Summer 8-2012

SIMULATION, DEVELOPMENT AND DEPLOYMENT OF MOBILE WIRELESS SENSOR NETWORKS FOR MIGRATORY BIRD TRACKING

William P. Bennett Jr
University of Nebraska-Lincoln, wm.paul.bennett@gmail.com

SIMULATION, DEVELOPMENT AND DEPLOYMENT OF MOBILE WIRELESS SENSOR NETWORKS FOR MIGRATORY BIRD TRACKING

by

William P. Bennett, Jr.

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Mehmet C. Vuran

Lincoln, Nebraska

May, 2012

SIMULATION, DEVELOPMENT AND DEPLOYMENT OF MOBILE WIRELESS SENSOR NETWORKS FOR MIGRATORY BIRD TRACKING

William P. Bennett, Jr., M. S.

University of Nebraska, 2012

Adviser: Mehmet C. Vuran

This thesis presents CraneTracker, a multi-modal sensing and communication system for monitoring migratory species at the continental level. By exploiting the robust and extensive cellular infrastructure across the continent, traditional mobile wireless sensor networks can be extended to enable reliable, low-cost monitoring of migratory species. The developed multi-tier architecture yields ecologists with unconventional behavior information not furnished by alternative tracking systems at such a large scale and for a low-cost. The simulation, development and implementation of the CraneTracker software system is presented. The system is shown effective through multiple proxy deployments on wildlife and has been operational for 10 months at the time of writing.

DEDICATION

To the one and only Allison.

ACKNOWLEDGMENTS

I am very grateful for the support and guidance given by Dr. Mehmet C. Vuran. The invaluable experience and lessons learned, will be in no question, beneficial to the rest of my career. I would also like to thank Dr. Matthew Dwyer and Dr. Sebastian Elbaum for serving on my committee. Their extensive guidance definately helped shape the outcome of the crane project, it would not have been the same without them.

David Anthony has been an irreplaceable friend and colleague throughout the project. I hope to continue to work with Dr. Vuran, David, Dr. Dwyer, and Dr. Elbaum on projects in the future, as I feel we make a productive and successful team. I would also like to thank Collin Lutz, Caleb Huenefeld, Derek Homan, and Aldo Arizmendi for their help. Without their assistance, the work in this thesis would not be complete.

The Whooping Crane Tracking project would not be possible without Anne Lacy, Mike Engels and the rest of the staff at the International Crane Foundation. Many thanks to Walter Wehtje, who also enabled the success of the Whooping Crane Project. The Holland Computing Center has also facilitated this work.

I also want to thank my family for the contionus support throughout my graduate studies. I would like to thank my wife Allison, who I owe indefinately for the continous love, patience, and support throughout my academic carreer. Allison, I promise we will take a vacation after this.

GRANT INFORMATION

This work was supported in part by NSF CAREER grant CNS-0953900 and NSF award CSR-0720654.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Mobile Wireless Sensor Networks

For over a decade, Wireless Sensor Networks (WSNs) have been conceptualized as a dense set of low-cost and low-complexity devices that are capable of sensing phenomena [5]. The nodes in these prior networks collaboratively monitor the phenomena by utilizing short-range, multi-hop communication behavior. The nodes are also considered disposable and often, fail. To withstand these failures, the nodes are deployed in high density. To ensure desirable life-time requirements, the nodes feature low-energy consumption characteristics in both hardware and software. These described features make WSN applications ideal for numerous military, environmental, health, home, and commercial applications.

The low-cost and energy efficient aspects of WSNs motivate its use towards migratory bird tracking. Unfortunately, the extremely mobile behavior of the migratory birds make utilizing traditional WSN methods impractical. Therefore mobile Wireless Sensor Network (MWSN) approaches must be considered instead. Mobile WSNs differ from traditional WSNs in many ways. In traditional WSNs the nodes are dense, static, and

use collaborative techniques to disseminate and collect data. It is also assumed that these nodes are always connected, even in cases where some of the devices fail. In the mobile setting, this is not the case. In most cases, connectivity cannot be guaranteed in mobile settings and alternative approaches of communication must be utilized. The energy characteristics also differ in mobile networks since the location of the neighboring nodes are usually indeterminable. Thus, mobile WSNs are unable to benefit from methods used in the traditional WSNs, where the static configuration allows the network to coordinate energy saving efforts across all the nodes.

The enhanced sensing, life-time, and communication capabilities available at a low-cost make utilization of mobile WSNs practical. However, existing mobile WSNs [72, 94] are far from ready to fulfill requirements of a system that monitors a mobile migratory species over a large geographical area. To utilize the mobile WSNs, the behavior of the mobile entity must be predictable or understood. Predictability of the migratory birds mobility currently falls short, where the end-points of the migrations are mainly understood. The known end-points suggest the use of delay-tolerant networks for monitoring the migratory species [11]. However, the current state of the art, which utilizes satellite technology [11, 56] is more responsive than this method. Migration over great geographical areas make dependency on solely delay tolerant networks inadequate for tracking purposes. Instead, the acceptable method to reach similar responsiveness, would be to deploy a large set of access points over the geographical area. This is clearly too costly and impractical. For tracking efforts, it was observed that pre-existing cellular networks could be utilized to add connectivity to compete with the state-of-the-art migratory tracking approaches. This approach is also novel for monitoring species with wireless sensor networks. In this thesis, the approach to monitor the endangered Whooping Crane with multi-modal sensing and communication devices is described. In addition to these efforts, the prior sensing capabilities of the ecologists are enhanced by utilizing a solid-

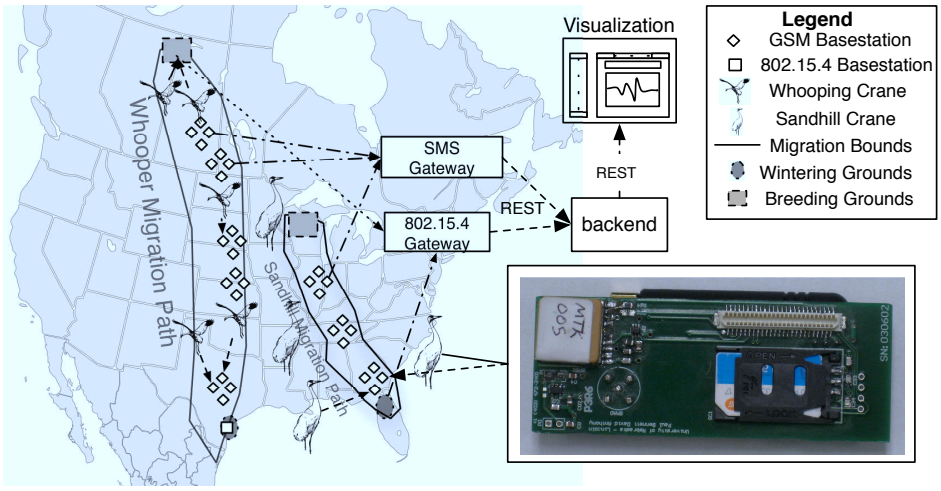


Figure 1.1: Proposed architecture for Whooping Crane monitoring.

state compass for behavior identification. Furthermore, this work provides insight on the process to develop a mobile WSN of this type. In this process the development, simulation, and deployment techniques created and utilized to make a functional system, are described.

This work aims to address issues found in monitoring migratory species. The prior efforts, which will be discussed in Chapter 2, are costly and are incapable of providing necessary behavioral information. Moreover, the timeliness that the information is received, exceeds the usefulness for field observations and collection of perished birds.

The mobile WSNs exhibit ideal characteristics for tracking migratory birds. However, the short-comings of prior wildlife tracking efforts with WSNs motivate creation of a more capable tracking system. The short-comings will be discussed later in Chapter 2. This work aims to address these short-comings by development of a tracking device, back-end and visualization system, simulation tools, and evaluation of the system through proxy deployments.

The final system is illustrated in Figure 1.1. In this system, the devices are mounted to the migratory birds. As the birds migrate they off-load sensor data through the SMS

gateway. Alternatively, when the birds are located in wintering or breeding grounds, the data can be off-loaded with short-range radio gateways (802.15.4). Once the gateways receive the data, they are then made available to researchers through back-end services and visualization components.

1.2 Contributions

The contributions of this work can be broken up into the following areas.

Novel Tracking Platform: The idea to increase connectivity and behavior sensing capabilities using the cellular network and a solid-state compass for mobile WSN tracking is proposed. A prototype is developed to demonstrate the capabilities of the platform. This prototype is later improved by David Anthony, so it is able to be deployed in the wild [10]. The prototype development set in place the initial software used in the final application.

Embedded Software: In the development effort, driver software is written for the novel components since they were previously unavailable. In these efforts, an interrupt-based I^2C, system time, and minor bug fixes are made to TinyOS. In addition, high-level components that utilize the developed drivers are created and tested for use in the tracking efforts. This software has been used in alternative research [103] and course projects.

Back-end Software: A back-end and visualization tools are developed to manage and visualize network data. The back-end is composed of web-services that are responsible for processing, managing, and facilitating access to information, with little delay. The visualization utilizes the web-services to illustrate network behavior. To this end, the system has been operational for a period of 10 months at the time of writing. Moreover, the effectiveness of the developed software has proven useful in more than migratory

bird tracking but in alternative projects [103].

Simulation Tools and Evaluation: Multiple simulators were created in the early stages of the project to help estimate performance and testing of system components. For performance analysis, GPS and Crane simulations are developed that assist with decision making by estimating the performance of the system before it is deployed. For testing purposes, compass and GSM simulators are created that emulate the internal behavior of the respective devices. This capability improved testing efforts when hardware was unavailable.

Deployments: The endangered status of the Whooping Crane limits testability of the system, making it difficult to justify its use in migratory tracking efforts. To this end, a set of proxy deployments are carried out. Each deployment effort was designed to share a similar characteristic with the target migratory species. The deployments added necessary feedback into the testing process and allowed the system's effectiveness to be evaluated. The evaluation led to interesting observations and confirmed the approach proposed by this work is useful for migration tracking efforts.

Testing Framework: The available tools for general purpose development are unavailable for embedded software testing and development efforts. This is especially the case when utilizing the TinyOS operating system [67]. During the development process, a rich set of tools are created and made available to support embedded testing efforts. The testing framework features a port of a standard assertion library, unified instrumentation framework, improved unit testing capabilities, runtime verification techniques and a system integration testing tool. The use of these tools play a large role in the success of the tracking efforts described in this work.

1.3 Project History

Understanding the history of the Whooping Crane tracking project is necessary to best present the work described in this thesis. Initial wireless sensor network Whooping Crane tracking effort was by Rahul Parundare and Chris Thiel. This work was carried out in collaboration with Dr. Felipe Chavez-Ramirez at the Crane Trust [99], and Dr. Matthew Dwyer and Dr. Sebastian Elbaum at the university to fulfill requirements of the Wireless Sensor Networks (CSCE896) course project. Upon completion of the course, David Anthony [10] and I were brought on-board to continue work on the project. The initial effort by us to utilize delay-tolerant techniques and the multi-modal prototype development was performed in collaboration with Dr. Chavez. Later the effort was carried out in collaboration with Walter Wehtje. This relationship was beneficial, as Walter Wehtje assisted with our effort in the initial Wild Turkey deployment. Shortly after the Turkey Deployment, Walter Wehtje introduced us to ecologists at the International Crane Foundation (ICF) [45] at Baraboo, WI. The rest of the work evolved in collaboration with Anne Lacy and Mike Ingles. The result of this collaboration led to successful deployments on captive Siberian and wild Sandhill Cranes. With their assistance, the developed system was able to be proven useful for migratory bird tracking efforts [11].

In the rest of this thesis, the development, simulation, and deployment of the tracking system are discussed, including the outcome of the work, with exciting results. The background and motivation beyond this effort is discussed in Chapter 2. The design and development of the tracking device and embedded software is discussed in Chapter 3. In Chapter 4, the design and development of the supporting software, the back-end and visualization components, are discussed in detail. Subsequently, in Chapter 5 the developed simulations and analysis of these efforts are explained. Finally, in Chapter 6 the efforts to portray the usefulness of the system, through deployments, are described.

Following this discussion, the appendix presents a set of testing tools and methodologies developed that assisted with development efforts.

Chapter 2

Background and Motivation

This chapter describes the background and motivation of the work described in this thesis. The motivation behind this work is towards conservation efforts of the Whooping Crane. This is described in Section 2.1. The ideal characteristics of the mobile wireless sensor networks (WSN) make them applicable for monitoring the Whooping Crane. The details of prior mobile WSN deployments and their employment towards wildlife tracking are described in Section 2.2. From the prior effort, it was clear a new system was necessary. In Section 2.3, the requirements created in collaboration with ecologists are described. Finally, the challenges of meeting these requirements are discussed in Section 2.4.

2.1 Whooping Crane

The Whooping Crane (Grus americana), depicted in Figure 2.1 is a highly endangered species domestic to North America with only 575 birds known to exist. The Whooping Crane can be broken up into two populations, the Aransas-Wood Buffalo Population (AWBP) and the re-introduced population. The AWBP population is the only remaining

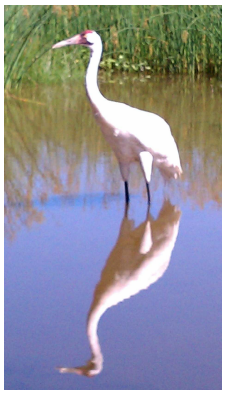


Figure 2.1: Captive Whooping Crane

wild population with only 279 birds. The re-introduced population consists of 300 birds, that exist solely from ecologists' efforts to re-establish the bird population [62]. The cranes are extremely mobile migratory birds capable of traveling distances up to 950 km/day. The AWBP population migrates from Canada to Texas every year, covering a distance of 4,000 km. The re-introduced population migrates a distance of 1,800km from Wisconsin to Florida every year. The re-introduced population was actually taught by the ecologists to perform this migration utilizing ultra-light gliders [54]. As depicted in Figure 2.1, the cranes are very vivid and identifiable birds. They are white and very large. The Whooping Crane is about 1.5m tall, and can weigh up to 7kg [11].

The ecologists are very interested in observing the behavior of the bird during the migration and while not migrating. A great amount of conservation effort has been put forth to evaluate the birds' geographical behavior [61, 62, 108]. In these observations, the ecologists are interested in identifying the particular causes of death, the flight dynamics, and how the birds spend their time while not flying. In the prior efforts, tracking was accomplished with VHF and satellite transmitters [11, 61, 62].

Although beneficial to conservation efforts, the prior trackers are inadequate for multiple reasons. In the early tracking efforts [61], the VHF transmitter tracking method

required the unit to be manually monitored with a receiver device. This was accomplished using an airplane [61] or by following the birds in a vehicle. This is inefficient and demands many man-hours to observe an entire migration that can take many weeks to complete. This method is still utilized since it is still more cost-effective, when compared to that of the satellite transmitters.

The satellite transmitters are useful to the ecologists, since they do not require them to do anything except capture and mount the devices. The limited recharge potential and the inability to store data leave large holes in the data received by the ecologists [62]. On top of the reliability, the timeliness in data reception is found to be around 48 hours [108]. In cases where bird observation is desired, the 48 hour delay exceeds the usefulness of the position information, since the birds are extremely mobile. For these reasons, a more responsive and reliable system is sought-after [11].

The researchers are interested in studying more than the bird location behaviors and causes of bird mortality. As an example, the ecologists want to understand why the re-introduced cranes are unable to reproduce [19]. This is considered to be one of the more pressing issues that are impeding with the reintroduction efforts. The ecologists suggest that black-flies in the Wisconsin breeding grounds negatively influence the birds breeding ability. The ecologists propose that when harassed by the flees, the birds flee from their nests. In addition, the ecologists advocate that the lack of energy, due to malnutrition, cause the birds to flee the nests in search of food [19]. It is suggested that malnutrition may be caused by human dominated landscape. It is proposed that the orientation and acceleration can be utilized to confirm either of these conjectures. In other words, if the behaviors can be identified by the system, the system can notify the ecologist when and where they occur. This information can then be used to confirm or reject their claims [19]

In previous efforts, studies were carried out utilizing MEMS sensors to identify micro-

behaviors [49, 90, 111, 112]. However, these devices rely on data logging techniques [112] and recapture for data collection. This is clearly not acceptable for Whooping Crane tracking efforts, where recapture of the birds is an unlikely occurrence. Utilizing the multi-modal sensing and communication as proposed in this work, it is believed these behaviors can be identified and reported remotely [19].

2.2 Mobile Wireless Sensor Networks

As described in Chapter 1, traditional Wireless Sensor Networks (WSNs) are static networks made up of many low-cost wireless devices. The traditional networks are designed to be very dense with a quantity of nodes large enough to handle the natural faulty charateristic of the wireless devices [4]. The desired use for these devices included military [4], environmental [3], health [21], home [2], and commercial [34] applications [4]. The data generated from the sensing devices tend to be lightweight. This is necessary so that the network can handle routing the information with collaborative multi-hop communication. However, these networks are designed in such a way that significant mobility can cripple the network with minimal effort [47]. The increasing importance of wireless sensor networks and vast number of applications where they could be applied, quickly motivated use in mobile applications [47].

The devices that reside in mobile WSNs are usually equipped with processing, sensing and communication capabilities. The duration for which the devices are useful, are influenced by how well the devices manage the power available to them. The use of this energy must be tightly managed. Moreover, the nodes must be equipped with energy harvesting capabilities [89] to extend life-time. The devices in mobile WSNs also feature power-aware characteristics [70]. This is necessary to maintain application lifetime since the devices are equipped with limited energy supplies [70]. Great effort goes into

the design so each aspect of the system (processing, sensing, communication) is power aware [70].

The processing on the devices are accomplished utilizing an efficient micro-controller. The micro-controllers operate in the mega-hertz range, usually never exceeding 20-MHz [75]. This characteristic extends lifetime, however impacts processing performance. In addition, the micro-controllers feature micro-amp low-power modes that are utilized to extend application lifetime [14].

The devices in mobile WSNs also feature simple sensors used to observe phenomena. In some cases, the devices can be equipped with complex sensors for capturing more interesting phenomena (i.e gieger counter, soil moisture, compass, and GPS unit [11, 31, 40]).

The devices of these networks are furnished with low-power short-range radios [4]. The radios enable the devices to create ad-hoc networks, and use multi-hop techniques to disseminate and collect data [51, 53]. Since the devices are low-cost, the communication protocols have dynamic characteristics to handle possible faults that can occur during deployment [60]. The networks have dense characteristics and overlap to maximize connectivity [4]. The transmit-power of the radios are also limited in order to conserve energy over the life of the device [4].

2.2.1 Deployment

The many ideal characteristics of mobile WSNs make them ideal for use in many monitoring applications. The effort made to track migratory species, fall into the environmental monitoring applications category [4]. Because of this, the scope of deployment background will be limited to environmental application category, even though there are many deployment efforts in other areas [2, 4, 21, 34].

The environmental monitoring WSN deployments can be classified as pre-defined, random, and mobile deployments [4, 10]. In pre-defined networks, the units are placed in known areas. These units are deployed in such way so the nodes can take advantage of known communication protocols, eliminating the need to behave adaptively. One example of this type of network is in a large scale deployment observing red-wood trees [102]. Due to hardware difficulties, the researchers had problems keeping the network alive for a substantial amount of time. This effort motivates better testing practices and power monitoring capabilities to the proposed system .

In [4], random deployment of nodes is considered for forest fire detection systems [115]. However, these types of deployments must be dense to facilitate communication. As a result, the random deployments may lead to difficulties in guaranteeing connectivity, efficient routing of data, and lifetime during deployment. These types of networks require adaptive communication behavior, thus increasing complexity of the software.

The mobility architectures used in mobile WSN deployments can be classified onto three areas [6]. These areas include flat, 2-tier, and 3-tier architectures [6].

Flat Architecture: The mobile nodes use ad-hoc communication to connect with stationary nodes [6, 7]. In addition to the mobile node communication, the stationary nodes can communicate with other stationary nodes. An example of this type of network is that of found in underground wireless sensor network deployment [31]. In this deployment, the nodes are deployed in the soil and if close enough, are capable of communicating with each other. Alternatively, a center pivot irrigation system can be equipped with an above ground node that is capable of storing data as it sweeps across the network. The center pivot system is an irrigation method where a mobile pipe waters crops in a circular motion. In this case, the researcher must go to the site and offload the data from the mobile node [31].

2-tier Architecture: The nodes utilize two separate networks for collection and dis-

semination of data. One network is for communication between the mobile nodes, the other is for communication with the static nodes. The mobile nodes are able to coordinate efforts to meet up and mule information nodes. It is also assumed the mobile nodes are able to be controlled [6]. An example of this deployment is shown by the NavMote system [6, 41]. The NavMote system utilizes dead-reckoning to monitor pedestrians. In this system, the NavMote is carried by the pedestrian. As the pedestrian walks around, the NavMote transmits movement data to RelayMotes (tier-1 network). The relay motes communicate with each other to transmit back to a collection gateway(tier-2 network). The data eventually is forwarded to a user to identify the pedestrians location.

3-tier Architecture: In this type of network, one tier contains heterogeneous nodes that are capable of communicating among each other. In addition, the heterogeneous nodes are capable of communicating with the mobile nodes (ad-hoc, tier-1). The mobile nodes can communicate with each other on a separate network (ad-hoc, tier-2). In the 3-tier architecture, the mobile nodes are capable of communicating with an additional network (tier-3) [6]. The third-tier is usually purposed to offload data over great distances [6, 103]. This architecture closely resembles that of the system developed in this thesis [11]. In this network, the mobile node can off-load to the static network in the breeding and wintering grounds (tier-1), capable of communicating among each other through ad-hoc networking (tier-2), and can communicate with the cellular network over great distances (tier-3).

2.2.2 Wildlife Tracking

There have been existing efforts to monitoring wildlife by deploying WSNs for zebras [69], badgers [72], and sea birds [94]. These types of deployments are implemented usually with two types of networks, infrastructure-based networks and ad-hoc networks [11].

The infrastructure type of deployment requires the devices to be carefully placed in the environment where the wildlife exists [11]. This is seen in one of the first large scale wild-life deployments. The Duck Island deployment, where seabirds are monitored by observing the breeding environment [94]. As the birds move through the environment, the light sensors are used to sense movement around nests. The researchers must spent a remarkable amount of time strategically placing devices throughout the island to observe behavior.

In a similar infrastructure deployment, efforts to track badgers with RFID and magneto-inductive technologies are carried out [72]. In this deployment, magneto transmitters are placed throughout the badger's habitat. As the badger (tagged with a receiver) moves around the environment, the device on the badger records when the magneto transmitter signal was received. This occurrence is recorded and stored until it can be off-loaded when the badger exits the borrow. In migratory tracking efforts, the required infrastructure of this approach is impractical since the species covers a great geographical area during migration [11].

This is evident not only with the migration tracking effort, but as well with the efforts to track zebras, as found in the ZebraNet deployments [69]. In the ZebraNet deployments, multi-hop ad-hoc networking and data muling techniques are used to collect Zebra movement information with a GPS collar [69]. The work performed for the zebra tracking efforts can not be directly applied toward tracking migratory birds since they act quiet differently than the zebras [11, 69]. More specifically, the population densities of the zebras outweigh that of the Whooping Crane, thus solely utilizing multi-hop communication for data collection would not be sufficient for monitoring Whooping Cranes. In addition, the geographical area covered by migratory birds are greater in size than the zebras [11, 69].

Tracking a migratory species that cover great distance require more than sole use of

multi-hop and data-muling capabilities. The system must also be able to utilize long-range communications such as cellular communications to maintain connectivity over a large area. There are alternative cellular trackers [77, 95] are capable of utilizing cellular communications for migratory bird tracking [11]. However, these devices are not equipped with ad-hoc capabilities to take advantage of desired ad-hoc behavior [11]. Furthermore, these tracking devices do not have sensors that can observe behaviors such as preening and foraging [77, 95]. Cellular devices have also been deployed on seals [11, 68] and people [68, 73]. The delay experienced through the seal tracking efforts exceeds the desired latency for tracking migratory birds [68]. Moreover, the smart phone devices used in the people tracking effort, do not have adequate energy resources or recharge capabilities to be used for migratory bird tracking [73].

The system deployed in this work utilizes greater sensing, and communication capabilities than work discussed in this section. In the proposed architecture, a hybrid approach is taken, where the existing cellular infrastructure is used for long range communications and ad-hoc communication is used short-range communication attempts [11].

2.3 System Requirements

The frailty of the prior efforts described motivate a new approach for monitoring Whooping Cranes. To develop a new monitoring platform, a set of requirements and goals are developed in accordance with ecologists. The ecologists have extensive background in crane conservation efforts, and their experience assists with justification of the requirements discussed in this section [11]. To satisfy the ecologists needs, the following requirements must be carried out by the system [11]:

- Weight: $<$ 120 grams

- GPS: 2 samples per day
- Location Accuracy: $<$ 10m desired, $<$ 25m acceptable
- Compass:
  - 0.5Hz sampling rate
  - 3D Acceleration
  - 3D Orientation
- Communication Latency: $<$ 24 hours
- Life-time:
  - Multi-year (3-5 year)
  - Enclosure for protection
  - Solar panel for energy harvesting

In the tracking efforts, the system must be able to withstand the demands of monitoring an unpredictable and highly mobile behavior of the birds. The system must also fulfill rigid timeliness requirements so the ecologists can gain insight on behaviors at the stop over locations. The ecologists will use this information to determine the environmental and health status of the bird. In the unfortunate event of mortality, the ecologists must be able to recover the perished crane to determine cause of death [11].

For research efforts, the system must localize the bird at least twice per day, once during the day and the other during the night. The accuracy of the acquisitions must fall within 25 meters of the definite position, in order to be useful for tracking efforts. The position data collected must be accurate enough to determine mobility patterns and vegetation that birds inhabit. The data can also assist with understanding the influence of specific geographical regions that play a role in the behaviors of the birds [11].

One of the pressing issues with previous efforts is the timeliness of the data. It is very important to the ecologists to receive the data collected within 24 hours. The timeliness of the positions influence the ecologists ability to visually evaluate the bird in field observations and in the unfortunate case, recover a perished bird, before ingested by a predator. With satellite tracking efforts (48 hours [108]), the delay is too extreme to accomplish this desired responsiveness [11].

The ecologists must understand and characterize the behaviors than they were previously capable of before. In other words, they must be able to quantify visual observations with actual collected information. The developed system, is capable of this characterization utilizing the solid state compass. The solid state compass is a type of MEMS sensors, and as discussed in Section 2.1, has been used to identify behaviors. With this capability, a better comprehension of behaviors can help expose any irregular behavior [11]. The orientation and acceleration data can be utilized to determine if the bird is dead or if the unit may have fallen off during deployment. This capability further assists with efforts to determine cause of death, since the compass is able to observe phenomena not available in prior tracking efforts. Thus, to observe these behaviors with the compass, the high sampling rate of 1Hz for 10 seconds every four hours, is necessary as backed by the data collected in this work [11].

The cranes are extremely endangered and difficult to capture, and almost impossible to recapture. Consequently, the system must be able to thrive over multi-year deployments with out interruption, to collect statistically valid information. More importantly, the system must be able to continuously observe bird behavior so ecologists can be responsive when necessary. Hence, the system must be able to withstand the harsh environments, and have the ability to recharge its energy sources [11].

The uniqueness of the study carried out in this work, necessitates a flexible design and operation. The actual sampling rates and duty cycle required for such operation

is not directly known. The studies carried out by the tracking efforts in this work, will play a crucial role as a baseline for future efforts. The developed system must sample a variety of data and attempt to communicate at a fixed rate. In addition, the developed system must be configurable enough so deployment results can be act as feedback for immediate future deployment efforts. In any case, it is difficult to tailor predictable sensing, communication and sleeping strategies while maintaining the desired high level of reconfigurability [11].

The negative influences of prior tracking systems, on flight and copulation, motivate the ecologists to utilize alternative approaches in future tracking systems. The prior approaches employed leg-bands [61, 62, 108]. These leg-bands have been observed to add drag and unbalance during flight. In addition, it is unacceptable for the tracking device to interfere with crane copulation, since this is a pressing issue. To mitigate these problems, the birds require the developed unit to utilize backpack mounting techniques during deployment. Consequently, the back-back approach will be used the developed system [11].

To facilitate the network and data management for Whooping Crane monitoring, it is necessary to have a back-end system. The requirements for this back-end system are discussed in the following section.

2.3.1 Back-End

Mobile WSNs are only useful if they are appropriately governed and made readily accessible to researchers. The following requirements are necessary for the back-end system to be useful in Whooping Crane monitoring efforts [78].

1. **Distributed:** The back-end must not have a single failure point. This is necessary so the nodes on the network are able to off-load data whenever possible, since the

connectivity of the mobile nodes cannot be determined.

2. **Deployable:** The back-end must be reliable and simple enough so it can be executed on any server. Alternatively, to load-balance during production the back-end must be able to launched in multiple instances to handle expanding network loads.

3. **Configurable:** The back-end must support changing different network parameters, add and remove exchange formats, and easy modification of internal parameters.

4. **Interoperable:** The back-end must be able to exchange information between multiple devices, and handle an evolving network.

5. **Loosely coupled:** The back-end must not really on implementation details or knowledge of the underlying network. This is necessary for handling heterogeneous, and evolving underlying networks.

6. **Self managed:** The back-end must auto-discover new nodes joining the network. In addition, the back-end should handle nodes continuously connecting and disconnecting from the network. The back-end must also handle reconfigured nodes, so failure node's identifiers can be re-purposed.

7. **Handle large data volumes:** The back-end must never reject information transmitted by the network. This information should be retained and analyzed externally to whether it is accepted or not for analysis. Furthermore, the back-end must handle the expanding data that grows over-time. The devices deployed in this effort are also designed to operate for multiple years. Finally, the back-end must handle hundreds and possibly thousands of devices capable of pushing thousands of bytes simultaneously.

8. **Integrable with other systems:** The ecologists have utilized prior systems to manage ecological data. The new back-end must be capable of integration with systems such as this, and additional systems in the future.

9. **Reusable:** The back-end must be designed in such a way that it can be purposed for alternative usage.

10. **Visualized:** The data must be able to be visualized by the ecologists, so network state and the monitored species can be observed.

There has been a significant of effort to apply standard practices to network and data management through middle-ware applications, in WSNs. The types of tools available include standard visualization tools [104], network querying techniques [71, 114], and service oriented middle-ware approaches [16, 24, 78]. The usefulness of networking and data management system can be based on the back-end requirements [78].

One of the original management tools for WSNs is MoteView [104]. MoteView was designed to handle large data volumes, observe the health of the network, and visualize the data generated by it [104]. The system is configurable, made data available through TCP/IP, abstracts details of the wireless sensor network, and observes network behaviors. However, MoteView is incapable of enabling external parties to access the data. The data is also modeled in such a way that it can not adapt to data generated from a dynamic mobile network [78]. Moreover, MoteView relies on a single database and SQL for storing and retrieving data. This is problematic when the system needs to be fault-tolerant and handle multiple nodes. If a failure occurs with the single database, the system may fail to record important data from the network. In a critical application this is unacceptable. A network comprised of many heterogeneous networks must be complemented by a network and data management system able to adaptively han-

dle intermittently connected nodes in a distributed manner, to be considered useful for migratory bird tracking.

The idea to observe WSN phenomena from a single database can be undesirable. TinyDB [71] and Cougar [114] take the approach of executing queries on the network itself, rather than querying the database after data is collected and stored. However, the centralized and reactive approach used in this method is undesirable for a mobile and intermittently connected network [78]. In mobile wireless networks this approach becomes a problem when connectivity cannot be predicted. Thus, the queries can never be determined accurate and complete since the query cannot be ran against all or most of the nodes in the network. In addition, this approach adds complexities in software and the data model of the deployed devices. For example, the network and data management system in this approach must be coupled into a single data model to ensure the queries are consistent. This coupling becomes a problem when the network is heterogeneous, rapidly changing and/or increasing in quantity over time with significant software changes and improvements. Moreover, TinyDB assumes a traditional ad-hoc network, where the nodes feature a single communication medium and the devices are not actively changing. The network and data management of the system proposed in this thesis is more suitable for tracking efforts by being adaptable, abstracting WSN details, and able to handling data collection from multiple communication mediums (interoperable).

The characteristics found in mobile WSNs motivate the use of the service oriented architecture (SOA) for managing the network and its data. Thus, there have been many efforts in Wireless Sensor Networks to utilize SOAs in the middle-ware [16, 22, 36, 59, 78]. The network and data management back-end for tracking whooping cranes must be interoperable and be able to be easily deployed, most likely in the "cloud" (interoperability and runtime support [78]). Furthermore, the data and network management capabilities

must handle dynamic and rapid deployments found in development of a migratory bird tracking device. Thus, the back-end must self-manage and auto-discover nodes in the network [78]. Likewise, the back-end must be able to handle a large volume of information and simultaneous connections that can occur during deployments [78]. Moreover, the communication delay requirements of the ecologists must be satisfied and supported by the back-end [78]. Finally, the data collected during the deployment of the networks must be able to integrate with existing and external systems [78].

In existing work [16, 22, 36, 59], the interoperability expected from these SOA systems does not exist [78], thus rendering them not useful to migratory tracking efforts. It is necessary to have this for migratory bird tracking, where many evolving and heterogeneous devices required to exchange information. The runtime support also required for mobile WSNs lack in prior efforts [16, 24, 36, 78]. In addition, the ability to handle the dynamic nature of the desired network with self-organizing capabilities is currently non existent [16, 24, 36, 78]. Moreover, the previous efforts did not design their systems to handle large volumes of data and traffic, again limiting their usefulness in bird tracking efforts [16, 22, 24]. Furthermore, the lacking Quality of Service requirements in previous efforts [16, 36], make them ineffective for tracking efforts were rigid timeliness requirements of the ecologists exist. Finally, the back-end system must integrate well with the legacy and future systems utilized by the ecologists, unfortunately the systems are incapable [22, 59].

2.4 Challenges

The limitations illustrated by prior migratory tracking efforts, suggest tracking migratory species is extremely difficult. For the system to be considered for deployment, it must follow rigid weight and size limits as enforced by the ecologists. The device must

way less than 110 grams. This weight restriction is necessary so the tracker does not negatively impact the birds health and cause irregular behaviors. To mitigate possible side-effects, the weight limitation is comprised from a rigid 2% body weight limit. On average, the birds weigh approximately 6kg, thus the 110 gram limit. For comparison, 1 gram is comparable to a standard paper-clip [11].

The weight requirements are extremely stern when compared to other wildlife tracking efforts. For example, the unit developed for zebra tracking weighed approximately 1.1kg [58]. Conversely, in badger tracking efforts, the tracker has similar weight characteristics (105g [72]). However, the operational life-time of the system has lower expectations (5 months) than the system proposed in this work (multi-year) [11, 72].

The cranes are capable of covering great distances during the migration periods [61, 62, 11]. The rate at which this is accomplished has been verified to reach up to 950km in one day. This behavior makes it nearly impossible to accurately predict flight paths, stopover points, and flight durations between migration points [11]. As discussed in previous mobile WSN studies in Section 2.2, the ability to predict the mobile node behavior heavily influences the communication performance. It is intended to use the breeding and wintering grounds as locations to setup ad-hoc infrastructures in the future, since this behavior is able to be predicted.

The required lifetime to perform valid studies by the ecologists, is over a duration of 5 to 7 years [11]. Thus, the management of the energy in standard and faulty operation is necessary. The power requirements of the components to accomplish the sensing and communication tasks, make operation with only a battery impossible. Consequently, the system is equipped with solar harvesting components to replenish resources for continuous operation. Measures are also in place in hardware to recover from fatal faults that occur over long periods of time.

