, the ability to add new rulesets, without technical knowledge of the hardware of each node, means that users of the network have the ability to add knowledge in the form of rules as they learn it.

ITA Sensor Fabric

The ITA Sensor Fabric is collaboration project between IBM, the US Army and the UK MoD. Sensor Fabric, or Fabric, is a two-way messaging bus and set of middleware services connecting network assets to each other and users [**?**].

The core difference between the Fabric middleware and others is that not every node is sensing all of the time, sensor nodes are tasked when there is a requirement and they stop as soon as that task has been fulfilled. Similar to sinks in a traditional WSN, Fabric utilises Fabric nodes, which run the following three pieces of software:

1. Message Broker - Provides the communication infrastructure.
2. Fabric Registry - Holds information about the current deployment, such as all nodes deployed, all assets, routing information and tasks. Deployed in the form of a database.
3. Fabric Manager - The main service on the node to track the status of connected sensors, establish communication channels, provide a container for processing, plug-ins and to extends the capabilities of the Fabric.

Fabric runs on a Publish/Subscribe model, a sensing requirement is sent to a messaging broker as a subscription and this is distributed through all Fabric nodes and, thus, all sensor nodes. Sensor nodes then publish their data and the relevant data is sent to all applications that have subscribed to the data.

The plugin structure of Fabric makes it stand out from existing middlewares, allowing its functionality to be extended through web interfaces.

Because Fabric has been developed for military purposes that cross countries, policy enforcement has been implemented to restrict access to the granularity of sensed data but these access levels do not simply apply to a military context. Using our motivating scenario, researchers and professors should see animal images whereas the Sabah Wildlife Department should see images of hunters and people in the forest.

GSN

The Global Sensor Networks (GSN) is a middleware that has been developed to manage heterogeneous sensor networks and be suitable for those without any technical knowledge [**?**].

GSN provides hardware abstraction through the use of *virtual sensors*, a data stream that abstracts implementation details from the actual sensed data. A virtual sensor can be comprised of many streams and it can even consist of many virtual sensors.

Virtual sensors are described using XML, with tags that consist of metadata for the sensor, the structure of the incoming data stream, SQL queries for processing the incoming data and querying times. What makes GSN stand out is that virtual sensors do not have to be sensors deployed within your network, or even sensors at all, some examples of GSN show virtual sensors being added that read in data from the weather websites. This also means that the underlying structure of the network is irrelevant to GSN, as well as the physical locations of the nodes. Unlike some middlewares that have an expectation of how data will be routed, GSN is decoupled from the routing protocol, allowing them to act independently.

GSN is completely open source and, while it does not provide the plug-in architecture that is available in Fabric, the Java code can be modified to suit a specific deployment.

Figure **??** outlines the architecture of GSN, showing that the virtual sensors are stored on a central node and their inputs are managed and stored. GSN also comes bundled with a web interface to show all active sensors and their most recent recordings, as well as the implementation of web services to access the data outside of the interface.

Data from virtual sensors pass through the virtual sensor manager to the storage layer. Once the data has been stored, the query manager is invoked and queries are loaded from the repository and executed by the manager. The results of the queries are then handled by the notification manager and also made available to the web interface. Notifications can be extended to support many different forms of communication, such as SMS, email or web

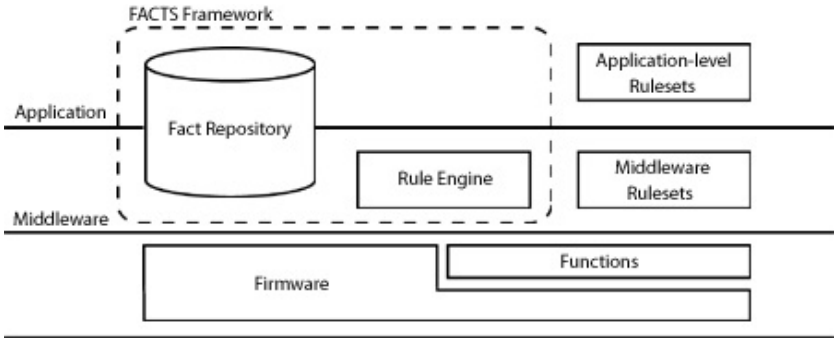

Figure 2.1: FACTS Architecture

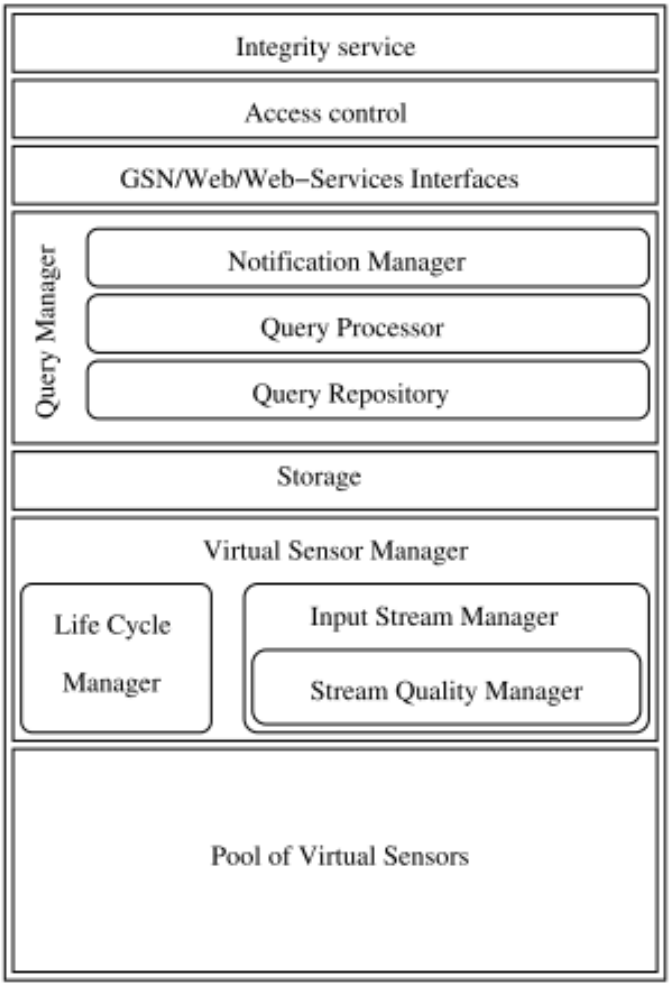

Figure 2.2: GSN Architecture

services.

Virtual sensors do not natively support all hardware, although new virtual sensors can be described using XML, and there may be a need to implement an entirely new virtual sensor. In this case, technical knowledge is required, and new sensors can be implemented through use of the Java programming language. This provides more control over the use of XML and allows users to specify how sensed data is stored in a database, use external libraries to receive proprietary data, specify processing workflows before the data is stored or implement new notification methods for the users of the network.

To show the simplicity of a basic virtual sensor, [**?**] describes a temperature sensor that we reproduce here in Listing **??**. The file is human-readable and has a shallower learning curve than programming languages, with tags that explain what data they contain. In this example, the output structure shows that only a temperature reading is received and that the data should be stored permanently. The stream source specifies the content of the stream and the query details the standard query that should be used to extract data from GSN.

```
<life-cycle pool-size="10" />
<output-structure>
        <field name="TEMPERATURE" type="integer"/>
</output-structure>
<storage permanent-storage="true" size="10s" />
<input-stream name="dummy" rate="100" >
<stream-source alias="src1" sampling-rate="1" storage-
    size="1h">
<address wrapper="remote"> <predicate key="type" val="
    temperature" /> <predicate key="location" val="bc143
    " /> </address>
<query>select avg(temperature) from WRAPPER</query>
</stream-source>
<query>select * from src1</query>
</input-stream>
\caption{Virtual Temperature Sensor}
```

```
\label{bg:lst:gsn}
```

The modularity and flexibility of GSN makes it different to existing middlewares as it has not been designed for any specific hardware and modules of the middleware can be replaced, such as the database.

2.4 Biodiversity Sensor Networks

In this section we will cover existing WSNs that are related to our motivating scenario or, more specifically, biodiversity focussed WSNs that have been deployed to monitor wildlife and/or the environment. WSNs for habitat, and wildlife, monitoring are especially important because these are areas that often need to be untouched by humans. Areas with high human disturbance can influence the abundance of species and some habitats, i.e. underground burrows, may be impossible to monitor without destruction.

One of the most well known WSNs to monitor habitat is the network deployed on Great Duck Island (GDI), an island off the coast of Maine, USA. A network consisting of 32 nodes was deployed to monitor a bird, known as the Leachs Storm Petrel [**?**]. This network used a clustering approach for groups of nodes to send data to a gateway, which would then route it back to the base station. The base station, located a few kilometres away on the island, has internet access and uploads the data to allow users to browse and process the data.

A multihop approach was used here as they found that, for sufficient coverage, single hop connectivity would not cover all of the island. Acrylic enclosures were developed to ensure the nodes were weatherproofed for the conditions of the island, while maintaining the functionality of each sensor and not impeding transmission range. While the nodes, their casing and their sensors have been designed specifically for the deployment on GDI, the success of the network, running for 123 days in the early stages of WSN research [**?**], shows that this approach can be used elsewhere with similar effects; allowing hard to monitor and/or inaccessible areas to be continuously monitored.

On a smaller scale, INternet-Sensor InteGration for HabitaT monitoring (INSIGHT) is a single-hop WSN that allows remote access for data and reconfiguring of nodes [**?**]. While this network does use commercial hardware, their findings do show that their nodes could survive for 160 days on a single battery, supporting their claim that a single hop network allows for a longer network lifetime.

The key feature of this network is the ability for humans to remotely set reporting thresholds for sensor nodes. This means a user can prolong the lifetime of nodes by limiting the threshold they report on, as well as the fact that these thresholds are a way for users to add knowledge, albeit primitive, into a network.

While there is research on cameras used to monitor animals [**?**, **?**], these networks are generally cameras deployed with their memory cards manually retrieved and processed. In recent years, however, the use of wireless technologies and image-based WSNs has increased, [**?**] uses wireless cameras to monitor the movement of animals between roads. Using commercial hardware and controlled sleep scheduling, this solution employs the use of nodes to detect movement and wake up more power-hungry camera nodes. While the nodes are wireless, the distance of the network from civilisation means that the data does still need to be collected manually and uploaded to a computer.

Due to the advent of smartphones and tablets, as well as the improvements in 3G technology, projects taking advantage of more modern technologies have grown in popularity. Using 3G enabled cameras, [**?**] have deployed a number of devices in locations all over the world, such as: Kenya, Indonesia and the USA. The images captured are transmitted to a server and a website allows the general public to not only see the images in near real-time, but to classify the images as well. This crowdsourcing of collective knowledge lets people, that may not have domain knowledge, vote on an image and those votes are used to make classification easier.

Over the past fifteen years, WSNs have grown from a concept to a real solution for monitoring the habitats, movements and eating habits of wildlife all over the world. Whether it is using GPS collars to monitor the movement

of cattle [**?**], monitoring animal habitats on a remote island or using cameras to capture the animals themselves, the popularity of these networks has grown considerably and advances in technology have allowed these networks to be deployed in places that humans cannot.

2.5 Local and Global Knowledge

The environment of a sensor network is rich and varied and we believe that patterns in the data sensed can be used to inform the network on decisions surrounding the transmission and processing of newly sensed data. As our research began, we simply called this knowledge but, as our work continued, it became apparent that it could be split further.

While we believe that we are the first to use the concept of local and global knowledge within the wireless sensor network domain, the terms have been around for many years. In 1999, a book that referred to local knowledge as *indigenous knowledge* defined local knowledge as systematic information that remains in the informal sector, usually unwritten and preserved in oral traditions rather than text [**?**].

Over the past twenty years, local knowledge has been used in various contexts, from researching lending and the credit market [**?**] to extracting local knowledge from natives to improve farming techniques [**?**]. This research, as well as work that will be covered later, showed us that there are two kinds of knowledge: global and local.

It was from agriculture research that we were able to refine our definition of local knowledge, [**?**] defines local knowledge as knowledge that farmers have derived locally through experience and experimentation. They also say that indigenous knowledge is different in that it is culturally specific. From this definition, as well as our work with our motivating scenario, we were able to generalise the definition and expand upon it.

We now define local knowledge as *knowledge of an area, held by a domain expert, that has been gained through experience or experimentation.* This then means that global knowledge is *knowledge of an area that can be accessed by anyone, without the need to visit the area directly.* The weather of a region is

global knowledge because it can be found through a variety of media, whereas the level of a rain for a field within that region would be local knowledge, as it would require experimentation.

Using these definitions, we believe that encoding local and/or global knowledge onto sensors will inform routing decisions to make better use of the bandwidth in resource-constrained WSNs by sending data that it believes to be important first, rather than just chronologically. Patterns in the data, and knowledge of the environment surrounding a node, will allow a node to infer what the data may be classified as, automate the classification process, learn from previously sensed data and utilise global knowledge of ongoing projects within the network to determine what data is thought to be of a higher priority.

2.6 Relevant Existing Networks

In this section, we discuss existing networks that have done work relating to our research question and/or our motivating scenario. While most of the existing networks relevant to Danau Girang have already been covered in Section **??**, there are some WSNs that are not directly related to biodiversity but have been deployed in harsh conditions or involve interdisciplinary collaboration.

2.6.1 Context-Awareness

While standard WSNs have been prevalent for many decades, new research on the Internet of Things [**?**] has brought about interest in context-aware sensing, in some cases this is for small wearable devices to track fitness but the applications are much broader. Here we will look at WSNs that use context to make informed routing decisions, save power, or prioritise the transmission of data.

Health

Health monitoring is one of the more obvious choices for context awareness as classifying readings can often help determine the health of someone, rather than their self-reports. AlarmNet is a WSN that uses context to provide long-term health monitoring for people in assisted-living environments.

AlarmNet employs context-awareness to learn about the activity levels of the patient and uses that knowledge to determine when changes in the readings may mean that the patient is at risk. Once the AlarmNet system has completed its learning period, deviations, from what it has recorded as the normal activities for the patient, are sent to nurses and doctors, with the idea that this information can assist with a diagnosis.

As some nodes in the system will be battery powered, AlarmNet also employs a subsystem, called the Context-Aware Power Management System (CAPM), to control the power consumption of a device based on the recorded activities of the patient. The system uses policies, based on the context, to save power for all nodes in the system. For example, the system could put mains-powered nodes into a low-power state and disable all nodes outside of the bedroom when it detects that the patient has gone to sleep.

This use of context has not only shown that it can assist with the lifetime of a network, but it can also provide valuable insight into sensed data to provide data enriched with semantics.

Wearable Devices

As wearable devices, such as the Fitbit and the Pebble smartwatch, are increasing in popularity, context-awareness is a useful tool to differentiate between the many activities that a person can undertake. The MObile PErsonal Trainer system (MOPET) is a wearable fitness devices that uses context to record data on jogging and fitness exercises [**?**].

While the project is five years old now, it is one of the earliest proof-of-concept devices created and shows how simple rules applied to sensor readings can be used to apply context. For example, the device consists of a GPS sensor and, when active, it is constantly recording positions. Using

pairs of points, it is able to calculate the speed and, if the speed is consistent with jogging, then it records a running exercise.

Fitness is not the only purpose for wearable computing, the eWatch is an early design for a *smartwatch* that uses a microphone and light sensor to record the locations that the user visits, storing previous recordings in flash memory and matching them with current recordings [**?**].

UNETS

*UNETS is tiered and context aware

2.6.2 Harsh Environments

There are a number of situations where we may want to record data in an environment that is not safe for humans, such as a volcano [**?**], or that may be difficult for electronics to survive. In these cases, special considerations must be made during the design and deployment of the WSN, in order to ensure maximum network lifetime with reliable readings.

GLACSWEB

Glacsweb is a sensor network to monitor the rate that which glaciers are melting. Deployed in Norway, specially designed sensors have been drilled into glaciers to monitor pressure, temperature, orientation and strain. Due to the high pressure and exposure to moisture, a polyester casing was used so that, once bonded, the node inside would not be recoverable.

Reventador

Volcano Reventador is in northern Ecuador and a WSN has been deployed on there to monitor eruptions. Similar to Glacsweb, weatherproof enclosures were used to prevent ash and moisture from breaking the sensors and long-range external antenna mounted to a pole was used to achieve communications over large distances when wireless communications have proven to be difficult [**?**].

Rainforest

There have been many studies on how the rainforest affects the range and quality of wireless links [?, ?, ?]. In [?], the humidity was shown to reduce 802.11 range by up to 78% and [?] explains that periods of rainfall reduce the link quality up to 100% in some cases, resulting in the loss of a hop and this could prevent data from reaching the endpoint.

2.7 Summary

In this chapter we have explored the components that make a WSN, as well as some existing deployments that are relevant to our motivating scenario. Sensor middlewares have come a long way in the past decade and increased capabilities for sensor nodes have allowed for more intense processing to be carried out before any data is seen by a human. Context is now used on sensor nodes to infer the activity it is undertaking or even to determine when it should wake to sample.

Local knowledge is not a new concept and it has been used for many years to extract information from indigenous people and in industry to adapt their processes to a local area, such as farming. However, we believe that our work is the first to apply local knowledge in the context of WSNs, but context-aware sensor networks support our hypothesis that knowledge of its surroundings, or of the data it has sensed, can improve network performance.

The two primary components of a WSN that could be injected with local knowledge is the middleware and the routing protocol, each providing different benefits. Existing work has shown middleware that uses context-awareness can use that to make global changes to the network, such as power management, whereas routing protocols affect the data that is sampled and sent to the endpoint(s) of the network, such as adaptive thresholds.