The data observed in prior efforts is limited, and does not feature aspects required

to support operational design choices. This is a result of the prior tracking efforts to be limited in sensing and communication capabilities [11]. It is not directly known how to configure the parameters in software operations. Thus, the system must act as a baseline for future deployments. The system has hardware and software measures in place to recover from faults not exhibited during testing.

The most crucial challenge is to not harm the endangered Whooping Crane. New technology in wildlife monitoring must first ”prove” its value towards conservation efforts. Accordingly, to validate the systems usefulness, it is necessary that the system is heavily tested and deployed on alternate, less influential species [11]. Moreover, it is extremely important for all wildlife to be handled by trained professionals, so no negligent harm is brought to the studied species.

Chapter 3

Design and Implementation of the Tracking Device

To assist with realizing the requirements discussed in Chapter 2, a hardware platform is needed that is capable of capturing intimate details of the migratory species. To determine if a platform is suitable for such needs, it is necessary to evaluate the platform in terms of sensing, communication and life-time capabilities. The sensors on the platform must sense acceleration and orientation timely enough to observe behavior (0.5Hz). In addition, the communication capabilities must be able to maintain connectivity while the migratory species is mobile (data received $<$ 24 hours), covering great geographical ranges. Finally, the system must be able to survive multi-year deployments [11].

After a hardware platform is considered capable, the platform must become animated through software. This software must facilitate intrinsic abstractions of the hardware through drivers, so higher-level components can be designed around them with ease. The higher-level software components must also feature energy-aware characteristics and refrain from abusing system resources. However, the higher-level software components must not be too conservative or else they may miss important details of the

observed migratory species [11].

The effort made to achieve this hardware and software functionality is described in this chapter. The chapter is organized into the following sections: First, in Section 3.1 the division of responsibilities between Dave Anthony [10] and I in development efforts, are discussed. Next, in Section 3.2 the motivations towards the tracking device design are covered. Following, Section 3.3 discusses the hardware development efforts made throughout the project. The efforts in hardware development are composed of the initial off-the-shelf approach, development of the prototype and then the final hardware used in the deployments. Finally, the software developed for the tracking devices are discussed in Section 3.4.

3.1 Division of Responsibilities

This project was heavily developed in conjunction with David Anthony. The responsibilities of the project are divided mainly into hardware and software. However, the initial idea of utilizing the cellular infrastructure and compass sensor, was developed by myself for Embedded Systems (CSCE 436/836) course project under the direction of Dr.Vuran. The result of this work is a prototype to exhibit the potential of a platform composed of multi-modal sensing and communication capabilities. This prototype exploited the use of cellular networks to increase connectivity of the tracking platform during the migration, instead of solely relying on delay tolerant networks. The prototype also employed a compass to collect acceleration and orientation data. Use of this type of sensor for Whooping Crane research is novel. The final product of this work is based on this previous work performed during a course project.

To make the system deployable on actual wildlife, Dave had made tremendous changes and additions. Dave also contributed efforts in the software development. The

initial delay tolerant manager, flash storage, MoteWorks GPS driver revision, short-range base-station protocol, and numerous fixes were performed by Dave. As Dave did with the hardware prototype, the software has changed substantially as documented in this thesis.

3.2 Motivation

As discussed in Chapter 2, the motivation to utilize Wireless Sensor Networks for monitoring migratory birds transpired from problems that manifested in prior tracking efforts [61, 62, 108]. In prior efforts [61, 62, 108], the satellite and VHF transmitters utilized are not capable of meeting the sensing and delay requirements of the ecologists. In addition, they are expensive ($> \$1000$/unit). The low-cost, sensing, and communication capabilities exhibited by WSNs motivate their use towards migratory bird tracking.

To fulfill the ecologists' requirements, the WSN must be able to monitor the location and the behavior of the migratory birds. In addition, they must be able to collect data in a timely manner as discussed in Chapter 2. To be able to monitor the location of the birds, the nodes in the WSN must be equipped with a GPS receiver. To monitor the behavior of the bird at the observed locations, the nodes must be furnished with additional sensors, such as an accelerometer and magnetometer. In order to collect the location and behavior data, the nodes must be equipped with a radio.

In the initial effort, delay-tolerant techniques are attempted. The details of this effort are described in Section 3.2.1. This hardware was equipped with sensors and communication components thought to be adequate for migratory bird tracking. However, through these efforts, it was found that this approach did meet the delay, sensing, communication, and life-time requirements of the ecologists.

The short-comings that transpired from the effort to use off-the-shelf devices, moti-

vated the development of a new platform that was able to fulfill the requirements of the ecologists. The approach featured multi-modal capabilities not previously available in Wireless Sensing Networks. In Section 3.2.2.1, the motivation behind the multi-modal communication is discussed. Following, in Section 3.2.2.2 the motivation for improved sensing capabilities are explored.

3.2.1 Initial Off-the-Shelf Approach

The hardware used in this project has undergone multiple revisions. Initially, the goal was to utilize an Iris wireless module connected to an MTS-420 sensor board. The Iris mote is an off-the-shelf mote equipped with a processor, flash memory, and short-range radio. Using the off-the-shefl mote would minimize extra development effort by eliminating hardware development all-together, and providing code to construct an application to monitor the Whooping Cranes. The Iris mote developed by Memsic, is composed of flash storage IC, an Atmel 8-bit micro-controller, and an RF230 wireless communications module. The MTS-420 sensor-board also developed by Memsic, composed of a GPS, accelerometer, magnetometer, humidity, pressure, temperature, and light sensors.

The initial tracker software was developed using Memsic's MoteWorks platform [28]. This platform software was a commercially maintained version of TinyOS-1.x. Throughout the rest of this thesis, reference to TinyOS will essentially refer to MoteWorks. However, the source developed that is targeted for the MoteWorks platform can be executed successfully on TinyOS-1.x software branch.

The ecologist at the beginning of the project, did not require immediate update requirements. Instead, by Dave, the initial software application was developed to rely on delay-tolerant techniques for data collection. Therefore, this initial state machine was tailored to acquire sensor reading at specific times during the day, store sensor data into

flash memory, and then go to sleep during the rest of the migration. After the migration, the data would be collected from the devices, at predictable breeding and wintering locations.

In the initial approach, the GPS driver provided by TinyOS was extended to support the (Global Positioning System Fix Data) GGA message provided by the NMEA1083 specification. This task was carried out by Dave. In addition, Dave developed a crane manager that utilized the flash, radio and sensors from the MTS-420 that would best fit the sensing and delay requirements of the ecologists at the time.

3.2.2 Pitfalls of Off-the-Shelf Components

The off-the-shelf approach displayed many problems. First, the power source of the off-the-shelf mote was inadequate. This problem was realized after analyzing the energy performance of the GPS device during component experiments. At a sampling rate of 2-fixes a day, the mote would only last for 14 days, utilizing the power supply advised by Memsic. This falls short of the multi-year deployment requirement. The next problem was connectivity. The connectivity issue arose after performing communication range tests. The tests results displayed that the device was only capable of communicating a distance up to 150 meters. To maintain connectivity to compete with related trackers, the ecologists would be required to deploy an unrealistic quantity of base stations over the migration path. The final issue was the sensing capability. The ecologists sought-after a better understanding of flying behaviors, ground behaviors (i.e foraging, resting), and fatality detection. The MTS-420 was not capable of capturing these behaviors adequately since it only featured a two-dimensional magnetometer and accelerometer. From all of these shortcomings, the off-the-shelf approach was found unsuitable for migration tracking and motivation to find a better solution was explored.

3.2.2.1 The Multi-Modal Communication Approach

As discussed Chapter 2, short-range multi-hop communication schemes are insufficient for maintaining connectivity in a highly mobile sensor network. This was evident during the initial off-the-shelf approach. However, this functionality could not be removed since the well-known breeding and wintering grounds provide an opportunity to establish ad-hoc communication with the cranes [11]. The ecologists still needed to maintain connectivity to perform field observations, thus the migratory bird monitoring effort calls for a multi-modal communication scheme. In a multi-modal scheme, multiple communication mediums are utilized to enhance connectivity of the system.

For a long-range communication solution, two alternatives exist: satellite and cellular communication [11]. As discussed in Chapter 2, the previously deployed satellite transmitters exhibited undesirable performance, thus are not adequate for the new platform. The cellular communication method is chosen as an alternative because the necessary infrastructure is already deployed throughout the migration path. Moreover, the operational cost is significantly lower than that of satellite systems [11].

At the time of development, two main technologies were available when choosing the cellular device: CDMA and GSM. GSM technology was chosen as the technology used because of its international deployment. This property enables future deployments all over the globe [11].

As depicted in Figure 3.1, the idea behind the new platform is to exploit an extensively deployed cellular network designed to cover the human population. Moreover, the cellular network covers a majority of the known migration path. Exploiting this phenomenon for long-range communication enables the system to frequently update the ecologists, rather than waiting several months to recover the data in a delay-tolerant network. In situations where coverage is minimal, the data will be persisted to flash

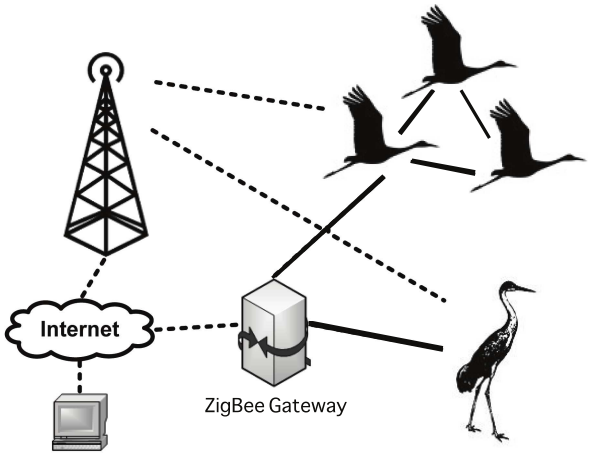


Figure 3.1: Proposed Architecture for Whooping Crane Monitoring

memory. Alternatively, the previously available 802.15.4 base-stations can be deployed at known locations at the winter and roosting grounds, where the data can then be forwarded to the back-end systems [11].

3.2.2.2 Unveiling Novel Behaviors with Multi-Modal Sensors

The tracking device was required to sense location and observe behaviors of the migratory birds while they migrate and reside in wintering and breeding areas. The off-the-shelf device did not have adequate sensing capabilities to capture behaviors with the granularity desired. Therefore, a sensor capable of sensing movement through acceleration in *three* dimensions is needed. In addition, it is desired to be able to sense the orientation in *three* dimensions, which the MTS-420 was not capable of [76]. The MTS-420 is able to sense location with a GPS unit. However, on the event of sensor failure, the node is unable to localize. An additional method to localize the tracking device is needed by the new platform.

To provide the ecologists with these capabilities, a new multi-modal communication and sensing approach is necessary. In Section 3.3.1, the prototype developed to display

the potential of a multi-modal communication and sensing platform is discussed. Following this discussion, the final hardware platform, developed by Dave, is explored 3.3.2.

3.3 Hardware

To monitor migratory species with Wireless Sensor Networks, the network nodes must be equipped with sensors able to determine location and identify behaviors exhibited through movement. In addition, the sensor nodes must be able to process, store and communicate data over the network. Initially as discussed in Section 3.2.1, off-the-shelf nodes capable of sensing location, and two-dimensional acceleration and magnetic fields. In addition, the node is capable of ad-hoc communication short-range radio communication. Following the initial approach, it was found that the off-the-shelf components were inadequate in sensing, communication, and life-time aspects (Section 3.2.2).

From the pitfalls, the motivation to utilize multi-modal communication and sensing techniques were discussed in Sections 3.2.2.1 and 3.2.2.2. To realize the potential of a multi-modal communication and sensing platform, a prototype is developed. The details and evaluation of this prototype are discussed in Section 3.3.1. Finally, in Section 3.3.2 the final hardware platform used in the deployment efforts, developed by Dave, is described. The result of the deployment efforts will be left for discussion in Chapter 6.

3.3.1 Initial Multi-Modal Prototype Hardware

As described in Chapter 2, the ecologists are interested in observing the flight patterns, intermediate roosting grounds, bird deaths, and micro behaviors, with data collection latency comparable to the previous tracking devices. As identified with prior off-the-shelf experiments, this approach was not sufficient for tracking the Whooping Crane over long distances for long periods of time. Thus, a prototype is developed to satisfy

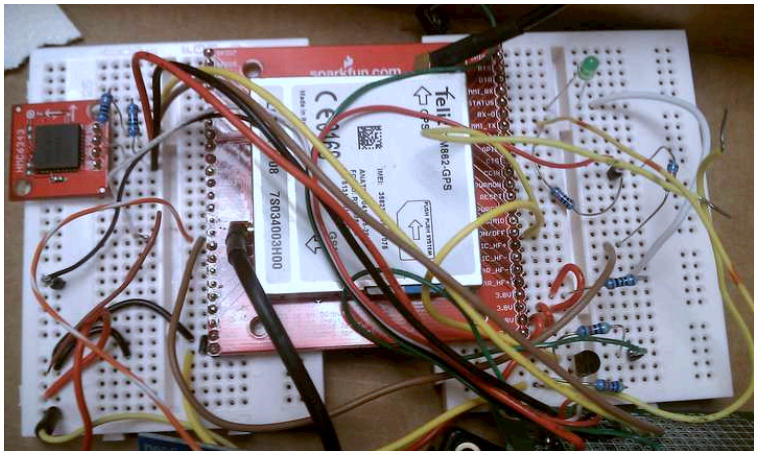


Figure 3.2: Prototype that includes the compass (left in red) and GM862-GPS in the middle.

the requirements of the ecologists and demonstrate the potential of a device capable of multi-modal sensing and communication. This is accomplished by utilizing the Iris mote and GSM cellular chip. The prototype introduces improved sensing capabilities by employing an integrated GPS receiver and a solid-state compass.

The initial prototype hardware is developed to illustrate the capabilities of a multi-modal communication and sensing platform. The prototype utilizes the Iris mote for processing, storage and short-range communication. The prototype adds a long-range communication device (GSM), GPS sensor and compass. These additional components come together to form an platform able to fulfill the requirements of the ecologists for Whooping Crane tracking efforts.

The initial prototype, illustrated in Fig. 3.2, makes use of an MDA100 prototyping board to interface components with the Iris mote through a 51 pin expansion connector. The prototype features a Honeywell 3-axis digital compass with tilt compensation [55], and a Telit cellular quad band module with an internal GPS (GM862-GPS)[98]. The compass is capable of sensing heading, pitch, roll, temperature and 3D acceleration data. The Telit module encompasses a GPS device and a GSM transceiver. The dual-purpose of

the GM862-GPS, eliminates the need to interface an additional GPS sensor. In addition, the GM862-GPS module contains a battery charging circuit to assist with charging the battery when connected to a DC source. All of these integrated components reduced initial prototyping costs and design complexity.

In the following sub-sections, the components that make up the prototype are described in detail. In Section 3.3.1.1, the solid-state compass that is used to observe behaviors is described. The GSM and integrated GPS are discussed in section 3.3.1.2. Finally, in Section 3.3.1.3 the components necessary for operation are explored.

3.3.1.1 Solid-State Compass

To capture granular orientation and acceleration behaviors of the bird, a solid-state compass is added to the prototype [55]. The solid-state compass represents orientation in the form of pitch, roll, and yaw (heading). The yaw is 2^o accurate according to the data-sheet [55]. Furthermore, the data-sheet states that the pitch and roll is accurate to $\pm 1^o$ in the 0^o-15^o range, and $\pm 2^o$ accurate in the 15^o-60^o range. In addition to orientation, the compass yields acceleration data in meters per second squared, in three-dimensions. Lastly, the compass equips the user with temperature data in Celsius. The temperature data is used to observe the environment the bird inhibits.

With these sensing capabilities the orientation and movement can be monitored with high-fidelity. In addition, the environment can be monitored through the temperature sensor. The potential of the compass is brought-out in the deployment efforts discussed in Chapter 6.

3.3.1.2 GSM Cellular Module and GPS Receiver

The short-range radio on the Iris mote did not exhibit adequate communication performance in the off-the-shelf approach. Thus an alternative communication medium is

necessary to meet the requirements, as discussed in Chapter 2. To extend the range of the system and maintain the sensing capabilities found in other tracking devices, a GSM module is used. The GSM module chosen for the prototype is the GM862-GPS [98], which features a quad-band radio capable of use all over the world. In addition, the GSM module utilizes a 20-channel SIRFstarIII GPS receiver capable of initialization capabilities to enhance time-to-first-fix acquisitions. The estimated position resolution can be less than 2.5m under ideal conditions with the GM862-GPS. The Telit module is interfaced with UART, and implements the Hayes *AT* serial command protocol for interaction with external entities [96]. As discussed in the Appendix B.1, there are custom commands for configuration, sending/receiving data, and retrieving the position information.

3.3.1.3 Hardware Implementation Details

For the prototype to function properly, additional circuitry and components are required. A voltage divider is used to drop the battery supply voltage from 3.7V to 3.3V for the Iris and compass, since they operate at different levels than the GSM module. The data sheet also suggests using a $100pF$ tantalum capacitor is added in parallel with the battery. $100k\Omega$ pull-up resistors are added to the UART RX/TX lines to facilitate communication between the Iris and the GSM module. A NPN bi-polar junction transistor is added and purposed to toggle the GSM module on and off. For debugging purposes, a status led is also added to the circuit. The blink rate of the LED signifies the association status of the module. It was not necessary to add pull-up resistors to the I^2C bus as specified by the compass data-sheet [55], since the micro controller included this capability. The GPS receiver and the GSM transceiver require external antennas for position acquisition and communication, respectively. The battery on the prototype is charged utilizing the charging mechanism of the GM862-GPS module and a variable DC power supply.

All the components of this prototype are available by many distributors and have

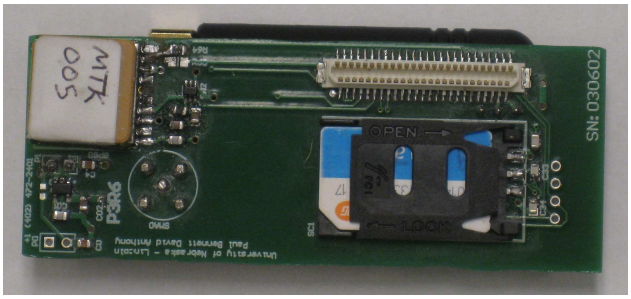


Figure 3.3: The final version of the release hardware.

a combined unit cost below $400 USD, which is substantially lower in cost than the state-of-the-art monitoring devices used for migratory birds. The transmitters used for tracking efforts cost around $5,000 USD and include only feature a GPS unit and satellite transmitter. As with the satellite transmitter, the GSM module requires a subscription for usage. However, the GSM module has a significantly lower usage cost (20$/mo unlimited SMS fee w/ family plan) when compared to the satellite transmitter ($>$ 1000 per unit [108]).

The capability of the prototype components is shown by a proof-of-concept application discussed in Section 3.4. The proof-of-concept application demonstrates the sensing and communication capabilities.

3.3.2 Release Hardware

The prototype (Figure 3.2) was incapable of being deployed on wildlife due to the large size and form-factor. It also lacked essential circuitry to monitor energy, maintain health, and extend the lifetime of the platform. Because of this, Dave developed a platform that was capable of being deployed that utilized the same compass and a similar GSM component from Telit. This section describes the modifications made to the platform so it was capable of deployment on wildlife. The final platform is depicted in Figure 3.3.2 and is described in more detail in Dave's thesis [10]. In this effort, an improved cellular

module, GPS unit, solar-panel and power circuitry were added. The Iris and solid-state compass were still used from the prototype [10].

The GM862-GPS was replaced with a GE865 [97] and a PA6B GPS receiver [50]. Each component was interfaced using UART communications. In addition, each component included a switch that was capable of toggling the supply power to the components. This functionality enables the platform to save energy by powering off components when not in use. The cellular chip still featured a quad-band radio capable of being utilized internationally. The cellular chip had similar energy characteristics but was in a smaller form factor [10]. The GPS receiver featured a built-in compact patch antenna, which eliminated the additional antenna required by the GM862-GPS [98]. In addition, the receiver had a higher receiver sensitivity and consumed less energy than the GPS encompassed in the GM862-GPS [10, 98].

The solar panel and power circuitry were added to extend life-time and maintain battery health of the system. The power circuitry allows the platform to monitor energy characteristics and protected the health of the system. The protection components of the power circuitry disable all components when insufficient power is available, thus protecting the battery. As observed in deployments discussed in Chapter 6, the protection components are successful in maintaining desired life-time of the system.

3.3.3 Power Consumption Analysis

Modeling the power consumption of a sensor platform is essential for developing energy efficient solutions. To analyze each component in the system, a data acquisition unit is used to measure the current through and voltage across the components. The analysis inspect the energy of each component at different operational states. This is discussed for each component in the prototype next.

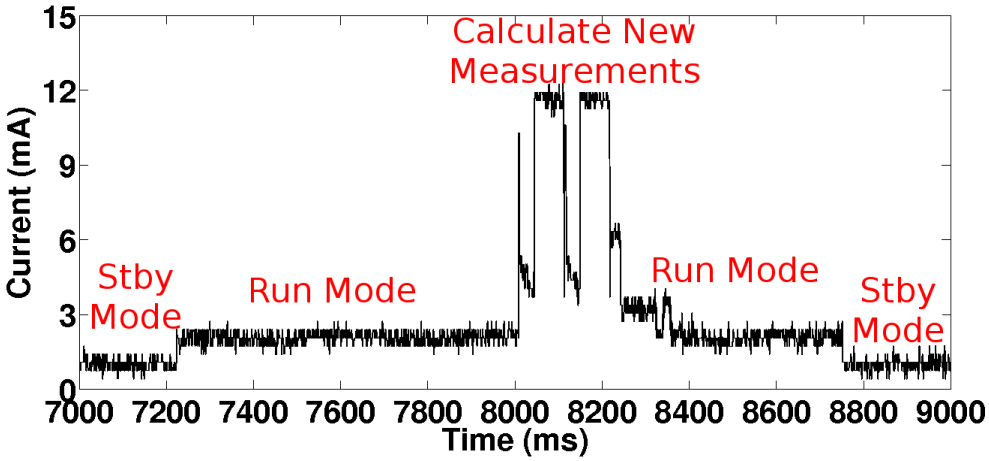


Figure 3.4: The compass current over time switching from stand-by to run mode.

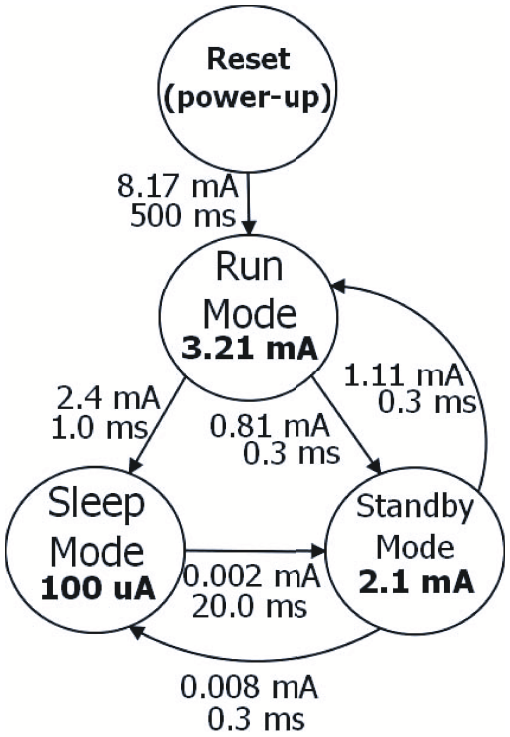


Figure 3.5: The compass power-schema considering average current.

3.3.3.1 Compass

The compass features three operational modes: run, standby and sleep. Sleep Mode is defined as featuring the analog circuitry powered off (lowest power consumption) while power is applied. Standby mode includes the compass being fully powered, but no measurements are performed. Run mode is the mode when all the internal sensors are activated while waiting for commands.

The compass load in milliamperes over time, during stand-by and run-mode is depicted in Figure. 3.4. During the Run Mode, when a new reading request is sent by the Iris mote (at 8000 ms in Figure 3.4), the module calculates heading, pitch, and roll values using raw acceleration and magnetic readings (8000-8200 ms in Figure 3.4). Following the acquisition, the information is then relayed back to the MCU (8200-8400 ms in Figure 3.4). It is observed in Figure. 3.4, that the acquisition peaks at 12mA during the reading acquisition. During the rest of run mode, compass draws 2.4mA. After successfully completing the sensing period, the device transitions down to standby mode until the next operation, where a current draw of 1.1mA is measured. Unfortunately, the sleep mode current is so low that it is unable to be measured with the data acquisition unit, but the compass is specified to sleep below 100μA [55].

Using the data from the experiments, the energy schema for the compass is illustrated in Fig. 3.5, where the costs of operational modes and transitions are shown. The arrows indicate the flow of transition along with the respective current and time delay costs. From this schema, it is obvious that energy can be saved by utilizing the sleep modes instead of completely powering off the devices since the compass consumes 8.17mA of current over 500ms at start-up. However, this may be only beneficial to systems that are continuously using the compass. For crane tracking efforts, the power consumed by the start-up is negligible when compared to other parts of the system. In the grand scheme,

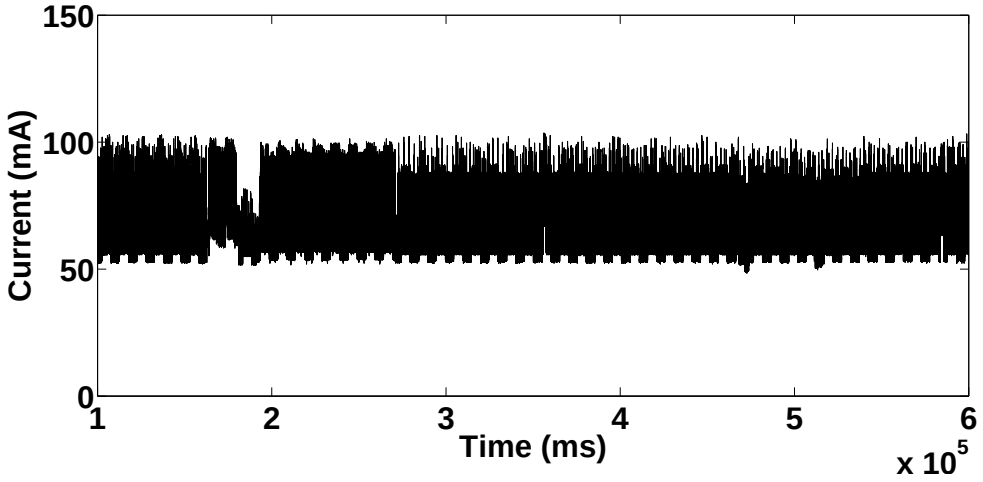


Figure 3.6: Energy consumption of position acquisition of the GPS unit.

the compass is utilized far more less (1 sample every 10 seconds for 10 samples) than sensors such as the GPS device (300*seconds*) , at significantly less energy levels (3.21mA vs 50mA). This understanding motivates utilizing the sensor more frequent behavioral observations.

3.3.3.2 GPS Power Consumption

The Time-To-First-Fix (TTFF) is plotted over time while attempting to get a GPS fix in Figure 3.6. When testing the energy consumption of the integrated GPS of the GM862-GPS, the cellular and low-power transceivers are disabled. Therefore, the GPS operation is isolated to improve current sensing accuracy and interference minimization. In addition to the isolation, the GPS is reset to start in cold-start mode, which exhibits a more realistic approach to the crane tracking application. This approach is more realistic on account of the cranes' mobility, where they fly a considerable distance between acquisition opportunities. Therefore, it is assumed that the platform would not benefit from warm-start information. According to the experiments, the average time to first fix (TTFF), in 10 runs, is found to be 34.75s with a load of 67.25mA. The tests were conducted in

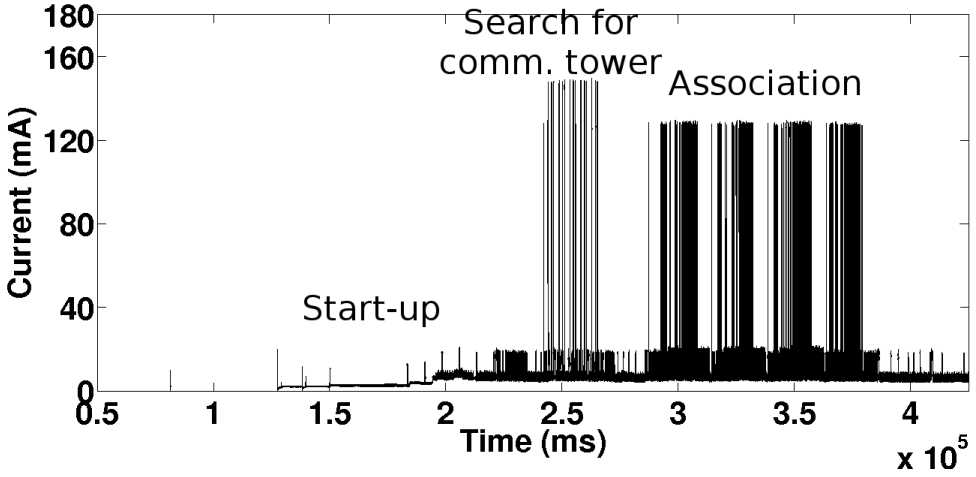


Figure 3.7: Current over time during GSM association with base-station.

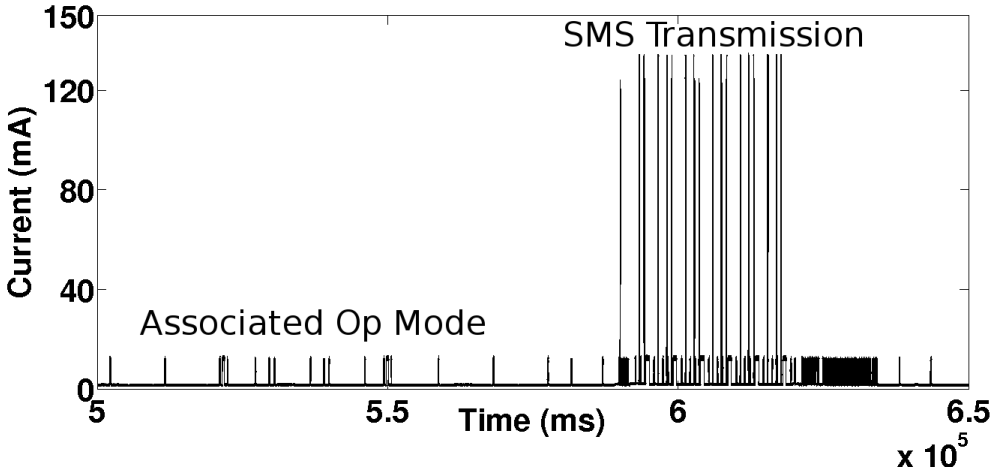


Figure 3.8: Current over time during SMS transmission.

partially cloudy conditions, and the performance falls closely to the expected data-sheet values [98]. This data reinforces the importance of limiting the amount of time the GPS is active to preserve energy resources.

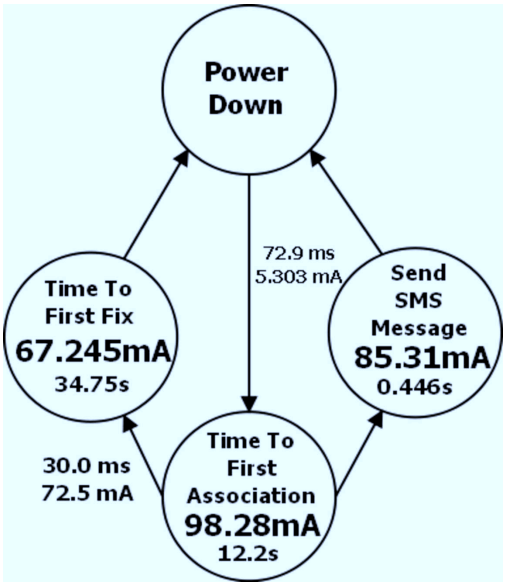


Figure 3.9: GPS and GSM Power Scheme for the GM862-GPS.

3.3.3.3 GSM Power Consumption

In the resulting tracking application, the GSM is used for long range communications. In efforts to ensure data is transmitted reliably, the GSM utilizes the SMS transmission capabilities since they function with the lowest network service available. This is in contrast to GPRS where it is required to maintain a continuous IP connection to a server for data off-loading. Instead, the carrier appropriately handles the routing of SMS messages. At a high level, the GSM (GM862 [98] and GE865 [97]) have two primary states were high energy consumption can occur in the tracking application. The first high energy consumption state occurs while the GSM is associating with a communications tower. The other high energy consumption state occurs while the GSM is transmitting SMS messages.

The energy measurements during time to first association, is depicted in Figure 3.7. After 10 trials, the average time for the GSM association (TTFA) is found to be 12.2s. In addition, the measurements during transmission of a single text message is illustrated in

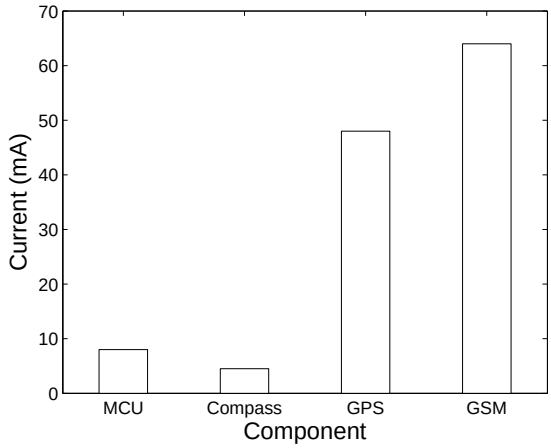


Figure 3.10: Power consumption of platform components.

Figure 3.8. The software user guide suggests that a time of 60 seconds should be allowed before determining that an SMS has failed [96]. Over this period of time, a considerable amount of energy can be exhausted. According to the experiment results, the GPS and GSM module (GM862-GPS) energy consumption can be characterized as depicted in Figure 3.9. In Figure 3.9, most of the energy is spent during association, when compared to sending an SMS message. The GE865 that is used in the final application features similar radio characteristics, since the underlying radio is of the same GSM family provided by Telit [97, 98]. This power analysis suggests that appropriate monitoring of the GSM device is necessary, so the platform does not deplete resources while attempting to associate in locations were service is unavailable.

The overall power consumption of each component of the platform is shown in Figure 3.3.3.3. It can be observed that GPS and GSM modules consume significantly higher energy compared to the rest of the components. Hence, it is necessary to limit the active time of the GSM and GPS units to sustain life-time requirements of the ecologists as described in Chapter 2.

Device	Processing	Storage	Sensing	Communication	Power Supply	Re-charge	Protection Circuitry
Off-the-shelf	Atmel 1281	AT45DB 512KB Flash	Acceleration(2D), Orientation(2D), GPS(ext. ant.)	RF230(short-range)	AA Batteries	None	Yes
Prototype	Atmel 1281	AT45DB 512KB Flash	Acceleration(3D), Orientation(3D), GPS(integrated + ext. ant.)	RF230(short-range), GM862-GPS(long-range)	3.7V LiPo Battery	None	No
Release	Atmel 1281	AT45DB 512KB Flash	Acceleration(3D), Orientation(3D), GPS(Integrated Antenna)	RF230(short-range), GE865(long-range)	3.7V LiPo Battery	Solar Panel	Yes

Table 3.1: A summary comparing hardware versions.