One issue in proving this hypothesis is the deployment of a network that utilises local knowledge. The deployment environment of our motivating scenario is not only interdisciplinary, it is in a region that is humid, dynamic and dense. Existing research has shown that the deployment of a WSN in

these considerations means that range will be greatly reduced and changes in humidity can prevent communication altogether, as well as moisture affecting the hardware itself. To deploy a network, hardware must be adapted and the right medium must be chosen in order to maximise link quality and minimise dropped connections.

In Chapter 2, we show how we used this research, as well as our own findings, to choose suitable hardware and experiments, that were informed by previous rainforest range tests in the literature, supported our choice for transmission medium.

Chapter 3

Technical

In this chapter, we explain our motivating scenario in more detail, explore the sensor hardware that we researched and outline the results of experiments we undertook in the Malaysian rainforest. As highlighted in Section **??**, we have been working with Cardiff University School of Bioscience to design and deploy a WSN that utilises local knowledge, using an area of rainforest in Malaysia owned by the Sabah Wildlife Department, called Danau Girang.

The structure of this chapter is as follows. Section **??** explains what we aimed to deploy in Danau Girang and what our considerations were. Secion **??** introduces sensor hardware that is in use today and details the choices we made. Section **??** details the transmission medium choices we tried and also shows the results of experiments performed in both the UK and Malaysia. Finally, Section **??** summarises our findings and explains the choices we made for the sensor nodes we used in DG.

3.1 Danau Girang

Based in Sabah, Malaysia, Danau Girang is a field centre located in Lot 6 of the Lower Kinabatangan Wildlife Sanctuary (LKWS), surrounded by secondary rainforest that had been logged up until the 1970s. Experiencing typical wet and dry seasons, the LKWS can receive more than 500mm of rainfall during the rainy season, dropping to lows of around 150mm during

the dry seasons [**?**], and up to 100% humidity all year round.

Danau Girang is uniquely situated in a rainforest corridor that joins two areas of rainforest together, with the corridor surrounded by palm oil fields on each side. Because of this, animals use the corridor to move between the rainforest regions and some use it to enter the palm oil plantation for new feeding grounds. This gives Danau Girang insight into the movement patterns of these animals in the corridor as well as in the rainforest itself, with a wide variety of species that are not commonly seen in other tropical regions of the world. Due to the remote nature of the centre, power is provided by a set of diesel generators which, typically, provide power from 10 am to 1 pm and 5pm to 11pm daily. Wireless Internet access is provided by satellite with speeds comparable to that of 56k, although the upload speeds are considerably faster than downloads.

As outlined in Section **??**, the corridor monitoring programme is a scheme that has been in place for more than five years to use wildlife cameras to track the movement of animals through the corridor and to capture species that are rare or unique to South-East Asia, such as the Bornean Clouded Leopard or Sun Bear. Currently, the use of Reconyx Hyperfire HC500 cameras are being used [**?**]. These are standalone cameras that store a pre-defined set of images to an SD card on each trigger, which is triggered by an infrared (IR) motion sensor when the beam is broken.

Images must be collected manually every two weeks from the cameras and the batteries are changed at that point as well, although a typical charge should last three months. The cameras are equipped with watertight casing but, due to the humidity and the opening of the cameras every 2 weeks, silica gel is used to prevent moisture inside the camera. We believe that humidity also reduces the battery life, as the charge drops from three months to around three weeks within the first few months of usage. However, the lack of constant power availability could also negatively impact the charge cycles of the batteries when at the field centre, reducing their capacity. Each unit is secured to a tree and more dangerous sites, such as known elephant paths, have protective cases as well.

In 2010, twenty cameras were deployed for six month periods and then

relocated based on the needs of the projects at that time. As of 2013, there are now ninety with a view to expand and dozens of projects within the field centre use the images gathered from these cameras. Initially, it was the job of visiting research students to collect the images but, since the number of deployed cameras has grown, full-time staff have been taken on to maintain them.

Our belief is that we can use the LKWS, and the locations of the existing cameras, to deploy a WSN that uses local knowledge gained from the researchers at DG to automate the collection of images, improve the battery life by not exposing the internals of the camera to the elements so often and, most important, prioritise the flow of data through the network by in-network processing.

Annual visits, lasting three weeks, have been made to DG to test out hardware, software and wireless choices, in an effort to optimise the network. These visits have also been used to extract local knowledge from the area and researchers, by semi-structured interviews and watching them work.

3.2 Hardware

Before any visits were made to DG, meetings with staff members of the field centre were held in order to gain a better understanding of the environment and the project. This is where we were alerted to the humidity of the region and the fact that the failure rate of the Reconyx cameras has been as high as thirty per cent.

Reconyx cameras have no external interface support and the only way to access the images is through the removable SD card, because of this there is no way of attaching external sensor hardware to the existing cameras. In-network processing is an important requirement for our WSN and this did limit our choices to nodes that are computationally capable than more common sensors, such as the IMote 2 [**?**].

In this section, we detail our research into suitable sensor hardware that met the following requirements:

1. Able to perform processing of images and metadata
2. Common interface availability (Serial, USB)
3. Wireless enabled
4. Battery-powered
5. Expandable memory

This section also details the modifications we made to the devices in order to ensure they would survive in a humid environment.

3.2.1 Pandaboard

Texas Instruments supported the development of a reference Single Board Computer (SBC) that had specs similar to that of a modern smartphone and was capable of running desktop Linux, known as the Pandaboard [**?**]. A dual core 1GhZ ARM processor with 1GB of RAM, support for external storage, expansion ports, USB, Wi-Fi and Bluetooth in a board the size of two credit cards can be a powerful addition to a data heavy WSN, especially one that deals with images.

There is no mention of Pandaboards in the literature being used in WSNs but, the low power of the system and advanced capabilities, make it suitable for processing and transmitting data simultaneously.

3.2.2 IGEP v2

The IGEP v2 is another ARM based SBC that uses a 1GhZ single core processor with 512MB RAM and similar connectivity features to the Pandaboard, but around half the size. This does result in a reduced power draw and the device is still capable of running desktop Linux.

Due to the smaller size, and easier commercial availability, the IGEP has been used as a sensor node to record, process and send readings from multiple devices, such as air temperature, GPS and oxygen saturation as part of environmental monitoring [**?**]. In their research, they found that the IGEP

achieved 9.1 hours of uptime using a 4000mAH battery, a capacity used in many modern smartphones.

3.2.3 Waspmote

The Waspmote is a general purpose sensing board that is designed to allow plug-and-play connectivity for multiple sensor modules. The node can be programmed with C++, using the prepackaged SDK. The processing power is not comparable to the more powerful SBCs, but the 600mAh battery is reported to last three months and the size is much smaller [**?**].

One of the benefits of the Waspmote is that they are commercially available with an actively maintained programming environment.

3.3 Wireless Range Experiements

In this section, we explain the experiments we carried out to test the performance of different transmission media in the UK as well as the Malaysian rainforest. While range is the most important feature, a data rate that can handle hundreds of large readings in a day is a requirement. Our motivating scenario is focussed on the transmission of sets of 3 images for every trigger; where a sensor can trigger hundreds of times in a day.

3.3.1 Wi-Fi

Wi-Fi was already available on our initial test platforms and the high data rate made it suitable for sending a large volume of images in a short period. We knew that current cameras deployed in DG were up to 1km apart and we did not expect to cover that range completely, but were did anticipate that coverage with intermediate nodes.

Research, outlined in Section **??**, showed us that the rainforest could reduce the range by up to 78% and the ideal maximum range of 2.4GhZ Wi-Fi is 100m [**?**].

We tested Wi-Fi range using two IGEP boards, powered by 4 D Cell batteries and running a lightweight Linux operating system. The IGEP nodes

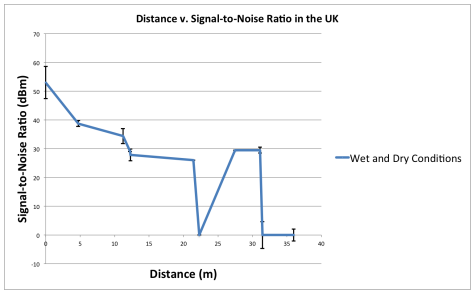

Figure 3.1: Signal-to-Noise Ratio for Wi-Fi in UK Woodland

we used did not have any additional hardware and the nodes were tested without the use of an external antenna. A Java application was written to periodically scan for available networks and store those results in a text file. One IGEP board was set as the base station and attached to a tree, at the same height it would be if it was attached to a camera, and another was walked to specified points around the base station at defined locations. These locations were chosen to include as many distances as possible and as many different forms of obstacle between the searching node and the base station, such as: line of sight (LOS), medium vegetation or thick trees.

This experiment was run in a wooded area in the UK and in the rainforest at DG. The specified maximum range of 802.11g is 120m. When considering attenuation and obstacles we were expecting the signal to be reduced by up to 50% in the UK. However, we found that we received a maximum range of 30m, with LOS. Figure **??** shows the results we experienced, while testing in Cardiff, some of the drops in signal can be attributed to dense foliage and readings that were not LOS, but a maximum range of 31m, with an SNR of 29.5 dBm, is less than we expected, as the UK does not experience high humidity often.

The graph does show a drop at 22m, this was due to the dense foliage that restricted the LOS between the base station and the receiving node, with five runs of this test we observed the same results. The primary aim of this experiment was to prove the viability of Wi-Fi and to to ensure our application functioned as intended, which it did. Further experiments could have been run to remove the anomaly but the results of the experiments in

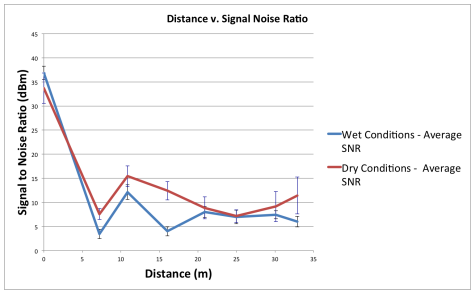

Figure 3.2: Signal-to-Noise Ratio for Wi-Fi in Malaysian Rainforest

Danau Girang were the focus.

Despite the poor range from the tests in the UK, it was consistent with other studies reporting signal degradation of up to 78% in areas with moderate foliage. We visited Danau Girang in 2011 to gather the requirements of the network and ensure the hardware is able to survive the humidity. Range experiments were run in the rainforest to see if a more humid environment impacts range any further, Figure **??** shows this.

Comparing figures **??** and **??** shows that the maximum distance to receive a signal is approximately the same in Malaysia as it is in the UK. There are more signal drops but this seems to be due to denser foliage blocking the line of sight. However, it does suggest that the humid environment of the rainforest does not have a significant impact on the received signal. It is clear that the denser rainforest does impact the signal-to-noise ratio in a much shorter distance from the base station but a link is still made, allowing for a successful transmission of data.

Due to the poor results of these experiments we researched alternative methods to increase the range without impacting the environment the network is to be deployed in. We considered using intermediate nodes, not attached to cameras, to account for the lack of range but, because some cameras can be up to 1km apart, we would need more than 30 nodes to create a connection between two locations.

We also researched wireless technologies that are more common in sensor networks. This does mean that the data rate is not as high as Wi-Fi and error correction in packet streams is not always as robust, but it is more

suited to sensor networks, using less power and providing longer range.

Finally, we considered using the researchers or animals at Danau Girang, as 'data mules', creating temporary links between nodes while they are in the forest. However, the trip to Danau Girang yielded the information that researchers generally do not cover those distances in the forest and data delivery would be sporadic.

Although the range of Wi-Fi is poor, for our requirements, in both Malaysia and the UK, it did show that the results we experience in the UK are very similar to the results in Malaysia. This means that tests run in the UK should be indicative of what we can expect in Danau Girang.

3.3.2 Digimesh

Due to the poor range results of Wi-Fi, we created a second prototype of the network, using Digimesh. Digimesh is a proprietary wireless protocol, based on the 802.15.4 standard and designed for devices with limited power. Using the same frequency as Wi-Fi, Digimesh has been reported to provide 7km of range, with a data rate of 250kbps.

In our prototype implementation, we are using Waspmote sensor boards [**?**], a general purpose node that is capable of transmitting through various communication mediums. Our Waspmotes are provided with Digimesh modules and a 2GB SD card to store sensed data.

When testing the range of the Waspmotes, we followed a similar method to that which is outlined in Section **??**. One board is in a fixed location and running a C++ application to poll for nodes in the network, once a node is found it sends a message to the node every 10 seconds. The second board is set to scan the network and receive packets as soon as a base node is found, this node is then moved to different locations.

The receiving node prints out variables related to the received packet, such as: RSSI, source MAC address and packet ID. However, not all packets are received so the RSSI can display 0 if there are errors reading or if packet collision occurs. We found this to affect the results and have just used the two nodes to identify the maximum distance they can be apart, while maintaining

a stable connection.

Initial experiments were run in a moderately vegetated area in the UK which yielded 497m of range. Limitations with buildings preventing us from testing any further but the signal strength still proved to be strong.

The initial results for the range tests proved positive and Digimesh does seem to be a viable solution to account for the lack of range when using Wi-Fi. As the frequency is the same as 802.11g, thus licensing it for worldwide use, we expected similar results in Danau Girang..

Experiments were run in 2 areas of the rainforest around Danau Girang and the results yielded were not the same as we experienced in the UK, and thick vegetation proved to have a significant impact on the range, reducing it by almost 50%.

In more open areas of the rainforest, we achieved 199m on average, more dense regions of the forest reduced this to 102m on average. While these results are not as high as we achieved in the UK, they are still suitable to use Digimesh in the deployment of a WSN. This could be because we were using low-gain antennas and little configuration had been made on the Digimesh radio.

3.3.3 Adapting for Harsh Environments

All of the hardware that we used for experiments had their components exposed and would have become compromised if moisture came in contact with them. To protect them from this, we used waterproof cases with a protective foam inside, known as Pelican cases, to keep the nodes watertight, but still allowing airflow, shown in Figure **??**. External antenna can be fed through the lip of the case, using thin cable, ensuring that the range of the transmissions is not affected by the case.

Figure 3.3: Pelican Waterproof Case

3.4 Software

3.4.1 Triton

Triton is a C++ program that makes use of the Open Computer Vision (OpenCV) library for performing popular image processing functions. Triton takes in a set of images, combines them and builds a background model. Using this model, the foreground is extracted from the image and this is scanned for objects. If an object is found, it is extracted to a separate and saved as a black and white template, to minimise space and assist with future classifications.

Primarily, we developed this tool to detect the huge number of empty images captured around Danau Girang, due to trees falling, movement of the Sun, fast animals or dirt on the lens. However, Triton proved to be quite effective when finding images of interest and it was modified to be used with our sensors. In this section, we explain how Triton was tested and compared with human observations. To ensure a valid comparison, we also compared both results to a random classifier.

During a typical three month deployment along the Kinabatangan River more than 40,000 images can be taken. A large proportion of these images can be classed as false triggers.

Using the images, taken in sets of 3, and separating the images taken by specific cameras, we build a Gaussian background model and used that to detect animals in the foreground, classifying the detected foreground as the

ROI, and extracting it.

As mentioned in Section **??**, Figure **??** shows an image taken by a wildlife camera. The dynamic background and light levels should be noted here. This image is number 1, in a set of 3, a background model is built from these images and added to the pre-existing background model built by that camera. The foreground of the image is then extracted and ROIs, larger than a threshold, are identified. The largest ROI is then extracted from the image and saved separately.

From our visit to Danau Girang in 2011, we collected images from two different three month deployments, in two different lots in Danau Girang, giving us just over 70,000 of images to test our approach. The images are sorted by the camera and the date of collection. We process every 3 images as one set, building a background model of all three images.

We manually process all of the images initially, marking images that are empty as false triggers. We then process the images, using our application, and a resulting processed image is created from every set of 3 images. If nothing is detected in a set then no image is created and that set is logged as empty.

There are four classifications that can be made with images sets:

True positive: An ROI is extracted that contains the animal in the set.

False positive: An ROI is extracted that contains nothing of interest.

True negative: A false trigger is correctly identified and no ROI is extracted.

False negative: An image with interesting contents is classified as a false trigger and no ROI is extracted.

The processed images are then compared with our manual findings. The accuracy of our application is calculated by the following equation:

$$Accuracy = (T_p + T_n)/TotalSets \tag{3.1}$$

Where T_p is the number of true positive sets extracted and T_n is the number of true negative sets.

True Positive	True Negative	False Positive	False Negative	Total Image Sets
77	201	5	17	300

Table 3.1: Classification Results

Table **??** shows experiments run on a camera deployed in Danau Girang, testing on 300 sets of 3 images. These images were manually processed and classified as interesting or empty, of the 300, 94 sets were identified as interesting. Using Triton, 53 of those interesting sets (*true positives*) were extracted, with another 17 *false positives* extracted. 201 *true negatives* were correctly identified and a further 5 were identified as *false negatives*.

Out of the 94 interesting sets, 77 were identified correctly, giving an accuracy of 82% for finding sets of interest. Furthermore, of 206 empty sets, 201 were found, which gives a 98% accuracy at finding empty images.

These preliminary results show that our method is effective at detecting false positives but is less effective at detecting false negatives. It appears that these misclassifications primarily come from black and white images taken at night, images where minimal movement of the animal has caused a trigger and images where an animal has caused a trigger but it has been too fast moving to be in the second two images.

After a longer deployment, we would be able to build up a more substantial background model to account for some of the animals being less dynamic in images and we expect this to decrease our error rate thus reducing the number of false negatives.