3.3.4 Summary of Major Hardware Versions

In table 3.1, a summary of the major hardware revisions are shown. This summary includes the Iris + MTS420 that was found inadequate for monitoring the Whooping Crane. In addition, the prototype is shown, that adds a higher resolution acceleration and orientation sensing capabilities, and an additional long-range communication device. Finally, the release hardware developed by Dave [10] is displayed. This platform includes the additional sensors and communication devices provided by the prototype. However, it features a more compact design, energy harvesting and protection circuitry necessary to fulfill life-time requirements.

3.4 Device Software

The tracking device features complex software that is developed to fulfill the sensing and communication requirements of the ecologists. The embedded software features low-level components and high-level components. The low-level components interact with the hardware to abstract the underlying behavior. The high-level components manage the hardware through the developed abstractions.

The goal of the device software development is to construct drivers that are capable of maximizing the potential of the underlying hardware. It is also desired for the abstraction to be simple enough so programmers are able to utilize the hardware platform without requiring extensive knowledge of the underlying hardware. In addition, the software aspires to maximize the usage of sensing and communication components while maintaining the desired multi-year deployment requirement. Finally, the device software aims to be reliable enough to recover from failures, and maintain the desired life-time.

The challenges experienced during device software development make reaching these goals difficult. To be able to maximize the potential of the devices requires a great understanding of the components. This process requires studying of data-sheets and user-manuals, and this takes a great amount of time to complete. In addition, it is difficult to guarantee successful operation of the provided software. For example, the GSM driver is required to provide upper layers with the ability to send SMS messages. However, the birds mobility characteristics make determining if cellular coverage is available, a difficult process. This erroneous behavior must be handled and adequate information about the underlying state must be available to upper layers. Furthermore, it is challenging to determine how long to sleep, sense and communicate. If the software does not provide enough time to sleep, the device will quickly exhaust available energy resources. Alternatively, if the software sleeps to long, the tracking device will not observe enough behavior to be useful to the ecologists. The mobility characteristics and endangered status of the bird reinforce the importance of testing. If the device is not adequately tested, the system may fail during run-time. However, testing the device is also challenging because of the limited processing and storage capabilities of the hardware.

The software development for the embedded platform was completed in two phases. The first phase was towards creating a function prototype to display the usefulness of

the multi-modal sensing and communication device. The second phase was towards the final tracking platform to be used in migratory tracking efforts.

3.4.1 Prototype Driver Development

To fully utilize the new components, drivers are developed. Each driver abstracts the hardware, enabling higher-level components to utilize hardware, mitigating device task management from the developer. The extensive knowledge of the underlying hardware required to develop driver software make this process difficult.

The firmware is developed using the TinyOS operating system. The TinyOS operating system does not feature a kernel space, and is compiled statically with the application before being uploaded to the device. Moreover, the TinyOS operating system does not carry-out management of component functional units such as I^2C or UART. As a benefit, these traits enable the developer with freedom to directly interact with physical-layer components. However, this can lead to unintentional harm to the system during runtime. By lacking separation of user-space from kernel-space and functional unit arbitration, developing drivers becomes even more difficult. Development becomes difficult because it is unknown if a hardware component is utilized by a different part of the system with out directly studying the source code of pre-existing components.

The prototype driver development work is organized as follows. In Section 3.4.1.1, the hardware interrupt-based implementation of the I^2C protocol is discussed. This development was necessary to facilitate communication with the solid-state compass. Subsequently, the solid-state compass driver development is described in Section 3.4.1.2. Following, the software driver developed for the GSM unit is explored in Section 3.4.1.3. Finally, the application software utilized to demonstrate the capabilities is discussed in Section 3.4.1.4. These efforts can benefit the TinyOS community since they are previously

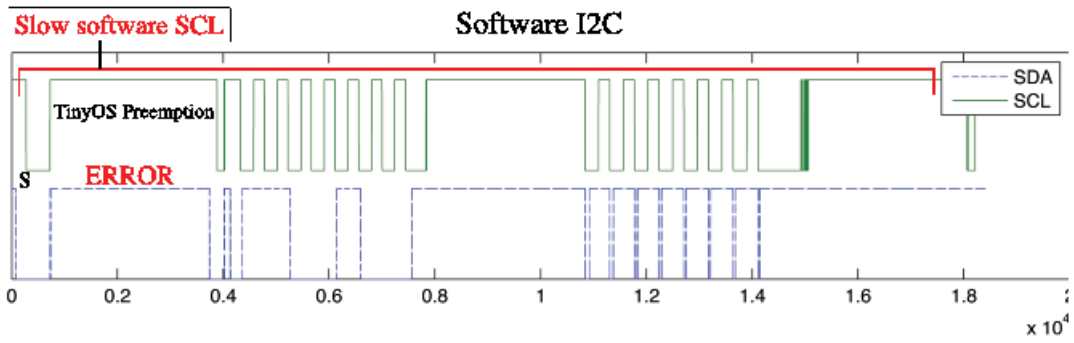


Figure 3.11: Software I^2C Protocol

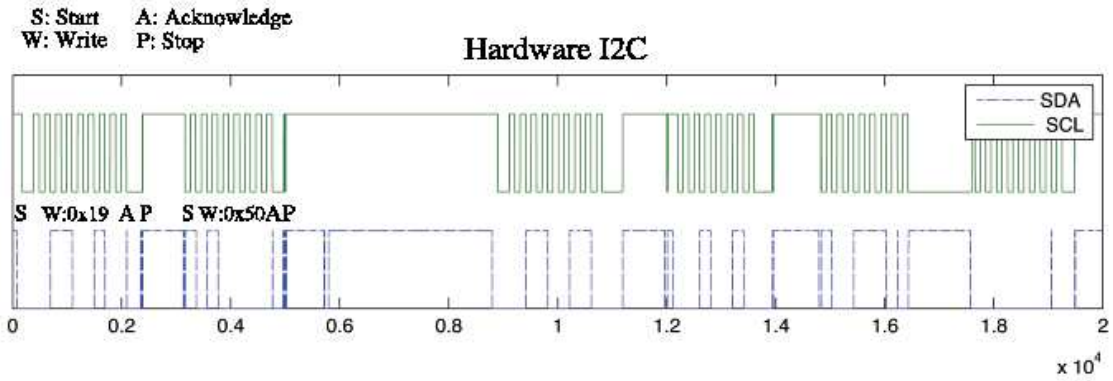


Figure 3.12: Hardware I^2C Protocol

unavailable.

3.4.1.1 Interrupt-based I^2C Protocol

The software implementation, provided by TinyOS, was intended for the compass driver. However it did not function properly. This erroneous behavior was obvious when utilizing a logic-analyzer, where the Serial Clock Line (SCL) was observed being incorrect as illustrated in Figure 3.11. The correct timing was necessary to ensure data is correctly

received on the Serial Data Line (SDA). An interrupt version was made using references to the Atmel chip design notes [27] and the wire library provided by [13]. To explore the problem in more detail, experiments are carried out. Figures 3.11 and 3.12 illustrate the output of the logic analyzer depicting the same higher-level code executed.

From these experiments, the comparison between the TinyOS bit-banged driver (software I^2C) and the interrupt-driven I^2C driver (hardware I^2C) can be evaluated. The same high-level code was executed for both examples so the IO behavior could be predicted. To exploit the timing issues, activity was captured for each test over a period of 180ms. The hardware I^2C depicted in Figure 3.12, follows the correct progress of instructing the compass into run mode. As a comparison, there are a three key observations to note: First, the timing delay for the software implementation in Figure 3.11 is 60% greater than that in Fig. 3.12. Moreover, the hardware I^2C performed the write command in a 0.08 milliseconds, when compared to the software I^2C attempting to perform the same command but actually fails doing so. The second observation, at the 2000th sample of Figure 3.11, TinyOS preempts the software I^2C driver and the master overlooks the acknowledgement. This delay in transmission corrupts the master device's ability to write the slave address. Consequently, the compass does not accept the next command.

While a software based implementation makes the driver more platform independent, it was necessary to modify the driver to correct erroneous timing between the compass and the micro-controller. The timing errors occurred as a result of pre-emption from other components of the operating system. With an interrupt based implementation, the pre-emption is no longer a problem, since the hardware interrupts take precedence over the regular program.

As a result of the hardware implementation, all of the apparent timing issues in the bit-banged implementation are eliminated. Thus, producing a more reliable and effective driver I^2C, the driver is intended to be released to the TinyOS community. The same

driver has also been utilized for debugging purposes when the UART is busy performing alternative tasks.

3.4.1.2 Compass Driver

The compass driver provides a well defined entry point to control and access compass sensor readings. In addition, the driver hides implementation details from components that utilize the driver.

The compass interface generalizes configuration, calibration, activation, and reading sensors from the compass hardware. The driver also utilizes a HMC6343 component that encompasses hardware specific communication details, that differ from a general compass device. By featuring this design methodology, future compass modules can be added without modification of higher level components. Therefore, programs that use the compass driver will not be required to change implementation details.

The novel TinyOS I^2C driver discussed in Section 3.4.1.1 is utilized for communication with the compass hardware. The compass driver is broken into two modules: the HMC6343 module and CompassDriver module. The HMC6343 module implements a command protocol specific to the compass hardware. The command protocol instructs the device to perform operations based on the payload bytes of the I^2C protocol. Throughout this operation, the master device (MCU) sends a 7-bit address (HMC6343 Compass), a 1-bit read/write, and waits for a 1-bit acknowledgement from the slave device. After the acknowledgment bit is clocked, the next three bytes clocked over are composed of the command to be executed, and argument bytes. When commands expect response bytes, such as those returned from a heading request, the data is clocked in the context of the previously sent command byte [55]. To guarantee communication success, the I^2C acknowledgement feature is utilized during communication.

The CompassDriver module implements general functionality of the compass. This

functionality incorporates reading acceleration and rotation information. In addition, the module provides a calibration function, that is used to calibrate the compass for hard-iron offsets. An event-based methodology is employed by this module so it does not block the application from performing alternative tasks. In addition to the compass behavior, the temperature sensor in the compass can be interfaced through this driver module.

3.4.1.3 GM862-GPS Driver

The dual-feature (GM862-GPS) GSM module provides both GSM and GPS functionalities is utilized by the prototype. The dual purpose device was thought to reduce cost and component count by not requiring an additional GPS unit for determining location information. In the prototype, the Iris communicates with the GM862 device through UART. In the driver software, the Hayes command protocol is utilized by the GM862-GPS for configuration, and text-messaging operations. This is described in detail in Appendix B.1.In the driver module, the Hayes command protocol is lexically analyzed and parsed for commanding and querying the GSM unit. In addition, the GSM driver abstracts initialization, text-messaging, and sensors to a higher level. The sensor available to other include GPS location information, communication metrics and the available cellular base-stations. Since each of the operations require time to execute, the event-based methodology is used by the driver to free-up the MCU during long response times such as the time required to send a text-messages.

3.4.1.4 Proof-of-Concept Application

The initial prototype is purposed to demonstrate the multi-modal capabilities of the new prototype. In this application, the sensors are used to identify behaviors and then respond based on the observation. For example, it is known that the crane harnesses

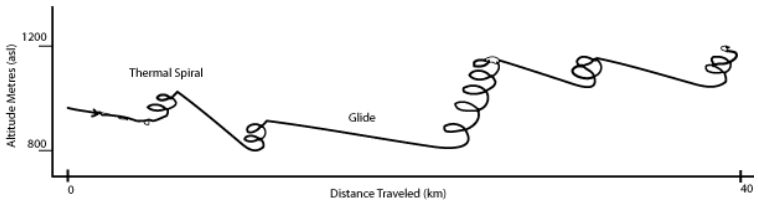


Figure 3.13: Spiral and glide behavior of the cranes during flight.

thermal updrafts, to "spiral and glide", exerting less effort during the migration as illustrated in Figure 3.13. At the prototyping stage of development, it was thought this behavior could be exploited to identify opportunities to use high-power sensors. It is thought during the spiral behavior, the tracker would experience minimal interference while attempting to acquire a GPS position. It was also assumed that the acquisition was likely to occur quicker than in any other environmental situation, thus minimizing energy exhausted by reducing the time that the GPS is active. In addition to the behavior identification, the prototype demonstrated utilizing each communication medium for a specific purpose. In the demonstration, the prototype uses short-range communication capabilities to transmit real-time orientation data to a visualization component. Following the circle detection, the prototype would send a SMS message to a Twitter account to display the position to the viewers. The visualization and additional details are discussed further in the visualization discussion of Chapter 4. This application is used to display the trackers capabilities during tours of the Cyber-Physical Networking laboratory.

3.4.2 Release Software

To successfully carry out the ecologists' missions and fully realize the capabilities of the multi-modal platform, necessary software developments are carried out. The software developed during the prototype phase are utilized in the final device release software

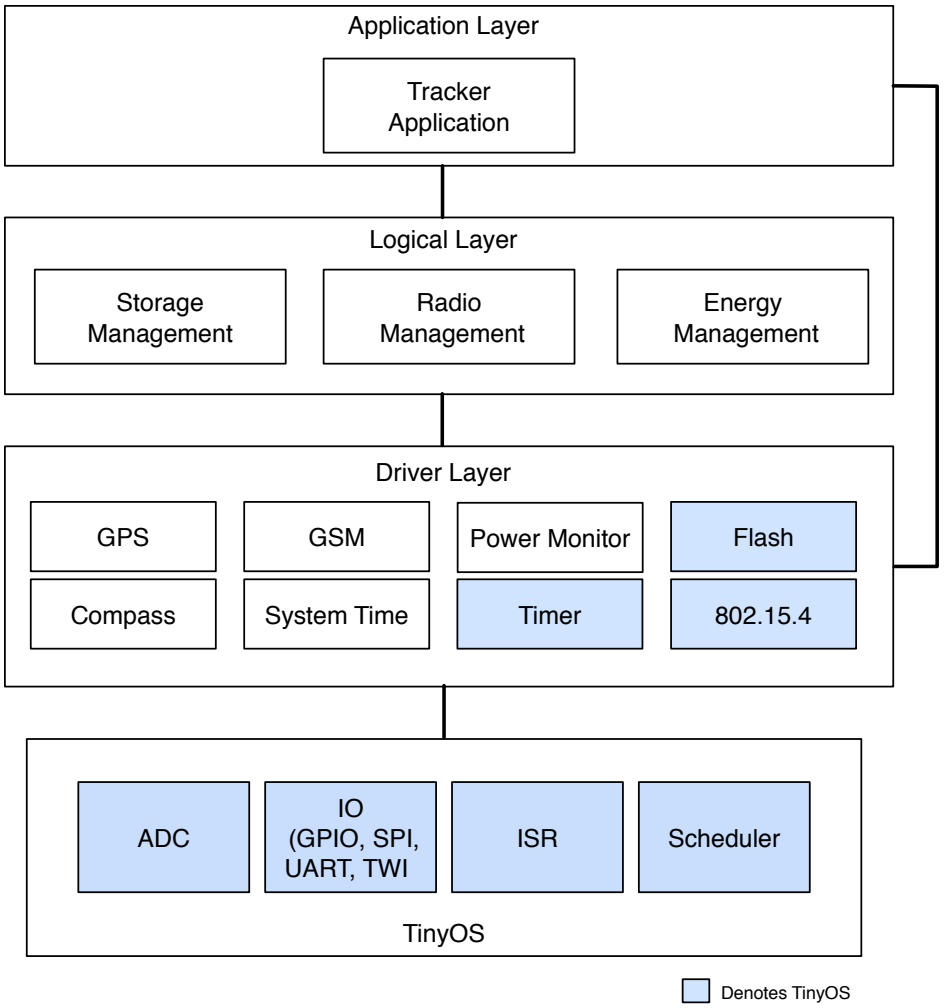


Figure 3.14: The embedded software architecture.

development. For the release software, a layered architecture is developed to abstract as many hardware details away from future developers, as possible. The layers of the platform as depicted in Figure 3.14, are broken up into the driver layer, logical layer (component management layer), and the application layer. The driver layer intimately interfaces hardware components. The logical layer provides logical component interfaces to store data and communicate. Lastly, the application layer utilizes the component management layer and sensor drivers from the driver layer to perform desired tasks during deployment.

3.4.2.1 Device Drivers

The previously developed drivers from both the off-the-shelf approach and prototype are extended to create a deployable platform with-out taking additional development time before deployment. The drivers that are reused include the extended MTS-420 driver, GM862-GPS driver, interrupt-based I^2C driver and compass driver. In addition, a new power-monitoring driver is developed to enable energy-aware management of the system. Finally, a driver to manage system time is added to TinyOS.

GPS Driver: The GPS driver that was extended by Dave for the off-the-shelf approach is utilized in the final system. This was necessary because protocol used in the GM862-GPS driver was incompatible with the new receiver. The new GPS receiver emitted receiver data utilizing the NMEA0183 protocol. Unfortunately the GM862-GPS position code was useless in this version of the platform, since it relied on the AT command set extensions supported by Telit.

The driver underwent several modifications. First, the ability to synchronize the system time from GPS time was added. In addition, the driver is modified to also feature device power-management. This is accomplished by interfacing the switch IC connected to the GPS device's power supply. Finally, the GPS driver is altered to behave in an event driven manner. In this fashion, the position data is signaled after it is acquired by the GPS. The event driven behavior enables the higher-level components to perform alternate tasks while the GPS is attempting to acquire fixes. Based on the requirements of the ecologists, the GPS driver only supports Global Positioning System Fixed Data (GGA) and Recommended Minimum Specific GNSS Data (RMC) sentences. From these sentences, the system is able to acquire latitude, longitude, altitude, speed, course, time, and date.

GSM Driver: The purpose of the GSM driver is to enable the higher-level compo-

nents to utilize the GSM module. The prototype GSM driver was developed to support the needs of proof-of-concept application and assist with illustrating usefulness to interested parties. Although successful, the prototype lacked necessary functionality to be deployed a production system. The driver developed for the prototype is still reused, but is extended to support additional features. The extensions include improved error-handling, fault-tolerance, power-management, parser improvements, and cellular metric collection.

It is important to handle erroneous behavior since the GSM consumes the highest amount of energy, out of all the components, when activated. Any failures from the driver can lead to degraded performance and even complete system failure. To improve error-handling and fault-tolerance, enumerated values are defined to represent error codes returned by the GSM device. The codes are used to handle failed SMS transmissions and cellular module shutdown due to registration errors. Furthermore, the driver ensures the device is disabled after a defined time-out period, utilizing a watch-dog timer.

The GSM driver is also improved by interfacing the switch IC connected to the supply of the GSM module. This switch allows energy to be conserved by eliminating GSM module leakage current even when powered off but still connected to the battery supply. When the device is powered off, but still connected to the battery, it consumes 62uA [97]. In addition, completely cutting the supply from the cellular module guarantees the device will not drain power in a situation where a faulty software behavior occurs.

The AT command parser, which was initially created for the prototype, was a quick implementation and was too bug ridden for deployment. Therefore, it was necessary for the parser to be improved and tested heavily. In modern software development, tools such as ANTLR [86] can be utilized to generate bullet-proof parser-generators. Unfortunately, the static nature of TinyOS eliminates the ability to utilize ANTLR since

the tool requires malloc and free functions of the standard C libraries. In TinyOS and the nesC language, the use of these functions is not acceptable, rendering ANTLR useless. The parser used in the prototype is ported from nesC to C, tested on a PC utilizing CUnit and XCode developer tools [12, 29], and then ported back into nesC. The porting from C to nesC method was not as demanding in implementation time when compared to time spent porting from nesC to C, because nesC is an extension to C. nesC supports all language features of C.

The GSM driver was extended to store more information about the cellular communications. Specifically, the useful data accessible from the cellular modem includes communication metrics and a cell-monitor report. The communication metrics are used to assist with understanding the channel quality of the environment after deployment. In addition, the cell-monitor report is utilized by the back-end to add another method of localization.

Compass Driver: The existing compass driver from the prototype was improved to support the new power switches. The switches eliminate energy consumption from the compass when the device is not in use. the compass takes a negligible amount of time (500ms) to start-up [55]. Therefore, there are no foreseen advantages to completely powering off the compass device, contrary to instructing the component to go into a deep-sleep mode.

Power Monitor Driver: The crane tracker is required to survive multi-year deployments. To accomplish this requirement, the tracker features a solar-panel that is purposed to recharge the battery. There are many unknown parameters that go into designing a system of this nature, since it has not been accomplished before. The main unknown is the amount solar energy exposure that the device would receive in a tracking setting. In addition, it is unknown how to properly utilize the multi-modal components. In order to define a model for these properties, the system must be able to monitor the

power from both the battery and the regenerative power source. Moreover, the application must be able to measure the health of the battery to maintain an operational system. To accomplish both of these tasks, the power monitor driver is developed (Depicted in Figure 3.14). The driver interfaces the ADC lines connected to the voltage output of the voltage/current measuring hardware for both the battery and solar panel. The ADC value is then converted into a standard voltage and current values with the precision of millivolts and milliamperes, respectively.

System Time Drivers: System time is invaluable and is available to developers in many operating systems such as unix [18]. However, TinyOS does not support this feature. To facilitate the platform with this functionality, TinyOS is extended to support a global system-time. A global 64-bit millisecond counter is added to the operating system to store the time in milliseconds since CPN epoch. CPN epoch is defined as January 1, 2009 00:00:00.000, the same year that the project started to take off. This epoch is used instead of the standard Unix epoch to remind the programmer that the time is in *binary* milliseconds. After converting from *binary* milliseconds, this offset can be added to the timer value to determine UTC of the system-time. The timer implementation that was modified for the Iris platform, utilizes the Timer/Counter2 because of its low-power count operation during sleep-mode. Therefore, there is no added energy penalty for adding the system-time feature. In addition, the system-time utilizes the GPS driver to synchronize the system-clock with the real-time. Synchronization is accomplished by extracting the date and time from NMEA01831 sentences. The developed system-time driver proves beneficial for determining when the system restarts, since the time is stored with the sensor readings. On a device reset, the device will initialize the system-time to zero. Until synchronization occurs, the time will increment from zero. Therefore, when a record is received with a sensed time less than a prior record, it can be assumed a reset occurs since the system only resets on initial start-up.

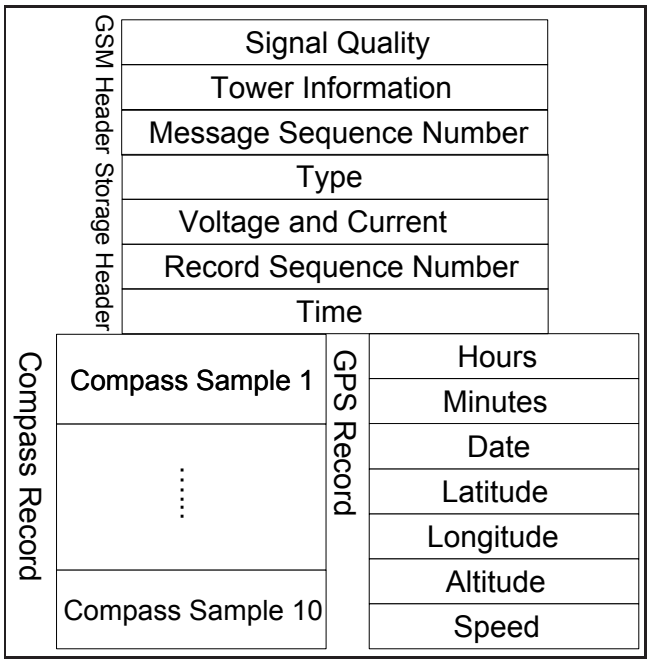


Figure 3.15: Flash storage format and SMS packet format.

3.4.2.2 Storage Management

The flash memory on the Iris is used to persist sensor readings before being transmitted to the back-end. The flash memory on the Iris is only capable of storing 512 kB of information. As illustrated in Figure 3.15, each reading is divided into GPS and Compass records. The records include a header followed by three different record types of the same size. The type of records stored include compass, solar, and GPS records. The header specifies the type of record stored and record number. In addition, the voltage of the battery and solar panel, electrical current of the solar panel, and system time of the when record was taken is stored.

The CraneStorage (Storage Management) component is responsible for writing and reading records to and from flash memory. The flash memory is managed as a FIFO queue in the CraneStorage module. The queue can contain up to 16,516 records. On the event that the flash is unable to store any more records, the oldest records are over-

written with new data. Managing data in this manner enables the most recent data to be collected by the ecologists.

The ecologists are interested in what occurs during the migration, breeding and wintering periods. Due to the unpredictability and immobility observed at locations where the breeding and wintering periods occur, there were concerns about retaining data from the observational periods were connectivity may become unlikely. To compensate for this, the data size (31 bytes x 10 data types) and collection period (4 hours and 5 minutes) is designed to observe a duration significantly greater than the breeding or wintering periods. This enables the tracker to observe 275 days before data becomes overwritten. It assumed that the bird will be mobile and capable of offloading data before this duration is breached.

3.4.2.3 Communication Management

The device is capable of communicating by means of the 802.15.4 radio and the GSM cellular module. Facilities to transmit flash records by either of these mediums is performed by the Radio Management (Figure 3.14). The management component is made up of the CraneRadio and GSMManager modules. The CraneRadio and GsmManager modules utilize the Storage Management components to read from flash storage and transmit stored information over a respective medium.

The CraneRadio manages communication with a base-station node through the 802.15.4 radio. Once the CraneRadio is activated, it will first listen for a beacon from the base-station. If the beacon is not received before a time-out, the communication attempt will complete and no records will be transmitted. On the event that the base-station beacon is received, the module will transmit stored readings until there are no more available records available or a time-out occurs.

The GsmManager manages communication by utilizing the GSM cellular module.

Once the GsmManager starts, the GsmManager checks the voltage to ensure the GSM can function, since the GSM requires at least 3.4V to operate. If there is sufficient voltage, the GSM attempts to associate with the cellular service. Once associated, the GsmManager collects the signal quality and base-station information. These are used to determine transmission quality and location from where the SMS was transmitted from. After the communication metrics are collected, the GSMManager fetches 2 records from flash, encodes them into a 7-bit format, and then transmits them over SMS. This is repeated until all records in flash have been transmitted or a time-out occurs. The 7-bit format is necessary so the AT command control bytes are not unintentionally sent over the UART line. Each SMS record contains a header composed of the network name, base-station identity code, local area code, absolute radio-frequency channel number, cell identifier, received signal strength indication, and bit error rate as shown in Figure 3.15. In addition, the SMS contains 5 generic records consisting of compass, GPS or solar sensor readings.

3.4.2.4 Energy Management

Energy saving techniques are applied to manage sensors. This functionality is necessary to stop faulty behavior, since this can lead to complete energy depletion. Each management component utilizes the driver switching mechanism, voltage checking, and time-out behavior. Each sensor is equipped with IC switches to enable supply cut-off when not in use. The drivers exploit this feature by activating the GPIO lines that interface the switches in the start and stop procedures of the driver. These start and stop functions are part of the driver interfaces. The management components utilize this functionality to save energy. In addition, voltage checking is used to protect the system from incorrect usage of components. Voltage checking is the process of evaluating if the battery voltage is high enough to use a component. Most of the components require 3.3V to operate. However, the GSM requires at least 3.4V to function. In this case, voltage-checking in

the GsmManager stops higher levels from using the GSM if the voltage is below 3.4 volts. Time-out behavior ensures devices are not over used and guarantee that devices are shut-off after a set time. As an example, the GsmManager limits the association time. This limitation on the GSM association time saves energy by disabling the GSM after it is unlikely that the GSM will be able to associate. This behavior ensures the GSM does not unintentionally deplete the system's energy resources.

Even with energy management in software, unforeseen software errors can lead to complete energy depletion. For example, timers are used to disable high energy consuming devices. If these timers ever fail to fire, the device will remain enabled and deplete available energy resources. Therefore, the platform features additional energy saving components in hardware. This separation of software from complete system control, enables the system to recover from these unexpected software errors. The benefits seen with this approach are manifested through the deployments, where it is shown the system recovering from an unknown errors, in Chapter 6.

3.4.2.5 Tracker Application

The tracker application (Figure 3.14), called the CraneManager, is a top-level component accountable for manipulation of the logical components. The CraneManager determines when the sensors, storage, and communication components should be utilized. The manager is power-aware and uses the power monitor component to determine if there are enough resources available before using any sensor or communication device. On the event that it is determined that there are no resources available, the CraneManager forces the device back to sleep. This feature will increase chances of operation in the future after the battery is able to recharge.

The tracker application software is developed as a time-triggered architecture [65], where static operations are defined and driven by a hardware clock. Initially, when

power is applied to the device the application configures itself, checks the system power level, attempts to use sensors, establish communication, and then transitions into a sleep state. The rate at which the device sleeps, uses sensors and communicates is configurable. However, in the application used in the tracking applications, a 4 hour and 5 minute cycle is used. The 5 minute time skew was added so sensor readings can be observed from different parts of the day, since the initial deployment acts as a base-line for future deployments. In cases where power is insufficient for communication, the device sleeps for approximately 20 minutes and then re-evaluates the battery voltage level. The evaluation of the battery voltage level consumes negligible energy, therefore waking up every 20 minutes is not a concern. The result of deploying this application on numerous species is discussed in more detail in Chapter 6.

3.4.3 Summary of Device Software

In table 3.2, a summary of the device software components are displayed. This table shows the layer that each software component is associated with and the function of the component. Furthermore, the lines of code for each component are provided to show a quantitative comparison between each component. This table shows that the GSM Driver is the largest implementation. In addition, this is considered the most complex since it handles communication, parsing and error handling of the GSM device. Alternatively, the Debug I^2C component is the least complex since it simply provides an instrumentation stream and utilizes the I^2C driver for communication.

Component	Layer	Function	Lines of Code
I^2C Driver	Driver	Hardware implementation of I^2C that is needed for interfacing compass	492
Debug I^2C	Driver	Provides instrumentation over I^2C	98
Compass Driver	Driver	Abstracts compass device functionality	675
GPS Driver	Driver	Interfaces GPS receiver over UART	1011
GSM Driver	Driver	Abstracts GSM device. Provides Hayes commands engine, configures GSM, sends SMS, and acquires position if hardware supports	2088
System Time	Driver	Provides system time for TinyOS, also provides library for manipulating time	726
Power Monitor Driver	Driver	Allows system to monitor solar panel and battery	271
Crane Storage (Storage Management)	Logic	Enables reading and writing to flash	1161
CraneRadio (Comm. Management)	Logic	Utilizes storage management to transmit data over short-range radio	631
GsmManager (Comm. Management)	Logic	Utilizes storage management to transmit data over long-range radio	934
Tracker Application	Application	Utilizes logical layers and sensor drivers to perform tracking task	963

Table 3.2: Table comparing hardware versions.

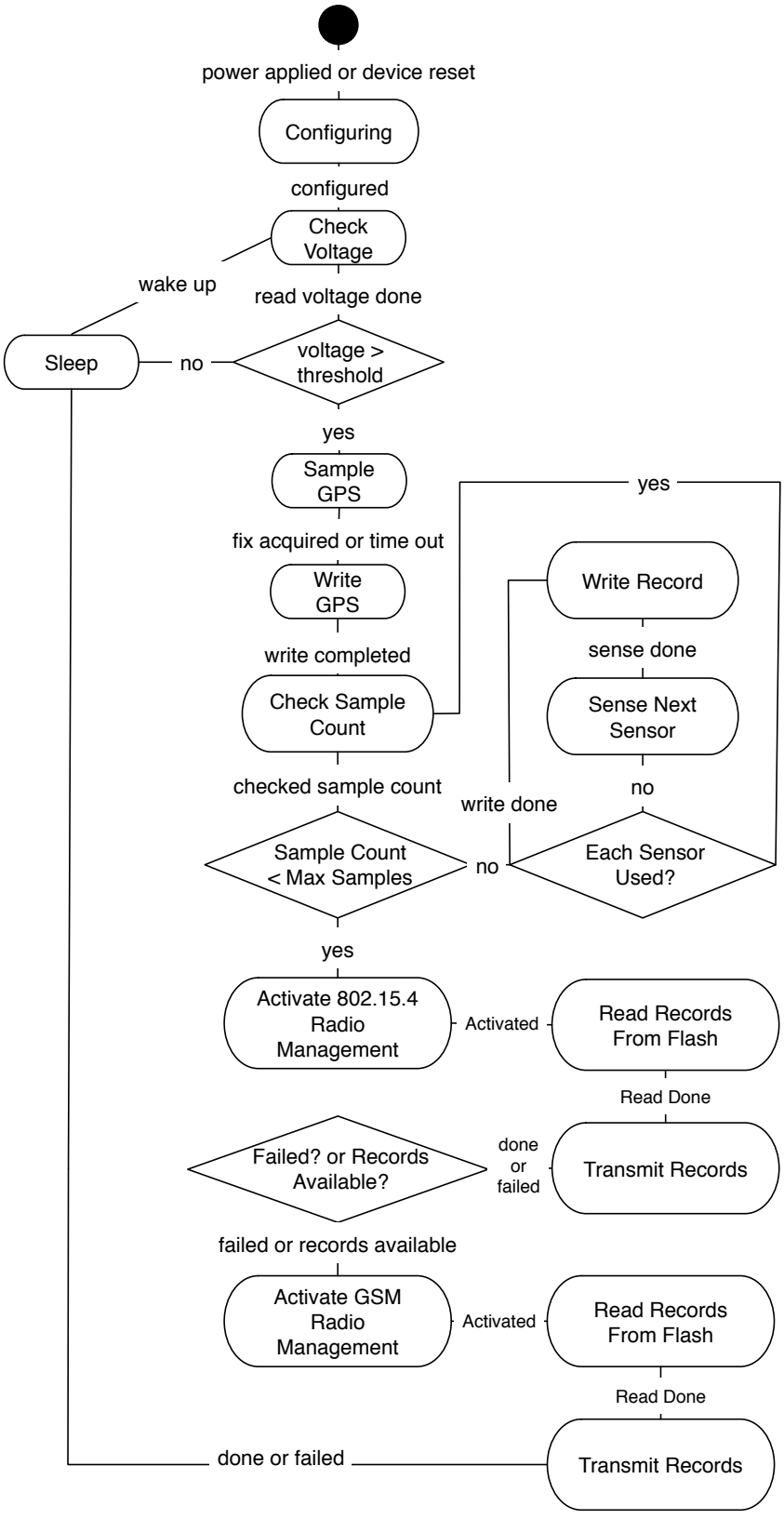


Figure 3.16: Tracker application control flow.

Chapter 4

Back-end Design and Implementation

As exhibited in Chapter 3, the CraneTracker is a highly capable platform. However, the platform is only as beneficial to the ecologists, as the back-end is capable of network management and making data available. The back-end can be determined capable by how well it is able to be deployed, and exchange information. In addition, its performance can be determined by its ability to be re-configured and self-managed. Moreover, the back-end's usefulness can be quantified by its ability to manage large volumes of information and its ability to visualize the information. The set of back-end requirements discussed in Chapter 2 are used to facilitate these discussed attributes.

In this chapter the back-end system is discussed. The chapter is organized into two sections: In the first section (Section 4.1), the back-end is discussed. The back-end consists of a set of services implemented in a Service Oriented Architecture (SOA). These services assist with handling network entities and their respective data. In addition, the services manage SMS data and decoding of data from the network. The second section (Section 4.2), covers the visualization components made to conceptualize the network data, behaviors, and the species' locations.