The results of Triton were compared to a random classifier, implemented in Python. Table **??** shows the outcome, with 72 *true positives* extracted, resulting in an accuracy of 80%. 73.3 *true negatives* from a total of 206 were found, giving an accuracy of 36%. Although this set does not compare to the number of images collected in a six month deployment, human classification is a time consuming process and to have 900 classifications is complex. Although these could be crowdsourced, the reliability becomes questionable, in part due to the specialist nature of the images. This does show that our approach is quite close to the accuracy of a human when finding images of

True Positive	True Negative	False Positive	False Negative	Total Image Sets
72	73.3	78.3	75.3	300

Table 3.2: Random Classifier Results

interest, but is able to do so in a fraction of the time, processing hundreds of images every minute.

Triton, coupled with a set of human eyes at the heart of the network, should prove to be an effective approach for prioritising images through the network, even when a classification cannot be made through the use of local knowledge. More importantly, in an area where the environment is as dynamic as the Malaysian rainforest, 97% accuracy for detecting empty images can be crucial, if only for saving network bandwidth.

3.5 Conclusion

In this chapter, we have shown the current hardware choices available for more computationally capable sensors and detailed our experimental results on how rainforest environments impact wireless transmissions. Although technologies, such as Wi-Fi, provide a high data rate, their range is limited and not suitable for sparse sensor networks; especially in a humid environment. Newer technologies, designed for long-range communication in sensor networks, are becoming increasingly more viable and, while they do have lower data rates and less robust protocols, they are more suitable for a resource constrained WSN that requires minimal power draw when transmitting. Using general-purpose hardware also means that there is no protection against water, humidity and animal or human intervention. We have adapted existing waterproof cases to suit our needs and two week deployments have shown that they are able to prevent any moisture from entering the case.

Chapter 4

Architecture

In this chapter, we explain our proposed network architecture that uses local and global knowledge to make informed routing decisions and to classify sensed data within the network. Our approach, K-HAS, uses a three tiered approach with each subsequent tier providing increased knowledge processing capabilities.

We believe that sensors capable of processing knowledge will provide a more efficient network and be able to prioritise data delivery that it believes to be important, rather than chronologically. We also believe that human input is a valuable learning process for such a network and feedback, from humans, on data that has been classified can be used to inform future classifications. To prove this, we have developed an architecture that uses different levels of knowledge processing throughout the network, the Knowledge-based Hierarchical Architecture for Sensing (K-HAS).

The rest of this chapter is structured as follows. Section 1 outlines the main aims of K-HAS and what it is capable of that typical sensor networks are not. Section 2 introduces an example, from our motivating scenario, that will be used to better explain each tier of the architecture. Section 3 explains the data collection tier. Section 4 explains the data processing tier. Section 5 explains the data aggregation tier. Section 6 concludes the chapter and summarises the key features of K-HAS.

4.1 K-HAS

K-HAS has been designed as an architecture for WSNs that is able to handle changes in the data sensed, as well as the structure of the network. By pushing knowledge bases out to the edge of the network, all nodes in the network have some awareness of the data they are sensing, as well as how important it is, based on the current projects that the network is involved in. This is achieved by using rules with different levels of granularity based on the knowledge processing capabilities of that tier.

4.1.1 Data Collection

The data collection (DC) tier is a very similar to standard nodes in a typical WSN, using hardware that has similar capabilities. These DC nodes are deployed at the edge of the network and tasked with sensing their environment, pre-processing the sensed data and using each other to relay data to the next tier.

DC nodes are capable of performing processing on data, such as the time it was recorded and its size, but their limited knowledge processing capabilities allow them to have an increased battery life and reduced size, making them suitable for a variety of deployments.

Knowledge Base

Reduced knowledge processing capabilities and low memory restrict the knowledge that these nodes can hold and they are limited to a static knowledge base that is encoded at the time of deployment. DC nodes only perform simple operations on the properties and content of the data that they sense, such as the time it was recorded, the location and its size. For more complex data, such as images and video, DC nodes do not possess the computational power required to process them and instead use the metadata associated. Unlike modern rule engines, these static rules do not use forward chaining and the outcome of one rule does not cause the rules to be fired again. Listing **??** shows an example of some of the rules in the knowledge base.

Listing 4.1: Example DC Node Rules

```
if(reading.dateCreated.month == JUNE AND reading.
   timeCreated.between(17:00, 19:00)
        data.write( Potential Otter sighting )

if(reading.temp == 37 AND reading.timeCreated.between
   (01:00, 05:00)
        data.write( Potential Leopard sighting )
        data.write( PRIORITY=HIGH )
```

When the data is recorded by the DC node, the knowledge base is fired and inferences are made about the contents of the data. Each DC node has a different knowledge base encoded based on the local knowledge of the area that it is deployed in. For example, a node deployed on the bank of a river would have a different knowledge base to a node deployed in the fields of a plantation.

Once a trigger has been processed, the data is packaged and then sent on to the Data Processing (DP) node.

4.1.2 Data Processing

DP nodes act as cluster heads of the network, serving a subset of all deployed DC nodes. When data is sensed, it is forwarded through all DC nodes to the DP node that is tasked with serving the originating DC node. These nodes have more knowledge-processing capabilities than a DC node and do not typically do any direct sensing.

Due to the greater capabilities, DP nodes have a much shorter battery life and a network typically consists of fewer DP nodes. This also allows DP nodes to run a complete rule engine and process complex data. When a DP node receives data, it processes everything associated, this includes metadata, the data itself and the inferences made by the DC node. If the DC node infers that the data is of a higher priority, then this data is processed first.

In our current implementation, DP nodes use two different radios, a Zigbee radio to allow long range communication from DC nodes and a Wi-Fi radio that provides short range communication that allows for higher data rates.

Knowledge Base

In our motivating scenario the network is image-based, this means that the DP node would perform image processing, as well as processing the image metadata. The increased knowledge processing capabilities allow DP nodes to run rules dynamically, learning from the sensed data and providing classifications that change based on changes in the environment. For example, if a DP node has not seen an elephant before, and it is not aware of the object in the image, then it will await a human classification. The node will then record the time period that it receives elephant pictures, i.e. June to July, and become more alert the following year. Similarly, the node will know not to look for pictures of nocturnal animals during the day. This local knowledge allows processing power to be saved and, thus, time; this ensures that the processing of sensed data is optimised as much as possible in order to reduce the time it spends in the network.

The rule engine used in our current implementation is Drools, a Java based rule engine that allows for rules to be defined in *.drl* files and these can be loaded dynamically into a knowledge base. This flexibility allows to be changed on the fly without the need to restart the device, or even require human access, as all of this can be achieved through network communication.

Upon receiving sensed data from a DC node, the rule base is fired on the metadata of each file received. If the rules determine that the data is of interest or, in the best case scenario, provides a classification, then the data is packaged and sent on to the Data Aggregation (DA) node.

4.1.3 Data Aggregation

Placed at the edge of the network, these are nodes with high knowledge-processing capabilities and would be accessible by users of the network. When

DA nodes receive sensed data, it is unpacked and stored in a folder that represents the node that it originated from.

Any information added by the DP node is parsed and classifications are extracted. If a classification is found, it is stored and the DA node checks for any active projects that contain the classification. If a match is found then all users involved in the project are informed via their preferred method of communication. Using the motivating scenario as an example, the people involved with projects could be researchers and professors and they may be looking for images of leopards, requesting to be informed via Twitter.

All sensed data received, regardless of whether it has been classified, is accessible through a web interface hosted by each DA node. The interface shows all of the sensed data from each deployment, along with the associated classification. More importantly, it allows users to classify the data using a voting system. Users have roles which give them different privileges within the system. Normal users are able to vote and the majority vote is seen to be the current classification.

However, privileged users are able to confirm a classification and prevent any further votes. Once a classification has been confirmed, it is then sent back to the DP node it originated from. If the classification made by a user is different to the one inferred by the node, then it updates its knowledge base and acknowledges receipt.

Knowledge Base

DA nodes do not typically experience the resource constraints that DC and DP nodes must compensate for. Because of this, they hold a global knowledge that contains a history of all observations made by all nodes, as well as the location and deployment times of all nodes in the network.

While DC do not store any of the observations they capture, and DP nodes only store part of the observation that can be used in future classifications, DA nodes store the complete observation made by every DC node, as well as any extra data that is added by users upon receiving the sensed data.

As well as this, DA nodes provide administrative operations on the network, such as the recording of node locations, time of deployment and viewing all active nodes. This allows the DA node to monitor active nodes and alert users if a node has not sent any data in a while.

4.2 Technological Components

In this section, we describe the technologies used in every layer of K-HAS and how they integrate in order to use local knowledge based on their respective knowledge processing capabilities. The majority of components, both hardware and software, used in K-HAS are used so they are applicable for any WSN, but some choices have been made to remain in line with our motivating scenario and, thus, are more specifically suited for the capture of scientific observations.

4.2.1 Data Standard

To pass sensed data through the network, we first had to choose a standard format that would allow us to encode the sensed data, as well as enrich it with inferences made through processing. Darwin Core (DwC) is a body of standards with predefined terms that allows for the sharing of biodiversity occurrences through the means of XML and CSV data files [**?**].

The Global Biodiversity Information Facility (GBIF) indexes more than 300 million Darwin Core records published by organisations all over the web, allowing datasets that were previously siloed from the public to be accessed by both human and machine. The main of Darwin Core it to provide a common language for sharing biodiversity data that reuses standards from other domains [**?**].

DwC follows a star record structure, where a record can contain many occurrences, which is the recording of a species in nature or in a dataset. In an occurrence, there is an *event*, a recording of a species in space and time, enriched with other terms such as *identification* and *location*. The core files in a Darwin Core archive are:

1. Meta.xml
2. EML.xml
3. Data files

Ecological Metadata Language (EML) is a metadata used by ecologists and the language is used to describe projects and those involved. This file acts as a form of certificate and descriptor as to what the data is related to and who owns it. The XML file, shown in Listing **??**, outlines a sample project and users involved in the project.

Listing 4.2: Darwin Core: EML.xml

```
<?xml version='1.0' encoding='utf-8'?>
<eml:eml xmlns:eml="eml://ecoinformatics.org/eml-2.1.1"
    xmlns:md="eml://ecoinformatics.org/methods-2.1.1"
    xmlns:proj="eml://ecoinformatics.org/project-2.1.1"
    xmlns:d="eml://ecoinformatics.org/dataset-2.1.1"
    xmlns:res="eml://ecoinformatics.org/resource-2.1.1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-
        instance"
    xmlns:dc="http://purl.org/dc/terms/"
    packageId="e71fda1c-dcb9-4eae-81a9-183114978e44/eml
        -1.xml" system="GBIF-IPT" scope="system">
    <dataset>
        <alternateIdentifier>e71fda1c-dcb9-4eae-81a9
            -183114978e44</alternateIdentifier>
        <title>Images from Danau Girang during the PTY
            Project 2011-12</title>
        <creator>
            <individualName>
                <givenName>Christopher</givenName>
                <surName>Gwilliams</surName>
            </individualName>
```

```
            <organizationName>Cardiff University</
                organizationName>
            <positionName>PhD</positionName>
            <address>
                <city>Cardiff</city>
```

The core file, *meta.xml*, shown in Listing **??** lists the files that contains the actual sensed data, as well as the terms used to describe it. Examples include: date, time, location, type of data, filename and species contained.

Listing 4.3: Darwin Core: meta.xml

```
<?xml version="1.0" encoding='utf-8'?>
<archive xmlns="http://rs.tdwg.org/dwc/text/" metadata=
    "eml.xml">
        <core encoding="UTF-8" linesTerminatedBy="\n"
            fieldsTerminatedBy="," fieldsEnclosedBy=''
            ignoreHeaderLines="1" rowType="http://rs.
            tdwg.org/dwc/terms/Occurrence">
                <files>
                        <location>set.csv</location>
                </files>
                <id index="0" />
                <field index="0" term="http://rs.tdwg.
                    org/dwc/terms/occurrenceID" />
                <field index="1" term="http://rs.tdwg.
                    org/dwc/terms/basisOfRecord" />
                <field index="2" term="http://rs.tdwg.
                    org/dwc/terms/recordedBy" />
                <field index="3" term="http://rs.tdwg.
                    org/dwc/terms/associatedMedia" />
                <field index="4" term="http://rs.tdwg.
                    org/dwc/terms/eventDate" />
                <field index="5" term="http://rs.tdwg.
                    org/dwc/terms/eventTime" />
```

```
                <field index="6" term="http://rs.tdwg.
                    org/dwc/terms/locationID" />
                <field index="7" term="http://rs.tdwg.
                    org/dwc/terms/scientificName" />
                <field index="8" term="http://rs.tdwg.
                    org/dwc/terms/identifiedBy" />
                <field index="9" term="http://rs.tdwg.
                    org/dwc/terms/dateIdentified" />
        </core>
        <extension encoding="UTF-8" linesTerminatedBy="
            \n" fieldsTerminatedBy="," fieldsEnclosedBy=
            '' ignoreHeaderLines="1" rowType="http://rs.
            gbif.org/terms/1.0/Image">
                <files>
                        <location>images.csv</location>
                </files>
                <coreid index="0" />
                <field index="1" term="http://purl.org/
                    dc/terms/identifier" />
        </extension>
</archive>
```

Data files contain the actual sensed data, based on how it is supported, and these files are linked in *meta.xml*. For example, temperature readings or direct human sightings would be stored in a CSV file and linked, however, images or video would require the metadata to be store in a CSV file and a filepath would be referenced in the XML. The structure of the CSV file contains a header line that matches the terms in the meta file and each line would be an observation. The terms are linked to the Darwin Core glossary so the archive can be validated and processed by a DwC archive reader.

All of these files are then archived and sent as a ZIP folder throughout the network. If the sensed data is media based, then the media is included as well. DwC archive processing libraries are included on both DP and DA

nodes.

Darwin Core is suited to K-HAS because its use in scientific observations matches our motivating scenario and the archive can be easily created by a DC node, as it does not require any heavy processing and all of the files are common formats.

4.2.2 Middleware

The knowledge-processing capabilities of DA and DP nodes are the same and this is part of what makes K-HAS different from most other WSNs; both types of node run the sensor middleware, but each for different purposes. DA nodes use the middleware for administrating the network, receiving and archiving sensed data and allowing users to provide classifications. DP nodes use if for the receiving, sending and controlling the flow of processing of sensed data before it is passed on.

Existing suitable middlewares have been detailed in Section **??** and our requirements for K-HAS were partially determined by the expertise of the users in our motivating scenario. Below is a list of our three core requirements:

Portability Heterogeneous WSNs utilise nodes with different architectures and capabilities, if middleware is to be used on the nodes it must be able to run on these varied devices.

Usability Users of K-HAS should not be expected to have knowledge of computer science or the underlying architecture, this network should be usable by almost anyone. The same must be said for the middleware as well.

Extensibility A closed-source middleware can be used, but it must then support all sensor nodes and data types, as well as receive regular updates. Open-source, or extensible, middleware can be used to add support for newer nodes.

GSN is a Java-based open-source middleware. New generic sensors can be added through XML files, while more complex sensors can be added through

custom Java classes. GSN is covered in more detail in Section **??**. Because GSN can run on any architecture that supports the Java Virtual Machine (JVM) then it meets our portability requirements and the web interface to provide administrative functionality makes it usable by those without any domain knowledge. Finally, the ability to add new sensors through XML means that it can be extended by almost any user of the network with very little guidance.

4.2.3 Knowledge Capture

GSN is packaged with a web interface that allows users to see all nodes deployed and view the latest sensed data received. The web interface is targeted towards users with domain expertise and has limited functionality focussed towards sensor administration. However, it does make GSN accessible by more than one computer, as well as a variety of different architectures. For example, the admin webpage could be accessed on the machine that runs GSN, or from a tablet computer connected to the same network. We used the same approach to develop a web-based tool that provides access to all sensed data, as well as a simple interface for performing tasks, such as uploading new rules or updating the location of nodes.

All sensed data is read from a MySQL database and users can view the metadata from each observation, such as location, date, time and temperature, as well as the data itself. From this, users are able to classify the data based on their role. Covered in detail in Chapter **??**, K-HAS uses roles to control active projects and classifications; there are administrators and researchers. Researchers are involved in projects and receive notifications when relevant data has been received. They have access to the web interface and can vote on classifications for sensed data. Administrators lead projects and can create/complete them, but they also have the ability to finalise classifications. K-HAS follows a knowledge hierarchy where administrators are at the top, with DC nodes at the bottom. When an administrator makes a classification, all prior classifications are ignored and the relevant knowledge bases are updated.

Figure **??** shows an observation where users can vote on the contents. An administrator can then confirm that classification and prevent further votes from being cast. This type of moderation means that it does not have to be specialists voting on sensed data that cannot be classified by DP nodes.

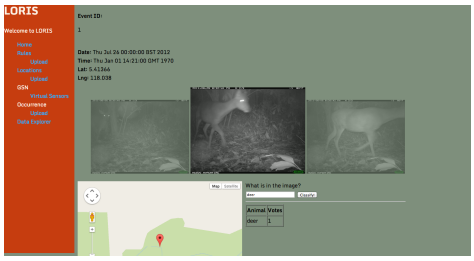

Figure 4.1: Web Interface for Observations

Viewing data for new observations is useful for gaining classifications and alerting members of a project, but viewing older sensed data allows patterns to be identified in order to create new rules. Figure **??** shows a map of all deployed nodes in the area surrounding the field centre in our motivating scenario. When users select a node, a table is populated with all of the classified observations that it has captured; this can then be used to extract patterns from the data and create rules. For example, the two observations of the Malay Civet are only seen late at night, if further observations also showed this, then we could create a rule defining the active hours of the Malay Civet and, potentially, list days that it is likely to pass. These rules can then be written and uploaded to the knowledge base, which we cover in the next section.