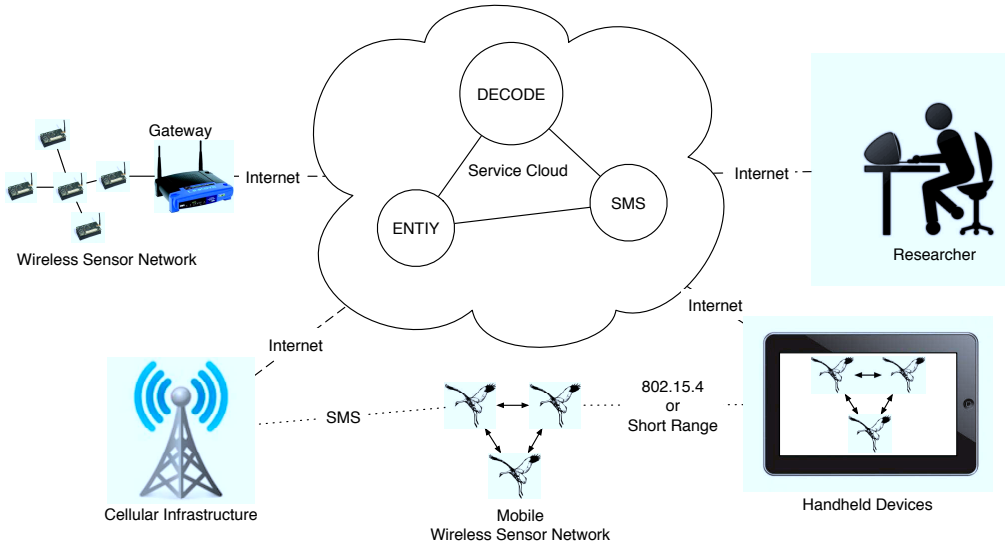


Figure 4.1: A depiction of the employed service oriented architecture.

4.1 Monitoring Birds in the Cloud: The Service Oriented Back-end

A Wireless Sensor Network is composed of many low-powered embedded wireless devices with multiple sensors and in some cases, multiple communication devices. These devices collect data in difficult conditions where humans usually are unable to exist. This aspect makes the information they collect of utmost importance. This importance is even more amplified because the devices are heavily prone to failures in communication, sensing, and life-time. Furthermore, the functional requirements of the devices differ from network to network, making the information generated highly domain-specific.

As identified in Chapter 2, the information collected by the sensor nodes range from positional to environmental information [4]. In addition, the data acquired is usually observed at very high frequencies by hundreds and even thousands of nodes [4]. Alternatively, some networks have minimal connectivity but provide critical and rare information [11]. Therefore, discarding or corrupting of information is simply unacceptable.

The back-end described in this work, is aimed to be deployable and interoperable between networks. More specifically, it is designed so it can handle a diverse set of sensor nodes, as they evolve over time. This has already been observed by this system, as it has handled the evolution of the tracking platform. In addition, the back-end works towards being extensible, so it is capable of supporting novel sensor data and different types of networks, such as underground Wireless Sensor Networks.

The back-end targets configurable and self-maintaining characteristics. This configurable behavior enables the system to handle nodes that change over time and even when they are re-purposed for alternative tasks. The self-maintaining characteristic of the network means that the back-end is capable of discovering and handling the intermittently connected nodes that exist in a mobile Wireless Network. This is a challenging process when determining if a node is out of network, or if the device has failed. The nodes in the mobile network utilize a monthly service fee, and in cases of failure, identifiers must be purposed to new nodes. It is undesirable to re-purpose identifiers if nodes are still functional.

The back-end also aims to support the large-amounts of data generated by a diverse set of networks. The non-deterministic connectivity of the mobile nodes make it difficult to predict when and where the mobile nodes will off-load data. Therefore, the system must handle instantaneous connections and the thousands of bytes generated by possibly hundreds of nodes every 4 hours.

Through the use of industry standard technologies [82, 105] and simple design methodologies, the existing solutions described in Chapter 2, can be improved with a flexible architecture and data models found in this work. The developed system facilitates a model to support every sensor network and can be deployed on the cloud. Lastly, the system's effectiveness and performance is magnified by a significant uptime of ten months at the time of writing. The developed back-end has also been deployed for use in underground

Wireless Sensor Networks [103].

The current system employs a Service Oriented Architecture (SOA) [33]. The SOA is a highly flexible architecture that enables the system to be distributed and manageable. This functionality is achieved by dividing logical aspects of the system into services that function independent of each other. Each service in this architecture provide a specification that allows domain specific computation to be hidden from outside parties. This benefits the system by being loosely-coupled and highly distributed, therefore very scalable.

The prior efforts of utilizing SOAs in Wireless Sensor Networks are discussed in Chapter 2, and will be left out of the discussion. The remaining parts of this section are organized into an overview covering the back-end components (Section 4.1.1). Then, the technologies used to develop the back-end are discussed in Section 4.1.2. Following, in Section 4.1.3, the Model-View-Controller and service controller employed by all services are described. The entity service, designed to model heterogeneous networks is described in Section 4.1.4. Subsequently, in Section 4.1.5 the service for managing SMS messages is discussed. Finally, the CPN Query Language developed for REST queries is examined in Section 4.1.7

4.1.1 Overview

The SOA implemented for this thesis is broken up into logical components. The services are loosely coupled and unassociated. Moreover, the services perform very specific functionalities. The services that exist in the system include entity service, SMS service and decoding service. The entity service is the service for managing, storing, and retrieval of data that exist in the system. The SMS service manages and interacts with the source of SMS messages. Finally, the decoding service handles the decoding of encoded data that

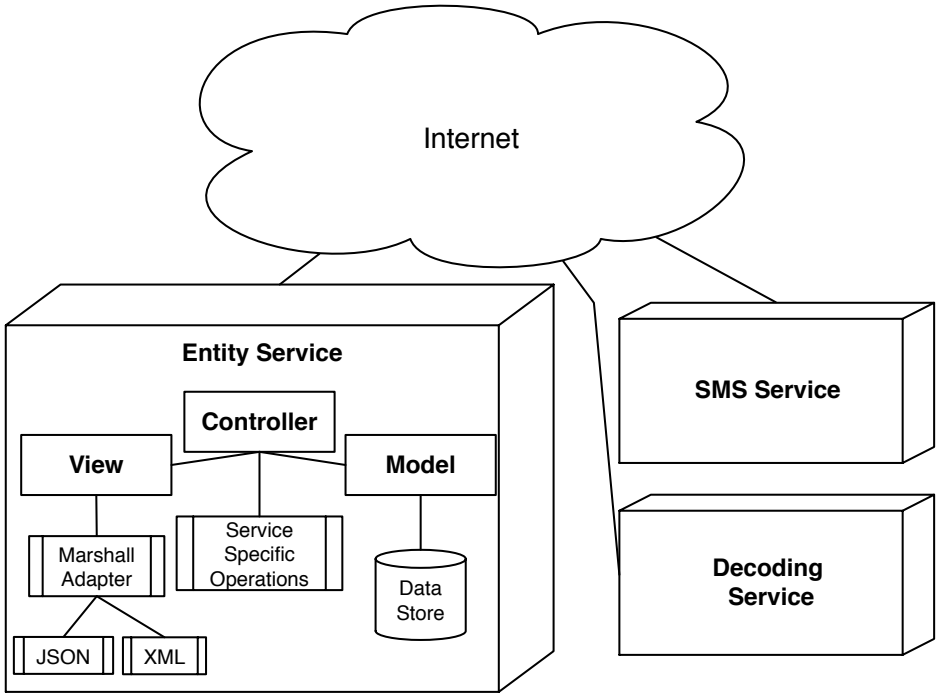


Figure 4.2: The logical parts of the back-end system.

is transmitted between networks.

Each service hides inner details and provide a schema that represents how data is consumed and produced by the service. Each service has the ability to notify other services or clients when updates have occurred in a specific service. This enables the system to be responsive and event based. This feature also results in reduced communication overhead by eliminating the polling behavior of services.

4.1.2 Play! Framework and Java Tools

The services and most of the functionality is implemented as web services and are developed using the Play! Framework. Play! is a part of the TypeSafe stack [105] and provides all the features and tools that JavaEE supports. JavaEE is an industry standard for web service development at the enterprise level. However, the stack is extremely bloated and

difficult to configure for a single person team. Play! is powerful by facilitating use of any Java or Scala library or API available to the Java language. The applications developed in Play! are test driven. Therefore, unit-tests and functional tests are part of the development process. Play! applications are deployable to Web application ARchive (WAR) [83] files and are intended to be deployed to the cloud.

4.1.3 The MVC and Service Controller

Play! applications are developed using the Model-View-Controller design pattern [48]. In this pattern, the model manages the behavior and storage of data. In the source code, the model is essentially a plain-old-java-object (POJO), that is annotated with an "Entity" attribute to denote it is handled by the Java Persistence API (JPA) [20]. This simplifies the implementation and minimizes the amount of code required to handle storage of data objects, since the developer is not required to manually transform POJOs into a relational data-store. Instead, JPA uses object relational mapping (ORM) [20], to persist object state into a relational database. This technique, in many cases, reduces the amount code written since the developer is not required to manually translate Java objects into relational data structures and vice-versa.

The view in the pattern handles the representation of the data that external parties will use or *view* the data. The services take advantage of Java Architecture for XML Binding (JAXB) [81] that automatically un-marshall and marshall annotated POJOs. It also provides schemas that are used to allow outside parties to bind to the data in a hassle and error free manner. By default, JAXB supports generation of XML and JSON. Additional adapters are created that support Matlab and CSV generation. The views of the services proposed by this system, include representation as XML, JSON, Matlab, and CSV. Finally, visualization tools transform the information into XHTML to be viewed as

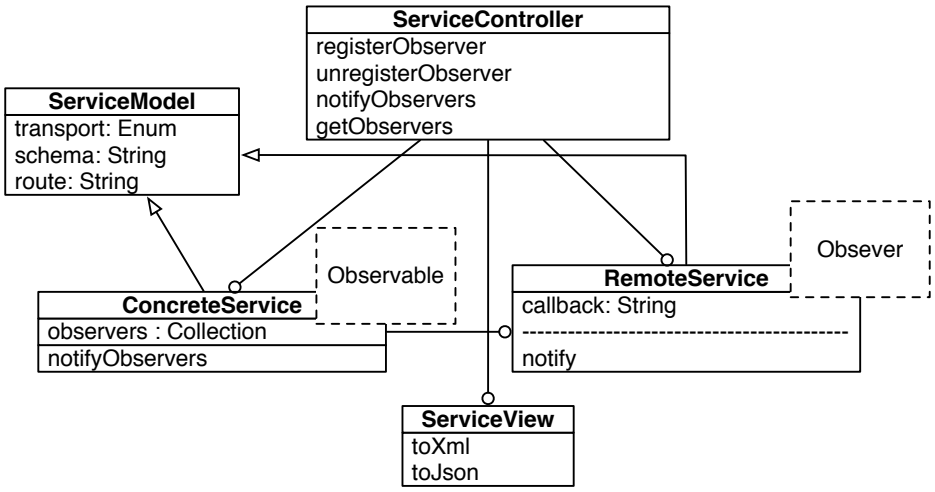


Figure 4.3: A UML representation of the generic service.

charts in a web browser. This is described in more detail in Section 4.2.

The controller is responsible for handling external requests. Each request is handled through a concept called routing [105]. Each of the *routes* maps to a static function in the controller, in which a specific task is performed. These handlers return the result of the request, that is in the form of data or status that corresponds to the outcome of the invocation. In the handlers, the controller manipulates and queries the model based on the request. If the request requires the rendering of a view, the controller instantiates and renders the appropriate view based on the model of the request. In the alternative case, the controller can perform service operations that do not return a result set.

Every service in the SOA implements the MVC illustrated in Figure 4.3. In this MVC, the service controller handles registering, unregistering, notifying and retrieving observers. This controller also works in hand with the service model, that represents services that are both observers and observable. In the general case, each service only contains concrete service models that feature multiple instances of remote service objects. The concrete and remote service models, are subclasses of the service model. The service model includes fields that denote the type of transport mode supported by the service,

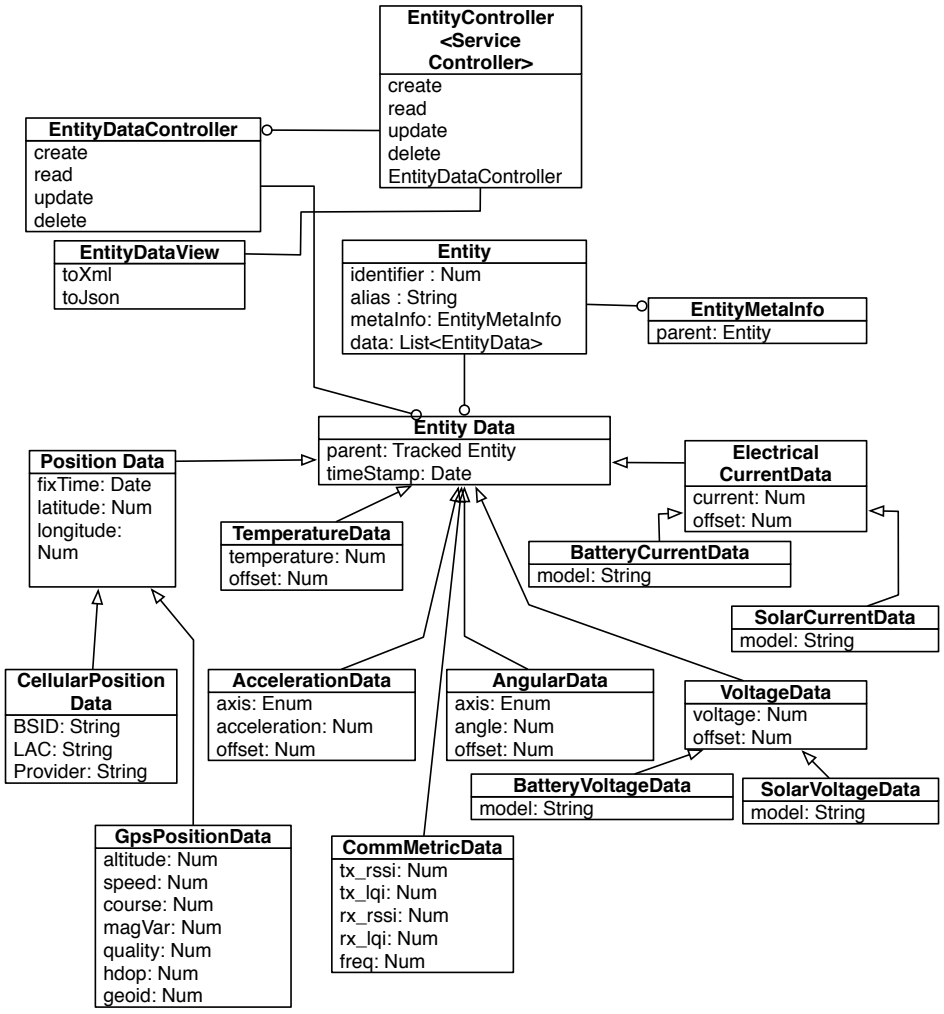


Figure 4.4: A UML representation of the entity service.

the route that which the service resides and also the schema at which data is consumed or produced. The concrete service model includes a collection of remote observers. Each remote service model includes a callback path. This path is used for invocation whenever the state of the service changes. The most heavily used service is the Entity Service, since it manages all network states and devices that exist on the network.

4.1.4 Entity Service

The entity service handles the management, storing and retrieving of information that exists in the underlying system. Again, the information that exists in the system is a product of each network submitting to it. The manner, at which the data is submitted, is handled by external parties. Some examples of this include a base-station connected to a static network, hand held devices that researchers use to submit the information, observational data that scientist observe in the field, and data that is submitted by services that the entity service is currently observing (as illustrated in Figure 4.2). In the developed system, the entity service is configured to observe the SMS service, on account that the mobile network of cranes are using SMS services to offload data. It is important to note that outside parties are responsible for correct configuration in mapping services that observe other services.

The entity service implements the MVC and the service controller described in section 4.1.3. The entity controller provides create, read, update, and delete (CRUD) of entities that are tracked. Examples of these entities include a single network node, network as a whole, mobile node, base-station on a network, hand-held devices, and even scientists that submit human observations.

There are two models used in this service: the Entity Model and EntityData Model. The Entity Model represents the physical entity that generates data (i.e Tracking Device). It has a unique identifier that discriminates it from alternative entities. In addition, it includes an alias that is human readable and understandable. The Entity Model also includes includes the EntityMetaInfo object which enables future services to add relations to the Entity object by sub-classing EntityMetaInfo object. The second model, the EntityData model, represents the data that is generated by the Entity model. The EntityData model includes a time stamp at which the data was generated in addition to

an implicit identifier. The actual data is modeled by sub-classing the EntityData into a better described object.

Currently, there are thirteen sub-classed POJOs that are general representations of the data generated by the tracking system. These include the following:

1. Position: contains the time that a position was acquired by an entity.
2. Cellular position: contains fields that identify base station location, area codde, and cellular provider at which the position was acquired.
3. GPS position: contains additional information including altitude, speed, course, magnetic variation, quality, horizontal dilution of precision, and GEOID.
4. Temperature: temperature acquired from any temperature sensor.
5. acceleration: represents acceleration data from a sensor that collects acceleration (i.e. accelerometer).
6. Angular: represents angular data such as heading, pitch, and roll. This data is provided by sensor such as a solid-state compass.
7. Voltage: represents a voltage across a electrical component.
8. Battery voltage: voltage across the battery.
9. Solar voltage: voltage across the solar panel.
10. Electrical current: represents current through a component.
11. Battery current: represents current through the battery source.
12. Solar current: represents current through the solar panel.

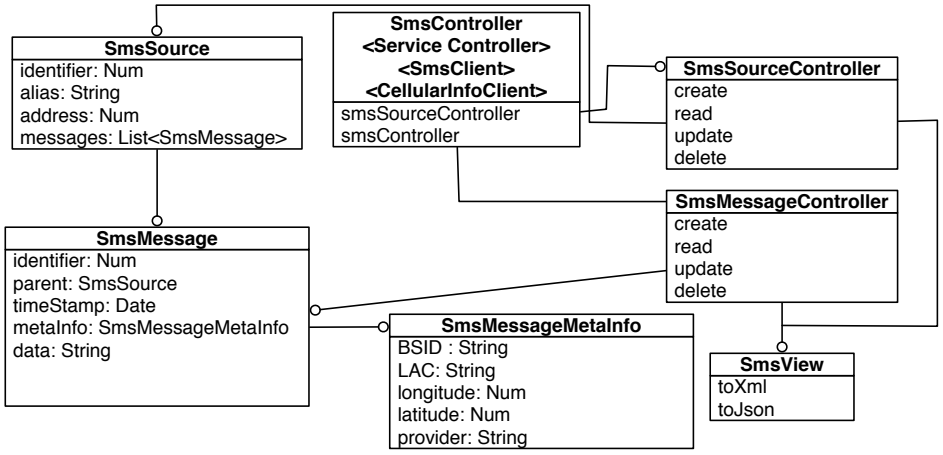


Figure 4.5: A UML representation of the SMS service.

13. Communication metric: represents performance metrics of a communication device on a tracked entity. Some examples include received signal strength indication (RSSI) and link quality indicator(LQI) from the GSM radio.

Each of these described models and additional models can be derived from an existing generalized model, or from the entity data object to be able to be persisted by the entity service.

4.1.5 SMS Service

As depicted in Figure 4.5, the SMS service implements the MVC pattern and models an SMS source and multiple SMS messages. In this relationship, the SMS source represents any device that can transmit SMS messages to the service. Such devices include network gateways, individual network nodes and even human observers.

The SMS message model represents arbitrary data in a many-to-many relationship with the SMS source. In addition, each SMS model encompasses meta information about the SMS. This meta information is extracted from each message using the decoding service described in sub-section 4.1.6. The service utilizes the Base-Station Identifier

(BSID) and Location Area Code (LAC) to determine the location where the SMS was sent from. In tracking efforts, this feature provides instantaneous localization every time data is received from the tracking device. This feature also facilitates an additional position sensing redundancy in the case of a faulty a GPS sensor, although the position tends to be not as accurate (Chapter 6),

As with all the services in the back-end, the view in this service, presents the data model in the form of XML and JSON. Each service is capable of also providing a data specification to govern the allowable data components that can be emitted and consumed by the service. If outside parties fail to meet the specification when submitting the data, unexpected behavior should be expected from the external party. These specifications can be used for taking advantage of tools such as the Java Architecture for XML Binding (XML) and web-frameworks for easy marshalling of data emitted and accepted by the services.

The SMS Controller extends the SMS Client, CelluarInfoClient, and service controller as illustrated in Figure 4.5. It also encompasses an SMS Source Controller and SMS Message Controller. As with all available services discussed in this section, the SMS controller extends the service controller to enable the service to register for events from external services. The SMS client interacts with an adapter interface that handles the physical data available from the devices. The SMS client will be discussed in further detail in the following Section. The CellularInfoClient class interacts with proprietary services to lookup base station localization data.

Finally, the SMS Service Controller handles sending arbitrary data through SMS to external parties. This is accomplished through the SMS client class. In the near future, the system desires to use this feature extensively to reconfigure the devices during runtime, after the devices have been deployed.

4.1.5.1 SMS Client

In industry, SMS data can be collected using three primary methods: SMS gateway, physical device, or free ad-supported voice services. Each method has its benefits and disadvantages.

The first method of receiving and sending data is through a proprietary SMS gateway. This method is the simplest method for interfacing with SMS devices. The companies which facilitate these services provide APIs to access the SMS facilities, permanently store each message received, and are contractually bound to providing a certain level of quality-of-service [15]. However, the per message compensation desired by the commercial entities make this method impractical for non-for-profit research. The second option for SMS data collection is to utilize physical devices to receive SMS messages and forward them to the back-end. This essentially requires a monthly service plan and many physical devices to handle the data generated by the network. In addition, the devices are required to be continuously powered and monitored. Hiring workers to monitor the system, on top of the monthly fees, could easily add up to be less cost effective than employing a SMS gateway. Furthermore, additional software to perform autonomous forwarding from the physical device to the back-end would still need to be developed and tested. The last option is to take advantage of Google's advertisement supported voice services [52]. Google provides many robust services that millions of people take advantage of every day. These services are funded by advertisements that force end-users to view during service usage. The Google voice service enables users to send and receive calls, SMS messages, and voice-mail. This service is exploited by this project to send and receive SMS messages to devices in our mobile network at absolutely no cost to the end-user.

When this project started, the Google voice service was in its infancy and at the time

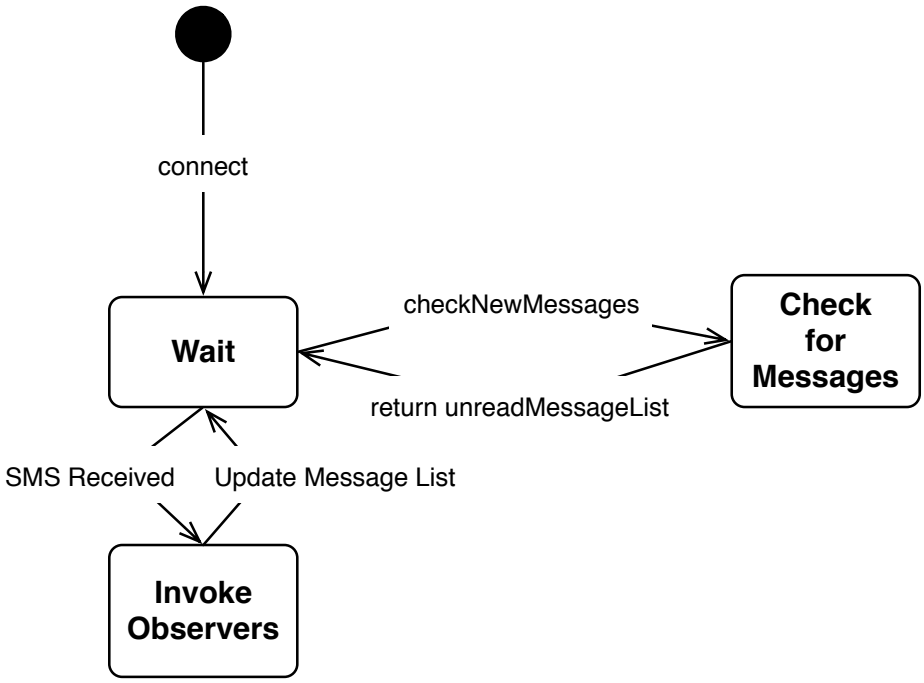


Figure 4.6: The SMS client process of handling SMS messages transmitted by the mobile network.

of writing, an API for interfacing the voice services was not provided. Instead, the voice service allowed the messages to be forwarded by email. This feature of the voice service is used by the back-end for handling SMS messages.

The SMS client manages SMS data forwarded by the Google Voice services. The interface to the actual SMS source is developed as an adapter pattern. Utilizing the adapter pattern will allow simplified changes in the future, where SMS messages may be handled by alternative means. During runtime, the client connects to the specified source, and invokes methods provided by the adapter interface. The invocation will eventually propagate to the concrete adapter. In this process as depicted in Fig 4.6, upon connection, the client can specify if it will function in an event or poll-based mode. In either case, the client can request for new messages by using the "checkForNewMessages" method. Internally, the concrete adapter will run as a scheduled job. On dispatch, the job will check for new messages. If new messages are available, the observers are notified of the

event. Alternatively, if clients are not registered for events, the messages will be cached. It is the responsibility of the non-event based clients to check for new messages. On the event that clients check for new messages, a result set of available data will be returned to the client. The client must mark the data as read, or else the returned result will consist of older data on future checks.

The SMS client also facilitates transmission of SMS messages to a designated endpoint (phone number). This will be used in the future to disseminate data to the network.

4.1.6 Decoding Service

In every deployed network, hardware devices have many platform specific types and feature different Endianness. The decoding service is designed to enable external parties to have a unified method to decode data for all platforms in the target network. In practice, the idea of having a coupled design is undesirable. However, to implement a generic specification on a constrained embedded device, would increase overhead in processing and communication. Specifications such as XML have a space efficiency of $2 * N + 5$ where N is the data field size [64]. This is inadequate for most platforms in wireless sensor networks. Furthermore, non-deterministic connectivity necessitates every communication opportunity to behave with minimum overhead on the device. Beyond the connectivity issues, the platform described in this thesis utilizes SMS to transmit data over a long distance. The cellular medium, unfortunately has properties of small message length and slow data rates. In addition, it consumes the most energy out of all components on the platform. Therefore, sending additional data to follow a general specification, such as XML, is unacceptable due to the limited energy resources of the platform. To counter these issues, the decoding service is required to account for different versions of the software and platforms. Accordingly, the platform specific

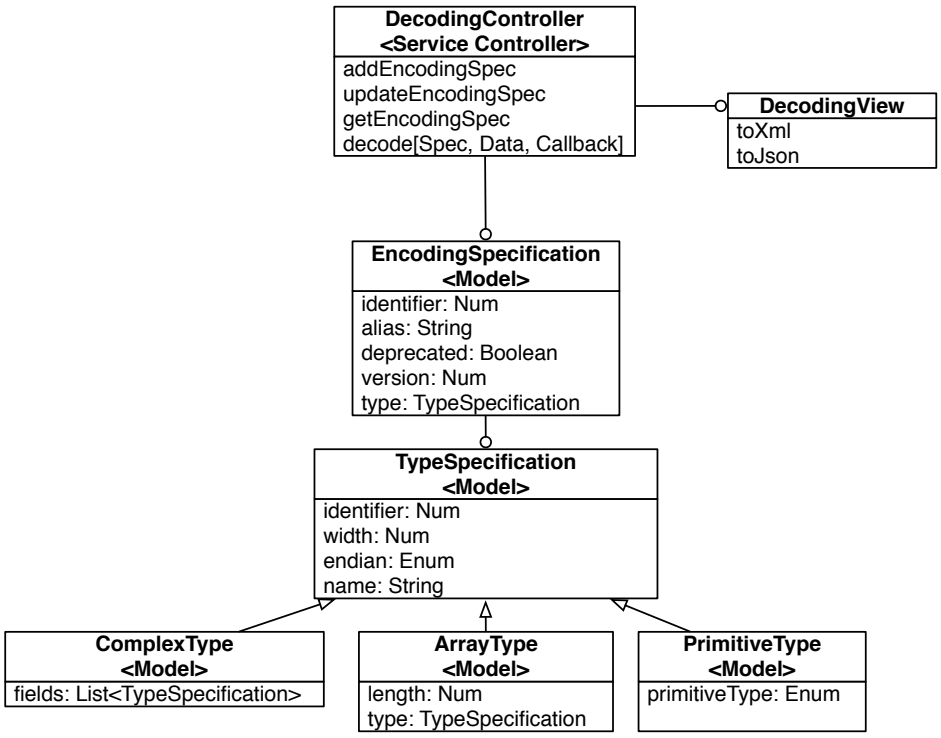


Figure 4.7: A UML representation of the decoding service.

encoded data can be unpacked in a reliable manner.

The decoding service implements a MVC, as depicted in Figure 4.7. The model of the decoding service is the specification of the data that is to be encoded and decoded. Users of this service must add a specification or conform to pre-existing ones. Each specification is specific to the hardware platform, in addition to the software version running on the platform. Each platform can have different versions of the data types transmitted by the nodes.

The encoding specification model is composed of an identifier, alias, deprecated flag, version, and type. The identifier is unique value that corresponds to a specific encoding. The alias is a string based representation of the specification. The version field denotes the version of the encoding specification, since the specification can evolve over time. The type is TypeSpecification model object that describes the type in the specification.

The type specification model has a unique identifier, data width, endian, and name. The width represents the size in bytes of the type specification. The endian field describes the endian of the platform, so users are able to verify the byte order of the data from the device. The name is a human readable description of the type. A good example is the GPS position type. The type specification model is extended by three additional types: ComplexType, ArrayType, and PrimitiveType. The complex type is a type that has many fields that model a specific entity. Such as, the GPS position. The GPS position type includes fields such as latitude, longitude, and altitude. The array type describes a collection of types. This model includes a count of how many reside in the array and what type the collections are. Finally, the primitive type is an enumeration of primitive values that can be used to construct a complex type. A good example is the GPS type's latitude field. The primitive for this field is a double precision type. This data model enables modeling any data-type being described, independent of the platform.

The view of the decoding service is the translation of the encoding specification to the standardized specifications, such as XML and JSON. This is acceptable to have in the back-end where the constraints don't exist as they would on a mote.

The controller of this service, as depicted in Figure 4.7, extends the service for long decoding jobs. External parties can register with a callback endpoint for a remote procedure call after the decoding is complete. In cases, where an endpoint does not exist, the callback can be left null to indicate that the result will be returned during the same request. A caveat of this service is the requirement to have the decoding operation implemented by the service. It is desirable to add in the future a method to submit a JAR file with black box decoding logic that follows an interface specification. This would eliminate the need for the service to be extended for custom compression and custom decoding methods.

As depicted in Figure 4.8, the decoding process works as follows. First, on application

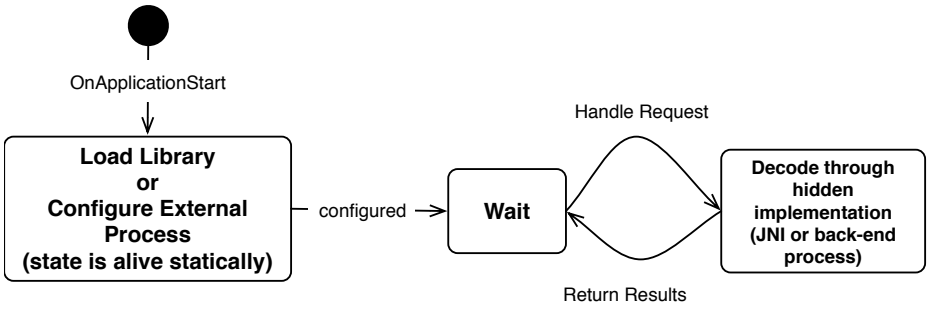


Figure 4.8: The hidden process of decoding firmware specific information.

start, the service initializes the library or external process that is active the life of the web service. On the event of a decoding request, the web service invokes the hidden process to handle the decoding. This can be handled by a native decoder invoking a hidden process or calling a native code through Java Native Interface (JNI). Once the decoding completes, the results are returned to the callee. In JavaEE containers it is common not to allow JNI (for security reasons). In this case, the decoding process must be handled in the service or by invoking a process outside the JavaEE container. This situation manifested in the final system, and a external C++ decoding process is invoked by the web service upon a decoding request. The C++ program was employed so the binary encoding can be easily decoded and handle unsigned types with no effort, a program language feature that is not available to Java.

4.1.7 RESTful Web Services and CPN Query Language

Representational state transfer (REST) [43] is an elegant architecture for a distributed system such as that presented in this thesis. Essentially, all the services act as a collection of resources that utilize a uniform resource identifier (URI) to describe the data that is serviced. The RESTful services have a stateless characteristic which is perfect for a decoupled system. In addition, an interaction with each service utilizes HTTP methods to POST (create), GET (retrieve), PUT(update) and DELETE. Finally, MIME types are

used to inform the user how the data will be returned. An example URI to access a resource is the following:

http://.../service/<service name>/<model identifier>/<model field name>/<type>/<query>

http://.../service/entity/1/data/gpsposition/all

From this URI it is obvious what service, model, and query will be returned to the user. The service is described by the ”service name” node of the URI. The ”model identifier” node is a query for a specific model. In the above example, the entity model with an identifier of 1 is desired. The URI can be further expanded into the fields of the Java class (i.e data of the Entity Model), where a specific type can be to be retrieved (GPS Position) from the collection of data in the entity model. The final node can then be queried further to limit the results, if the type is a collection. The standard URI query constructs such as, “all”, “id”, and ”$('?'(< param >=< id >)+)$” are supported. However, it is difficult to describe expressive queries using these simple constructs provided by the URI specification. There is a need to have more descriptive URI query system to filter unnecessary data returned in the result set of a given end point. For example, it is desired to support filtering invalid GPS data and ordering of the objects. By constraining fields of GPS data objects this can be easily achieved.

To satisfy this need, the CPN Query Language is created. This feature can be invoked by requesting the resource URI with a <query> of “?=query”, where the query is a string containing the desired query. The language is designed to be embedded into the URL for the requests. It is not meant to describe collections of complex types, since this can be described easily with hierarchical component of the URI. Instead, it is solely intended to add complex constraints based on the primitive fields of the object at a node in the URI. The Extended Backus-Naur Form (EBNF)/ANTLR description of the language is available in the appendix C.1.

Internally, the grammar is implemented using the ANTLR parser generator [86]. ANTLR is a parser generator that employs LL(*) parsing. The input to ANTLR is a grammar provided by a user. The output of ANTLR is a generated recognizer composed of a lexer and parser. During recognition, ANTLR will throw errors if syntax is incorrect. While parsing, the recognizer will nest actions in the grammar placed by the user. Every controller in each service has access to the CPN Query recognizer, and can only be utilized for GET based requests. This eliminates the concerns of deletion or modification of complex object graphs, that may have unintentional consequences.

Currently, Java Persistence Query Language (JPQL) is the only target output of the language generator. In the end of the query building process, the generated string is passed to the entity manager, and then a result set is returned. There is motivation to generate different target outputs and improve invalid query checking capabilities. It is not the most efficient method to generate a JPQL string, since the JPQL engine will be required to re-parse the string for query execution. Finally, there is a desire to improve type checking since it is solely handled by the reflection library included with Java, which is left for future work.

4.1.8 Integration with Visualization and External Components

As depicted in Figure 4.9, the services developed are utilized by the visualization components. In addition, the services are used externally for data analysis. In this integration, clients connect to the web services and invoke desired endpoints. For example, in Section 4.2.2 the behavior detection demonstration visualization uses web services to check the state of the tracking node. The state of the tracking device is updated by an additional client. This client uses short-range communication to communicate with the tracking device, where the state is forwarded and published to the service every update.