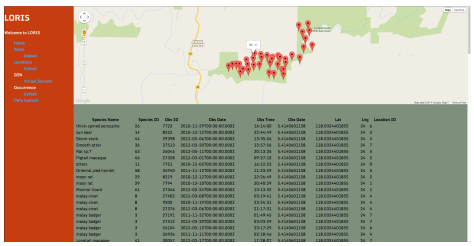

Figure 4.2: Web Interface for Classified Sensed Data

4.2.4 Knowledge Base

The Drools rule engine is a Java-based engine that uses forward chaining inference for the processing of rules, which means that rules are used to make meaningful inferences about data. Unlike DC nodes, which have limited knowledge processing capabilities, Drools is able to chain rules together and a rule that may not have been triggered at the start of processing may be triggered later if another inference is made. For example, a rule that is specifically for small mammals may not be triggered until an inference has been made that the image may contain small mammals based on the time and location of the observation.

Drools is able to dynamically update its knowledge base, adding rules and firing them on observations that have already been loaded, as well as newer ones. This allows DP nodes to adapt to new rules and local knowledge whilst they are deployed. The use of *drl* files use a mixture of Drools and Java syntax to define rules, allowing them to modify, or create, Java objects. This is one of the main reasons we chose Drools, as it can work with GSN and DwC Java objects, as well as the ability to run on any architecture that supports Java.

The functionality of Drools is extensive and it is a very powerful engine, however, it does require specialist knowledge to use and manipulate rules. Using the LORIS web interface, detailed in Section **??**, we created a simplified interface that uses a custom REST API for Drools, allowing users to create sessions, add rules, load data and fire rules, returning the output to the interface. Users can view, and load, existing drl files, shown in Figure **??**.

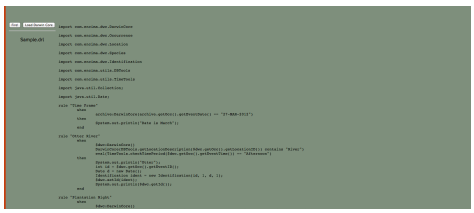

Figure 4.3: Web Interface for Drools Operations

Once a file has been selected, users can view the rules and use the controls on the webpage to perform common operations. The *Load Darwin Core*

button loads all DwC archives that are stored in the MySQL database into the current Drools session and the *Fire* button runs all loaded rules on the loaded data. If any of the rules trigger, then the output is presented to the user in the same page, allowing them to act on the results. For example, the location of observations could provide a narrowed down list of potential classifications, allowing an administrator to remove votes that do not match the list.

Currently, these implementations have been developed for our motivational scenario, but much of these tools are general enough to be repurposed in order to apply to a variety of different WSNs. The Drools API can be used for any kind of sensed data and the web interface would require minor changes to be extensible.

4.3 Walkthrough

In this section we will explain the steps involved in the capture, and processing, of an observation when using the K-HAS architecture. Each tier is responsible for performing different actions upon the observation to ensure it is received by the DA node with an inference as to what it may contain.

4.3.1 Scenario

This walkthrough will use our motivating scenario and the type of sensed data will be images of animals in the Malaysian rainforest. In this example, we have a collection of wildlife cameras, with nodes attached to them, deployed in the forest. Projects for the rare Clouded leopard and Sun bear are currently active at Danau Girang. The Clouded leopard is a nocturnal carnivore that uses existing paths and hill trails to travel through the rainforest and the Sun bear is the smallest bear in the world and sightings are rare. It is also nocturnal and claw marks can be seen on trees that they have climbed.All of this information has been encoded onto the DP nodes and DC nodes know that images taken at night will be of a higher priority, as well as to prioritise further images at night from DC nodes that are deployed on

Figure 4.4: Clouded Leopard Image Capture

ridges or existing trails.

4.3.2 Data Collection

A DC node is deployed along a ridge in the rainforest and consists of a wildlife camera with a wireless node attached. At 0200, the infra-red sensor detects movement and the camera triggers a set of 3 images to be captured. The DC node creates the DwC archive for the observation. Terms in the observation, such as time, date, species identified and location, are added to the meta.xml file and links to CSV files that contain the data for each term. A separate CSV file is created that holds the filename of each image that was taken. The image is shown below in Figure **??**.

The DC node runs its rules on the metadata of the images and infers that the image may contain a Clouded leopard, this is because the image was taken in the early hours of the morning and the camera is deployed on a ridge.

The inference is included in the archive and is compressed. The node then sends it through every DC node between the originating node and the DP node assigned. To achieve the long range communications in the forest, Digimesh is used; the low transfer rate does mean that an archive takes several minutes to send but it allows a distance of up to 1km.

4.3.3 Data Processing

The DP node receives the observation and it is unzipped and processed by the Darwin Core library. The images are read from the filenames provided in

the CSV and processed using two methods. The EXIF tags in the image are extracted and the images themselves are processed using the Open Computer Vision (OpenCV) library. A unique feature of the DP node is that it uses two radios to allow links to both DA and DC nodes. DC nodes send archives using Digimesh, to achieve long range communcation, and DA nodes use Wi-Fi, to provide a faster transfer rate than Digimesh and a more standard method that allows other devices to connect, such as mobile phones or laptops.

EXiF

EXIF (Exchangeable Image Format) tags are written to images at the time of capture. Examples of these tags can be time, date or camera serial no. The capabilities of the camera do affect how detailed the EXIF is, for example, a camera with GPS capabilities will enrich the image with the location.

Wildlife cameras have more functionality than common digital cameras, with details like moon phase, temperature or GPS location. Some devices even include the saturation, brightness and hue of each image. These capabilities allow the EXIF to be extremely detailed and this metadata can be used to find patterns in pictures that, when accompanied by local knowledge, assist with the classification of sensed data.

In this example, the knowledge base on the DP node is aware that Clouded leopards and Sun bears are nocturnal, but Clouded leopards have previously only seen when the temperature is between 30 to 35°Cand only when the moon is not full. However, data on Sun bear is not as complete and the knowledge base only shows that the bear is nocturnal and can be seen at any time of night in any area of the rainforest. The DP node identifies that the moon phase is not full and that the temperature is 32°C, from this it determines that the image could contain either animal and it cannot make a final conclusion.

Image Processing

OpenCV is a C++ library that allows popular image processing functions to be performed on one or more images. A program, we have called Triton,

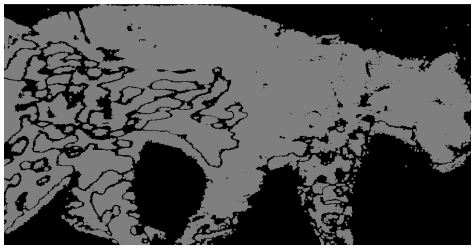

Figure 4.5: Processed Image of Clouded Leopard

that utilises OpenCV, is run on the set of three images. These images are converted to black and white and combined to build a background model for the complete set. The detected background is then removed and the final image is then searched for objects, where objects in the foreground will be shown with white pixels. The largest object is then found in the image and extracted to create a template, shown in Figure **??**.

Processed images of previously sensed images are stored on the DP node and associated with the confirmed classification, confirmed by a human or a node. Although the memory available on a DP node is typically around 32GB, this could easily fill in a matter of months if 3 full HD images were stored for every observation. Storing a single black and white template that contains a portion of the image is much more efficient and can still easily be associated with the classification made. The extracted image is then compared with the existing images, using the knowledge base to prioritise templates for comparison. In this example, nocturnal animals are prioritised and especially nocturnal animals with active projects associated. If the DP node has received an observation from the same DC node recently, then it will check for a classification on that and check for a match there first.

This observation is the first trigger from the DC node in the past few hours, so there are no recent classifications. However, processing of the metadata showed that the image was taken at night, so the DP node uses its knowledge base to match the images to templates of nocturnal animals first. Triton then finds a match to an existing template of a Clouded leopard and completes its classification.

Classification

The metadata processing of the image shows that it could be any nocturnal animal that is known to come out when the moon is not full and the temperature is 32°C. This is not a complete classification but the image processing has found a match. These findings are written into the DwC archive, using the ID of the DP node as the person that identified the image and the Clouded leopard as the species identified in the image. The archive is then zipped and sent on to the DA node.

4.3.4 Data Aggregation

Upon receiving a DwC archive, it unarchived and processed by Darwin Core libraries, called by the middleware running on the node. The resulting archive is then inserted into a MySQL database and the files themselves are stored in a directory that maps to the DC node that captured the original observation. At the field centre, three users of the system have subscribed to updates for observations of Clouded leopards.

As the archive is processed by the library, the species is extracted and this triggers a rule to notify the subscribers. The rule then queries the database and finds their preferred method of communication. In this case, one is a lecturer and wants to be emailed while the two remaining are students and want to be notified via Twitter, a social networking website. An email is drafted that contains the time, location and content of the observation, with the images attached, and sent on to the lecturer. The students are sent a short tweet that tells them a Clouded leopard has been spotted and a link to the middle image in the sequence is provided; the middle image is used because this local knowledge has shown that it is the most likely to contain the full subject in the image.

The middleware on the node supports a web interface to allow users to perform administrative functions on the network, such as deploy a new node, on top of this there is a custom made website that shows all observations for every DC node. This allows users to log on and classify the images. In this case, the lecturer receives the email notification, reviews the attached images

and clicks on the link to access the website to inform the DP node that the classification was correct. Due to the administrator position the lecturer has on the system, he is able to stop any users voting on the image and to simply confirm the classification.

If there was no classification, then users would be able to vote on the contents and use the classification with the highest vote, or the classification made by an administrator. Once a classification has been made, it is stored in the database and written to the archive. This triggers the DA node to send that classification on to the DP node that sent the original archive. In this case, the DA node informed the DP node that it has been confirmed as a Clouded leopard and the DP node then stores the extracted image in the directory of Clouded leopard templates, to assist with future classifications.

4.4 Conclusion

In this chapter we have explained the architecture we have developed to allow knowledge to be encoded and utilised within a wireless sensor network. Using tiers of nodes, with varying levels of knowledge-processing capabilities, we can process observations within the network and deliver data that has, where possible, already been classified. Working as more of a subscribe-push method, users do not have to check a base station for new data, instead it is sent to them if it has been found to be part of a project they are subscribed to. If not, then the data is accessible to all users of the network through a web interface.

One of the key features of K-HAS is that it is not a static deployment. The knowledge that the network holds at the time of deployment will rarely be the same as the knowledge held after a few months. Humans enrich the existing knowledge base and the nodes are able to make inferences about the data they are sensing, improving their classifications the longer they are deployed.

We believe that K-HAS is an architecture that suited to many different applications of WSNs and its general design allows it to be adapted to new scenarios with ease.

Chapter 5

Ontology

In this chapter, we explain the ontology that we developed to formally define the structure of K-HAS, as well as the format of the sensed data that is passed through the network.

This ontology provides those wanting to use K-HAS with a representation that is easy to reuse and deploy in a number of different scenarios, or even to select parts of the ontology that meet their requirements and implement those.

5.1 Background

K-HAS was developed in order to provide a generic architecture for wireless sensor networks to utilise the local knowledge contained within their environment to process sensed data and, therefore, make more efficient use of the network bandwidth by prioritising sensed data that is deemed to be more valuable []. We have defined local knowledge as knowledge of an area that has been gained through experience, or experimentation.

Before we were able to implement K-HAS, we needed to model the flow of knowledge within the network. Developing an ontology for K-HAS means that we can do this, as well as provide a computer-readable model for all of the classes and components used by K-HAS, and the relationships between them.

During the development of the K-HAS ontology, we researched existing ontologies that were popular in the domains that K-HAS covered. These included scientific observations and sensor hardware.

Looking into existing ontologies, we found that there had been many surveys on representing sensors in the semantic web [**?**], [**?**], outlines the existing work. These surveys clearly highlighted that these ontologies had been split into two branches; observation-centric and sensor-centric.

Observation-centric ontologies focus on the data that is sensed, and its content; whereas sensor-centric ontologies detail the components that make up a sensor, and the operations they perform to turn sensed data into an output.

We found few ontologies that linked these concepts together to show the flow of data within the network and the role that each sensor plays in delivering this data. Because of this, we developed an aligning ontology that reuses existing ontologies, where possible, and introduces new classes that allows these ontologies to interlink.

K-HAS has been developed to be used with any sensors and is not specific to wildlife cameras, therefore we also looked into sensor-based ontologies that concentrate on the hardware and the individual capabilities of each device within the network.

From this we have categorised relevant existing ontologies into Observation-Centric and Sensor-Centric ontologies.

5.1.1 Observation-Centric Ontologies

OBOE

The Extensible Observation Ontology (known in reverse as OBOE) is a popular suite of ontologies used to represent scientific observations. Initially starting as base ontology for ecological observations, it has now grown into a suite of extensions that make it suited for chemistry, bioinformatics, anatomy and others. OBOE is represented by OWL-DL [**?**] and allows the characteristics of a generic scientific observation to be linked to domain specific characteristics.

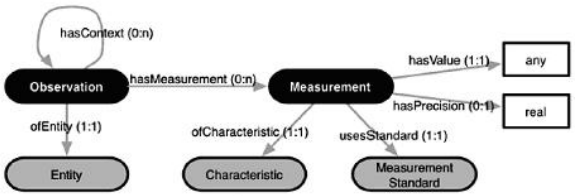

Figure 5.1: The Core of an OBOE Observation

OBOE focuses on the concept of an *observation*, which is made up of an entity, a measurement and a characteristic [**?**]. An example of an observation could be a researcher observing an animal (the entity) and recording the gender (the measurement) as male (the characteristic). A single observation could then consist of multiple observations within it, such as gender, location, species and the number of species observed.

Figure **??** shows the basic structure of a core OBOE observation, outlining the five key classes that are linked by seven properties. While this is the core structure of OBOE, domain specific extensions have been implemented by utilising 'extension points' that are part of OBOE's core. It is clear to see that OBOE follows the O&M Standard (below) very closely, providing extensions to the core classes that allows more information about each to be encoded, adding context and enhancing the value of an observation.

The primary benefits of OBOE are that it is generic enough to cover almost all types of scientification observation and domain extensions allow for more specific details to be stored.

O&M

The OpenGIS Consortium (OGC) Observations and Measurements Standard aims to provide a framework suitable for recording any observation made by a sensor, regardless of the domain [**?**].

The key difference to note with O&M is that *observation* and *measurement* are not just classes. They also denote an action. An observation is an action that causes a result, yielding a value and a measurement is a set of operations that provide some result(s).

O&M provides a conceptual model, as well as XML encoding for observations and measurements. The listing **??** shows an observation of a vehicle in a given time and place. Similar to the encoding of a measurement with the standard and protocol used in OBOE, O&M provides support for the recording of the procedure used to gain the measurement for the observation.

Listing 5.1: An Observation of a Vehicle encoded in O&M

```
<?xml version="1.0" encoding="windows-1250"?>
<om:GeometryObservation gml:id="geom1610"
xmlns:om="http://www.opengis.net/om/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:gml="http://www.opengis.net/gml"
xsi:schemaLocation="..Specialization_override.xsd">
  <om:samplingTime>
    <gml:TimeInstant>
      <gml:timePosition>2009-09-16T17:22:25.00</
          gml:timePosition>
    </gml:TimeInstant>
  </om:samplingTime>
  <om:procedure xlink:href="
      urn:ogc:object:procedure:ifgi:GPS"/>
  <om:observedProperty xlink:href="
      urn:ogc:def:phenomenon:OGC:Shape"/>
  <om:featureOfInterest xlink:href="
      urn:ogc:object:feature:vehicle"/>
  <om:result>
    <gml:Point srsName="urn:ogc:crs:epsg:4326">
      <gml:pos>40.7833 -73.9667</gml:pos>
    </gml:Point>
  </om:result>
</om:GeometryObservation>
```

The listing shows that an observation centres around a *feature of interest*

that can be a physical object and the measurement is the detection of the vehicle at the recorded location. Figure ?? shows a basic diagram of the ontology. While the event of a feature is linked, it is clear that the main focus is on the observation and the measurement associated with it.

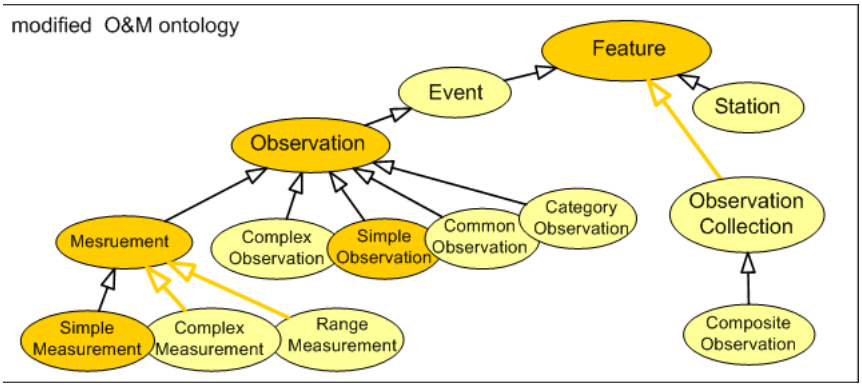

Figure 5.2: The OGC O&M Ontology

The structure of an observation within O&M is focussed on the action and the result, this makes it suited for sensor networks across many domains that perform a wide range of observations. Because the users of the O&M standard are spread across many domains, each with their own terms and definitions, the creation of an ontology for the standard aimed to remain as generic as other observation-centric ontologies.

Darwin Core SW

Darwin Core (DwC) is an approach to storing ecological observations, made by a sensor or a human, in a star record format. While it does not have the extensions available to OBOE, Darwin Core is extremely concise for recording observations within the biological diversity domain and aims to be a standard reference for sharing these observations.

The standard structure of a Darwin Core Observation is a star record; an archive of files with a core metadata file that describes the content of all other files within the archive.