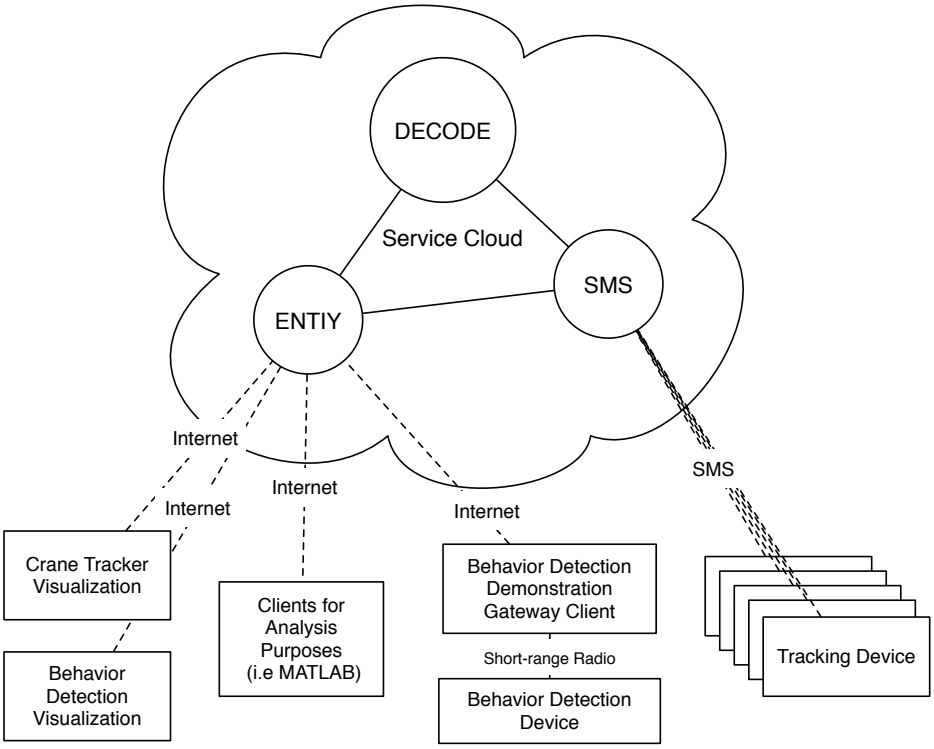


Figure 4.9: Illustration of the services interacting with external components.

For viewing network state during deployments, the Crane Tracker visualization utilizes the entity service. The visualization queries the entity service for the available trackers, their sensor data, and the location of the tracking device. The tracking devices submit SMS messages to the phone number of the SMS service. The data is then decoded and eventually propagates to the entity service so the data from the nodes is available to interested clients.

4.2 Visualization

Wireless Sensor Networks have great potential and can be utilized in any application where electronic devices can be employed to observe a physical environment [4]. In addition, these networks can be very useful for decision making and understanding

phenomena. Unfortunately, the data generated by these networks are very abundant and complex. For example, the compass on the tracker can sample up to 10Hz and contains seven different data types for each sample. To make decisions and understand phenomena, effective tools are needed for illustrating each aspect of the system.

As discussed in Chapter 2, existing tools are designed to model networks of stationary devices. However, the migratory birds tracked in this thesis are extremely mobile. In addition, the current tools, as described in Chapter 2, are incapable of displaying real-time three-dimensional behavior. However, the CraneTracker is capable of generating this type of information. To demonstrate these capabilities, it is necessary to have a visualization tool that manipulates a multi-dimensional model. In addition, the prior visualization efforts require execution on a desktop machine [85, 104]. It is desirable for the final visualization to be accessible through a web-interface so the data can be visualized on any platform.

The rest of the section will describe the visualization components created during the development efforts. First, in Section 4.2.1, the visualization component used to illustrate the multi-modal sensing and communication capabilities is described. Then, in Section 4.2.2, the visualization component capable of illustrating the behavior detection capabilities is discussed. Finally, the tracking application used to visually observe the network state is examined.

4.2.1 Compass Capability Demonstration

In previous tracking efforts, ecologists employ devices that simply recorded location [11]. While recording bird location is important, the ecologists are also interested in behavior beyond just location information. Therefore, the platform is equipped with a three-dimensional compass that can be used to capture these behaviors over time.

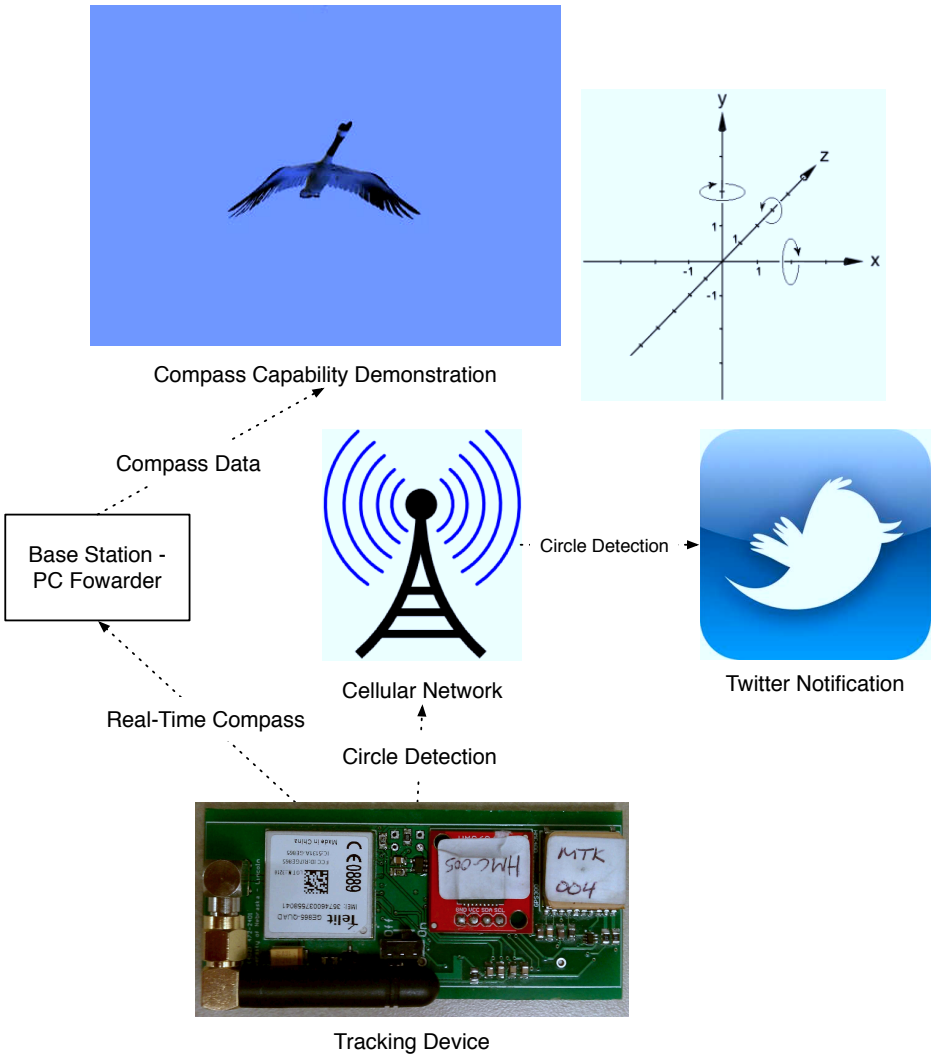


Figure 4.10: The compass capability visualization components.

Although useful, it is difficult for a human with no mathematical or physics background to understand information of the data produced by the compass sensor. To help illustrate the usefulness of the device, an interactive tool is created. This tool demonstrates utilizing the solid-state compass to detect when a circular movement is performed. Then this event is sent to a twitter account to exploit the cellular modem. The tool encompasses a three-dimensional model that orients itself in the same orientation as the tracking platform.

Figure 4.11: A screen-shot of the compass demonstration tool.

This visualization tool, depicted in Figure 4.10, is composed of three parts. The first component of the demonstration is the firmware running on the mote. The second component of the tool utilizes short-range communication and handles data forwarded from the base-station. The final component handles rendering in the visualization tool. This object mimics the orientation of the hardware platform in real-time.

The rendering component, illustrated in Figure 4.11, is a multi-threaded application, where the state of the bird model is shared across each thread. One thread is responsible for registering a serial port with the base station. The same thread handles communication events generated by the base station. Furthermore, it is responsible for parsing and translating the data into types usable by the rest of the rendering program. After data processing is complete, the thread updates the shared rotation and position vectors of the model to be use by the rendering thread.

The second thread manages graphics configuration, user input, and rendering of the model. During the configuration process, the graphics device and view port of the dis-

play are configured. In this process, the tool automatically adjusts the graphic enhancing capabilities of the system to perform at a more suitable setting, with respect to the target machine. This enables the tool to run optimally on the target desktop and make best use of the tool. After configuration, the thread runs through an infinite loop continuously executing handleInput, updateModel, and draw. In this process, the handleInput function checks and handles user input. Some controls that are accessible to the user include changing the camera position and selecting the appropriate serial port with respect to the base station connection to the computer. Next, the updateModel function notifies the skeleton of the three-dimensional model and the camera to reflect the change in orientation and position. Finally, the draw function governs the drawing operations to portray the three-dimensional state of the model and camera. In the drawing process, the function orchestrates clearing frames, updating the view matrix, computing projection of the model relative to the camera, updating the mesh of the model, and iterating through an animation over time.

4.2.2 Behavior Detection Demonstration

The motivation to utilize the crane tracker platform goes beyond position tracking. Currently, Whooping Cranes' inability to produce offspring is one of the most pressing threats to the success of the reintroduction efforts. It is believed that with the ability to observe movements besides position, ecologists can confirm speculation on root causes of inability to reproduce. Fortunately, the cranes exhibit distinct movements related to the behaviors they are engaging in. These behaviors are broken into comfort, locomotion, foraging, and social behaviors. By utilizing the solid-state compass, the platform can be used to identify these behaviors [19, 39].

During the tracking efforts, the device is assumed to be mounted on the back of the

bird with the positive z axis pointing towards the sky. In addition, it is oriented with the positive x axis pointing toward the head of the bird. For demonstration purposes, the behavior detection is accomplished using a jerk threshold-based method [1]. The jerk can be determined by taking the derivative of the acceleration values from the compass. The jerk thresholds are determined by manually simulating each behavior. The following behaviors, descriptions, and their respective thresholds are described as follows:

1. Resting Behavior (0.0 m/s^3): This movement corresponds to no movement. The resting behavior is assumed to occur when other behaviors are unable to be detected through jerk comparisons.

2. Dancing Behavior (0.3 m/s^3): This movement corresponds to the social movement exhibited by the cranes during mating. This movement is detected by computing the jerk over time in the y axis and comparing it to a value defined as a jerk that occurs while dancing.

3. Flying Behavior (0.3 m/s^3): This movement corresponds to general movement in the z axis. The computed jerk is compared against a threshold is assumed to be the jerk experienced during flight.

4. Foraging (0.3 m/s^3): This movement corresponds to a general stabbing movement in the x axis. This is determined by comparing the jerk in the x dimension against a value that constitutes as a foraging motion.

A visualization program is created to illustrate the crane tracker platform's ability to detect coarse behaviors. The visualization is composed of an HTML5 web application, a firmware application, and a serial-to-PC application for receiving the state changes. The HTML5 application utilizes WebSockets to perform real-time updates on state changes of the device. The WebSocket provides a bi-directional, full-duplex communication channel

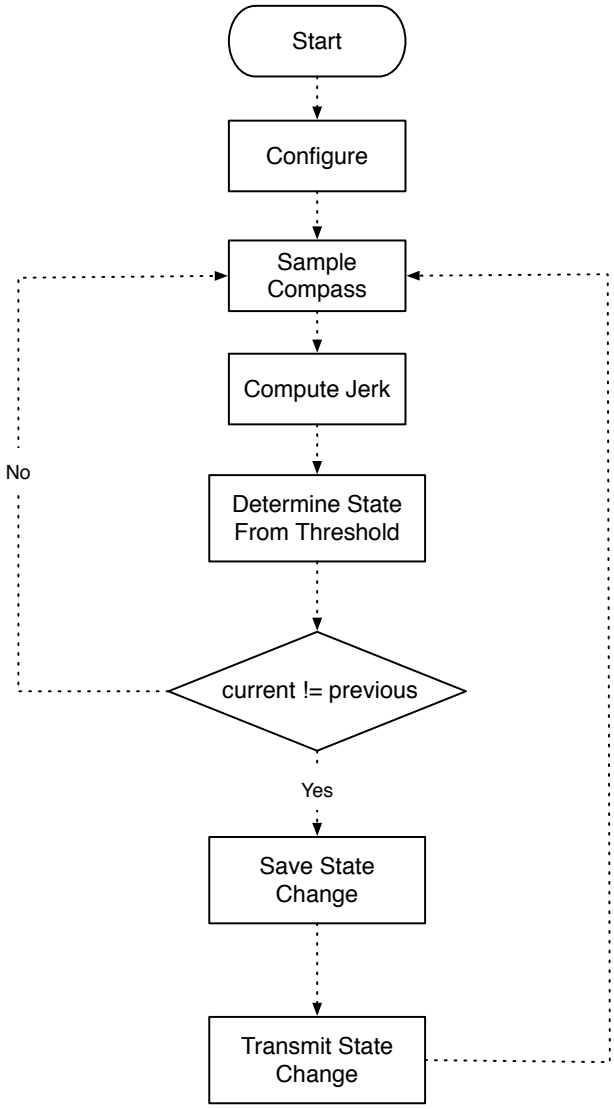


Figure 4.12: Behavior detection algorithm.

over Transfer Control Protocol (TCP). The WebSocket waits for state messages from the server. On the reception event of a state message, the application modifies the animation of "Ali the Whooper" to correspond to the behavior detected by the device. The different states detected by the demonstration can be broken up into four categories: dancing, flying, foraging, and resting.

The next component of the visualization tool is the device firmware. This component

is responsible for behavior detection and transmission of state changes to the base station. The algorithm, illustrated in Figure 4.12, performs as follows: In the first step, the device is configured. This involves specifying a infinite impulse response filter (IIF) threshold and sampling rate provided by the compass. The IIF assists with filtering noise so there are minimal invalid changes in the reading received by the platform. The second step computes the derivative of acceleration sample acquired, with respect to time. The result of this is the jerk. In the third step, the jerk is compared to the thresholds decided at compile time. As described previously in this sub-section, the thresholds employed by the algorithm, are determined by manually simulating the movements to find the proper threshold values. In the case, where the threshold is accepted by more than one variable, the axis that has the greatest absolute jerk is considered to be the present behavior. The behavior mapping to an axis is defined earlier in this section. If none of the axes surpass the threshold, the device is considered to be in the default resting state. The last step, after determining the behavior, is to save the state and to notify the user. However, the state information is only transmitted if it is different then the previous state. Hence, resulting in minimal communication.

The serial-to-PC component of the demonstration is responsible for forwarding the behavior changes of the tracking platform to the back-end of the web application. The component also maintains connectivity with the tracking device. In the case of erroneous program behavior, the component automatically notifies the web application to reset the state of the visualization. This feature improves usability of the tool.

4.2.3 Crane Tracker Visualization

Previous wireless sensor network visualizations rely on a client application to be installed on the end-user's system [104]. Moreover, many clients require the end-user to be

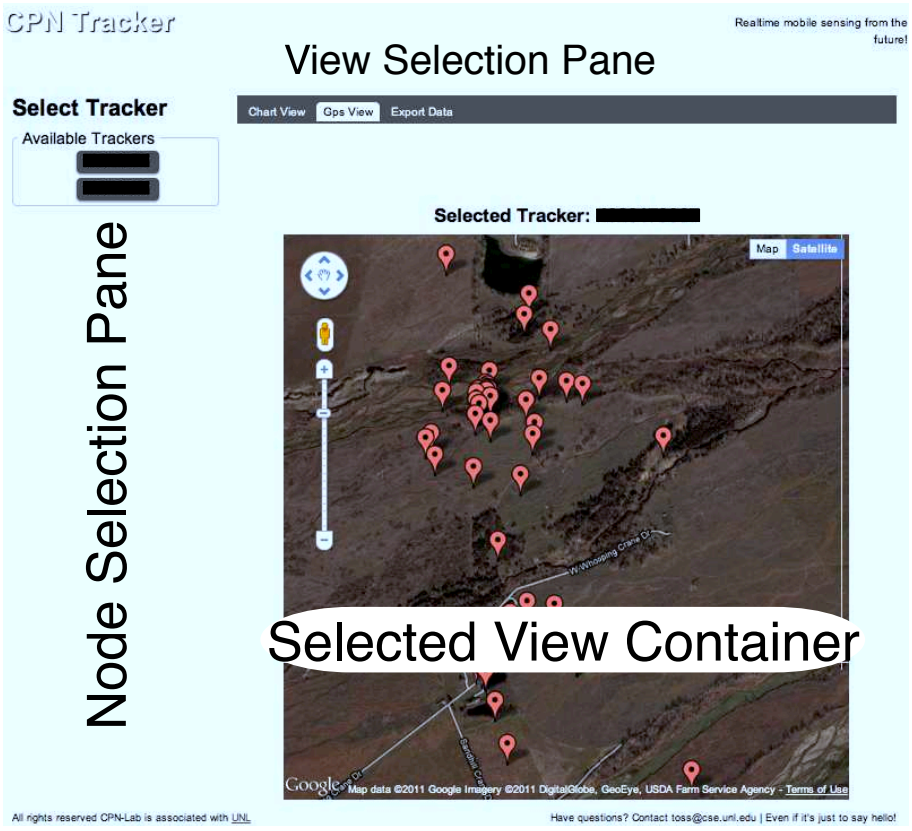


Figure 4.13: A screen shot of the Crane Tracker visualization tool.

physically connected to the base-station, either by serial port, or by a TCP socket directly connected to serial forwarded connection [85, 104, 114]. In this work, a different approach is used to present the data to the end-user. This approach utilizes a web-interface to display collected by the network. Using a web-interface allows the data to be accessible from anywhere and at any point in time from a desktop computer, mobile device, or any device with a web-browser. This design choice also eliminates cross-platform issues and reduces maintenance challengesthat are experienced with full-fledged desktop or mobile applications.

The user interface depicted in Figure 4.13, is a MVC similar to the services described in Section 4.1. However, in this MVC, the controller returns a view consisting of HTML and JavaScript representations of the data. To eliminate complete page requests, the

JavaScript in the returned view utilizes asynchronous JavaScript methods (AJAX). This eliminates full page refreshes by simply ”patching” parts of the user interface that need to be updated after user interaction.

As captured in Figure 4.13, the interface can be broken up into the following parts:

1. Available Trackers Column: Presents the list of trackers that are available on the network. This list is updated every time a new tracker joins or leaves the network. The main view reflects the displayed data based on the tracker selected from the list.

2. View Selection Panel: Enables the user to switch between the main views. To this end, only live sensor readings and the GPS view are available.

3. Main View Controller: Displays the main content of interest. The main content of interest is a charts view or a GPS view. The chart view plots the select sensor data collected over time, with respect to the chosen tracker from the available trackers column. The sensors can be chosen from a list corresponding to the available sensors on the device. The GPS view displays the position data as markers on a zoom-able map.

The Crane Tracker visualization was tested with manual unit testing methods. To simulate how the final system would behave, tracking devices and their data are generated on the back-end.This was carried out manually so the plotting and locations could be rapidly verified. The desired timeliness of the tool completion, motivated utilizing manual methods instead of automated testing tools.

It is clear that the Crane Tracker visualization is in its infancy. However, it is useful for visualizing the high-level state of the system. It is desired to extend the user interface

Component	Type	Function
Entity Service	Service	Manages network, stores network data, and allows access to network data.
SMS Service	Service	Provides access to SMS data and location of where the SMS originates.
SMS Client	Service	Client used by the SMS service to gain access to SMS messages from SMS source.
Decoding Service	Service	Provides decoding services of encoded network data (i.e encoded SMS messages).
CPN Query Language	Service	Language for information retrieval through URIs. The recognizer allows read-only, constrained information retrieval at URI end-point (i.e request only valid GPS data).
Compass Capability Demo	Visualization	Visualizes orientation of bird based on device orientation.
Compass Capability Demo Client	Visualization	Forwards short-range radio data to the visualization.
Twitter Viewer	Visualization	Depicts twitter notifications transmitted by the demo device.
Compass Capability Device	Visualization	Device program that detects events, tweets event changes, and transmits real-time compass data.
Behavior Detection Demo	Visualization	Visualizes the current behavior provided by the demo device.
Behavior Detection Demo Client	Visualization	Updates the back-end services with real-time device state.
Behavior Detection Device	Visualization	Determines behavior and transmits state to demo client.
Crane Tracker Visualization	Visualization	Shows the devices on the network, and data generated by the devices.

Table 4.1: A summary of back-end developments.

to utilize all the available web services, since it only takes advantage of the entity service. This is left for future work.

4.3 Summary of Back-end Developments

In Table 4.1, the back-end and visualization components are summarized. This shows the extensive quantity of components involved to utilize the developed tracking device (discussed in Chapter 3). The services in the architecture are used to manage the network, and provide access to data generated by the network. The visualization components are used to illustrate the deployed network. In addition, some the visualization components are used to visualize specific capabilities of the device (compass and behavior visualizations).

Chapter 5

Simulation

Rapid advances in embedded technology enable deployment of wireless sensor networks for scientists at low cost. These embedded sensors can and are being put to use in many applications [4], playing important roles in industrial and scientific projects [4]. The wireless sensor networks are being utilized to solve problems in domains that have not been studied before because of the reduced cost [4]. In the crane project, the many unknown requirements made it difficult to design a system that supported the needs of the ecologists. Therefore, simulation was necessary to evaluate performance in the development process.

A typical embedded wireless sensor mote features a processor with limited volatile memory and processing power, a radio for communication between devices, and sensors. To satisfy the tracking requirements of the ecologists, the platform employs a GPS sensor to determine position. Studies carried out with GPS sensors revealed that the GPS device consumes the largest amount of amount of energy (5 times greater than compass), when compared to the rest of the sensors [9]. Therefore, energy-aware usage of the GPS sensor is necessary to sustain the operational life-times required to observe an entire migration. Furthermore, the crane migration duration and mobility make recapture at-

tempts infeasible after deployments. To fulfill the tracking requirements, development of device simulators are necessary to evaluate the system. The results of the simulation will influence the performance and longevity of the system.

Simulation is invaluable to development efforts of tracking systems, primarily in cases where access to the species is difficult or impossible. Simulation allows the components, target behavior, and energy characteristics to be evaluated. This is beneficial when it is not exactly known what is physically needed during the design phase of the development process. In testing efforts, simulation permits the platform software to be executed without requiring actual hardware. For these reasons, the modeling capability of simulation places a crucial role in the development of Wireless Sensor Networks.

The simulators used in WSNs are centered around modeling the multi-faceted behavior of a wireless network. These simulators focus on having extensible, reusable, scalable, complete, and high fidelity characteristics. The most popular network simulator is NS [79]. NS is targeted towards modeling traditional networks that behave differently than WSNs [25, 79]. NS is known to be difficult for customization and integration into the application layer. Thus, little effort has been made to tailor NS for use in WSNs. SENSE is a simulator developed based on the short-comings of NS [25, 79]. SENSE can be easily modified and is fast compared to NS-2 [25, 38]. However, the need for direct integration into developed applications make use of NS-2 and SENSE undesirable [25, 38, 79].

TOSSIM is a popular WSN simulator centered around the TinyOS [35]. TOSSIM can directly execute logic developed for TinyOS without modification, at a high-level. TOSSIM is also capable of WSN simulation at all network levels [35]. Furthermore, it is capable of simulating packet and bit-level networks [35]. However, TOSSIM is incapable of executing more than one application on the network, and does not capture important behaviors including CPU and energy characteristics [38]. Energy modeling functionality was eventually added through the research efforts of PowerTOSSIM-Z [87]. In efforts to

capture CPU behaviors and preemption not available in TOSSIM, Avrora was developed to model cycle accurate behaviors of a WSN device [101]. The binary created for AVR devices, is actually executed by Avrora. This eliminates the dependency on TinyOS for simulation behavior of WSNs [35, 101]. However, this comes at a cost of nearly 50% more execution time than seen with TOSSIM. Despite being cycle accurate, Avrora does not model clock drift and originally did not model energy consumption [38]. Based on [63], Avrora is now capable of modelling energy consumption [101].

The Contiki operating system is an alternative operating system to TinyOS, used in WSNs [32]. Contiki features a built-in simulator called COOJA. COOJA is useful for simulation of Contiki programs at the code level, similar to TOSSIM [32, 38]. COOJA is a cross-layer simulator capable of simulating different applications simultaneously at multiple layers [84]. The cross-layer behavior exhibits increased granularity in simulation outcomes not available with TOSSIM [35, 38, 84]. However, the extensibility COOJA provides comes at a cost of efficiency [38].

All of the previously described simulation frameworks [32, 35, 84, 101] are incapable of modeling complex sensors, long-range communication, and migratory bird behavior. These aspects are necessary for development and testing of a system that monitors migratory birds. All of the previously defined simulation frameworks fail to model mobility. NS supports many mobility models, however NS is incapable of incorporating the application and sensing behaviors into the simulation. The work proposed in this thesis attempts to improve these simulation aspects as discussed in Chapter 5.

As discussed previously discussed, many simulators exist in the WSN community. However, these simulations focus on simulation of program execution, the wireless network, and power consumption of simple ADC, single radio and processor models [35] [87]. To the best of our knowledge, there are no WSN simulation solutions of peripheral components, such as GPS, GSM, and compass devices. In addition, there

are no mobility simulation studies that focus effort towards modeling a migratory bird during the migration period. Simulation of hardware components and crane behavior is necessary to demonstrate the capabilities of WSNs for crane migration tracking.

As described in Chapter 3, extensive studies are performed to assist with understanding energy consumption and operational performance of the devices intended for use in crane tracking. The research results are used to provide input parameters for simulations. The accurate inputs make for more realistic simulation results. The simulators described in this chapter utilize this data to accurately model system behavior.

In addition to modeling power and operation performance, simulation of actual crane and device behavior is used to assist with testing. In this chapter, individual simulators and emulators that are developed are discussed. The developed simulators/emulators include a crane migration simulator, GPS simulator, compass simulator, and GSM simulator.

This chapter also introduces the concept of *Wsn IntegeRatEd Simulator* (WIRES) which is an architecture for integrating multiple simulators together into a unified simulation system. The integration of multiple simulations together improves modeling granularity and testing capabilities. The granularity is improved because the cross-domain performance is considered, where the result of one simulation model can influence the results of the other simulation. The testing capabilities are improved because software can be developed without development or purchasing of the target devices.

The WIRES framework is beneficial for both simulation and testing. First, the WIRES concept is discussed in Section 5.1. Then, the simulation capabilities of WIRES is described and evaluated in Section 5.3, where the GPS simulator and crane behavior simulator are created and integrated together utilizing WIRES. With this configuration, an estimation of GPS performance and crane fatalities are provided. Then, the TOSSIM extensions created to assist with software development are described in Section 5.2. Finally,

as an alternative for evaluation, the testing potential of WIRES is described in the GSM and compass simulators in section 5.4. This section discusses the GSM and compass simulators, including their integration with WIRES.

5.1 Integrating Simulations Together with WIRES

Existing simulators such as TOSSIM [35], only simulate TinyOS code and a wireless ad-hoc network. It is necessary to simulate more than these aspects of the system, since the devices that reside in modern networks feature components not modeled in these previous efforts. An example of a modern network is the tracking system described in this work.

WIRES is developed to integrate multiple simulators into a working unit. The framework is a distributed-discrete event simulator [42], where independent simulation modules work together to simulate a functionality, hardware, and a WSN.

WIRES is inherently broken up into two simulation entities; simulation agents, and event sources. In this architecture, the event sources generate events that are then processed by simulation agents. Each simulation agent processes or simulates a specific aspect of the modeled system. Each simulation runs independently of another and in most cases invoke alternative event sources to handle and possibly generate more events. In this architecture, the simulation computation can be distributed to heterogeneous computing platforms, and communicate utilizing message passing.

As illustrated in Figure 5.1, the simulation agents in WIRES consist of three types of simulation agents: component, environment, and actor. Each of these simulation components can be run either independent of each other or dependent on each other. Each simulation is not capable of advancing simulation unless the simulation's dependencies have advanced their simulation clock beyond the clock of the local simulation clock.

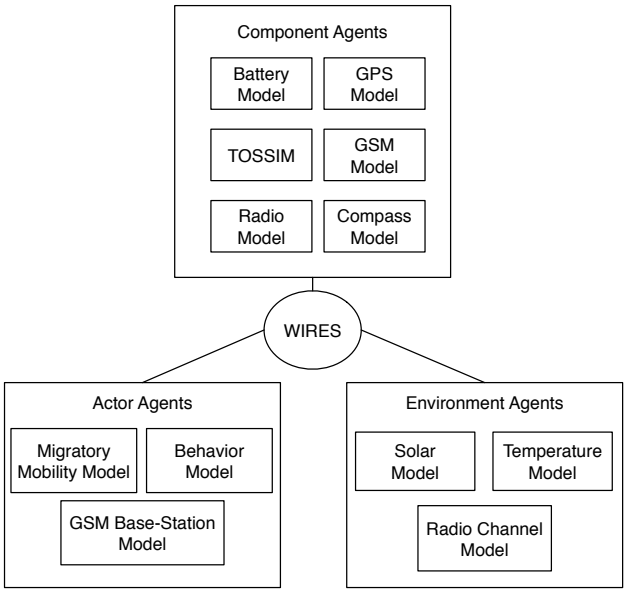


Figure 5.1: An illustration of the types of simulation agents in WIRES.

The component agents are an abstraction of hardware and software components in the modeled system. For example, TOSSIM is included in this category because the logic of the mote software is encapsulated and modeled by TOSSIM. Hardware components such as the GPS and compass sensors fall into this category as well. Power and energy harvesting components such as batteries and solar panels are also simulated as components. Many of these components are dependent on other components. In addition, the components are dependent on environmental and actor components.

The environment simulation agents simulate physical environment factors that influence the properties of the components and actors in the network. Some examples of possible environment simulation agents include temperature, solar energy, and the radio channel model agents. It is assumed that there are several dependencies on environment agents, since the environment influences many parts of the system. For instance, the temperature sensor is dependent on the temperature simulation agent and the solar panel component is dependent on the solar energy simulation agent. As another example, if

an improved radio channel model simulation is developed, it may be desirable to evaluate the influence it has on existing MAC protocols. Instead of creating a new network simulator, the channel model encompassed in TOSSIM can be replaced with a generic version that depends on the data provided by the external radio channel simulation. The two simulations utilize WIRES to message pass and advance simulation state.

The actor simulation agents are immobile or mobile entities that are usually equipped with a mote. An example mobile actor simulation includes the crane simulation, in which the migration of the crane is simulated. In the immobile case, static nodes are still considered actors. An example of the static case, are the base-stations distributed in the cellular network across the continent. A simulation of these nodes can be considered an agent simulation.

There are many challenges with developing a simulator of increased fidelity. First, utilization of existing simulations is challenging. It is difficult to correctly identify the aspects that should be connected to alternative agents. Second, it is difficult to control and properly simulate interactions between dependent simulation agents. As simulation fidelity increases, more CPU time is required for simulations. Finding a balance between fidelity and practicality is important to consider. Finally, it is important to ensure that the simulation framework is valid.

In discrete event simulation, a discrete event is an occurrence of an event in an instant of time [42]. Examples of this in WSNs include a timer firing from a clock on an MCU, a crane flying to a different position, a GPS acquiring a fix, or a packet broadcast onto a network. In simulation, the events are inserted into a queue and then processed by the time the events occur, in ascending order.

By distributing the simulation work load, it is obvious that potentially larger and more complex systems can be modeled [42]. Without considering the validity of a specific simulation model, to perform a valid *distributed* simulation, it is important that

system simulation must never process events from the past. Otherwise, the simulation becomes invalidated since the past will not be included in the future state. To mitigate these issues there are two common methods employed. First, the simulation agents can optimistically run and when past events occur, the simulator will go back in time and recompute all events. This requires large data structures to store state to recompute history, which is undesirable [42]. Alternatively, as employed by WIRES, the simulation nodes maintain knowledge of dependent event sources. When required, the simulation nodes query the event sources before processing an event to insure that the simulation node never receives an event from the past.

5.1.1 WIRES Architecture

As illustrated in Figure 5.2, the WIRES architecture is composed of the WIRES Messaging Service object, topic connection, topic session, topic publisher, topic subscriber, topic and message. To communicate between simulation agents and event source, WIRES utilizes message passing techniques in the form of a publisher-subscriber model [48]. This technique is designed to loosely couple communication between each simulation entity. This loosely coupled design enables WIRES to be interoperable and extensible, which is a feature desirable by any simulation framework. The design of the messaging service for WIRES is heavily influenced by the Java Messaging Service (JMS) API [88]. Hence WIRES employs a mediated, observer-observable design pattern. JMS was not utilized because of the dependency to the JVM and J2EE stack. However, the software utilized by sensor motes is written in C-based languages making JVM difficult to use. To follow the same principles of JVM, the same ideologies are utilized, but are implemented with the C++ programming language. WIRES is composed of the WIRES Messaging Service, topics, messages, topic connections, topic sessions, topic publishers, and topic

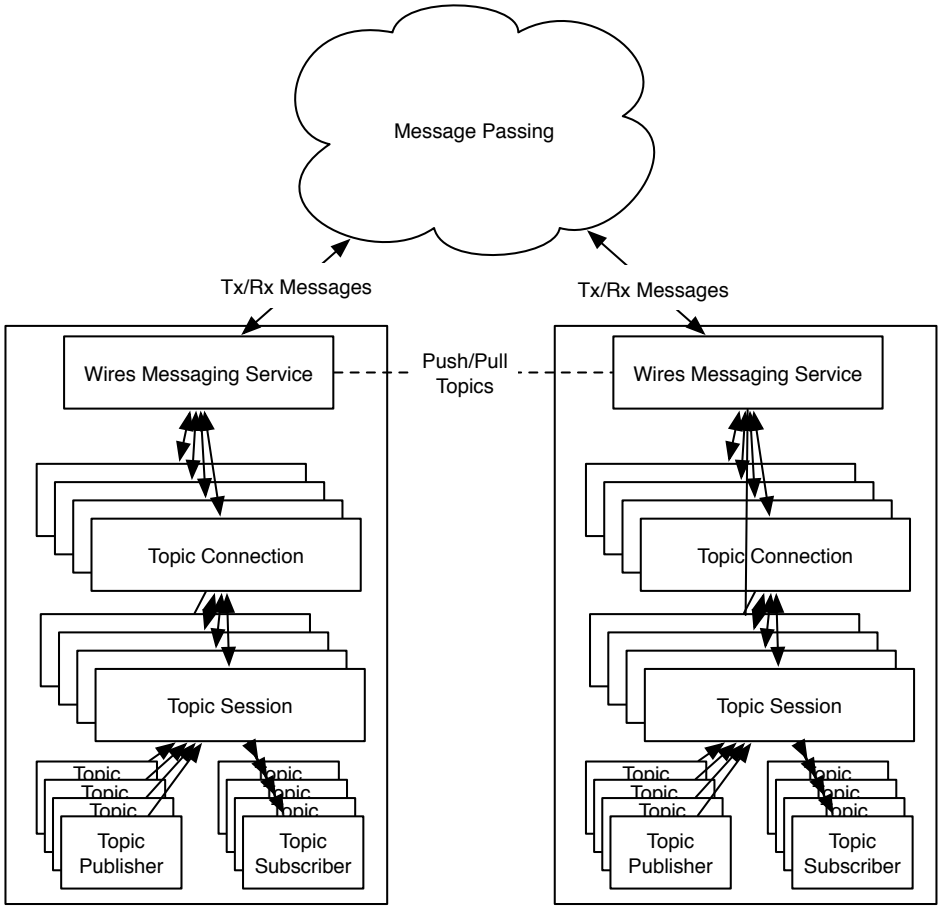


Figure 5.2: High-level illustration of WIRES architecture.

subscribers.