The record shown in figure ?? represents a DwC archive that conforms to the star schema. The ecological metadata language document contains

all of the details about the project, such as who is involved, the institution code, contact details and the project(s) related to the observation, shown in Listing **??**.

Listing 5.2: Darwin Core Ecological Metadata File

```
<?xml version='1.0' encoding='utf-8'?>
<eml:eml xmlns:eml="eml://ecoinformatics.org/eml-2.1.1"
xmlns:md="eml://ecoinformatics.org/methods-2.1.1"
xmlns:proj="eml://ecoinformatics.org/project-2.1.1"
xmlns:d="eml://ecoinformatics.org/dataset-2.1.1"
xmlns:res="eml://ecoinformatics.org/resource-2.1.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:dc="http://purl.org/dc/terms/"
packageId="e71fda1c-dcb9-4eae-81a9-183114978e44/eml-1.
    xml" system="GBIF-IPT" scope="system">
<dataset>
<alternateIdentifier>e71fda1c-dcb9-4eae-81a9-183114978
    e44</alternateIdentifier>
<title>Images from Danau Girang during the PTY Project
    2011-12</title>
<creator>
<individualName>
<givenName>Christopher</givenName>
<surName>Gwilliams</surName>
</individualName>
<organizationName>Cardiff University</organizationName>
<positionName>PhD</positionName>
<address>
<city>Cardiff</city>
<administrativeArea>Cardiff</administrativeArea>
<postalCode>CF24 3AA</postalCode>
<country>Wales</country>
</address>
```

```
<phone></phone>
<electronicMailAddress>C.Gwilliams@cs.cf.ac.uk</
    electronicMailAddress>
<onlineUrl>christopher-gwilliams.com</onlineUrl>
</creator>
<pubDate>2012-07-26</pubDate>
<language>en</language>
<abstract>
<para></para>
</abstract>
<keywordSet>
<keyword>Danau Girang</keyword>
<keyword>Darwin Core</keyword>
</keywordSet>
<intellectualRights>
<para></para>
</intellectualRights>
<coverage>
<geographicCoverage>
<geographicDescription></geographicDescription>
<boundingCoordinates>
<westBoundingCoordinate>-180</westBoundingCoordinate>
<eastBoundingCoordinate>180</eastBoundingCoordinate>
<northBoundingCoordinate>90</northBoundingCoordinate>
<southBoundingCoordinate>-90</southBoundingCoordinate>
</boundingCoordinates>
</geographicCoverage>
<taxonomicCoverage>
<generalTaxonomicCoverage></generalTaxonomicCoverage>
<taxonomicClassification>
<taxonRankName></taxonRankName>
<taxonRankValue></taxonRankValue>
<commonName></commonName>
```

```
</taxonomicClassification>
</taxonomicCoverage>
</coverage>
<contact>
<individualName>
<surName>Gwilliams</surName>
</individualName>
<electronicMailAddress>c.gwilliams@cs.cf.ac.uk</
    electronicMailAddress>
</contact>
<methods>
</methods>
<project>
<title>Using Local and Global Knowledge in Wireless
    Sensor Networks</title>
</project>
</dataset>
<additionalMetadata>
</additionalMetadata>
</eml:eml>
```

The descriptor file is an xml document that contains the column headers in the attached files and the mappings of those headers to DwC terms, an example file is shown in Listing **??**. This file shows that this is an archive containing the sighting of an individual, and that the sighting has been split into two files. The largest benefit of Darwin Core is its modularity. Extension files can be added to enrich the data for each occurrence. In this example the extension file is named as *images.csv* and contains image-based evidence to support the observation.

Listing 5.3: Darwin Core Descriptor File

```
<?xml version="1.0" encoding='utf-8'?>
<archive xmlns="http://rs.tdwg.org/dwc/text/" metadata=
    "eml.xml">
<core encoding="UTF-8" linesTerminatedBy="\n"
    fieldsTerminatedBy="," fieldsEnclosedBy=''
    ignoreHeaderLines="1" rowType="http://rs.tdwg.org/
    dwc/terms/Occurrence">
<files>
<location>set.csv</location>
</files>
<id index="0" />
<field index="0" term="http://rs.tdwg.org/dwc/terms/
    occurrenceID" />
<field index="1" term="http://rs.tdwg.org/dwc/terms/
    basisOfRecord" />
<field index="2" term="http://rs.tdwg.org/dwc/terms/
    recordedBy" />
<field index="3" term="http://rs.tdwg.org/dwc/terms/
    associatedMedia" />
<field index="4" term="http://rs.tdwg.org/dwc/terms/
    eventDate" />
<field index="5" term="http://rs.tdwg.org/dwc/terms/
    eventTime" />
<field index="6" term="http://rs.tdwg.org/dwc/terms/
    locationID" />
<field index="7" term="http://rs.tdwg.org/dwc/terms/
    scientificName" />
<field index="8" term="http://rs.tdwg.org/dwc/terms/
    identifiedBy" />
<field index="9" term="http://rs.tdwg.org/dwc/terms/
    dateIdentified" />
</core>
<extension encoding="UTF-8"71linesTerminatedBy="\n"
    fieldsTerminatedBy="," fieldsEnclosedBy=''
    ignoreHeaderLines="1" rowType="http://rs.gbif.org/
    terms/1.0/Image">
<files>
<location>images.csv</location>
```

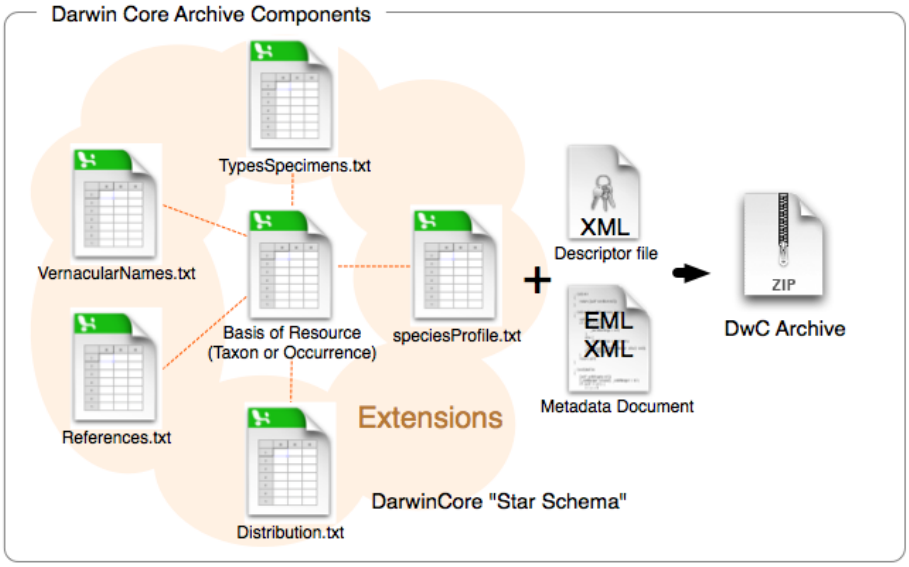

Figure 5.3: A Darwin Core Star Archive

Listing ?? shows the comma-separated central file (Basis of Resource) containing the core details of the observation, i.e. the animal observed, and the column headings map to the descriptor file. Other linked files are typically linked by the unique ID of the observation, containing information that extends the observation and provides further context.

Listing 5.4: Darwin Core Occurrence Data

```
eventID,basisOfRecord,recordedBy,eventDate,eventTime,
    locationID,scientificName,identifiedBy,
    dateIdentified
3,MovingImage,1,2010-12-22,16:14:35,1,,,
```

In order to represent DwC occurrences in an ontological format, work has been done to represent Darwin Core terms, as an ontology, in OWL. Darwin-Semantic Web (Darwin-SW) [?] is the project that aims to do this and many of the core terms associated with an occurrence have already been formalised.

While Darwin-SW does not represent all of the classes within the DwC namespace, it contains the core classes required to record an occurrence, to a

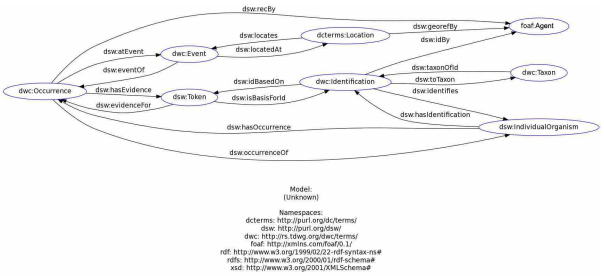

Figure 5.4: Darwin-SW

more-detailed level than OBOE. The downside of this is that the specificity of the terms limits DwC to ecological observations, making it far less generic than OBOE and other alternatives. There are positive aspects that, for those that need to record ecological observations, DwC allows for a greater level of detail and can be mapped to OBOE with ease.

5.1.2 Sensor-Centric Ontologies

Sensor-centric ontologies are more focussed on the structure of the sensor, the network and the sensing processes involved.

SensorML

The Sensor Markup Language (SensorML) Standard has been developed by the OGC, and complements the O&M Standard, to enable the discovery and tasking of internet-connected sensors [**?**].

SensorML provides an XML schema for describing a sensor, its capabilities and the processes available. At the core, SensorML comprises of:

1. Component - A physical process that transforms information from one form to another.

2. System - Model of a group of components.

3. Process Model - Atomic processing block used within a Process Chain.

4. Process Chain - Composite block of Process Models.

5. Process Method - Definition of the behaviour of a Process Model.

6. Detector - Atomic part of a Measurement System.

7. System - Array of components, relates a Process Chain to the real world.

8. Measurement System - Specific type of System, mainly consisting of sampling devices and detectors.

9. Sensor - Specific type of System that represents a complete Sensor.

These definitions outline the core concepts of SensorML, a *system* that performs one (or more) process(es) and is comprised of a group of *components* [**?**]. A SensorML document allow for a general, formal specification of a *sensor* and its capabilities. The document describes a *component*, outlining what data it reads in and the output once it has been processed. Several of these *components* can the be used to create a *system* and the primary goal of SensorML is to describe the process of how an observation came to be, focussing on the technical featuresof the node.

SSN

The Semantic Sensor Network (SSN) Ontology is the most fitting ontology for our requirements, as it covers systems processes and observations. Developed by the W3C after an extensive review of existing ontologies [**?**], the SSN ontology is designed to allow focus on a variety of perspectives, such as the sensor within the network or the data that has been observed.

Figure **??** shows the model for the SSN ontology, displaying the relationships that connect each class, as well as their associated properties. It also highlights the modular approach that has been taken, separating the system from the process and the observation.

The observation pattern of SSN is centred around 'Stimuli-Sensor-Observation', which can be simply described by an event causing a sensor to trigger and create an encompassing observation to store details about the event, as well as the device that recorded the event. While SSN is focussed on sensors,

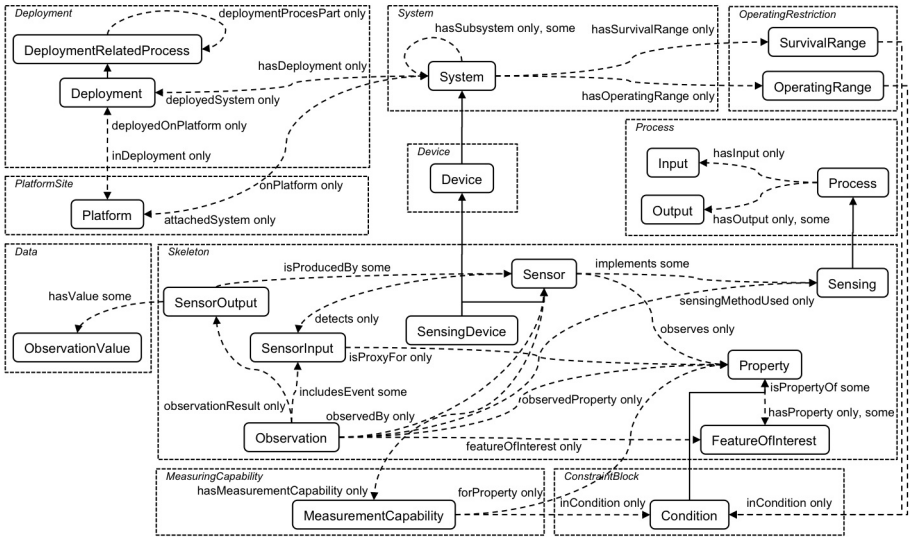

Figure 5.5: SSN Ontology

capturing the measurement capabilities of each sensing device that makes up a system and specifics on its lifespan, the development of the ontology arose from the review of sensor-centric ontologies as well, recording data about observations is a structure very similar to O&M.

Findings

There are many ontologies that are suited for sensor-based scientific observations and ontologies that allow for the description of sensor hardware are very complete. However, in our research, we were unable to find a complete ontology that allowed for a sensor to be described as well as specifics of the observation. SSN was the closest that we found for this, but it was still very hardware focussed and aimed at a technical audience.

We found that many of these ontologies satisfied many subsections of K-HAS completely, meaning that it would be unnecessary to reproduce these concepts. From these findings we have looked to develop an alignment ontology that connects ontologies across multiple domains to allow for a complete implementation of K-HAS, creating an ontology that is both sensor-centric and observation-centric.

5.2 Method

Before we began development, we researched existing methods that have been used to develop current ontologies. From this, we found three well-documented methodologies; the methodology used to develop the Toronto Virtual Enterprise (TOVE) ontolgy, the Methontology and the methodology used to develop the Enterprise Model ontology.

The methodology by Uschold and King [**?**], used to develop the Enterprise ontology, is a four step method that was most suited to the processes we expected to follow. The steps are not covered in great detail in the original paper but, research since provides greater detail for each step.

5.2.1 Identify Purpose

The aim of this step is to identify why the ontology is being built and what requirements it is supposed to fulfil. This includes considerations, such as the audience, the intended purpose and the specificity of the ontology.

Other methodologies have used more structured methods to informally identify the requirements of the ontology. Gruninger and Fox [**?**] propose competency questions; these are questions that are used to identify the problems that the developed ontology is developed to solve. These questions can act as a benchmark, showing that an ontology is sufficient if it can solve the questions raised here.

5.2.2 Build the Ontology

Building the ontology can be broken down further into 3 smaller steps: capture, code and integrate existing ontologies.

Capture

The capture stage involves defining the concepts and terms that the ontology will model, as well as how they map to the real world. This can be done in one of three ways:

1. Top Down - Starting with the core conepts, create the more specialised classes until you have identified all subclasses.

2. Bottom Up - This process begins with the more specialised classes and grouping them into the more general classes towards the top of the heirarchy.

3. Middle Out - A combination of the two, this process involves specialising, and generalising, the classes identified in the middle layer of the hierarchy.

As for the capture of the knowledge used to identify the classes needed, this is an area that has been documented, but primarily provides recommendations. [**?**] suggests interviews with domain experts, iteratively brainstorming with a group actively involved in the development of the ontology. This stagehas often been referred to as the knowledge-acquisition stage.

Code

This step involves coding the terms identified in the previous step into a formal language, such as the Web Ontology Language (OWL).

Integrate Existing Ontologies

Some of the literature (REFERENCE) recommends that this step is carried out at the same time as the two steps prior, so that the ontology is developed with existing ontologies in mind. This also allows overlap to be identified, and incorporated, early. Being aware of popular ontologies within the domain, before development begins, is a more logical approach and does avoid the need to create new terms for existing concepts.

Existing ontologies can be integrated by importing them and linking the existing terms to the terms identified in the ontology that is being developed. However, there is also the method of developing the ontology completely, so that it is consistent without the need to rely on existing ontologies and creating 'sameAs' relationships between the terms identified and only the required terms in the existing ontologies.

In [?], it is recommended that, when importing external ontologies, the whole ontology is not used. Rather, one should aim to extract only the required fragments from the external ontology that are relevant to the concepts in the developed ontology.

5.2.3 Evaluation

For the evaluation, Uschold and King adopt the definition of [?]: to make a technical judgement of the ontologies, their associated software environment, and documentation with respect to a frame of reference ... The frame of reference may be requirements specifications, competency questions, and/or the real world.

There are some well documented methods for evaluating ontologies in the literature, [?] proposes using the competency questions, used in the first step, to ensure that they can all be fully answered by the finished ontology.

5.2.4 Documentation

Although this step is listed at the end of the development cycle, it seems more fitting to document all major aspects of the ontology as it is being developed. Documentation of the ontology should include: any assumptions, all concepts introduced, all ontologies that have been incorporated (by whatever meeans) and any primitives used for the definitions of concepts.

5.3 Results

This section details our results when using the Uschold and King methodology, accompanied by more recent research for particular steps, to develop the K-HAS ontology.

5.3.1 Identify Purpose

The need to create the K-HAS ontology was partly due to the reasons outlined by Gruninger and Fox [?], we had identified a problem as we were developing

Table 5.1: Competency Questions

Competency Questions
Find all **Occurrences** of an **Individual**
Find all **Occurrences** of an **Individual** at a **Location**
Find all **Occurrence** within a specified **Date** and **Time** range
Find all **Sensors** that have recorded an **Occurrence** of an **Individual**
Find all **Locations** of an **Individual**
Find the storage location of a **Project**
Find all **Projects** containing an **Individual**
Find all **People** involved in a **Project**
Find all the **Evidence** that supports an **Occurrence**
Find all the **Types** of evidence that supports an **Occurrence**

a sensing architecture that utilised local knowledge: how could we formally represent the flow of knowledge and sensed data throughout a wireless sensor network?

To determine the scope that K-HAS should cover, we identified a set of competency questions that represent what we expect K-HAS to cover within the domain. In order to present this, we used an approach similar to that of [**?**], which can be seen in Table **??**.

These questions allow us to identify the core concepts that the ontology needs to represent, as well as serving as a tool to evaluate the completed ontology.

5.3.2 Build the Ontology - Capture

This part of the development cycle is the identification of the concepts and their implementation. The first step is capture the knowledge that will be used to identify the core concepts within the ontology.

To capture the knowledge for the ontology, we used a combinatorial approach of those outlined in the literature. We interviewed domain experts, as well as overseeing a basic implementation of a system, and we iteratively

brainstormed the concepts throughout the development cycle.

Over the course of eighteen months, which involved 2 field visits and several brainstorming meetings, we identified the core concepts that the ontology would need to contain, as well as the properties that would link them. From this, it would seem that we followed the *Top-Down* approach, but the first visit with domain experts also allowed us to identify some of the more specialist classes early on in the development cycle. Thus, it seems that the *Middle-Out* suits our methodology more closely.

Table **??** outlines the core concepts we have identified for K-HAS to be complete, as well as the definitions we have used for the architecture. When integrating existing ontologies, the definitions of similar terms would need to match with our definitions or we would not deem them the same as the K-HAS concepts.

Mapping these concepts, a diagram of the base K-HAS ontology is shown in Figure **??**. The next step is to create the ontology and map the concepts identified to existing ontologies within the same domain space(s).

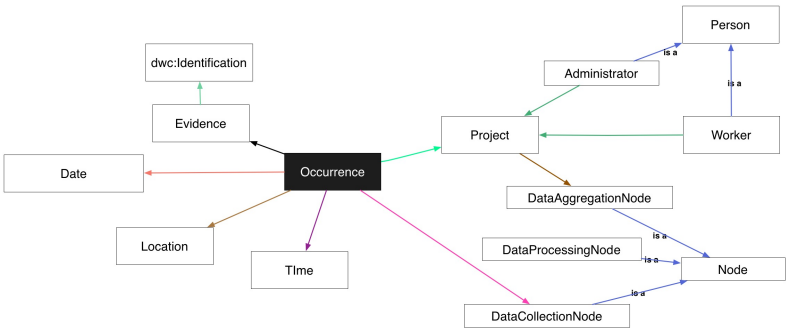

Figure 5.6: K-HAS Base Ontology

5.3.3 Build the Ontology - Code

Once the concepts had been identified (and agreed upon), the ontology must be coded. We used Protege 4.2.2 [**?**] and implemented K-HAS in the Web Ontology Language (OWL), creating a core file that could be expanded; should we need to import existing ontologies.

Table 5.2: K-HAS Concepts

Concept	**Subclass Of**	**Definition**
Occurrence	-	
Identification	-	A text-based recording of the content of an Occurrence
Evidence	-	Media to support the Identification, such as a photo or recording.
Location	-	The location of the Occurrence.
Date	-	The date of the Occurrence.
Time	-	The time of the Occurrence.
Project	-	Project(s) that can contain many Occurrences
Node	-	A device with sensing capabilities.
Data Collection (DC) Node	Node	Node with limited knowledge-processing capabilities charged with sensing a feature (or features) of its environment.
Data Processing (DP) Node	Node	Node with knowledge-processing capabilities charged with serving a subset of DC Nodes and processing their sensed data.
Data Aggregation (DA) Node	Node	Node with knowledge-processing capabilities that stores all knowledge and sensed data for the whole network.
Person	-	
Administrator	Person	Person in charge of a project (or projects).
Worker	Person	Person involved with a project.

5.3.4 Build the Ontology - Integrate Existing Ontologies

Our implementation of K-HAS is focused on scientific observations. Because of this, the focal point of our existing ontology research is of ontologies that are centred around scientific observations, this allows us to create a more generic ontology that is still able to capture all of the semantic details associated with a wildlife observation.

As explained in Section **??**, our research of existing ontologies covered two categories: observation-centric and sensor-centric. We identified ontologies, within multiple domains, that satisfied many of K-HAS' requirements, but not all. Using the core concepts outlined in Section **??**, we created a minimal ontology and used the results of our research to map K-HAS concepts to those that had already been identified.

During this process we found that we had determined concepts that mapped to existing ontologies, but may not share identical structures, or even the same name. For example, the OBOE ontology describes the concept of an occurrence, which is very similar to our occurrence and the occurrence in the Darwin Core ontology. When we found these mappings, we used *sameAs* relationship to form a link that allows data stored according to these existing ontologies to map directly to K-HAS. However, the structure of an OBOE observation is, as outlined in Section **??**, is more generic for all types of scientific observation and depicts the observation of an entity, containing a measurement of a particular characteristic. Whereas the structure of a Darwin Core observation is more limited to scientific observations of taxa which allows it to have more predefined terms, such as location, species name and the evidence for the recorded individual. Because of this, it was difficult to create a structure that encapsulated both OBOE and Darwin Core.

Research showed that Darwin Core maps to K-HAS' requirements more completely, as well as the fact that there has been work to represent Darwin Core Observations in OBOE [**?**], K-HAS occurrences map directly to Darwin Core and elements that are similar to OBOE have been linked by the *sameAs* relationship. This does mean that OBOE observations cannot map directly,

but K-HAS, and Darwin Core, occurrences can be converted, if necessary.

For the sensor hardware of K-HAS, we found that the SensorML ontology maps directly to our concepts and we also realised that we did not need to recreate concepts that may already exist in popular ontologies outside of the domain spaces we researched. For example, the Friend of a Friend (FOAF) ontology [**?**] is an ontology designed to create machine-readable pages that describe people, so it is more logical that K-HAS reuses existing terms from popular ontologies to allow pre-existing data to be mapped with ease.

INCLUDE TABLE TO MATCH TERMS TO EXISTING ONTOLOGIES??

5.3.5 Ontology

Whilst researching popular ontologies, we became aware that some of classes identified for K-HAS were not satisfied by what is currently available. The final step for creating the ontology was to add these concepts to the linked ontologies, we call these *extension concepts*, concepts unique to K-HAS which do not exist in any other ontology. Much of these terms were linked to the unique layered architecture of K-HAS and we defined them within a new K-HAS namespace, changing our ontology from an alingment of existing ontologies to an extension of these.

The final ontology is shown in Figure **??** and shows how each concept maps to existing ontologies. The figure shows that there were only five extension concepts unique to K-HAS that can be subclassed from the *person* concept in the FOAF ontology and *node* in the SensorML ontology.

5.4 Evaluation

Uschold and King's ontology development method does not explain how to evaluate the created ontology and much of the literature describes a number of methods that can be employed, [**?**] outlines a number of different evaluation techniques and separates them into categories. These categories are listed below:

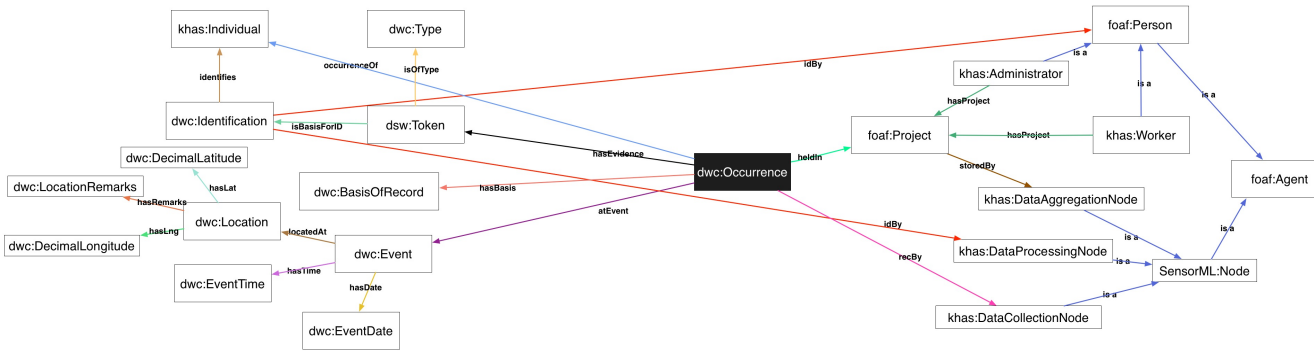

Figure 5.7: K-HAS Ontology

1. Comparing the ontology to a 'golden standard'.

2. Using the ontology in an application and evaluating the results.

3. Comparing the ontology with a collection of documents from the domain to be covered.

4. Manual evaluation by humans to test if the ontology meets a set of predefined criteria.

The last category mentioned is similar to the method in Section **??** that uses the competency questions to determine the effectiveness of the ontology. Because there is no agreed method,we have evaluated K-HAS by testing it in the application it was developed in (Protege), mapping it to existing documents within the intended domain and ensuring that it can satisfy all of the competency questions we outlined in Table **??**.

5.4.1 Using the Ontology in an Application

We developed the K-HAS ontology in Protege, a Java based ontology editor, that also provides the functionality to reason over the ontology and check for inconsistencies. Using the Hermit reasoner that is built into Protege, the ontology was determined to be logically consistent.

To confirm the validation provided by Protege, we also used a web-based ontology validation tool, called the OntOlogy Pitfall Scanner (OOPS), that

scans an ontology for common errors, such as defining incorrect inverse relationships or using recursive definitions, that can occur during the development phase [**?**]. The results of this tool showed that no pitfalls had been detected.

We have a customised build of K-HAS running that receives data from a variety of sources and stores it in a MySQL database that allows users to classify through a web site. The schema of the database maps to our ontology, scientific observations are received, unzipped and stored in the relevant fields in the database. Using real sensed data, we have successfully stored over two hundred occurrences and mapped existing DwC occurrence to K-HAS.

5.4.2 Competency Questions

In Table **??** we have identified some basic competency questions that we expected K-HAS to be able to satisfy with the concepts we had identified, Table **??** shows how the original questions are satisfied by K-HAS. We have mapped the terms we originally used to the terms used in K-HAS (and the linked ontologies) and shown the concepts involved with each question, as well as the relationships linking them.

The table shows that all of the competency questions outlined in Section **??** have been satisfied by the final ontology and each concept used maps quite easily with little to no change.

5.4.3 Comparing the Ontology with a Collection of Documents

Within the domain of scientific observations, Darwin Core is a popular choice for observations of wildlife and plants, it was because of this that we chose to use many of the pre-existing concepts from Darwin Core in K-HAS.

In the data driven approach suggested by [**?**], a corpus of documents, related to the domain that the ontology covers, and the keyword content is matched with the terms used in the ontology. Because Darwin Core observations are structured into archives of files, explained in Section **??**, we

evaluated the K-HAS ontology by combining the approach of comparing a corpus of documents related to the domain with human evaluation of ensuring an ontology met a set of requirements to create a method that ensured existing scientific sensed observations could be mapped to K-HAS with little to no modification.

As we have previously explained, an archive of Darwin Core files consists of a minimum of 3 files: a metadata file that contains information about the creator of the archive and the project it relates to (eml.xml), a metadata file that describes all of the files that contain the occurrence and the fields within them (meta.xml) and a csv file that contains the data relating to the occurrence itself. Within the archive, the core files that need to be mapped to K-HAS are the meta.xml and eml.xml files as this allows us to store what and who.

We used the example files from the archive in Section **??** to extract the terms associated with an occurrence and we mapped these terms to K-HAS concepts, the results can be seen in Table **??**.

Each term within a Darwin Core observation maps to K-HAS with only a few changes to the terms, while the definitions do not change. As K-HAS is an alignment ontology, there are extra concepts that do not map to Darwin Core, but can encapsulate each occurrence. For example, a project in K-HAS can contain many occurrences. As previously mentioned, [**?**] shows how a DWC archive can be represented in OBOE, which means that it would not be too complex to map an OBOE observation in K-HAS.

5.5 Conclusion

In this chapter we have presented an ontology for the K-HAS architecture and described our methodology for development, as well as showing that existing ontologies are not complete enough to cover K-HAS' requirements. The K-HAS ontology we present is an alignment of ontologies that are spread across multiple domains and provides a complete solution for a sensor network that deals with scientific observations and we believe this is the first alignment ontology that combines sensor-centric and observation-centric on-

tologies. We have also extended this alignment to include concepts that model the unique features of K-HAS, but this ontology should be suitable for any sensor network that deals with observations.

Our evaluation has shown that the ontology is logically consistent and that existing scientific observations, in other formats, can be mapped across to K-HAS with few changes. The competency questions we identified during the design phase of the ontology can all be satisfied by the resulting ontology and our K-HAS network currently stores data with a schema that follows the design of this ontology.

In the future, work would be done to make the Darwin Core terms more modular and users will then be able to 'plug in' their occurrence structure of choice.

Table 5.3: Competency Questions

Competency Questions	**Concepts**	**Relationships**
Find all **Occurrences** of an **Individual**	Occurrence; Individual	Occurrence occurrenceOf Individual
Find all **Occurrences** of an **Individual** at a **Location**	Occurrence, Token, Individual, Location, Identification	Occurrence hasEvidence Token; Token isBasisForID Identification; Identification identifies Individual
Find all **Occurrence** within a specified **Date** and **Time** range	Occurrence; Event; Date; Time	Occurrence atEvent Event; Event hasTime Time; Event hasDate; Date
Find all **Data-Collection Nodes** that have recorded an **Occurrence** of an **Individual**	DCNode; Individual; Occurrence	Occurrence recordedBy DCNode; Occurrence occurrenceOf Individual
Find all **Locations** of an **Individual**	Occurrence; Individual; Event; Location	Occurrence occurrenceOf Individual; Occurrence hasEvent Event; Event locatedAt Location
Find the storage location of a **Project**	Project, Data Aggregation Node	Project storedBy DataAggregationNode
Find all **Projects** containing an **Individual**	Individual, Project; Occurrence	Individual occurredIn Occurrence; Occurrence heldIn Project
Find all **Persons** involved in a **Project**	Person; Administrator; Worker; Project	Administrator hasProject Project; Worker hasProject Project
Find all the **Evidence** that supports an **Occurrence**	Occurrence; Token	Occurrence hasEvidence Evidence
Find all the **Types** of evidence that supports an **Occurrence**	Occurrence; Token; Type	Occurrence hasEvidence Token; Token isOfType Type

Table 5.4: Evaluation of K-HAS against Darwin Core Occurrence Terms

Darwin Core Term	**K-HAS Concept**
occurrenceID	Occurrence
basisOfRecord	Basis of Record
recordedBy	Person/Node
associatedMedia	Token
eventDate	Event Date
eventTime	Event Time
locationId	Location
scientificName	Individual
identifiedBy	Person
dateIdentified	Identification

Chapter 6

LORIS

In this chapter, we detail the design and deployment of K-HAS in the Malaysian rainforest. We also highlight the issues we experienced deploying the architecture in a humid, dense rainforest for use by those without domain knowledge of WSNs.

Experiments carried out in the rainforest had already shown that the range of wireless communications could be reduced by up to 80% and we expected that the lifetime of nodes in such conditions would be affected.

Yearly visits were made to the Danau Girang Field Centre (DGFC) to test different nodes, gather knowledge and trial iterations of K-HAS. The first visit showed us just how much the rainforest affected communications, but also allowed us to gather knowledge from researchers and cameras that had previously been deployed. Subsequent visits were then used to test our own nodes and software, based on the knowledge we had gained from the first visit. We developed K-HAS with a view to deploying it in Malaysia, however, it soon became clear that our design was beyond what was currently available, as well as the time constraints.

Developing a camera with both wireless and processing capabilities, as well as being waterproof, proved to a be an extremely difficult task, while wireless wildlife cameras were already commercially available with range far beyond what we had experienced. Before realising this, we attempted to use various node types connected to a camera, such as the Raspberry Pi and

Waspmote nodes, both yielded problems with mounting an SD card that was readable by both camera and node. At the time, we were unable to create a camera combined with a node that could be left untouched for months at a time in a rainforest.

However, we still needed to implement a network to prove our hypothesis and we developed a modification to the K-HAS that utilised the latest commercially available hardware to provide an architecture that provides similar capabilities. Because these changes were made due to issues with deployment in Danau Girang, we chose to name it after a famous animal in Malaysia: LORIS. This stands for Local-knowledge Ontology-based Remote-sensing Informatics System.

LORIS has been developed specifically for the Malaysian rainforest, our motivating scenario, and, as such, much of the hardware and software has been implemented to reflect this. However, the same approach could be used for other WSNs focussing on scientific observations with minor changes required. Using the same ontology and underlying software as K-HAS.

The rest of this chapter is structured as follows. Section **??** highlights the changes we had to make from K-HAS and explains the hardware used at each tier. Section **??** explains how we planned and deployed the network and Section **??** details the results from the deployment in Malaysia.

6.1 Architecture

The aim of LORIS was to keep as much of the architecture of K-HAS as possible and reuse the ontology without the need for modification; serving as a form of validation. In order to do this, we identified the tiers of the network that were feasible and the areas that needed to be addressed. Regular power and cheap computers with good knowledge-processing capabilities meant that DA nodes were simple to deploy and, with the growth of powerful microcomputers like the Raspberry Pi, DP nodes were also in abundance. However, finding a reliable camera that could integrate with a node capable of processing observations that was also reliable enough to withstand the humid rainforest proved to be difficult.

6.1.1 Data Collection

Experiments were run yearly in Malaysia to test the performance of variations of hardware. As covered in Section , the first year involved testing the range of Wi-Fi with an IGEP board. The limited range of 30m meant that, despite the high transfer rate, it was not possible. Wi-Fi was not designed for use in resource-constrained sensors, so the power consumption of the radio meant that it limited the lifetime of the node.

The following year, we used the Digimesh protocol, explained in Section , which provided a longer range and was developed for use in sensor networks. While the range was suitable for the rainforest, it was not as much as we had anticipated and finding a method to mount an SD card on both the existing wildlife cameras and the Waspmote nodes.