A **message** encapsulates the data actually transmitted across the WMS. It is composed of a header and a body. The header features the topic in which the message is sent, the time at which the message was sent, a hash containing the WMS entities visited, and a universally-unique identifier tag. The hash is necessary so the message is not received by multiple WIRES components (topic, session, publisher, subscriber) numerous times. The body of the message is by default a ”null” message, but is a template enabling transmission of any message type. However, the body must implement a serialization method so it can be utilized by WMS. These messages are passed with respect to a topic,

by publishers and subscribers.

A **topic** is the topic at which subscribers and publishers communicate messages. This is the basis at which publishers are subscribers are mapped. Topics consist of a topic name, which is a REST influenced string to represent a topic. The string name is a human readable description of the topic. In addition, a topic contains additional meta-information that enables the user to control the scope of the topic. In other words, the topic specifies whether the messages of the topic can be broadcast through the local subscriber, session and connection layer. This eliminates unnecessary dissemination of messages to all parts of the connected services.

Topic subscribers consume messages for a given topic. The topic subscribers are instantiated from a parenting session object, based on a topic that the parenting process will receive. The subscriber registers with the session to receive messages based on the topic object. It is the responsibility of the end-user to bind a call-back upon the event of a received message from the specified topic.

The **topic publisher** produces messages for a given topic. The topic publishers are instantiated from a parenting session object, with a given topic to publish. When instantiated, the session will percolate state to all the WMS layers to inform that messages of a topic will be produced. After registering with the other layers, the topic publisher object's creator will be able to use a publish function to transmit messages based on the initial topic.

The **topic session** is responsible for management of publisher-subscribers, queueing messages from upper and lower layers, and routing messages between connections, publishers and subscribers. The topic session is composed of hash map of subscribers, publishers, and a message queue. When publishers are created, a session binds a callback to all publishers, which are invoked whenever a publisher "publishes" a message. The topic subscribers are bound to observe a signal from the session when messages are re-

ceived. All processing of messages is performed in the background and asynchronously. The topic session acts as a factory for message creation. During creation, the session decorates the message with important meta information utilized during routing.

The **topic connection** is responsible for managing sessions, and communicating with the wires messaging service object. The topic connection stores sessions in a hash table and processes messages in the background. The purpose of the topic connection is to allow thread-safe operation for local and simulation node specific operations.

The **wires messaging service** object facilitates the messaging between computing nodes. When created, it searches for computing nodes and shares newly registered and already registered topics. Furthermore, the service maintains a publisher/subscriber list that is percolated to other service objects on the network. The WMS object manages connection creation, messaging queue, connection creation, and message routing. The services utilize callbacks to bind in-bound and out-bound messages to external parties and the connection layer.

5.1.2 WIRES Simulation

The WIRES simulation core consists mainly of simulation events, agents and event-sources. The relationship between each simulation entity, the agents and event-sources, is described through topics. In the framework, simulation events represent events that exist in a simulated model.

Simulation events are the actual events that are passed between simulation agents and event sources. The event contains meta-information that simulation entities use to handle events. These event objects are composed of the topics that determine what publisher and subscriber will produce and consume the event type. The event also possess a time property that is used by the agents for the execution process.

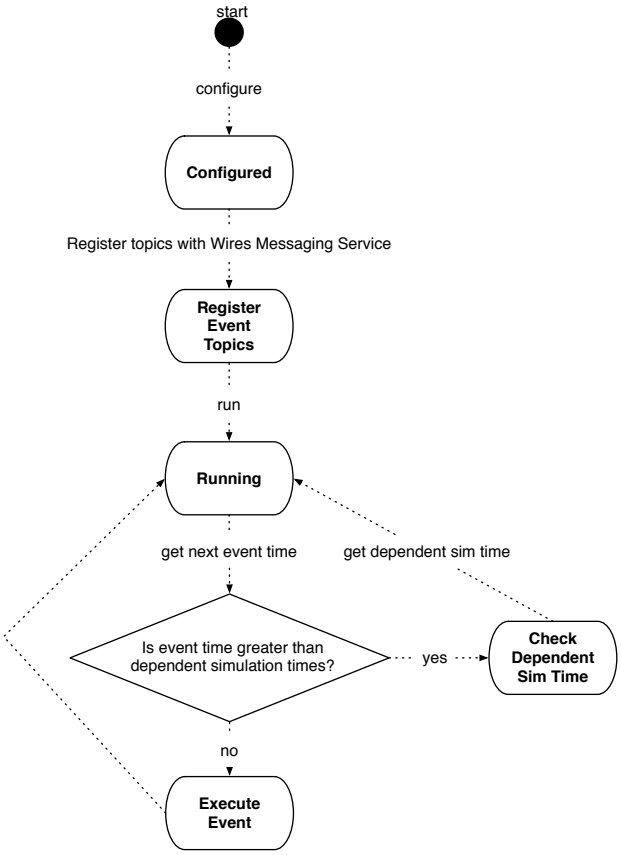


Figure 5.3: The high-level flow diagram of WIRES simulation.

The simulation event source generates the events that exist in the simulation. During instantiation, the source registers the desired registration and time topics with WMS. Before the simulation starts, the user must enroll the event source with the simulation related topics, so agents will be able to receive simulation events generated by the source.

The simulation agent is responsible for processing events, synchronizing time between event-sources, and executing events. Each simulation agent is composed of an event queue and an event dispatcher. The event queue is prioritized by time and the event dispatcher processes the event. The agent also maintains an event-source table to ensure that events will never be received in the past. The entire simulation is considered

valid in the sense of general discrete simulation, since the simulation cores ensure that events are never executed from the past. However, the validity of the simulated context is relative to the agents validity. For example, consider a WIRES simulation that consists of crane and GPS simulation agents. For demonstration purposes, in this simulation the crane simulation is invalid. Since the crane simulation is invalid, this may render the entire simulation invalid, dependent on what is invalid in the crane simulation.

5.2 The TOSSIM Extensions

As discussed in Chapter 3, the CraneTracker software is developed using TinyOS and NesC. The TinyOS operating system features a software simulator called TOSSIM [35]. TOSSIM is a discrete-event simulator that is generated during compilation of TinyOS programs. TOSSIM executes the same code that runs on the target device. To accomplish this, TOSSIM abstracts hardware details, such as hardware events, into discrete simulation events. TOSSIM is powerful and utilized by numerous researchers. However, TOSSSIM focuses solely on simulation of network behaviors, and lacks simulation of peripheral devices making it difficult to develop new complicated platforms. In this work, efforts are made to integrate TOSSIM to external simulators utilizing the WIRES framework.

To integrate with external simulators through WIRES, the serial communication abstractions for I2C and UART are implemented. In addition, TOSSIM is modified to enable the simulation time to be observed by external parties. Moreover, access to the event queue and event processor are also added to enable external control of TOSSIM. These extensions are necessary in order to synchronize with external simulations through WIRES.

The serial communication abstractions are necessary because these communications are utilized by the MCU to interface peripheral devices. These abstractions are only sim-

ulated at the byte level, instead of bit-level. It was decided bit-level simulation was not necessary since the drivers utilized only byte-level functionality of the serial communications. This eliminates bit-level events, thus, reducing the number of events inserted into the event-queue. With these abstractions, the device software remains unchanged and can be tested without requiring extra hardware.

5.3 Crane and GPS Simulators

The GPS sensor consumes a significant amount of energy and requires rigid timing and voltage characteristics to operate correctly. The system software must monitor the sensor and change the state accordingly. It is also important for the system to observe positions to determine if the bird is alive or dead. Simulation of the migration and GPS sensor is vital in the outcome of the crane project.

It is important to estimate the performance of the system before deployment. First, the crane migration is simulated. This consists of the paths that the Whooping Cranes will take during migration. Second, the GPS behavior and performance is simulated for estimation of energy consumption. Furthermore, the simulation and integration of the GPS with TOSSIM, through WIRES, allows the system software to test against more realistic GPS behavior.

At the time that this work was completed, the ecologists at the Crane Trust only provided a single Whooping Crane path for simulation. The data from this path (heading and distance traveled over time) is used for the parameters of the distribution used in simulating the Whooping Crane paths, which provide true paths for simulation and influence the output of the GPS. The data from the ecologists is fitted a normal distribution for generating random distances and headings of the Whooping Crane paths. It was concluded by applying a normally distributed accuracy error and acquisition error

(i.e getting a fix or not). Finally, the simulated path is illustrated in a graphical display using Google Earth [57].

5.3.1 Objectives

The overall objective of this work is to simulate crane migration and GPS performance as a function of time. The GPS performance is simulated with respect to energy consumption, time-to-first-fix, and acquisition accuracy. The parameters used in the simulation are determined based on the requirements of the ecologists and the results of preliminary studies [9]. The limited data and unknown behaviors resulted in assuming normal distributions. In addition, the simulations are limited to a single migration period from Texas to Canada, which results in a terminating simulation. The crane simulator simulates 60 birds and respective GPS devices over a migration period. This is performed for 100 replications to facilitate an adequate output analysis.

5.3.2 Model Conceptualization and Statement of Assumptions

Two simulation models are brought together to form a single simulation with WIRES. In the first simulation model, the simulated paths of the migrating cranes and the likeliness of death are computed. Alternatively, the GPS simulator, simulates the energy performance, time-to-first-fix (TTFF), acquisition, and fix error. In real life, the path that is followed by the migrating birds is from Aransas National Wildlife range, Texas, USA, to Wood Buffalo National Park, Alberta, Canada. Typically, the birds take between 4 to 6 weeks to traverse this distance. The simulator is developed to model a simulated path for 60 birds, which is the number of birds permitted to be tagged by the ecologists at the time of the study. Therefore, the simulation model is limited to a finite population of 60 devices.

A number of scenarios are represented in the model. In the first scenario, an ideal case in which a bird's path is successfully tracked from origin to the destination occurs. In the second scenario however, some birds perish at various points before the final destination. As a result, their paths are terminated before completion of the final simulation path. The paths for both cases are simulated at once, with all 60 birds migrating from the origin at the same time and only those in the first scenario will reach the destination.

A number of assumptions are used to model the system. First, during the migration, it is assumed that the device will get a fix or not, and that this will happen over a normally distributed duration. Therefore, the model in our system can be considered a binomial distribution, in which 1 represents a success when a position is acquired and 0 represents a failure when no signal is acquired. Based on experimental analysis, a real GPS device takes 36 seconds to 3.5 minutes to acquire a signal. At the time of work, there was no data available to assist with determining time-to-first-fix of the acquisitions on migratory species. The unknown environment, high-mobility, and unpredictability of the Whooping Crane motivated utilizing the normal distribution, with a mean time of 120 and a variance of 84 seconds. This GPS average range was observed in experiments performed with the MTS-420 mote. In the simulation, the number of GPS fix attempts is limited to one per day due to restrictions of the system and the power model developed in the early stages of the project. In practice, GPS sensors do not acquire GPS signals accurately due to signal attenuation. This behavior must be modeled by the simulation. The horizontal accuracy observed in this study [66], observed the first-fix errors to be as high as 100 meters. However, the GPS receiver on the MTS-520 claims accuracy of 5 meters [106] in the worst case. To compensate for the vast horizontal error, the simulation uses a normal distribution with a mean error of 50 meters and a variance of 15 meters used to model the fix error of each simulated position acquisition.

The random number generator (RNG) that was chosen was the Wichmann-Hill gen-

erator [110], since it is part of the standard C-library. The period of the generator is 6,953,607,871,644. Terminating the Crane and GPS simulation, in addition to utilizing an independent seed, is long enough for eliminating instances of overlapped random variates throughout the simulation process. The overlap elimination is necessary for reducing correlation caused by the random numbers and insuring that the components are independent and identically distributed.

5.3.3 Input Data Collection and Analysis

The input data and parameters required for realistic simulation output include a distribution of distance and heading values, distribution of TTFF, duration of the migration, number of cranes to simulate, mortality rate, initial position, GPS accuracy error, the GPS fix rate and a battery model. To improve the performance metrics of the simulation, empirical data taken from component evaluations, prior studies [66], and ecologists are used. The parameters and inputs can be easily modified to adhere to improved data set and modification of requirements.

Prior to this study, ecologists collected data from previous crane tracking attempts. Accordingly, the average crane travels 71.2 miles a day with a deviation of 48.7 miles and has an average bearing of -27 degrees from true north with a deviation of 22.5 degrees. Since the provided data has been collected using existing monitoring techniques, limited sample points exist. To compensate for this, we assume the distributions of both headings and degrees without loss of generality, are normal distribution, with a mean distance 71.2 miles and variance of 48.7 miles; a mean heading of -27 degrees and a variance of 22.5 degrees. Additionally, the ecologists provided the location that the captured whoopers would be released, resulting in the initial position. The prior studies suggest that 6%-8% of the birds die during each migration [74]. This statistic is also included in

the simulation to simulate mortalities.

To model the distribution of the duration it takes to get a first fix, several experiments are performed. The GPS device is capable of operating in different modes that influence the time it takes to acquire a GPS fix. These modes include cold-start, warm-start, and hot-start. Unless the power is removed from the GPS receiver, information from the initial acquisition can be used to improve first-fix acquisition time. In the best case mode (hot-start), the GPS is capable of acquiring a new fix after being disabled after approximately 5 seconds. In the best case, this will take approximately 34 seconds. It is assumed that the device is always in cold-start mode, since the cranes are extremely mobile, and the prior acquisition information can not be used. Without the prior information, it will take a longer time to search for signals to compute position. This assumption is also reinforced by the fact that the proposed system at the time, was only to be enabled once per day. Therefore, the device is unable to exploit rapid TTFF advantages of the GPS sensor.

TTFF experiments were performed to analyze the off-the-shelf GPS sensor [106]. Accordingly, the average time it takes the sensor to acquire a fix is approximately 111s. This time deviates 46 seconds throughout the tests performed. Using the TTFF experiment results, the GPS simulator generates realistic TTFF times. This duration is also used to compute the energy used by this device during the migration period. According to [92], the average accuracy of GPS sensors varies from device to device. The average error found by this study found the average error of the GPS receivers to be in the range of 21.2 to 76.9 meters. For realistic simulation results, these bounds are used to define the range of error in the GPS simulator.

The GPS performance varies from device to device, the need for analysis with a variable fix rate is important. Combining the known TTFF duration from prior results and trying multiple fix rates for a valid fix, the simulation platform is used to find the

expected energy usage for each platform. In the application, it is assumed that the device can take up to 5 minutes to acquire a fix.

To test the reliability of the generated paths the Chi Squared Goodness of Fit test is applied to the generated distances and headings for 60 birds over 42 days. The null hypothesis is that the data for distances and headings are from a random sample, and are normally distributed with level 5 of significance. If the test fails to reject the null hypothesis, our distribution follows the Normal distribution. If the null hypothesis is rejected our distribution does not follow the specified distribution. Utilizing MATLAB it was found that tests failed to reject the null hypothesis. Therefore, our system fits correctly to the assumed distributions.

5.3.4 Model Development

The GPS sensor entity models the GPS device behavior. This behavior is modeled by generating events into the simulator. As the events are executed the behavior state is updated. The Crane entity in the Crane simulator, is responsible for modeling the movement of the crane and bird fatalities. As with the GPS sensor, the entity generates events. As the events are executed, the state of the crane model is updated.

The GPS simulator and Crane simulator are integrated together through WIRES to make a complete simulation. As illustrated earlier, the high-level flow of a WIRES simulation is illustrated in Figure 5.3. In Figure 5.4, each simulation is depicted, and explained. In this configuration, the GPS simulator is dependent on the crane simulation to get the theoretical position of the bird. Therefore, it must wait each step in time for the location of the bird at that instance in time. Since the crane simulation is not dependent on the GPS device, the Crane simulation can complete before the GPS simulation. The crane simulation also simulates whether or not the bird died during the migration. Using the

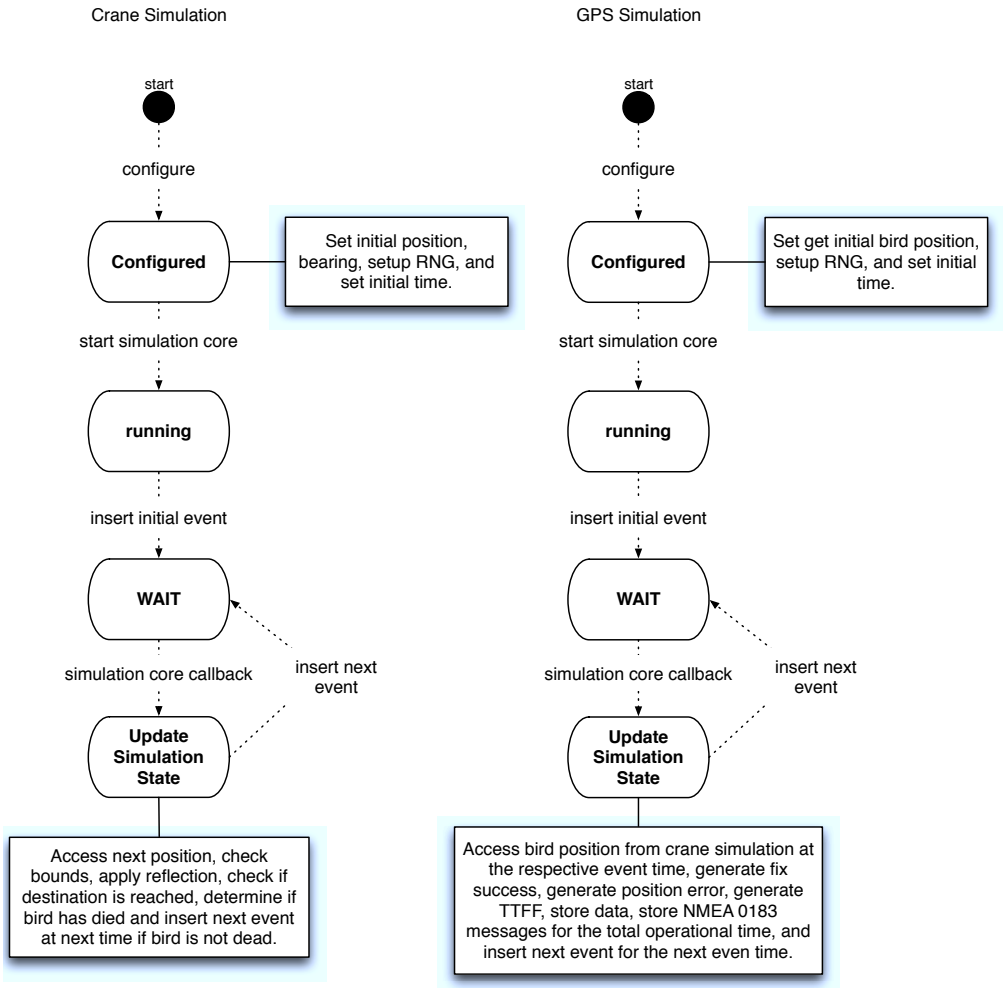


Figure 5.4: GPS and GSM simulation flow diagrams, with respect to WIRES core interaction.

theoretical position of the crane, the GPS simulation generates a position with added error more realistic to that provided a real receiver. In addition, the GPS simulator simulates the TTFF and if the device acquired a fix. The GPS simulator emits data files containing energy consumption profiling, fix-rate, the NMEA0183 sentences, and a KML file containing the positions in the sequence they were observed. The Crane simulator emits the simulated migration path in KML, and also statistics on whether or not the bird perished during the migration.

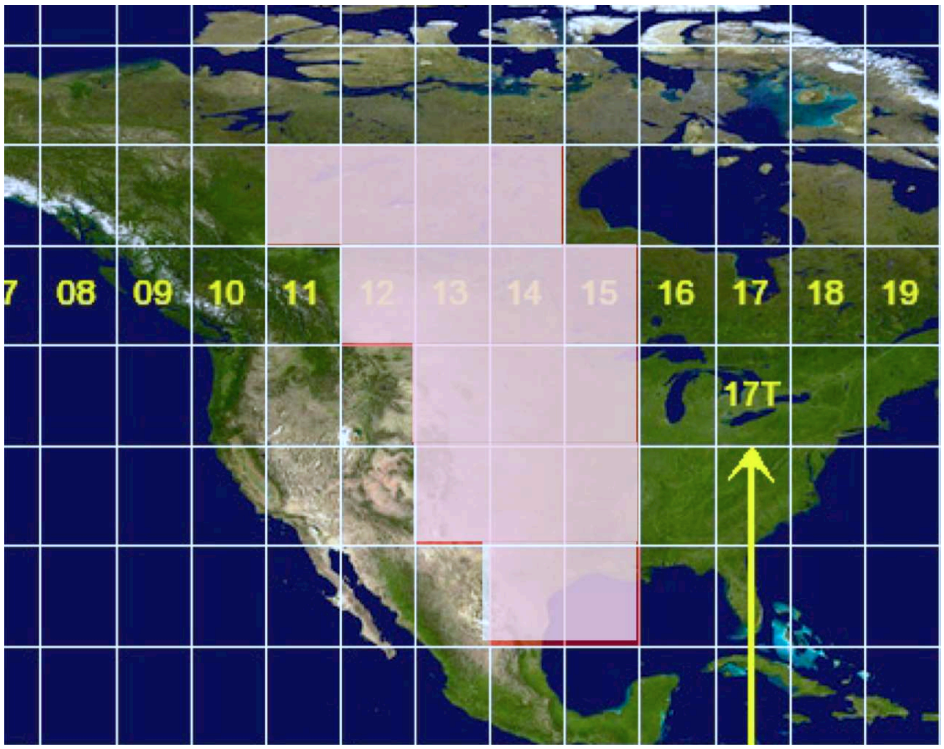


Figure 5.5: Bounding regions masked above the projected earth.

5.3.5 Validation of the Simulation Model

At the time of development, the available Aransas Wood Buffalo Population (AWBP) data was limited for reasons of bird scarcity and the limited number of trackers mounted on the Whooping Cranes. Therefore, a subset of the data was provided by Dr. Chavez at the Crane Trust. To ensure valid crane paths, the ecologists provided a bounded region where the birds were known to migrate in. This bounding region is depicted in Figure 5.5.

At any point that the generated position violates the bounding constraints, the model applies a reflection to keep the cranes paths inside the correct regions of migration. The effect changes the trajectory of the cranes' path. This results with a constrained path with bias towards the known breeding or wintering ground. This effect does not affect distance traveled; only the heading the crane will use for the next position. This

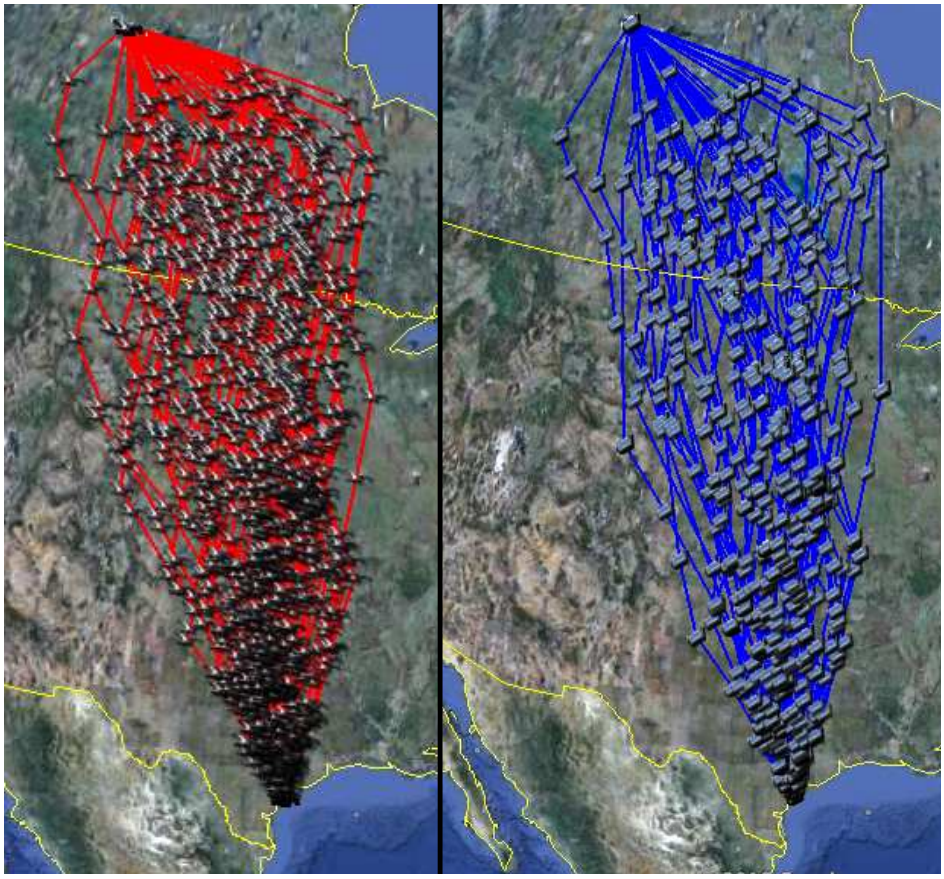


Figure 5.6: Generated paths of crane paths(red on left) and GPS acquisitions (blue on right).

behavior is modeled after the Random Walk Mobility Model (RWMM) [23], where the mobility area is bounded empirically, the course and distance are based on a probabilistic model. The RWMM is utilized any many efforts and is considered standard by the community [23]. In Figure 5.6, the migration paths (red on left) and sensed positions of the motes (blue on right) are plotted on Google Earth [57]. All of the generated points fall in the known migration region, and each represent a possible migration path traversed by a Whooping Crane.

5.3.6 Analysis of the Simulation Data

The simulator was ran for 100 repetitions with 60 birds. In the following, the results of the simulation are discussed. This discussion features evaluation of the simulated deaths, number of GPS fixes, GPS fix error, the GPS accuracy error, time-to-first-fix, energy consumption, and the possible encounters that occurred during the simulations.

5.3.6.1 Simulated Deaths

From prior work, it is known that six percent of Whooping Cranes are incapacitated per migration. The cause of these deaths are undetermined. The fatalities are simulated for 60 birds, over 42 days, for 100 replications. The results verify that our simulator models mortality correctly. In the results, four birds died on average over the simulated period with a 95% confidence interval of (3.91, 4.27). This reflects the known percentage of 6% of birds that perish per migration in real life.

5.3.6.2 Number of GPS Fixes

The limited data, unknown environmental characteristics, unpredictability, and high-mobility of the Whooping Crane, motivate use of a binomial distribution with a probability of 1/2 to estimate the success of GPS fix. The results show the average number of fixes per device for each replication. The average number of GPS fixes is 21 with a 95% confidence interval of (21.03, 21.34). This is 4.2 times higher than the previously recorded average of 5 fixes per path found in empirical data provided by Dr. Chavez. These results show that, assuming the fix opportunity is binomial and if the given data set is valid for the population, the GPS devices will provide better performance than the satellite transmitters. Although successful, the results are not as great as observed during the deployments (Chapter 6), where 97% of the fixes attempted were actually

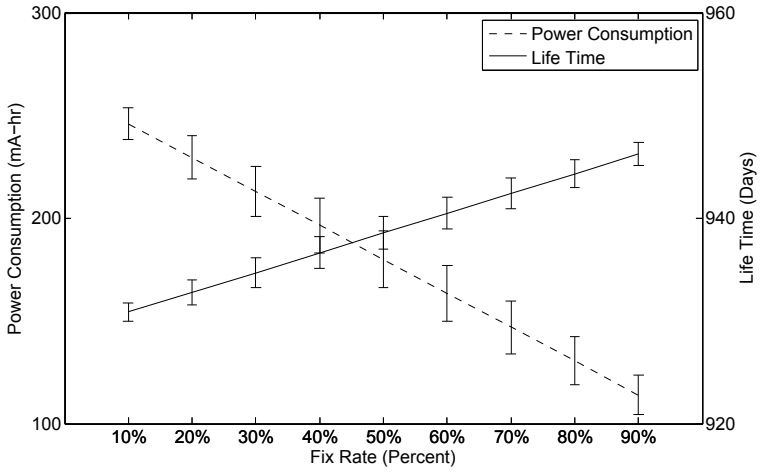


Figure 5.7: TTFF during one migration (60 cranes, 42 days).

acquired [11].

5.3.6.3 GPS Fix Error

The error of the GPS is important to model in simulation. The GPS error influences the ecologists' ability to accurately study habitat and even recover perished cranes. In the simulations, the average error distance for the GPS signal acquisitions is 77.26 meters with a 95% confidence interval of (75.76, 78.75). This falls within the standard range of 5-100 meters from the true (0,0) position on the X,Y coordinate axes. These results also agree with the experiments performed in later deployments, discussed in Chapter 6. Furthermore, the data falls correctly in the $8.2 - 108.2$ meter range reported in[66].

5.3.6.4 Time-to-First-Fix, Energy Consumption, and Life-Time

In the following analysis, the processing (6mA), radio (24mA), and sleep ($<$ 500uA) power consumption is not accounted for since it is considered negligible when compared to energy consumed by the GPS sensor (50mA in Figure 6.3). In Figure 5.3.6.4, the average TTFF duration and the corresponding estimated lifetime is shown as a function

of the GPS fix rate. For every fix that is not acquired, the timeout duration, i.e., 300s is reached. For every invalid GPS fix the system will accumulate a 300 second penalty against energy. This can be seen in the variation of 16 days of operation in the lifetime as the GPS fix rate changes. The lifetime of the application can be significantly impacted by the GPS fix rate. The result of this simulation motivates the importance of higher accuracy sensors. Furthermore, it motivates reducing the current time-out duration since the likeliness of acquiring a fix appears infeasible after 300 seconds.

5.3.6.5 Number of Encounters

It is of interest to understand the number of cranes that may encounter another crane during the migration. This information can be used for adding ad-hoc behavior to the tracking system. Determining an encounter is accomplished by computing the distance between each combination of cranes for each day of the migration. Since cranes generally roam around their layover points up to 5 miles, it is assumed if the cranes land within a 5 mile distance, a communication opportunity occurs. This analysis is based on a data from the generated crane paths.

The results are shown in Figure 5.8 and Figure 5.9, where the histogram for encounters with the same bird and the number of encounters in a day are shown, respectively. It is interesting to note that there is non-negligible chance that a mote has multiple opportunities to encounter another particular mote. Additionally, as shown in Figure 5.9, the greatest opportunity to communicate occurs during the earlier part of the migration and that the window for communication becomes random throughout the later part of the migration. These results motivate the need for probabilistic solutions that exploit the opportunities to communicate to develop *networking* solutions for Whooping Crane monitoring.

The overall simulated average number of fixes for the sixty birds is 21 during 42 days.

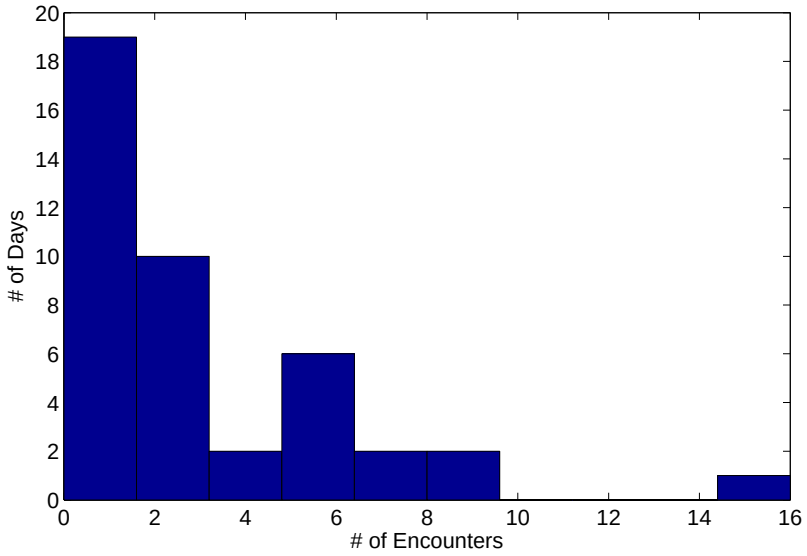


Figure 5.8: Histogram of encounters during one migration (60 cranes, 42 days).

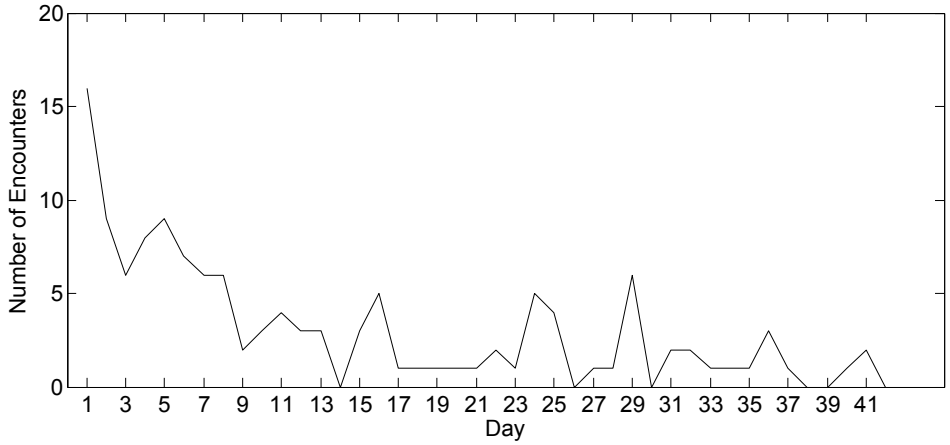


Figure 5.9: Number of encounters during one migration (60 cranes, 42 days).

This average was computed over the total of 60 birds, of which 4 perished on average. In this simulation effort, the TTFF was modeled. In combination with the fix-rate data, the TTFF assists with estimating the life-time of the mote. The result of this work motivates reducing the time-out value of the GPS sensor to increase the life-time of the tracking device. Finally, the number of encounters between each bird is explored and assists with

understand potential of future ad-hoc applications.

5.4 GSM and Compass Simulation

In efforts to progress software development before the hardware was available, it was necessary to develop simulators to accurately model device behavior. This capability is partially available through the GPS simulations, where the NMEA 0183 messages are emitted from the GPS simulator. The messages from this simulation, are then utilized by the GPS parsing software to verify correct functionality. However, the system described in Chapters 3 and 4 features two novel components: the GSM and compass, that initially lacked software to function properly. To facilitate rapid and controlled development of platform software, GSM and Compass simulators are developed. The simulators enable debugging, testing and analyzing of the software without a functional hardware components. In the following, the modeled behaviors are discussed. Then, each simulator will be described in detail.

It is impractical and unnecessary to model each and every aspect of the GSM and compass, so it is important to identify the behaviors that are critical enough to be modeled by the simulation. The properties to model of the GSM include connectivity and Hayes command protocol. Since the GSM does not have continuous connectivity, the system software must be able to handle cases when connectivity does not exist. In addition to the connectivity, the GSM is managed by utilizing the Hayes command protocol. This is a very complex protocol and only the subset of functionality is simulated.