From these experiments, we began to look into commercial alternatives that combined both the node and the camera. The most usable solution we found was the Raspberry Pi coupled with a camera attachment [**?**], but it was not ready for use in the Malaysian rainforest or to be used for an extended unsupervised period. Existing wildlife manufacturers were then looked into and we found that wireless cameras had been manufactured, but they had no local processing capabilities and had not been used much in research. Based on these findings, we used Buckeye X7R cameras [**?**]. Using the same Digimesh protocol as the Waspmote, they had a tested range of 1 mile and had been developed to withstand harsh environments.

We expected to achieve around 800m of range and these proved to be suitable DC node replacements, despite the fact that we had to forgo local processing or the storage of any local knowledge.

6.1.2 Data Processing and Aggregation

Due to the lack of processing available on the DC nodes, and limited power availability, we combined the DP and DA nodes into one machine stored at DGFC, that contained both a Digimesh radio and an Internet connection. The benefit of using commercial grade hardware was the software that accompanied it; remote management and configuration software allowed users

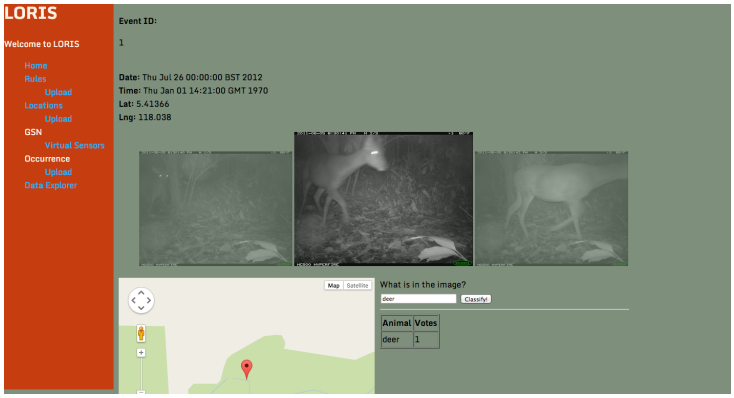

Figure 6.1: LORIS Web Interface

to modify the settings on each camera, as well as handling the retrieval of images from every node deployed. EXIF tags written to images could be modified, as well as how many images were captured for each observation. The software had been created to be used by those without any specialist knowledge and was easily used by researchers in the field centre.

Each observation was saved into a directory that matched the ID of the camera it originated from and this meant that the existing software used in K-HAS could be used without modification. Sensor middleware, such as GSN, is running on the same machine and virtual sensors listen for changes in each directory; where each virtual sensor represents a deployed camera.

When new observations are detected, Drools runs on the images to start metadata and EXiF processing. Some rules had already been extracted from the processing of 120,000 images collected from our yearly visits to DGFC; other rules had to be added by users of the WSN. When an observation had been processed, rules were run once again to determine if a match could be made to existing projects and, if so, who should be notified.

The web interface for classifying and viewing all observations was accessible by those within DGFC, as well as uploading rule files and new locations. Using this interface proved to be easier for those without a WSN background and the GSN interface had a steeper learning curve. Figure **??** shows the basic metadata of an observation, outlining where it was captured and when; as well as the images themselves.

Figure 6.2: Camera Locations Around Danau Girang

6.2 Deployment

In June of 2013, a three week visit was made to Danau Gurang, with three Buckeye cameras and the software required to deploy LORIS. The network was first tested within the field centre and one of the Buckeye cameras had been broken during transport, giving us only two cameras. Due to the protected nature of the forest, we were unable to nail any cables to trees to use the high gain antenna and it was no possible to use the cables without first securing them; because animals tampering with cameras was a common occurrence.

For the initial week of the deployment, we wanted to test the robustness of the network before focussing on the data. This meant placing the cameras in locations where they would be triggered often and in an area where they would be affected by rainfall and humidity. Figure **??** shows the locations of the cameras during both of the weeks that it was deployed. Buckeye 1 was deployed for the first week along the main path leading to the field centre from the river, this experienced the most traffic due to the number of researchers present at the time.

6.2.1 Direct Connection

For six days, we tested two Buckeye cameras that were near enough to the field centre to maintain an active connection to the base station. This meant that no hopping was involved and cameras were not reliant on each other to transmit images. Camera 1 was placed on a main path that guaranteed human foot traffic and Camera 3 was placed on a trail in the forest that, while experiencing minimal foot traffic, was expected to yield small mammals and birds.

For the duration of the six days, the cameras did not report a drop in battery levels and a total of 1076 images were sent. Camera 1 was in a more open environment, despite being further away, and average speeds of 4KB/s were achieved.

Transmissions from Camera 3 were significantly hampered by the dense forest that it was surrounded by, as well as the fact that a low-gain antenna was used. We also speculate that the field centre itself acted as a barrier to the signal, as the base station was located on the opposite end. Because of this, we experienced an average speed of 1.2KB/s, taking around three minutes to receive an image. This is consistent with experiments conducted in previous years where dense, humid forest has led to a decreased range of, up to, 78%. Lower frequencies do experience a longer range, but the data rate is significantly impacted.

6.2.2 1-Hop Network

For a further five days, Camera 1 was moved further into the forest and Camera 3 was set up as a routing camera that forwarded images onto the base station, 63 images were taken during this period.

The location of the moved camera is also shown in Figure **??**. As was expected, the traffic of the network reduced significantly when the cameras were moved from the main path, with almost all 63 pictures being of animals. Due to animals moving a lot faster than humans, the number of pictures taken per trigger was increased to two, this would ensure network traffic was not overwhelming while increasing the chance of capturing the subject.

Surprisingly, the speed of transmission from Camera 1 was faster than Camera 3, despite being routed through it, with an average speeds of 1.8KB/s.

6.3 Results

Deploying a network for two weeks is not a robust experiment and does not show that LORIS can withstand months of running without human intervention. However, it does serve as a proof of concept that, using commercial hardware, a modified K-HAS network can be implemented and utilise the knowledge of its environment.

The pictures of humans from the first six days are not of use for the current motivating scenario, focussed on animals in the rainforest corridor, but one could see this network also being tasked to warn the authorities of hunters in restricted areas of the rainforest. However, the images we have of animals, while few are animals of interest due to the proximity of the cameras to humans, were processed and used to create templates for future observations.

Figure 6.3: First Image of Lizard by Camera 3

More interestingly, we were able to infer further rules from the short deployment that helped us to narrow down the choice of animals to match to

based on the time of day and location. For example, Figure **??** shows a lizard crossing the path of Camera 3 in the middle of the day and Figure **??** shows the lizard walking past in early afternoon on a different day. This allows us to encode a window of time in which the lizard is more likely to appear onto the static knowledge base of DC nodes or, in the case of LORIS, onto the DA/DP node. Images an hour later were captured by the camera, showing the lizard walking in the opposite direction. While this was not every day, and the deployment time was too short to establish a pattern, it does help us to infer that a lizard passing by in early afternoon is likely to walk the same path approximately an hour later.

Figure 6.4: Second Image of Lizard by Camera 3

6.4 Rules

This section explains the rules used in K-HAS in more detail, to highlight how classifications are made as well as how nodes in the network are monitored. Any examples made here relate to our motivating scenario. The rules on each node are a vital way of encoding local knowledge and a practical use to show how local knowledge, from users or previously sensed data, can be used classify newly sensed data.

As highlighted in Section **??**, K-HAS uses the Drools rule engine, a Java based rule management system that uses forward chaining based inference. Forward chaining starts with some data, in our case a set of images, and uses inference rules to extract more data. We use this method to make meaningful inferences from properties of the sensed data. For example, an image taken at 8pm could be run through Drools and an inference rule will tell the system to start by looking for nocturnal animals.

6.4.1 Data Collection

As explained in Section **??**, DC nodes are unable to run the Drools engine and instead utilise static rules that are encoded at the time of deployment. Instead of utilising forward chaining to draw a single conclusion, these rules are executed one by one and the outcome of one does not affect the outcome of another. An example of such rules could be a rule to determine the time of day that an image was captured (such as morning, afternoon, evening or night) or a rule to determine what animals it is more likely to be based on the temperature and location of the node, shown in Listing **??**. No rule engine is used by the DC node and the rules are executed in the code before the observation is archived and sent on.

Listing 6.1: Encoded Data Collection Rule

```
if img.temp > 32 && img.temp < 38 then
classification.potentialAnimals = [
   clouded_leopard, malay_civet, sun_bear]
if img.time > 1800 && img.time < 0500 then
classification.animalBehaviour = nocturnal
```

6.4.2 Data Processing and Data Aggregation

The increased knowledge-processing capabilities allow DP nodes to run a complete implementation of Drools, this means that forward chaining can be used and the knowledge base is dynamic, i.e. it can be updated whilst the

network is deployed.

Rules are written in *drl* files that are loaded into a knowledge session. Sessions, as well as the firing of rules, are handled by the Java code. The benefit of using Drools with a Java based middleware, such as GSN, allows the simple integration of the two technologies, allowing a sensor class in GSN to insert an archive into the session and run the rules. When sensed data is received from a DC node, it is unarchived and Drools is run on the DwC files describing the original observation, the metadata, the originating DC node and the knowledge base of the DP node.

Listing **??** shows an example of some more simple rules that would be contained within the knowledge base. In this file, the rules use existing local knowledge with the properties of the observation to infer the contents and then update the DwC archive, or return suggestions as to what the contents may be. For example, the *Clouded Leopard Ridgeline* rule uses the location ID of the camera to match with a human description and uses local knowledge of the area and global knowledge of the time to infer that the image may contain a Clouded leopard.

Listing 6.2: Drools Rule File

```
rule "Otter River"
        when
                $dwc:DarwinCore()
                DarwinCore(DBTools.
                    getLocationDescription($dwc.getOcc()
                    .getLocationID()) contains "River")
                eval(TimeTools.checkTimePeriod($dwc.
                    getOcc().getEventTime()) == "
                    Afternoon")
        then
                System.out.println("Otter");
                int id = $dwc.getOcc().getEventID();
                Date d = new Date();
                Identification ident = new
```

```
                    Identification(id, 1, d, 1);
                $dwc.setId(ident);
                System.out.println($dwc.getId());
        end

rule "Plantation Night"
        when
                $dwc:DarwinCore()
                DarwinCore(DBTools.
                    getLocationDescription($dwc.getOcc()
                    .getLocationID()) contains "
                    Plantation")
                eval(TimeTools.checkTimePeriod($dwc.
                    getOcc().getEventTime()) == "Night")
        then
                System.out.println("Sun Bear or Civet")
                    ;
        end

rule "Clouded Leopard Ridgeline"
        when
                $dwc:DarwinCore()
                DarwinCore(DBTools.
                    getLocationDescription($dwc.getOcc()
                    .getLocationID()) contains "Ridge
                    Line")
                eval(TimeTools.checkTimePeriod($dwc.
                    getOcc().getEventTime()) == "Night")
        then
                System.out.println("Clouded Leopard -
                    Most likely male");
        end
```

```
rule "Location Description"
        when
                $dwc:DarwinCore()
                eval($dwc.getOcc().getLocationID()==1)
        then
                System.out.println("Occurence in site
                    1, animals here are: Clouded Leopard
                    , Malay Civets and Small Mammals");
        end
```

These rules are very simple and only scrape the surface of the local knowledge that can be utilised, as well as how DwC archives can be effectively used to carry the data of these inferences, but it does show that Drools is a powerful rule engine and that these rules can be extended further to make full use of the knowledge base.

DA nodes use the same rule engine as DP nodes but rules are used for the post-processing of Darwin Core archives. For example, when a classified archive is received from a DP node, Drools is run to check if the classification matches an existing project. If it does, then it searches for those that are subscribed to the project and notifies them via their selected medium. Drools can also be used to monitor the nodes deployed in the network and check their status. Rules are run to check for archives from DC nodes and, if a node has not sent an archive in a set number of days, then administrators are informed that it may have run out of battery.

More importantly, rules are vital for the exchange of knowledge between nodes in the network. If a user classifies an archive on a DA node, then rules are fired that identify the DP node that sent it and update its knowledge base so that the template it uses matches the updated classification.

6.5 Conclusion

K-HAS was developed, with the motivating scenario in mind, as an ideal, general-purpose network architecture that is able to utilise the knowledge of

its environment. However, experiments to support the design and deployment of such a network within the timeframe of this PhD showed that the current technology was not ready to support a network that would run without human intervention for extended periods. LORIS is our more specialised, cut down approach that can be implemented using hardware that is readily available, as well as software that we have open-sourced.

Our deployment in Malaysia has shown that local knowledge can assist with classifications and can be gained from even a few images collected within the first few days of deployment. Ideally, we would like to leave this network running for an extended period to infer rules from the sensed data and test the reliability of the network in harsh conditions.

LORIS has shown that local knowledge can be used to enrich sensed data, and automate classification, when it has been received at an endpoint, but we still maintain that K-HAS' aim of pushing local knowledge right out to the edge of the network will make the network more efficient and allow for a more timely delivery of important sensed data. The main drawback of LORIS is that it relies on the chronological delivery mechanism used by the commercial cameras, whereas we believe that K-HAS can improve this mechanism by sending the data that it infers to be of a higher importance, rather than what was simply captured first.

Chapter 7

Simulations

In this chapter, we detail simulations that we developed to test our hypothesis with a complete implementation of K-HAS. LORIS was implemented because current technology does not fully support the requirements of the K-HAS architecture, because of this we implemented a simulation of K-HAS that satisfies all of its requirements.

Using RePast, a java-based network simulation tool, we created a network to emulate K-HAS. Repast is an agent-based modelling system that allows for agents to be created and placed on a grid. Ticks denote a period of time and simulations can run for a fixed number of ticks, or until stopped. Ticks can also be used to schedule events, such as searching for neighbours, by calling methods that last for a set number of ticks, or begin at a particular tick. Similar to the chaining of rules we described with Drools, ticks are the core of Repast's scheduling mechanism which can be used to schedule single events, as well as chaining events. Consider a train simulation. A train may take three hundred ticks to arrive at its destination, from the train arrival a scheduled event to open doors and make announcements which could schedule other events and so on.

Agents are created, using the RePast SDK and Java classes are used to manipulate their behaviour. Simple networks may only contain basic agents with only a few variations from those provided by RePast. However, for more complex networks, a hierarchy of agents is required and Java's inheritance

can then be used create subclasses of an agent.

A 2D (or 3D) space is used to display the grid and the simulation is run within Repast's own GUI, that provides functionality such as editing the properties of classes, integrating with Matlab, taking screenshots and saving different configurations of the same network.

The rest of this chapter is structured as follows. Section 2 describes the implementation of the network. Section 3 outlines the results and Section 4 compares these with LORIS and the current solution in our motivating scenario. Section 5 concludes our findings and highlights areas that require further experimentation.

7.1 K-HAS Implementation

As described in Chapter **??**, K-HAS consists of three tiers: Data Collection, Data Processing and Data Aggregation. The DC tier focusses on capturing sensed data, performing basic processing and routing sensed data to the DP tier. The DP tier has more knowledge-processing capabilities and nodes are responsible for processing sensed data. Using a rule engine and a dynamic knowledge base, DP nodes process more than just the metadata of sensed data in order to create a classification. DP nodes have a direct connection to the DA node, which is the the endpoint in the K-HAS network, an Internet connected node that stores all sensed data from the network and acts as the central knowledge base.

K-HAS was originally intended to be deployed around Danau Girang and, over the course of three years, we have collected several thousand images and local knowledge interviews in order to configure K-HAS before deployment. Because of this, the data we have collected was used to create a deployment of K-HAS using scientific observations, in order to fit with our motivating scenario. LORIS had already been deployed to capture scientific observations, so this made a comparison of the three (current manual solution, LORIS and K-HAS simulation) possible and because we already had metrics on areas of the network such as battery life, transmission time of a Darwin Core archive and processing time.

Before implementing, we designed the agents required based largely on the existing ontology. Using that, we created a hierarchy of nodes inheriting common properties from a node object. As previously mentioned, we had metrics on range and transmission times from previous experiments and the deployment of LORIS, we used these to create properties for each transmission medium that could be used by each node object.

We also needed to create an object to represent the DwC archives, a Drools REST API had been implemented to work with the LORIS web interface, and much of the DwC archive code was reusable within RePast.

The structure of the simulation is shown below. Network builder instantiates all the nodes, places them on randomly on the grid and schedules events once the simulation has started. The nodes then use the properties of their transmission medium to find nodes in range and create a connection, depicted by a line between the node. The simulation can be run in one of two modes: random or based on a 2010 DG deployment.

DG mode means that it models a 6 month deployment in Danau Girang from 2010. All images taken in that time had the time they were taken, the original camera ID and the size of each image recorded into a CSV file. This file is read by the Network Builder and the contents are provided to each node. The first tick starts at the same time as the first image is taken. The number of deployed nodes matches the number of deployed cameras and, upon each tick, they check whether an image was taken by them at that tick. If so, an image is captured, archived and forwarded to the nearest DP node. This process runs, using the scheduler in RePast, for the six month deployment and the details of each archive (time captured, size, route taken, time taken to reach DA node) is saved into a CSV file when it reaches the DA node. Once the six month deployment has run, the simulation can either stop or continue randomly.

The random mode uses metrics extracted from the images taken at Danau Girang, but the chance of an image being captured at a camera is based on the average capture rate of a camera. The fire rate has been calculated by the average number of pictures captured in a day taken by each camera.

- Network Builder
- Node
  - Data Collection
  - Data Processing
  - Data Aggregation
- Darwin Core
  - Identification
  - Location
  - Occurrence
  - Image
  - Species

7.1.1 Darwin Core

The Darwin Core class represents a DwC archive, encapsulation Identification, Location, Occurrence, Image and Species. The images we have collected from Danau Girang were processed to find details such as the average size when taking at night and day, how often an average camera triggers and the percentage of images with animal content. These data were then used to specify how often a randomly placed node should capture an observation per tick.