The compass is not as complex as the GSM, but still must be modeled to enhance testing capabilities. A proprietary communication protocol [55] is simulated by the compass simulation. Each of the simulators utilize the TOSSIM extension described in the following section to interact together utilizing WIRES.

Simulation Component	Function	Simulated Aspects
Wires Messaging Service	Service used to communicate between simulation agents.	None, used for communication
Wires Simulation Core	Discrete simulation engine used for discrete simulation by agents.	Synchronization with dependent agents, executes events, advances clock
Crane Simulation	Simulates the whooping crane during a migration.	Single migration, 42 day simulation, 60 birds, simulates mortality, used to determine encounters
GPS Simulation	Simulates the GPS performance.	Simulates TTFF, GPS Fix Accuracy, acquisition rate
TOSSIM Extension	Extends TOSSIM application simulator to interface external behavior simulators.	Adds WIRES simulation core, inserts external IO events, simulates UART, simulates I^2C
GSM Simulation	Simulates the functional GSM behaviors.	Simulates Hayes command communication protocol over TOSSIM extension and returns arbitrary state.
Compass Simulation	Simulates the functional compass behaviors.	Simulates compass communication protocol over TOSSIM extension and returns arbitrary state.

Table 5.1: Summary of simulation developments.

5.5 Summary of Simulation Developments

A summary of the simulation developments are shown in Table 5.1. This is composed of WIRES components used to integrate multiple simulators into a single simulation. In addition, to show the validity of the simulation, WIRES is used in performance and testing simulation. The performance simulation capability is exhibited by the Crane and GPS simulations. The simulation capabilities towards testing are provided by the TOSSIM extension, GSM and Compass simulations. These components are used to assist with understanding system performance and testing code before hardware is available.

Chapter 6

Deployment

A diverse set of applications have been conceptualized for Mobile Wireless Sensor Networks [4, 72, 73]. These applications motivate the deployment of mobile WSNs towards monitoring wild migratory species, such as the Whooping Crane. However, the limited experience in deployment of mobile WSNs renders this process difficult. The novelty of monitoring an endangered species with mobile WSNs add additional burdens not found in standard deployments.

This chapter discusses the pre-deployment phase of this work, where the tracking platform is evaluated through component tests. From these system integration evaluations, the initial operational parameters are generated for use in proxy deployments before use on the Whooping Crane. The proxy deployments explored in this chapter assist with justifying the use of the tracking system for monitoring the endangered Whooping Crane.

In this chapter, efforts toward deployment of the developed system are described. In each deployment, the corresponding results are described. In Section 6.1, an overview of the deployment efforts are discussed. Following this discussion, in Section 6.2, the enclosure development efforts are described. The preliminary efforts for the evaluation

the system prior to the proxy deployments are discussed in Section 6.3. Afterward, the proxy deployment on Wild Turkeys is discussed in Section 6.4. In Section 6.5, the deployment of the system on captive Siberian Cranes is described. Finally, the deployment of the system on an immediate relative of the Whooping Crane, the Sandhill Crane, is analyzed in Section 6.5.

6.1 Overview

The Whooping Crane is an endangered species. Thus, the effectiveness of a new platform must be proven through deployments on proxy species. To illustrate the effectiveness, the system is validated with preliminary experiments to motivate further utilization in alternative deployment efforts. Following the preliminary efforts and to prove the effectiveness of the platform, the platform is deployed on "proxy" species that share similar characteristics with the Whooping Cranes [11]. The Wild Turkey (*Meleagris gallopavo*) was considered a good candidate due to their high abundance and low mobility. More importantly, the Wild Turkey share similar habitat where cranes are known to utilize. The turkey deployments enable initial evaluation of performance expectations of the system in terms of cellular-coverage, latency, mechanical durability and initial recharging characteristics [11].

The successful deployment of the units on Wild Turkeys led to the deployment of the units on captive Siberian Cranes (*Grus Leucogeranus*) at the International Crane Foundation (ICF), in Baraboo, Wisconsin. The purpose of the Siberian Crane deployments were to provide an additional measure of performance before placing the units in a migratory setting. In these efforts, the durability, short-range communication, and compass are evaluated [11].

The result of the Siberian Crane deployment effort led to the deployment of the

system in a realistic migratory and inaccessible setting, on wild Sandhill Cranes (*Grus Cahadensis*). The Sandhill Crane is an abundant, immediate relative of the Whooping Crane, that behaves similar to the Whooping Crane. In this effort, the tracking device is deployed on five wild Sandhill Cranes. In the deployment, two of the birds have completed migrations from Wisconsin to Kentucky and Wisconsin to Florida, respectively [11].

6.2 Enclosure

The system is required to follow rigid weight and multi-year deployment requirements. Therefore, the enclosure must be durable enough to withstand harsh environments and protect the enclosed electronics. In addition, the enclosure must not exceed weight requirements designated by the ecologists. Finally, a solar panel is needed for harvesting energy. It is necessary for this panel to be attached to the encolsure in such a manner that has optimal harvesting potential [11, 10].

The Whooping Crane can thrive in extreme blizzard and hostile weather conditions [44]. The habitat, during wintering, usually consist of salt water marshes. Thus, any exposure of the electronics from these elements will usually result in system failure, because the components may start to erode. Designing for these conditions usually result in a larger and heavier solution. Therefore, a balance between weight and durability is necessary in the enclosure development efforts.

Traditional Whooping Crane tracking devices are mounted on the leg of the Whooping Crane [62]. The ecologists expressed concerns with the leg mounted approach, and suggested utilizing a backpack mounted approach. The ecologists are concerned that the leg mounted tracker was interfering with copulation. Furthermore, the lengthy antennas on the prior tracker was found to be constantly pulled at, and over time could become

removed from the tracker. As a result, the tracker would become useless since the unit would not function properly without the antenna. The ecologists are also concerned with the lifetime of the unit. In other-words, the leg-mounted device would permanently be fixed to the bird, beyond the usefulness of the device [11, 10].

The backpack mounted approach permits the tracker to be mounted near the bird's center of gravity, minimizing the effect on copulation [11]. In this position, the backpack design has the potential to benefit from extended sun exposure and movement accuracy, when compared to the leg approach. To determine the best approach, different types of back pack designs were explored. Each design was employed in a series of deployments that consisted of an epoxy potted enclosure, printed plastic enclosure, a pouch, and flexible plastic tubing method. In the backpack mounted approach,the ecologist route a cloth strap around the breast bone, underneath the wing of the bird and then back around to be secured on the enclosure. The cloth strap benefited the bird by eventually falling off after the cloth strap frayed. In the leg band approach, the tracking unit was known to exist for the life of the bird. This was due to metal rivets utilized to keep tracking unit on the bird.

The *epoxy based enclosure*, was used since it is heavily used in numerous tracking efforts [37]. However, potting the device was found to add too much weight, difficult to set-up, set and extract. So this process was not considered since these problems were observed during the preliminary testing efforts.

The *printed plastic enclosure* (Figure 6.2), was designed with a computer automated design tool [93] and was fabricated utilizing a selective laser sintering (SLS) process [10]. This enclosure was never deployed on a crane because of the weight, irregular shape, lack of waterproofness and difficulties required for deployment. These properties were not known until deployment on Wild Turkeys, when it was found water breached the enclosure causing it to corrode and fail.

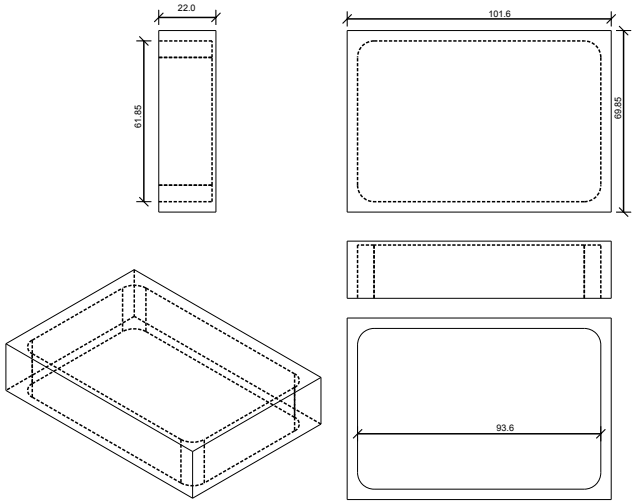


Figure 6.1: Illustration of the printed plastic enclosure.

Figure 6.2: CraneTracker on captive Siberian Crane (top), close-up view (bottom).

The *Oxford cloth enclosure* depicted in the top image of Figure 6.2, was designed to act as a quick and lightweight waterproof enclosure in response to issues found with the previous enclosure during the turkey deployment. The Oxford cloth is composed of fiber similar to nylon, woven to block water from passing through. One side of the pouch featured adhesive that was bound together into a pouch when applying direct

Figure 6.3: The VHF transmitter that inspired final enclosure design.

Figure 6.4: Flexible heat-shrink tubing being sealed with improved adhesive.

heat. The main idea of this enclosure was to seal the device inside of the cloth. This kept the device protected from the elements. The solar panel is then attached to the outside using glue, and the wires are routed from the solar panel to the device and sealed using epoxy. Unfortunately, the nylon fibers are naturally lubricated. Therefore, the cloth rejects any attempt to bond anything to the surface utilizing glue. This was evident when the solar panel would not stay fixed when the pouch was constructed. It was also found that not much force was required to separate the ”sealed” pouch. To evaluate the usefulness of the pouch, it wsa deployed on captive Siberian Cranes. The result of this is discussed in Section 6.6.

The *flexible heat-shrink tubing* (depicted in Fig. 6.2) was utilized to develop an enclosure. This design was inspired by a VHF transmitter (Fig. 6.3), that was used in a deployment in prior efforts. The prior transmitter was enclosed inside a PVC based heat-shrink tubing. Unfortunately, the heat-shrink previously used was only available in non-transparent material and was too narrow to contain the proposed platform. Instead, a wider and more transparent material was found to enable the solar panel to have optimal exposure to sunlight. It was necessary to contain the entire unit, including the solar panel, inside of the enclosure since it was extremely difficult to attach the panel to all the explored enclosures. Moreover, routing the wires from the panel to the inside added an additional failure point for water to breach.

The tubing which is fount to be the most suitable for the enclosure, was fluorinated ethylene propylene (FEP) heat-shrink tubing. The FEP tubing is a transparent plastic that does not deteriorate or discolor when exposed to direct sunlight. The plastic also withstands the harsh temperature ranges, harsh environments and harsh chemicals, including salt water found in the cranes' habitat. The tubing was also wide enough to fit the complete tracking unit.

The tubing was intended to be heat shrunk over the entity that required protection. This posed many difficulties. For example, when the tube was shrunk, it ended up compressing the solar panel. The compression inflicted damage to the wire terminals and minimized the surface area available to sun exposure. The effects were seen in the Sandhill Crane deployments discussed in Section 6.6. There was also a concern with the dangerous temperatures required to shrink the tubing. To shrink the tubing, it must be heated to 400 degrees Fahrenheit. This exceeds the maximum allowable temperature for ensuring the health of the battery. To ensure safety the ends were only shrunk over a long period of time. This allowed the batteries to remain at a safe temperature during seal

Figure 6.5: Flexible heat-shrink tubing being sealed with improved adhesive.

The process to seal the tube, as instructed by the product data sheet, requires sanding and application of etchant to the tubing. The result of the application, is a coarse surface to facilitate adhesion with an industrial strength glue. Even after the recommended process, the glue did not withstand force of testing. Due to time constraints, this method was still employed since it seemed the most suitable out of all approaches. For additional support, a PVC tubing featuring an internal adhesive was compressed and heated to 600 Fahrenheit on each end of the enclosure. The ends were far enough from the battery that there were no concerns about the health of the battery. The PVC tubing added an additional protective measure on each end of the enclosure. Following the result of the deployment of the first Sandhill Crane, this enclosure process was improved by exploring different sand paper grains and different industrial glues. The final enclosure utilized a single FEP tubing that was sealed by compressing the ends with a great amount of glue. The ends were then drilled for strap mounts and then rounded smooth. This eliminated

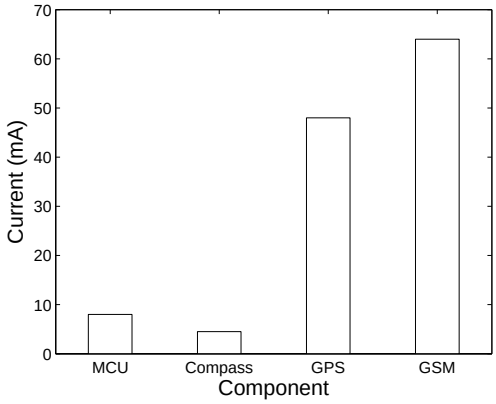


Figure 6.6: Power consumption of platform components.

the additional PVC tubing on each end of the tube. The final enclosure, illustrated in Figure 6.2 passed all stress tests. The only units that are still in operation at the time of writing, utilize this enclosure method.

6.3 Preliminary Evaluations

In the initial preliminary evaluations, it was necessary to evaluate the new communication capabilities. In this evaluation, a vehicle is used to drive through 160km of known migration area. This provided an initial measure on how effective the developed approach is towards the migratory tracking efforts. In this initial effort, the delay in communication and accuracy of the sensing capabilities are measured. From this evaluation, it was found that the sampling and communication delay was only 6.07 minutes ($\sigma = 1.54$ min). These measures are highly optimistic since human traversed highways are used in evaluation, but the delay is significantly less than the 48 hour satellite tracker delays [11]. Thus, the delay observed in the initial experiments motivated the deployment of the system on wild species.

In addition to the communication delays, an evaluation of the GSM localization accu-

racy is performed. In this evaluation, a distance of 160km is covered, and the tracking device was connected to 18 different base-stations over this area. Using the GPS device as a baseline, an estimate of error was computed. It was found that 50% of the error is less than 4km and in the worst case the error is bounded at 14km. This gives a promising redundancy in cases of GPS failure, or inability to acquire a GPS fix [11]. The success of these cellular results motivate deployment of the system on proxy species. The result of these deployments are described in the following.

6.4 Wild Turkey Deployment

From ecologists' field observations, it is known that two thirds of the Whooping Cranes' migration is spent foraging for food and roosting [11]. This time is spent in habitat similar to wild turkeys'. Therefore it was suggested that Wild Turkeys were a suitable candidate for deployment of the system. In this deployment, the tracking accuracy, communication performance, energy consumption and recharge performance in a limited area could be evaluated for a wild species.

In the turkey deployment, it was necessary to place not only a device on a turkey, but also a similar device in the open to act as a control. With the control, an idea of the available energy and the sensor accuracy of the turkey data could be established. In addition to the control, a tracker is attached to the back of a captured hen and then released for monitoring. The stationary control is placed in the open within 1km of the hen's known habitat. Accordingly, the central position provided an accurate idea of the environment. In the turkey deployment, the control and turkey units were deployed for 10 days [11]. During this week, each unit transmitted SMS messages every 4 hours containing GPS and compass data.

From the deployment, a total of 480 SMS messages, 105 GPS, and 9,170 compass-

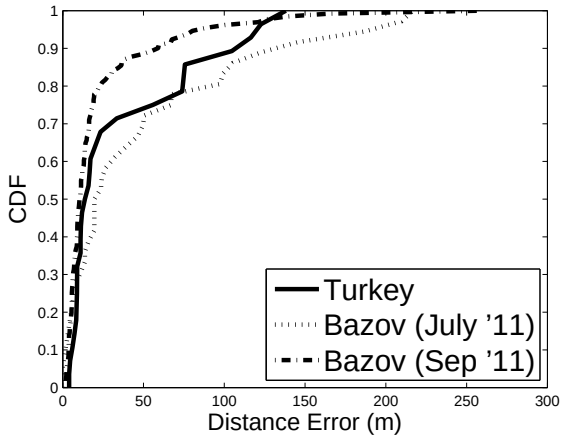
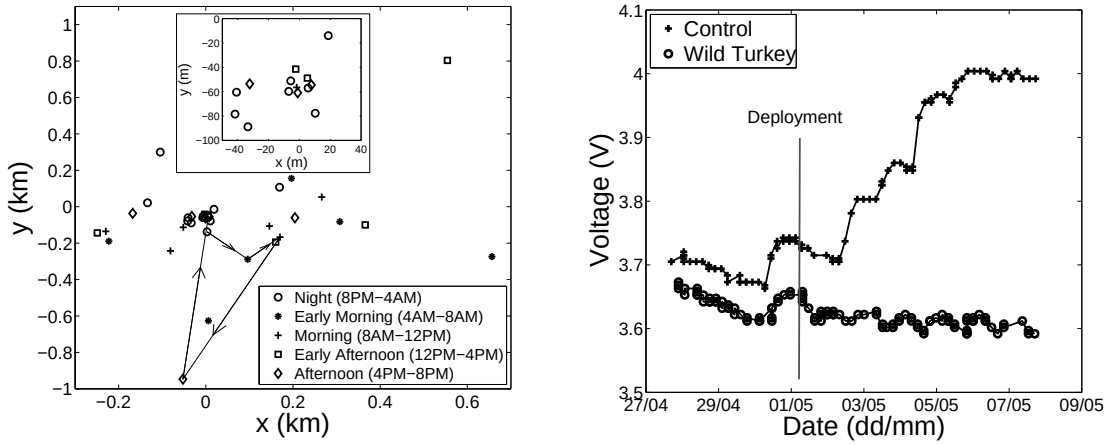


Figure 6.7: CDF of GPS error.

related data were collected over a period of 10 days [11]. In this deployment effort and at the specified duty cycle, it was found the system was capable of 100% reliability. However, in 9 cases, the SMS message was not received in the first communication window, but was capable of sending the data at the next communication opportunity.

To evaluate the performance of the GPS device, the known location is compared to the location acquired by the device. The result of this comparison is the GPS error. The error for this deployment is shown in Figure 6.4. The high error found during this evaluation, is due to the operational mode of the GPS device. During the deployment, the GPS device is deactivated after acquiring a single *first* fix. This behavior was enforced to improve energy consumption since the GPS would be active for a limited time. However, this characteristic causes the GPS to acquire a fix in the worst-case operational settings. Using the first-fix as the measure of position, leads to the worst errors since the unit will never have time to converge to the lower error bounds [11]. It is unable to converge to the lower error-bounds, since the initial position uses the minimal number of satellites to determine the position [66]. As more position estimates are computed and more satellites are visible, the accuracy increases.



(a) Turkey positions (lines connect consecutive fixes for one day).

(b) Battery voltage of control and turkey tracker.

Figure 6.8: The turkey positions (a), and battery voltage of control and turkey tracker.

This duty cycle allowed the turkey to be successfully monitored in a region of 2km x 1km using the cellular communications device. To measure the behavior monitoring capabilities of the system, the location data was divided into sets based on time of day (night, early morning, morning, early afternoon, and afternoon) as depicted in Figure 6.8(a). The results show that the birds congregated to roosts during night, early morning, and afternoon. It can also be observed that when the birds were more mobile than suspected in foraging periods, early morning and afternoon where the bird was captured moving as far as 1km over a 4 hour period [11].

The other purpose of the deployment was to evaluate the recharge performance of the system in a wild setting. In Figure 6.8(b), the recharge performance is visualized. In this figure, the deployment start is indicated with a vertical line. From this data, it is clear that the control device was out in the open, since the battery voltage was able to reach max capacity over time (4.2V). Although the device maintained operation, the turkey's recharge behavior did not increase as observed with the control tracker. The likely cause for the poor performance was due to the woodland coverage where the turkeys exist

to evade predators. In addition, from Figure 6.8(a), it is evident that the bird was in woodland coverage during peak sun exposure times. The data collected from the Wild Turkey deployment provide insight on how the system could function during a crane deployment [11]. For example, the control data is assumed to be similar to the recharge characteristics of flight, where the tracker is exposed to direct sunlight while the bird is in flight. The turkey data should represent similar recharge characteristics while the crane is on the ground [11].

In addition, the data collected by the turkey tracker provided insight on the behavior of the wild turkey. For example, it is known that wild turkeys spend up to 95% time feeding and are capable of moving at a rate of 200 m/hr while feeding [30]. The data collected by this deployment support this conjecture. More specifically, the GPS data collected while the bird was likely feeding (Figure 6.8(a)), showed movements of up to 1 km over a 4 hour period at a rate close to the 200 m/hr observations [30]. The ability to observe and confirm the behaviors, validate the usefulness of the developed system for wildlife tracking [11].

6.5 Captive Siberian Crane Deployment

From data collected in the Wild Turkey deployments, it was obvious that the tracking device could be used for cellular tracking of wild animals. However, the motivation behind utilizing the tracking device goes beyond position tracking and more into analyzing the *behavior* they are exhibiting. This is almost as, or in some cases more important to the ecologists. With the communication capabilities of the tracker coupled with the compass sensor, it was thought these behaviors could be identified. To confirm these assumptions, it was necessary to utilize the device in a controlled setting. In addition to exploring the behavior identification capabilities, the evaluation of the GPS driver was needed to de-

termine if GPS accuracy could be improved. Lastly, evaluation of a waterproof cloth enclosure was required to determine if it would adequately protect the device before deployment in a migratory setting. To assess these described aspects, deployment of the device on captive Siberian Cranes was carried out [19].

In July 2011, at the International Crane Foundation, in Baraboo, Wisconsin, a deployment to examine the behavioral sensing capabilities, communication, and waterproof cloth performance were carried out. In addition to testing the performance of the proposed system, the ecologists utilized this deployment to examine a new backpack mounting method not traditionally utilized in prior crane tracking efforts. In the captive crane deployment, the tracking device was placed on three captive Siberian Cranes: A. Wright, Bazov, and Hagrid. In this deployment, the birds were located in separate 10m x 10m pins. The devices on A. Wright and Bazov were purposed to evaluate the GPS, GSM and enclosure performance. The objective of the Hagrid deployment was to understand the behavior identification and evaluate the short-range communication capabilities of the device [11].

The tracking device was programmed with a 4 hour duty cycle to monitor A. Wright and Bazov. At each cycle, the position was sampled followed by compass sampling. Immediately after sampling the sensors, the software attempted to offload the data utilizing the GSM device [11].

As depicted in Figure 6.4, after 27 hours of operation and 135 acquisition attempts, the device attached to Bazov had achieved a GPS fix ratio of 97% [11]. Assuming the bird was located at the center of the cage, the results show that 58% of the GPS error was less than 25m. The high error observed was assumed to be caused by the sampling methods of the GPS driver software. In efforts to conserve energy, the GPS software uses the first acquired position as the location. Unfortunately, this position usually features the highest errors. To improve the error observed, the driver software was modified to collect 10

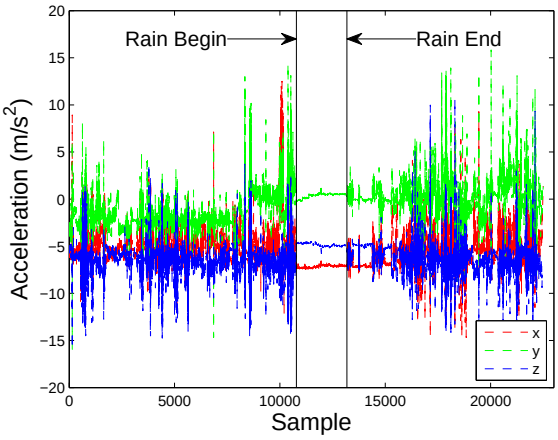
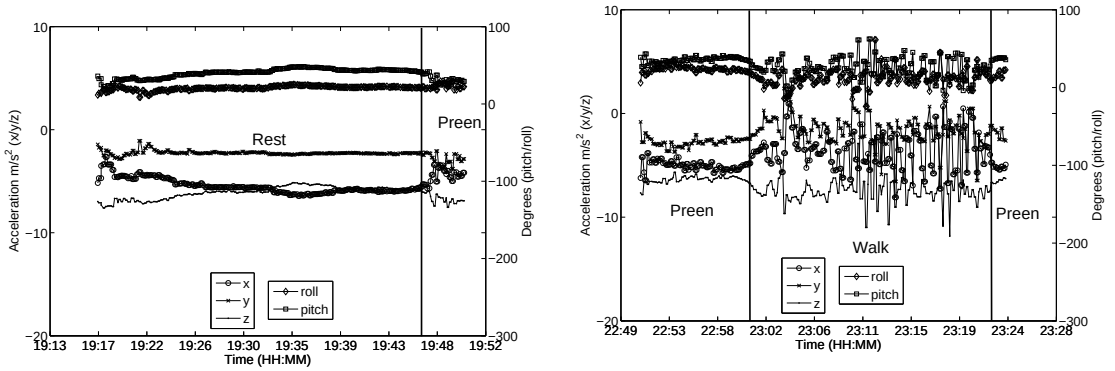


Figure 6.9: Three-axis acceleration readings from a Siberian Crane.

samples before signaling a successful acquisition event. This software was re-deployed in September 2011 on Bazov in the same pen. The Figure 6.4 illustrates improvement the software update made on GPS accuracy when compared to the previous deployments. The accuracy of the GPS position was improved so that 81% of the errors are less than 25m [11]. It is also important to note that a large portion of the errors came from the bird being mobile inside the pen ($\pm$7m). The other errors were produced by environmental issues and not operating the additional duration to save energy. The new results and quality of the data satisfied the needs of the ecologists [11].

The controlled captive Siberian Crane deployment also provided insight on performance of the waterproof pouch enclosure. During the deployment, the pouch successfully remained attached to the crane and prevented water from breaching the unit. However, A. Wright immediately had incapacitated the tracking device after deployment, by ramming the device into the cage [11]. This resulted in no data being collected. This effort motivated alternative approaches to enclose the tracking device as discussed in Section 6.2.

The device attached to Hagrid was used to exploit the behavioral sensing capabilities



(a) Behavioral movements rest to preen.

(b) Behavioral movements preen to walk to preen.

Figure 6.10: Illustrations of behavioral movements.

of the platform. This was accomplished by collecting solid-state compass readings at a high sampling rate of 10 Hz for 30 seconds every 3 minutes. The compass readings consisted of acceleration and rotation in three dimensions. The device utilized the short-range communication (802.15.4) capabilities to off-load data to a near base-station that was equipped with a high-gain antenna. During the deployment, the bird was also observed with a closed-circuit camera. The readings received by the base-station, were time-stamped so they could be synchronized with the camera recording. The behaviors observed through the camera feed were verified by an ecologist [19]. During initial observations, the behavior of the bird could be determined by one researcher and then confirmed by the other watching the video, solely utilizing the real-time values. As depicted in Figure 6.9, the behavior of the bird before and after a heavy rain can be identified [11].

The data collected over 4 hours during the Hagrid deployment consisted of 21,882 records. During post processing, the behavioral changes were annotated at data points as occurred over time. This method was valid because the video was synchronized in accordance with the recorded sample time-stamps. The result of this is plotted over

time in Figures 6.10(a) and 6.10(b), where the rotation in degrees is on the right y-axis and acceleration in three-dimensions in m/s^2 is on the left y-axis. The burst shown in Figure 6.10(a), shows the crane initially at a resting behavior. Then at time 19 : 47, Hagrid moves to a preening behavior. In the other burst shown in Figure 6.10(b), the crane starts in a preening behavior. At time 23 : 00 the crane starts to walk around the cage, abruptly at 23 : 22, the bird returns to a preening behavior. This provides evidence that distinct behaviors, beyond position, can be observed utilizing the developed tracking system [19]. This data also satisfies the needs of the ecologists. The ecologists intend to use this capability to model the energetic behavior of the cranes in the future. In my efforts, this work motivates use of the new sensing capabilities to conserve energy exhausted during communication, by reducing the information transmitted by the device [11].

There were many lessons learned during this deployment. One lesson that stands out is how the system performs when deployed at different locations. During the deployment at ICF, the orientation readings were inconsistent when compared to locations in previous evaluations. The cause for the inconsistency was due to the unusual magnetic field from a moraine located underneath the compound. This moraine influence was confirmed through the use of a smart phone where the inconsistency was observed as well. Fortunately, this only affected the orientation since the magnetometer, that utilizes the magnetic field to determine orientation [11].

6.6 Sandhill Crane Deployment

The success of the prior deployment efforts on Wild Turkeys and captive Siberian Cranes, motivate a capstone deployment of the system in a migratory setting. In this deployment, the device is deployed on five Sandhill Cranes. In this effort, the cranes are captured, assessed for ecological purposes, tagged with the device, tagged with colored markers,

Figure 6.11: A photograph of myself trained to assist with health assessment and tagging process.

and released into the wild. In ecological efforts, the deployment is used to evaluate the performance of the backpack mounted approach. In technological efforts, the deployment will assist with evaluating the enclosure, sensing, communication, energy, and monitoring capabilities of the system. The end result of this effort will be used to justify use of the system for monitoring wild migratory species at a continental scale.

In this deployment effort, the tracking device is mounted on five Sandhill Cranes. To identify the tagged cranes, they are designated with two letter prefix identities corresponding to where they are caught. The two-letter prefix is appended with the crane's gender: male, female, and chick. In the capture efforts, the Sandhill cranes are captured in the wild by trained professionals. This process was first accomplished utilizing a wire trapping method, where JB-Male was successfully caught. The wire-trapping process proved difficult and led to changing to a simpler narcotic-baiting method that enabled the ecologists to safely catch the birds. Once the cranes were narcotized, the professionals were able to easily walk up and capture the bird in a net without harm. Once the birds were captured, they were then ecologically evaluated. As illustrated in Figure 6.11, I was trained to assist with taking blood samples, rehydration of the bird, and mounting of the tracking device. Considering my technological background, this was an unforget-

Name	(days) Duration	# SMS	# Fixes	(km) Distance Traveled	Comments
JB-Male	13	0	0	-	Failed to recharge
JB-Female	41	2,795	208	20	Unknown
SH-Female	29	525	468	4.3	Fell off
SH-Chick	66	1,890	330	1,725	Fell Off
BB-Female	309	7,066	1124	603	Operational

Table 6.1: Deployment Summary

table experience. I believe this experience gives me an additional insight on the influence of my work in migratory bird tracking efforts. During this assessment, the bird has a blood sample taken, is rehydrated with intravenous therapy (IV), tagged with colored bands, and finally the tracking device is then affixed (Figure 6.2). The colored bands are used to identify the bird in case the tracking unit were to fall-off. Each of the deployed tracking devices are intended to monitor the complete migration over the course of a year [11].

For the deployment efforts, the software is scheduled to duty cycle every 4 hours and 5 minutes. During each cycle, the device wakes from sleep mode, acquires a GPS fix, samples 10 compass readings over at a 10 second interval, and attempts to establish communication with the short-range (802.15.4) and long-range (GSM) mediums. In this deployment, the GPS fix utilizes the 10 sample threshold method found to increase accuracy (Section 6.5). The long-range medium is only utilized if the short-range fails or all records are not sent. The 4 hour and 5 minute duty cycle was used to allow the unit to drift and observe an entire day over a long period of time [11].

The Sandhill Crane deployment offers significant insight on the performance of the system for its intended use. The summary of this deployment is shown Table 6.1. The success of the system is measured by looking at the performance of the enclosure and backpack mounted methods. The success of the system is also measured by the sensing and communication performance. Furthermore, the recharging and recovery behavior

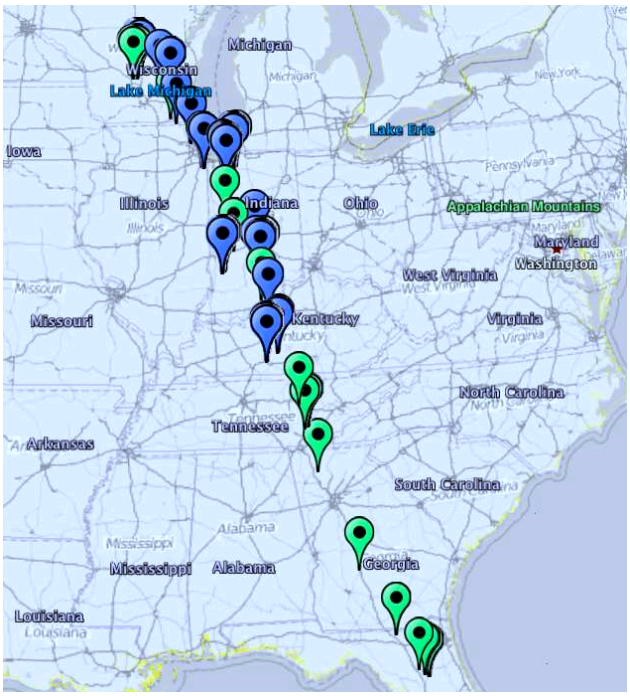


Figure 6.12: Migration paths of SH-Chick (white marker) and BB-Female (black marker)

of the system is examined to assess the longevity of the system. Lastly, gauging the observable behaviors in a deployment of this scale, the usefulness of the system can be determined. As depicted in Figure 6.12, the system was capable of monitoring the complete migration of two Sandhill Cranes. The migration that extended from Wisconsin to Florida was executed by the SH-Chick. The migration from Wisconsin to Indiana and back was conducted by BB-Female during the Fall of 2011. Although fascinating, this data suggests that the birds are extremely unpredictable and cover extremely long distances. Therefore, the assumption that determining static base station locations would be difficult and expensive, is justified by this behavior found in the data collected from the two migratory birds [11].

For the Sandhill Crane deployments, the enclosure was improved based on the captive Siberian deployment results, as discussed in Section 6.2. As shown in Table 6.1, only two of the tracking devices successfully completed migrations. The enclosure and back-

pack performances are the likely causes of the failures. For instance, immediately after JB-Male was released in July-August of 2011, the GPS was unable to acquire a fix and recharge. Over a course of 13 days the back-end was able to receive all of the transmitted GSM packets, but none of the packets contained valid GPS positions. In addition, the battery voltage showed no evidence of recharging. It was assumed from this collected data, that the compression of the enclosure process caused the terminals on the solar panel to become disconnected and possibly block the GPS from acquiring a position. The update to the enclosure process, as described in Section 6.2, assumed to have fixed these problems since recharging and position acquisition since the error was no longer found in data received from the devices.

The deployment also assisted with evaluation of the back-pack mounted approach. Since the backpack method was novel in crane tracking, it was not exactly known how to attach the harness to the cranes. The Sandhill Crane deployment also served as a trial for the ecologists to perfect the backpack mounting process for future efforts. The rest of the failures of the system appeared to be caused by the tracking unit falling off the bird not by the enclosure. This was confirmed after the units would no longer respond, but the birds were spotted (identified by colored bands) without backpacks. The final method utilized by the ecologists appear to be the best method since one device has been successfully operational for 10 months without failing or falling off.