Upon each capture, and based on the 'time' of day, images are created, given a random size based on the maximum and minimum size found in the 120,000 images collected from DG and the sum of the images is used to calculate the size of the archive. Using this size, a DC node calculates how long the archive takes to send based on the size and the transmission rate, we assume that the rate stays constant for the duration of transmission. When an archive is sent to the DP node, we used the average time for our image processing tool and Drools engine to run and attempt a classification, which is 143 seconds (ticks). To keep the classifications as general as possible, so

that the simulation applies to any WSN for scientific observations, archives are not classified down to the species level, they are marked as *interesting* or *empty* and then forwarded to the DA node.

7.1.2 Routing

The routing protocol used by K-HAS needs to be dynamic in order to adapt to nodes being added and removed during deployment, while minimising traffic in a resource constrained network. The approach we use a modification the Minimum Cost Forwarding Algorithm (MCFA), described in Section **??**. A cost is assigned to each node, based on how far they are from the DA node, with neighbouring nodes choosing to connect to the node with the lowest cost. However, in normal implementations of MCFA, all nodes are of the same type and simply need to connect to a base station.

In K-HAS, DC nodes cannot connect directly to a DA node, because processing would not take place. K-HAS MCFA works with a discovery phase and a transmission phase. The discovery phase is a scheduled event, taking place at the start of deployment but it can be run throughout deployment to react to nodes being added or removed.

Discovery

Discovery begins at each DA node, scanning nodes in range for DP nodes and sending a broadcast packet with a cost of 0 to inform them that they are within range of a DA node which, in our implementation, uses Wi-Fi. Once received, DP nodes increment the count and forward the packet to any DC nodes within range of them, where we use the range of Digimesh. We found that this method overloaded the DP nodes and all DC nodes within range would connect to the first DP node they receive the broadcast from. We then implemented a method, called *load balancing*, which uses the DC nodes connected to a DP node to calculate whether it should offload new nodes to a neighbouring DP node.

The maximum connections a DP node can have is determined by the total number of DC nodes in the network divided by the total number of

DP nodes, which is held in the knowledge base of the DA node. Once a DP node has the maximum number of connections allowed, it starts to offload to a neighbouring DP node that is also in range of the DC node requesting a connection. If there are no neighbouring nodes then it simply goes over the maximum number of connections allowed, to save DC nodes being left with nowhere to send their data.

If the DC node that receives the broadcast does not have an existing route to a DA node, or the cost of the current route is higher than the received route, it adds an edge to the DP node, increments the count and forwards it to all nodes in range. This process continues until the broadcast reaches the edge of the network. Nodes do not have global knowledge of the route to the DA node, only of their neighbour with the lowest cost.

This phase can be repeated throughout the course of the deployment, simply by scheduling it as an event to occur every n ticks. However, the simulation currently only uses the discovery phase at the beginning of the deployment.

Transmission

Once the discovery phase has been completed, providing nodes are within range of the DA node, the transmission phase begins where only DwC archives are then sent across the network. Observations are captured based on the mode of the simulation and sent to the lowest cost neighbour.

In order to manage transmissions, DC nodes have a *SendState* object that contains the next archive to send, the time to send it and whether it is currently sending. This is used to determine what operations to perform, once an archive has been sent, it is delete from the SendState and the sending flag is set to false. A new archive is then added and sent when the opportunity arises.

When a DP node receives the archive, it begins processing and we have implemented both sending and processing as serial processes. DP nodes use the SendState as well, but they do not add any archive, they add an archive once it has been processed and they then select the oldest archive that has

been classified as interesting, providing that an archive is not already waiting to be sent. The archive stores information about the route it takes, recording every hop, as well as the time it took from capture to DA node.

Scheduled sending events run every thousand ticks, which is configurable, to check the sending state of the node and send any archives in the SendState. The node then waits for the number of ticks that it will take in order to transmit the archive.

Once the simulation is completed, either manually or through a defined number of ticks, the archives in each DA node are iterated over and written to a CSV file, with details like the path it took, total transmission time and time of capture.

7.1.3 Capture

Using the existing data collected from Danau Girang, we calculated how often a camera triggers in a six month deployment, as well as how often the observation contained interesting content.

To calculate the count of interesting images, we processed every directory of images to extract the largest object in the foreground, using our Triton program. Once processed, we iterated through every directory, counted the total number of images and the total number of extracted images. We then calculated the average for each directory and then summed those averages divided by the total number of directories. This gave us a 20.7% chance of an image being interesting.

The chance of camera trigger was calculated by the total number of observations (13,399) divided by the number of seconds in six months (15,552,000). This gives a chance of 0.000861561. A random number is generated every and compared to this, if the value is fewer than or equal to the chance, then an observation is captured.

7.2 Results

As previously explained, the simulation runs in a random mode, or using data from an existing deployment. We added a property that ensured each simulation only ran for 15,552,000 seconds, simulating a six month deployment.

Before any experiments were run, we calculated the transmission time for the current solution at Danau Girang. Using the creation date of each image, and the date that the SD card was collected, we found the average number of hours between capture and being received at the field centre, which was 491 hours. It should be noted that this time does not include the upload or processing of each archive, but there was no accurate way of calculating this, and it does not distinguish between empty and interesting observations.

We performed one hundred runs in which the simulation recorded between 57,000 and 60,000 observations in each run. At the end of each run, a CSV file is written with the details of each observation that is stored on every DA node; it does not take undelivered observations into account. After counting the observations, we calculate the total interesting images and their average time to be delivered. The runs resulted in an average transmission time of 49.3 hours for interesting observations and 2043 hours for empty observations. Figure **??** shows the view of the simulation, with a list of the properties that can be set.

Mode	Avg. Total Duration	Avg. TP Duration	Percent TP Received	Avg. I
NK-ALL	597			
LK-ALL	677	133	27.3	
HK-ALL	491	133	27	
NK-LK	749	292	13.9	
LK-HK	687	18	17.1	

Table 7.1: Simulation Results

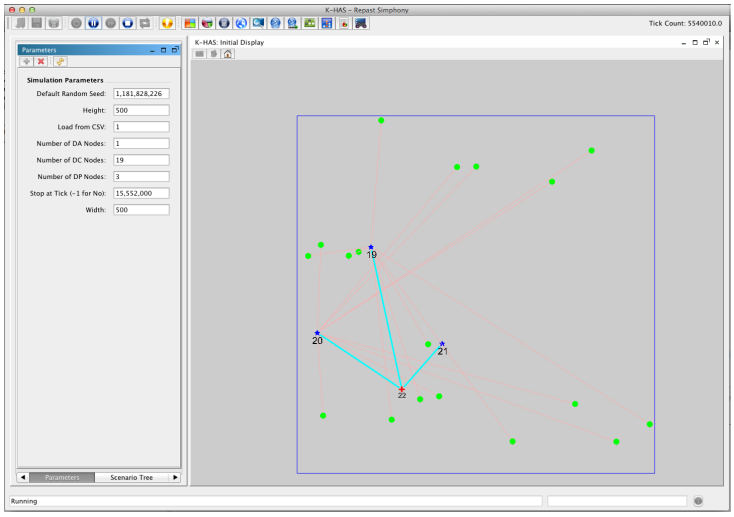

Figure 7.1: K-HAS Simulation

7.3 Improvements

Although the simulations do show an improvement over the current implementation, there are features that could be added to ensure that the simulation models a real-world implementation as closely as possible. Currently, the processing time of DP nodes is based on the average time to process an archive, using a random time within a range would make this less static.

All nodes within the network do not use a battery life and, due to the short deployment time of LORIS, we do not have accurate information on how long batteries last on each node. This is most important on DC nodes as there may be hundreds deployed at any one time and the DA node must report when a DC node does not report for a while. This also goes for transmission, using a fixed range and transfer rate suits a dynamic network in an open environment but obstacles and weather affect the rate and this could be taken into account in a future implementation.

In our implementation of K-HAS, we are using image-based scientific observations but the chance of a camera being triggered is the same for every DC node. In a future version, it would be more realistic to use different chances of triggering based on the cameras location, or based on random assignments. However, we believe that a more general, extensible simulation

that was able to handle many different types of sensed data would be the truest model of K-HAS; using property files, as we currently do, to provide details on the sensed data, processing times and a knowledge base.

7.4 Conclusion

In this chapter we have detailed the development and results of our simulation of the K-HAS architecture. Current technology limits the implementation of the original architecture we developed, so using a simulation framework is the best alternative. We have modelled all variables based on existing data from our motivating scenario and results have shown that using knowledge bases on deployed nodes for the processing of observations within the network allows 'interesting' sensed data to be prioritised and delivered to a DA node more than four hundred hours earlier than a manual solution; which does not include human processing time.

While the simulation is not feature complete, we believe it is accurate enough to show how K-HAS utilises the knowledge-processing capabilities at each tier to process, and prioritise, sensed data based on knowledge gained from the environment, previously sensed data and from humans using the network.

Chapter 8

Conclusion

In this thesis, we aimed to prove that utilising the local knowledge of an environment in a WSN improves the efficiency of the network by giving it the ability to prioritise sensed data based on the results of in-network processing. We believe that pushing knowledge out farther towards the edges of the network improves the overall performance. To prove this, we have developed a three-tier WSN architecture that uses knowledge-processing capabilities to process sensed data as it is forwarded through the network. Most current WSN implementations deliver data chronologically, or store it on the node to be retrieved by queries. We believe that this knowledge can be used to infer how valuable sensed data is and prioritise that through the network, delivering the most interesting data first. However, resources are still limited in WSNs and our architecture had to utilise these resources effectively, such as battery life and bandwidth, to maximise the network lifetime. Using Data Collection (DC) nodes at the edge of the network, they capture observations and use their limited knowledge-processing capabilities to enrich the sensed data before sending it on. Data Processing (DP) nodes use more powerful knowledge-processing features to attempt to classify observations and prioritise the sending of them to Data Aggregation (DA) nodes. DA node make the data available to users and use their classifications, and input, to dynamically update its knowledge base.

We used a collaboration with a research centre in Malaysia to implement

K-HAS in the Malaysian rainforest, in an area that had experienced logging and contained a diverse range of rare wildlife. Using K-HAS, we wanted to deploy a network that would prioritise images of rare wildlife and only send images of common wildlife when bandwidth was available. Over the course of 3 visits, we gathered knowledge from the area, researchers and locals to build a knowledge base and create rules that could be used to classify data. We also collected images taken from their current, manual solution to infer patterns and use existing classifications for new sensed data.

Current sensing technology means that K-HAS is not ready to be implemented as the architecture dictates. DC nodes are not a type of sensor that is readily available, and this is especially true for image capturing sensor nodes. Because of this, we modified the architecture to use commercial hardware with fewer capabilities that would allow us to deploy a sensor network in the Malaysian rainforest that does use local knowledge. We call this architecture LORIS, Local-knowledge Ontology-based Remote-Sensing Informatics System, and this solution combines the DA and DP node to hold all of the knowledge-processing capabilities.

LORIS has shown that even using local knowledge at the base station of a WSN means that sensed data is processed and organised within minutes of being received. This method also means that users are alerted to data that they have subscribed automatically.

Our simulations of K-HAS, the extension of LORIS, show that the bandwidth in a WSN is used more effectively when knowledge is pushed out towards the edge of the network, allowing nodes to perform knowledge-processing to make inferences about the contents of the data and prioritise, or delay, its delivery appropriately.

8.1 Summary of Contributions

In this section, we summarise the contributions detailed in this thesis, focussing on our deployment in Malaysia, the tiered architecture designed for general use and the alignment ontology created to formalise the architecture of K-HAS and the data standard that it uses.

8.1.1 K-HAS

In Chapter 4, we present a novel tiered architecture, K-HAS, for WSNs that uses local knowledge. We explored existing networks, those related to our motivating scenario in Malaysia and also those that use knowledge or context-awareness, and routing protocols that use the sensed data to determine how it is routed. We explain the purpose of each tier in the network and show how sensed data is enriched and routed as it progresses through the network. Using Darwin Core as a data standard, each node communicates in a common format and metadata is packaged with data in archives that can be read by any node in the network. We also show how rules can be used to infer the content of sensed data and the ability to run rule engines in the network allows sensed data to be prioritised based on its value; not just the time it was captured.

8.1.2 Ontology

In Chapter 5 we explain the aligning ontology created to formally represent the K-HAS architecture and the data standard used. We show how no ontology current joins observation-centric and sensor-centric ontologies to completely represent the hardware used in a WSN as well as the sensed data. We also show how the ontology is extensible and in no way specific to K-HAS; it can be used with any WSN that deals with scientific observations. The modular nature of the ontology means that parts of the ontology can be extracted and re-used in a network that the entire ontology may not be suitable for.

8.1.3 LORIS

In Chapter 6, we present the Local-knowledge Ontology-based Remote-Sensing Informatics System (LORIS), a system developed for our motivating scenario when we discovered that current sensing technology means that K-HAS is not ready to be implemented in its current form. DC nodes are not a type of sensor that is readily available, and this is especially true for image capturing

sensor nodes. Because of this, we modified the architecture to use commercial hardware with fewer capabilities that would allow us to deploy a sensor network in the Malaysian rainforest that does use local knowledge. This solution combines the DA and DP node to hold all of the knowledge-processing capabilities and then uses commercially available sensors to replace DC nodes.

This architecture is easier to implement but does not provide in-network processing, although it automates the delivery of sensed data and uses the increased knowledge-processing capabilities of modern PCs to process sensed data on arrival and inform users. We show how our deployment of LORIS was successful and highlighted how sensed data was delivered within minutes of being captured in some cases, and processed shortly after.

8.1.4 Simulations

In Chapter 7, we explain the implementation and results of our simulations to model an ideal deployment of K-HAS. We model every variable of the network on existing data collected from our motivating scenario and prove that the delivery of observations can be reduced by more than four hundred hours, when compared to the current manual solution. We also show how the network is able to prioritise interesting data, delaying the transmission of empty data. The simulation accurately models K-HAS and shows how basic inferences about data can be used to prioritise sensed data.

8.2 Future Work

The focus of this thesis is to show that local knowledge can improve the timeliness of interesting data by making inferences based on previously sensed data and knowledge of the environment. We have shown, using simulations, that this can be done with our tiered approach. However, another aim was to deploy such an architecture for our collaborators. We have explained that current technology means that this is not feasible, but sensing, and microcomputer, technology has moved forward significantly since the beginning of this PhD and it would be possible with a longer time period. Our deployment

in Malaysia was limited due to time constraints and we would like to leave a functional deployment active in Danau Girang for a six month period.

With more time, we could create custom DC nodes using webcams and computers, like the Raspberry Pi, encased in a watertight enclosure; allowing us to use knowledge-processing right at the edge of the network. A deployment of this length would also allow nodes to act on sensed data classified by users and update their knowledge base, responding to changes in the network dynamically.

K-HAS uses a combination of open source projects and some software created in-house, but it does require specialist knowledge to be deployed. We would like to create an installation candidate that was usable by those without any expertise that could provide basic information on the purpose of the network and the installation of all necessary packages would be automated.

On top of this, user interfaces that we have developed have been tested by researchers at Danau Girang but we have no metrics on the usability of the software. Testing the software on users to determine how users would score the different areas of the software, such as usability, response time, ease to learn, would help us to improve the software and ensure that it can be used by those without any knowledge of K-HAS or its underlying architecture.

Experiments have shown us that the range of wireless transmissions is often hard to predict and can be heavily influenced by weather and obstacles. If sensors were placed at the edge of range for a neighbouring sensor, then it is not guaranteed that transmissions could be made every time. We believe that using humans as an intermediate hop could be an interesting research opportunity. Using a mobile app to transfer data between nodes as they are within walking range could speed up data transfers and their knowledge could be used to prioritise data without the need to process it.

If a human views a recent observation when they are in range of a node, they could make an instant assessment on whether it is interesting or not. If they mark it as such, then it could be passed through the network with a higher priority and reach a DA node within a short period as it would not require processing, updating knowledge bases for future, similar observations as it is forwarded.

One important goal is to create a deployment of K-HAS for a completely different purpose than its current animal based implementation. There is a need in Malaysia to track hunters in the forest and alert authorities, this would not require any change in how K-HAS is currently implemented and would show how dynamic it is in its current implementation.

However, using K-HAS for a WSN that, for example, uses text based sensed data to monitor the temperature and lava level of a volcano is a completely different implementation, but the local knowledge could be used to prevent an emergency and predict eruptions. A deployment such as this would show that the benefit of using local knowledge in a WSN is not limited to our motivating scenario, but is versatile enough to benefit almost any network.

Using heterogenous sensor nodes within a K-HAS network would also show how local knowledge can be used in different streams, as well as how they can be combined to make more detailed inferences. Using motion sensors with microphones could be used to determine what person/animal is near the sensor and this information can then be used to infer patterns as fallback sources when an image based classification cannot be made.

Our simulation in Chapter 7 show how K-HAS can use local knowledge, but it is by no means a complete implementation. We need to implement a more modular simulation that allows K-HAS to be applied to any form of WSN. We would like to simulate individual knowledge bases on every node and experiment with networks that contain multiple Data Aggregation nodes to visualise the flow of sensed data through the network and see if there is an effect, positive or negative, on the speed of interesting sensed data. A more immediate goal is to test K-HAS when all of the network uses a communication medium with lower range but a higher transfer rate, such as Wi-Fi, to determine how much Zigbee slows the delivery of sensed data when compared with processing.

We would also like to experiment with different ratios of DC:DP:DA nodes to determine if there is an ideal ratio that maximises the flow of sensed data, delivering interesting data first and quickly but also delivering data that has been classified as not interesting within a shorter time period that would

allow humans to act on the data if it had been misclassified.