The most pivotal result taken from the Sandhill Crane deployment is the connectivity performance found during the deployment. In the collected results, it was confirmed that the cellular network was capable of covering enough area to track a migratory species at the continental scale. SH-Chick, the crane that migrated from Wisconsin to Florida, recorded a total of 330 total GPS acquisitions. In addition, the speed from this data verifies that the bird was actually flying while 12 of the samples were acquired. More importantly, 10 of these flying acquisitions were actually transmitted during flight. This

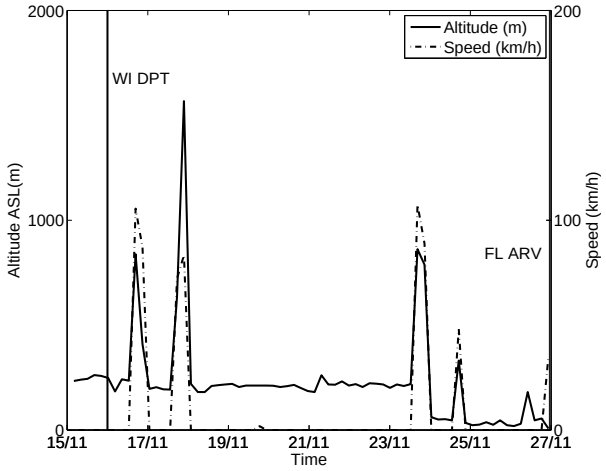
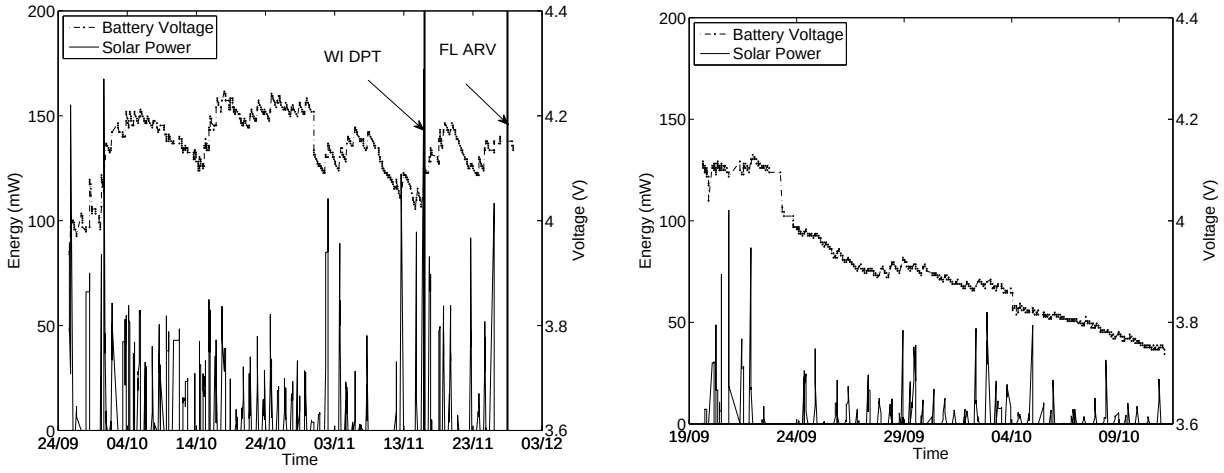


Figure 6.13: SH-Chick altitude and speed.

data confirms that the device can maintain connectivity in all behaviors exhibited by the crane [11].

Most of the birds' migration time is not spent in flight. In Figure 6.13 the altitude and speed during the migration from Wisconsin to Florida is shown. As assumed, this displays a high correlation between speed and altitude. The high altitudes and speed discovered by this data reinforce the difficulty of monitoring cranes, since they are extremely mobile [11].

The evaluation of energy characteristics are necessary since it is unknown how the system will behave in a migratory setting. In the Sandhill deployment, the system was able to observe a correlation between the solar power and the battery recharge rate. In Figures 6.14(a) and 6.14(b), the voltage (right y-axis) and the energy (left y-axis) is plotted over time. From this observation, depicted in Figures 6.14(a) and 6.14(b), it is obvious that power levels grater than 20mW led to battery recharge. If the system was under this level, the battery was unable to recharge. This was caused by the behavior of the recharge circuitry, since it is incapable of charging unless this power level is reached.



(a) Solar power and battery voltage for SH-Chick

(b) Solar power and battery voltage for JB-Female

Figure 6.14: Solar power and battery voltage for SH-Chick and JB-Female

These results motivate utilizing this information on the device to modify operational behavior. Furthermore, this data motivates possibly improving the recharge circuitry, so that it is capable of harvesting energy below the 20mW levels [11].

The distinct differences in energy between JB-Female (Figure 6.14(b)) and SH-Chick (Figure 6.14(a)) is assumed to be caused by either the physical location of the bird or how the device is mounted on the bird. While in the breeding grounds, the birds forage in heavy vegetation where the solar panel may not have adequate sun exposure. Alternatively, the different mounting approaches may have caused the solar panel to be covered by the wings while the bird is not flying. These speculations are backed by the fact that JB-Female featured a lower recharge rate when compared to JB-Chick. It is known each tracking device was mounted on the bird differently, JB-Female being mounted lower causing the wings of the bird to cover the tracker in a non-flying position. Furthermore, JB-Female spent significantly more time in the breeding grounds than JB-Chick. This may have caused a reduced recharge rate, since the bird would exist in covered habit.

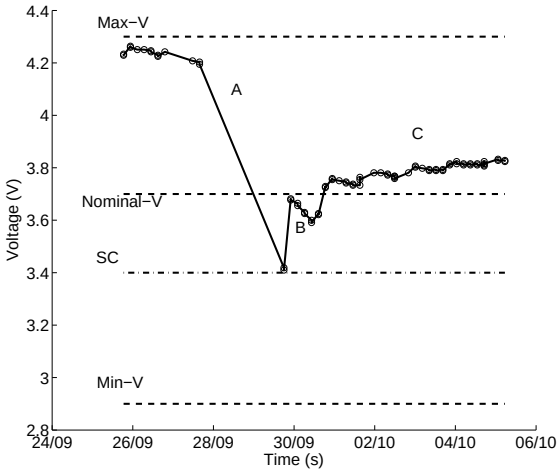


Figure 6.15: Low-voltage recovery.

The data collected during migration supports either case, since JB-Female was able to recharge in little time (days) during the migration. [11].

The battery protection and life-time enhancing mechanisms employed by the system were evaluated through the Sandhill Crane deployments. An example of where the recovery mechanisms are activated by SH-Female is depicted in Figure 6.15. The data collected show that these mechanisms were necessary to accomplish the 10 month run-time. The collected data show many cases where the system restarts due to energy depletion. The data also provides awareness on how to model operation in future deployments. During this recovery phase, the device is near full capacity at time *A* but fails to send due to an unknown error. The unknown error forces the device to deplete its available resources. Fortunately, the recovery mechanisms of the platform allow the system to recharge so the battery has enough energy to retransmit at a later time *B*. Following this recovery, the battery finally stabilizes at time *C*. Thus, this behavior is ideal and validates the usefulness of the protection circuit towards extending life-time of the system [11]. The failure found was suggested to be caused by a GPS failure, since no

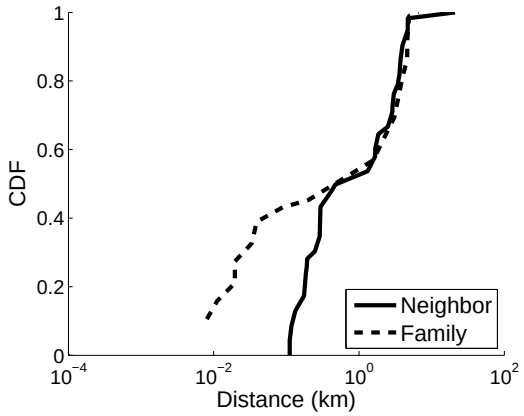
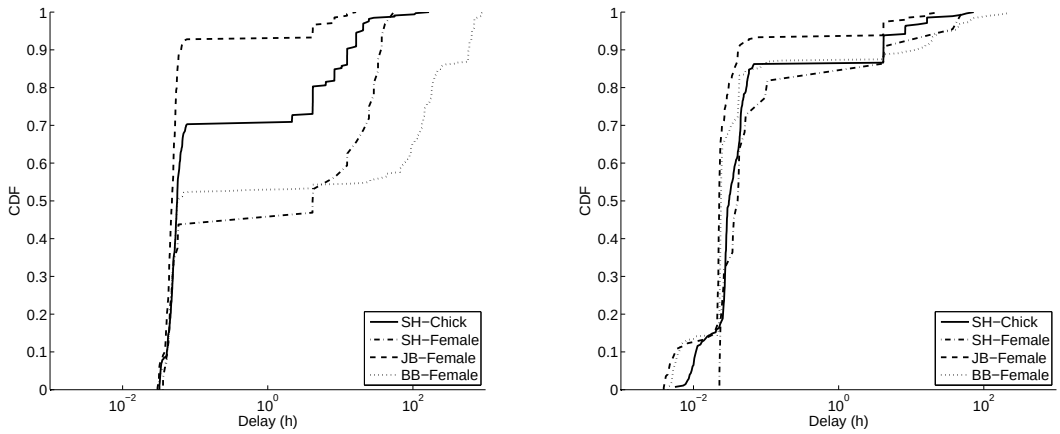


Figure 6.16: Distance between SH-Chick and family and neighbors.

position acquisitions occur after this unknown event. Although the GPS failed, localization was still available utilizing the cellular information in the SMS packets. It was later found that backpack harness errors caused the device to fall off, since the SH-Female was spotted without a backpack.

The Sandhill Crane deployment also assists with motivating future enhancements to the system. In Figure 6.16, the CDFs of distance between SH-Chick and SH-Female, and SH-Chick and JB-Male are plotted. These results show a close proximity between family members that motivate adding ad-hoc communication to the future software capabilities. Adding ad-hoc communication may enable improved data collection techniques, observation of social behaviors, recovery from device failures, and improvement of device lifetime through coordinated efforts [11].

From the Sandhill Crane deployment, performance of the network is also evaluated by observing the behavior of the network delay and queue sizes. The delay, in this evaluation, is considered to be the time from the sample acquisition to the time it is received by the back-end. The CDF of delay for JB-Female, SH-Chick, SH-Female, and BB-Female is shown in Figure 6.17. In these results, JB-Female and SH-Chick were able to transmit 98% of the readings in less than 24 hours. This is not the case for



(a) CDF of delay (comm.+queue)

(b) CDF of communication delay

Figure 6.17: CDF of communication delay.

Sh-Female and BB-Female where 72% and 55% of the data are transmitted in less than 24 hours, respectively. The cause of this delay is due to the queue build up in flash memory as a result of the FIFO queuing process implemented in software. These results motivate utilization of a LIFO queueing method to improve network performance [11]. In Figure 6.17(b), the delay due to missed communication opportunities are shown. The impact of utilizing a LIFO can be seen through this figure, such that more than 94% of the delay is less than 24 hours. This performance is greater than that observed with previous satellite tracking efforts (48 hours [108]), thus motivating the use of LIFO for future deployments. More importantly, the result of this data satisfies the requirements of the ecologists [11].

The Sandhill Crane deployment also illustrates the systems usefulness for ecological purposes. As depicted in Figures 6.18 and 6.19, the positions collected are plotted over 24 days. In addition, the markers are differentiated to signify different parts of the day. By differentiating the observed locations over time, it is obvious that the Sandhill Cranes spend a great deal of time at the roosting location during most of the afternoon and

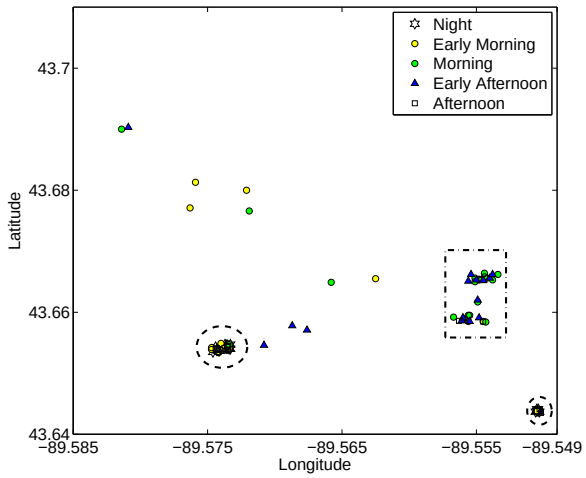


Figure 6.18: GPS locations of JB-Female over 24 days total overall fixes.

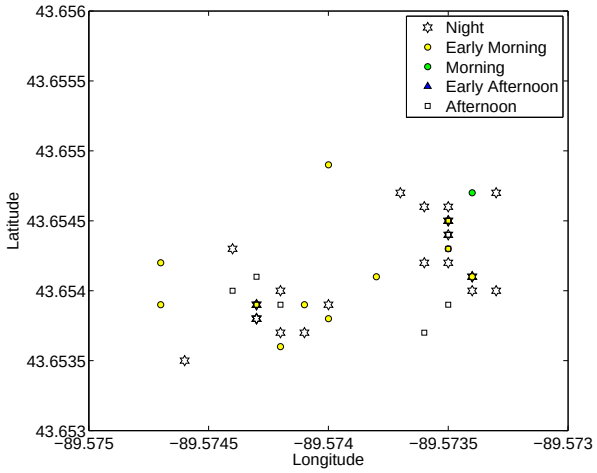


Figure 6.19: GPS locations of JB-Female over 24 days viewed as a zoomed circle cluster.

night. The observed roosting locations are known by the ecologists, thus validating the monitoring capability of the system. The rectangle also depicts the breeding territory where the birds exist during the early morning and afternoon. This territory is used for foraging and preening activities [11].

Although confirmed in the captive Siberian Crane deployment, the Sandhill Crane

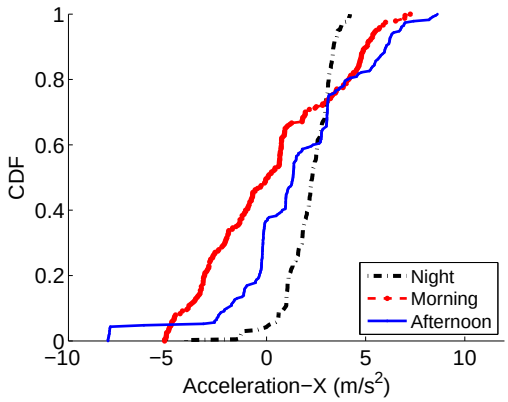


Figure 6.20: Acceleration for SH-Chick for x-axis.

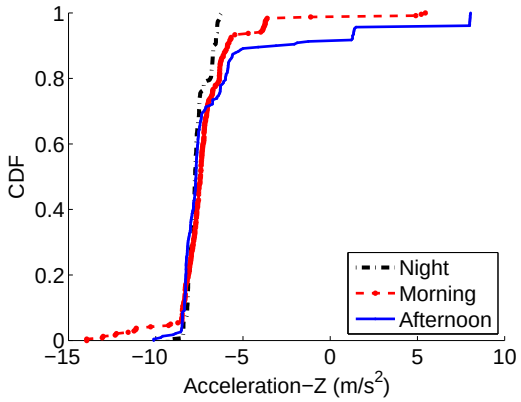


Figure 6.21: Acceleration for SH-Chick for z-axis.

deployment shows the capabilities of the tracker for determining more granular details about the birds. The general behaviors can be captured utilizing the compass readings as shown in Figures 6.20 and 6.21. The figures show the acceleration of the birds in respective z and x axes. The acceleration in these axes can show the posture (x-axis points towards the tail as illustrated in Figure 6.2) and movement of the bird over time. In the case of data collected from the z axis (Figure 6.21), the bird is immobile at night thus the observed mean is close to $9.8m/s^2$ with minimal variance. Conversely, the mean of $\sim 0m/s^2$, with a high variance and positive values suggest that the bird is flying in alternative parts of the day [11].

As seen with the z axis (Figure 6.21), the x axis (Figure 6.20) captures additional interesting behaviors for ecological purposes. More specifically, the posture of the bird can be used to determine the likely activity over time. For instance, the steep slope observed during the night falls greater than 0, suggesting that the bird was immobile. This is implication is further backed in Figure 6.18, where the birds are found at roosting locations in the morning and evening. As expected, this is not the case when the birds are feeding. In other words, the negative values propose that the bird is feeding, where the bird has its head oriented towards the ground. This further confirms the improved monitoring capabilities brought forth by the tracking platform [11].

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this work, the multi-modal mobile and associated tools for migratory bird tracking is presented. The endangered status of the Whooping Crane and unique challenges found in migratory bird tracking, motivate the use of mobile Wireless Sensor Networks in tracking efforts. The low population densities and extreme mobility exhibited by the endangered migratory birds, make prior mobile WSN wildlife tracking systems inapplicable towards conservation efforts. Instead, it is proposed and proven through deployments that the use of cellular sensor networks is a cost-effective alternative to prior systems. It can also be observed through the deployments, that the developed system can outperform prior tracking systems, in terms of sensing and communication delay capabilities of the developed tracking system. More importantly, the sensing capabilities of the system facilitate the ecologists with ability to observe more detailed behaviors applicable towards recent conservation efforts.

7.2 Contributions

The contributions showcased in this work are as follows.

Novel Tracking Platform: The idea to exploit the cellular network infrastructure in effort to increase connectivity of mobile WSNs is proposed and a prototype is developed. The prototype is equipped with additional sensing capabilities to facilitate observation of aspects not previously observable by trackers in prior Whooping Crane tracking efforts. The usefulness of the platform is evaluated through deployments. The advantageous capabilities displayed by the platform, stimulate its use in precision agriculture research [103] and student projects at the university.

Embedded / Back-End Software: The developed platform is made useful by the design and development of embedded and back-end software. The embedded software is heavily tested and utilized for deployment efforts. The usefulness of the software is exhibited by satisfying the requirements discussed in Chapter 2, in the deployments.

The back-end software bring more robust service oriented architecture principles to mobile Wireless Sensor Networks for network and data management purposes. The back-end is shown capable of satisfying the requirements described in Chapter 2. It is shown capable by managing the network and data of the demonstration tools, and through the deployments.

Simulation Tools for Performance and Testing Analysis: In the early stages of this work, simulators are developed to assist with analysis and testing of the tracking efforts before hardware was available. In addition, a novel method for integration of application, networking, and environment simulators is proposed. The GPS simulation assisted with providing GPS activation time used in deployments. It is shown that the fix rate during the deployment, out performed than what was initially thought in the GPS simulation (50% simulation vs 97% deployment). Furthermore, the proximity found during the de-

ployments had similar characteristics found during simulation. In the crane simulation, it was observed that the birds are more likely to encounter another bird while existing in the breeding and wintering grounds. In the deployment, it is also confirmed that it is more likely to communicate while in the breeding grounds, since it is observed that family members and neighbors are in enough proximity to communicate with short-range radios (capable of 400m communication distance).

Deployments: The limitations brought on by tracking the endangered Whooping Crane, led to deployments on proxy species. The deployments were necessary to illustrate the tracking system's ability to satisfy requirements and exhibit capabilities before use on the target species. Each deployment shared a similar characteristic with the Whooping Crane. Thus, the results from the deployments brought relevant feedback to the tracking system to verify requirements can be met, to show the resulting system is sufficient for tracking efforts. Each deployment displayed very interesting results that assisted justifying its use for migratory bird tracking.

7.3 Future Work

Although the capabilities of the system are significant, the maximum potential of the system has yet to be reached. Numerous enhancements and research is desired in the near future. The following discusses the areas aspired to be improved.

Back-end Service Improvements: The difficulties found during development motivate adding a logging service to the back-end, for handling system logging across the networks. In addition, it is desired to improve the data manipulation capabilities of the back-end, by adding a experiment service. The logging service will standardize how logging is handled across the network. In addition, the experiment service will enable researchers to virtually annotate data for manipulation of the data as desired. The exper-

iment service will allow researchers to organize data, note observations, add predicates to fire triggers (user notification on an event), view trends, and verify conjectures. The experiments will not be limited and allow ecologists to look at different aspects of the same data set. Furthermore, it is also desired to evaluate these services in alternative deployments.

Run-time Reprogramming: The difficulty of recapturing the birds and the current communication and lifetime capabilities exhibited during deployments, motivate reprogramming of the device. In this effort, it is intended for the tracking system to interpret instructions over the air so it can be reconfigured as much as possible without deployment of additional units. With this ability, the ecologists will be able to tailor data collection to focus on their current research. To accomplish run-time reprogramming is challenging due to hardware resource issues (i.e memory, processing) and communication problems (connectivity, bandwidth). In addition, the faulty nature of the devices in harsh environments perpetrates reprogramming a dangerous task.

Run-time Behavior Identification: The sensing capabilities of the system exhibited by the captive Siberian Crane deployment [19], motivate detecting the behaviors online. This will add numerous capabilities to the tracking system. Some of these capabilities include energy saving, efficient communication, life-time improvements, and push notifications when certain predicates are met during run-time. The behavioral data, correlated with visual observations, will act as a ground-truth for the detection capabilities. The ground-truth will also be improved by future deployments and field observations.

Ad-hoc Communication Capabilities: The ad-hoc communication capabilities of the system have yet to be explored during the birds' migrations. With ad-hoc communication potential, it is desired to study the social behavior of the birds and networking dynamics during migration. In addition, it is desired to explore energy saving and fault tolerant techniques that utilize a combination of ad-hoc and cellular networking capacities. To

accomplish ad-hoc capabilities, it is necessary to look into lower frequency low-power radios to increase communication distance. This will increase chances of communicating with family and neighboring birds. Furthermore, research and development of an adaptive and efficient communication protocols are necessary for communication with neighboring nodes.

Alternative Deployments: The success of this work in Sandhill Crane tracking efforts, motivate its use in alternative research. Effort has already been made to use the full system, and sub components in tracking wild pheasants, cows and foxes. The system has already been deployed in agricultural research efforts [103].

Appendix A

Testing

The Whooping Crane is a highly mobile species, and more importantly endangered. This makes deploying test versions of the system nearly impossible. Moreover, intermittent testing is impossible because the birds are difficult to capture and minimal interaction with the birds is desired. In addition, the recapture of these birds after deployment is impractical making it difficult to extract state of the devices. Unfortunately, these issues lead to slow development cycles and results for the ecologists. To test the tracking platform, existing approaches are brought to WSNs to assist with development efforts [9].

The device limitations of the embedded system also add to difficulties of testing. The devices used in this application have limited energy resources, limited storage, limited computational power, and are naturally error prone [9]. Therefore, it is difficult to continuously monitor, and save error states. Moreover, limited computational power limits the instrumentation on embedded devices. In this chapter, a unified and non-intrusive instrumentation framework is developed in effort to improve the current capabilities of WSN testing. In addition, runtime-verification techniques are developed and studied to examine their applicability in development efforts.

The unexpected failures not foreseen by the testing specifications append to the set

of issues. These problems are a result of misunderstanding of how components actually work, not adding enough successful tests, and not developing failure cases. To improve testing capabilities, a robust set of tools are brought together in a testing framework. This framework features better assertion library, improved instrumentation, an improved unit testing, and runtime verification techniques that are currently unavailable for TinyOS programs.

The chapter is organized as follows. In Section A.1, the properties necessary to identify, the defects to classify and faults observed are discussed. Then in Section A.3, the developed testing framework is described. In this section the available assertions, unified instrumentation framework, continuous unit testing, runtime verification techniques, and system testing capabilities are covered. Finally, in Section A.4 the analysis and validation results of the testing capabilities are explored.

A.1 Properties to Monitor and Defect Classification of Mobile WSNs

To successfully test a complex system for monitoring migratory birds, it is critical to correctly identify properties of the system that can fail, accompanied by a correct classification of the defects found to occur. Currently, the types of defects seen in a system as described in this thesis, are loosely defined. From research and development experience, this section looks to organize the types of properties and defects that can (and will) occur during the development process.

The types of properties to observe in a system such as this fall into indeterministic behavior, I/O arbitration, memory, processing, data, and energy. In some cases these properties can even be considered cross-domain properties. In addition to identifying

system properties, this work looks at dividing defects into areas prioritized by severity [9].

Indeterministic Behavior: System properties are definable behavior of the system. In the crane tracking work, an example of this is acquiring during a GPS fix acquisition. It is *never* guaranteed that the GPS sensor will ever acquire a fix, it is in-fact impossible to determine this. Therefore, it is important to observe control-flow in-order to verify the processor is never in a locked state [9].

I/O Arbitration: A pressing issue in systems such as the crane tracker include, arbitration of the input/output resources. For example, the crane tracker utilizes the SPI bus for flash and radio usage. If either of these components are enabled simultaneously, neither of them will function correctly. In verification, the system must be able to observe if and when these resources are utilized [9].

Memory Properties: The platform has limited program, static, and flash memory. It is easy to observe the amount of memory consumed for program and static memory. It is not important to observe dynamic memory in TinyOS, because TinyOS does not permit it. Unfortunately, this cannot be determined with flash memory since it changes dynamically. Physically, each of these memories have a limited number of write cycles that occur before they will no longer contain valid data. The system must be able to observe this to protect the longevity of the system [9].

Processing Properties: The computational power of the system is extremely limited. The microprocessor utilized for crane tracking operates around 8MHz. Therefore, performing tasks such as serializing data to XML for the back-end is not sensible. For these described reasons, the system must be capable of monitoring these hardware properties of the system to ensure users do not utilize the system improperly, resulting in faulty behavior.

Data Collection Properties: Monitoring data properties is pivotal for reason that data

collection is the ecologist's initial motivation behind utilizing the system. The quality of data collected solely influences the decisions made by the ecologists, which may cause the ecologists to inaccurately report the state of the wildlife. The properties to monitor with respect to data quality, can be broken into the following areas: clarity, accessibility, consistency, relevance, timeliness, completeness, and accuracy [109]. The importance of monitoring the data has already been observed during a development case. More specifically, shortly after deployment, it was observed that the parsed NMEA 0183 date from the GPS device was corrupted. The reason for this was unknown. Fortunately, the error was consistent across devices, so a manual solution recovery solution is found. However, the time required to fix the date influences the timeliness at which the data is available to ecologists. If this property was observed beforehand, this problem could have been prevented. Thus justifying the importance of identifying data properties to help monitor the quality.

Energy Properties: The last property to observe of the system is the energy property. It is important to observe energy because the target devices operate utilizing a battery. Even with a solar panel to replenish resources, it can not be determined if the system will continue to operate after the battery depletes. Therefore, it is important to monitor the charging properties after deployment. With the ability to monitor energy, fault prevention can be utilized in code. Monitoring the voltage from the battery can also ensure devices are not enabled when the system experiences inadequate energy supply [9].

A.1.1 Defect Classification of Mobile WSNs

In addition to properties is important to be able classify defects. This classification can be used to prioritize areas of the system that need to be better tested or improved upon in order to meet requirements. Defects of the system can be categorized into arithmetic and

logical, syntactical, resource and performance, timing, and semantic defects [107]. The defects in each category can then be prioritized by minor, major and fatal defects. Minor defects imply that the faults will not prevent the system from fulfilling requirements. Major defects imply that some of the functionality is missing and some requirements may not be met. Fatal defects imply that the system will not function or meet any of the requirements with an occurrence of this defect.

A.2 Background

Several approaches have been taken to develop, test and validate the correct operation of embedded systems and wireless sensor networks. In the following, existing approaches are discussed.

In the state-of-art unit testing for TinyOS there are two main entities, TUnit [100] and MUnit [80]. TUnit is a unit testing framework that brings modern unit testing philosophies to TinyOS. TUnit uses a driving node to trigger unit tests. MUnit is an alternative approach, that utilizes remote procedure calls to upload new tests to remote nodes. The results of these tests are collected, and together to verify state of the network as a whole.

TUnit is complex, requires a complete different program per test, and requires the tests to be implemented in TinyOS. Furthermore, it requires the user to define entire snapshots of TinyOS per test suite. This caveat can cause projects to easily bloat in size. In addition, TUnit requires a complete program to be defined, for each node implementing a test interface. This is undesirable when a testers intention is to test a ”unit” of code that is not necessarily different between nodes [80]. Moreover, TUnit requires the tests to be written in TinyOS. Doing this results in larger and more complex ”wiring” nightmares. This frowned upon when portable code is desired. Lastly, TUnit doesn’t

facilitate use of vital JTAG-ICE instruments, these instruments allow the developer to inspect and step through programs when string instrumentation fails.

MUnit takes a different approach by disseminating, loading, and execution of tests from a central machine through remote procedure calls. There are three issues with MUnit's approach. First, there is no support for multiple platforms. In addition, MUnit confuses unit testing from integration testing by testing functionality of code across multiple nodes. Lastly, like TUnit does not directly support JTAG-ICE debugging [80].

Both TUnit and MUnit require their testing motes to feature a radio, have guaranteed connectivity and have the default platform UART serial communication available for testing. Unfortunately, testing applications such as the crane tracker this is not possible. It is not possible because the tracker utilizes these serial components for alternative purposes, thus not limiting the usefulness of these tools. The testing framework proposed by this work, proposes an ”inline” unit testing method that does not, for the most part, require a new test harness for testing control. In addition, this method allows the programmer to determine how results are collected. And more importantly, the testing framework heavily supports testing with JTAG-ICE [26]. JTAG-ICE has proven extremely beneficial for catching bugs in TinyOS programs. However, for communication testing, the programmer is still required to invoke radio events externally. This is also performed by TUnit for testing radio event tests [100].

To go beyond unit testing and to verify program behavior of crane sensing application, log file analysis and aspect-oriented programming are used [9]. Log file analysis is performed off-line after the program has executed to determine faults. During execution, the program emits internal state. This process is completed with out interrupting program execution [8]. During analysis, the output is checked against a DFA representation of ”correct” execution. If the DFA fails to accept the output, it is determined that a fault has occurred.

The work performed by Andrews in his paper [8], shows that formal specifications can be effectively used to check program execution. However, the usefulness of this method is only as great as the formal specification. More specifically, determining if the ”correct specification” is actually correct. In similar work performed by Yang [113], the size of the log file can impede on the usefulness of this method as well [9, 10].

Aspect-oriented programming (AOP) is the concept of inserting program flow to observe complex cross-cutting concerns [46]. Cross-cutting concerns are properties of a system that are developed independently, and brought together to perform a specific task [46, 9]. The programmer uses expressions to define the properties to monitor (”aspects”) and in which instrumentation is injected into the functionality where these properties are observed (”weaving”) [9, 10].

A.3 Testing Framework

To facilitate testing of the proposed system, a testing framework is developed. The testing framework consists of an assertion library, unified instrumentation framework, unit testing, run-time verification tools, and system testing component as explained in the following.

A.3.1 Assertion Library

Assertions are essential for any programmer’s toolbox. Assertions enable programmers to define predicates in their programs that contend what they assume is ”correct” program execution. In TinyOS development, the bare minimum is available for testing [67], especially when compared to alternative resources in similar languages [29]. The prior assertion library brought forth by TUnit [100], lacks complex assertions such as memory comparisons and error reporting abilities. Therefore, an improved assertion library is

brought to TinyOS.

The assertion library, that is inspired by CUnit [29], is brought to TinyOS. CUnit could not be utilized due to the internal memory management of test cases, requiring facilities not supported by TinyOS. The implementation provided by the library is modified to suit the needs of embedded device development, where static memory allocation is necessary. It is also designed to be utilized with other parts of the testing framework. More specifically, it is intended to be used with the unit testing and runtime-verification components of the testing framework. The library is lightweight and uses compile time preprocessor macros to define assertions that can be used in programs. It also features a "fatal" behavior that permits the programmer to define what happens when defined fatal behavior occurs. By default, the fatal behavior is handled by entering an infinite loop and an LED notification. Upon notification, the programmer can connect JTAG-ICE [26] and inspect MCU state after fatal behavior occurs.

Similar to CUnit, the assertion library features essential assertion functionality. This includes standard "assert", true/false comparison, equality, pointer equality, null comparison, string comparisons, and memory comparisons. Finally, each assertion features a fatal version by simply adding a "_FATAL" suffix to the assertion.

A.3.2 Unified Instrumentation Framework

The unified instrumentation framework (UIF) brings together all instrumentation methods available to the embedded platform. The embedded device utilized by the crane tracking platform is capable of instrumentation through LEDs, serial communication, and JTAG communication. All of which can be reconfigured during run-time. Each method of instrumentation and debugging is used for special purposes. This subsection discusses each instrumentation capabilities brought forth.

To expand the expressiveness of instrumentation beyond switching LEDs, it is common to use serial communication to emit more verbose messages to the programmer. Serial communication permits ”printf” functionality during testing. Novel to TinyOS, the serial communication component of UIF facilitates instrumentation utilizing alternative UART ports, I2C, SPI, and the 802.15.4 radio. Previous instrumentation by these mediums are not available to TinyOS. A test bridge device is required to facilitate UIF serial instrumentation. The test bridge device is simply another embedded device that is interfaced by a free serial medium. This feature is important for testing the crane tracking platform, since both UART lines are coupled to the GSM and GPS devices, rendering the default serial method useless.

Although essential to any programmers toolbox, the overhead endured in program space and execution impact the usefulness of serial communication. Program space issues exist when MCUs have limited program and static memory available. In some cases, after compilation, the added instrumentation can yield a program that is unable to fit into micro-controller rendering the test program useless. This is seen in the analysis of runtime-verification, aspects are unable to be utilized on real hardware [9]. In addition, long execution durations from instrumentation can lead to indeterministic behavior. For example, if a programmer utilizes a interrupt service routine (ISR) to perform an action, with added instrumentation, it is possible for the programmer to miss a sequential interrupts because the instrumentation task took too long to complete. Lastly, serial communication instrumentation is incapable of observing micro-processor state without being ”weaved” into the program.

To avoid instrumentation that modifies program execution and to add improved observability during testing, UIF features JTAG debugging mechanisms. To reduce JTAG-ICE latency and missed signal issues, UIF advocates instrumentation of program traps and LED notifications of faults. A program-trap disables global interrupts on

the device (stopping the ISRs) and then enters a wait state controlled by a variable "GLOBAL_TRAP". The active LED notifies the programmer that a program-trap occurred, thus informing the programmer to inspect the MCU state with JTAG-ICE and GDB. After inspection, the programmer can set the "GLOBAL_TRAP" to false and then continue operation. This eliminates missing signal behaviors and slow stepping that occurs during testing by forcing the programmer to focus on writing assertions instead of "walking" through programs.

A.3.3 Unit Testing

Unit testing allows the programmer to single out small "units" of code and test functionality using assertions. In combination with the assertion library, the testing framework features a unit testing component, called EUnit. EUnit is intended to be a non-intrusive, and eliminate most testing harness mechanisms to execute the code. In this section, previous efforts will be discussed. Then, credit to the inspiration behind EUnit is given, including the differences between the libraries. Finally, the flow of the EUnit is explained.

As with the assertion library, EUnit is inspired by CUnit [29] because of features ideal for embedded testing including platform independent code and having lightweight properties. However, it is significantly different. First, CUnit utilizes a registry to maintain mappings of multiple test suites, and respective test cases. In doing so, the implementation relies on dynamic memory allocation. Unfortunately this is not available to TinyOS programs, forcing programmers to write programs with static memory. Secondly, CUnit returns results, controlled by the programmer, through an interactive console, and/or XML output. With EUnit, the results are not controlled by the programmer, rather the completion of test suites. It is the role of the programmer to invoke a function when a test case is complete. Moreover, EUnit does not utilize an interactive shell and is not

capable of emitting results in XML format for performance reasons. These two features were decided not beneficial or useful enough for embedded systems and therefore are not included.

With EUnit, it is the responsibility of the programmer to ensure the tests cases are invoked. For example, it is impossible for EUnit to invoke indeterministic code blocks such as those inside of events. To ensure tests are executed, the programmer must ”set off” the chain of events that will lead to the execution of the test case. In some sense, this forces code coverage, by making the programmer ”set-off” indeterministic code that reside in events to complete the test-suite. However, this could lead to test-suites that never actually complete so tests must be carefully observed.

The following programming example illustrates how to utilize the unit testing library:

```
#include "eunit/EUnit.h"
#include "avrtime.h"
#include "debug_constants.h"


module BlinkM {
  provides {
    interface StdControl;
  }
  uses {
    interface Timer;
    interface Leds;
  }
}
implementation {
  EU_TestSuite suite;
  EU_Test test1;
  EU_Test test2;
  void customHandler(EU_Test *test_ptr,
                     uint8_t expression,
                     uint32_t lineNum,
```

```
                      constChar_ptr condition,
                      constChar_ptr fileName,
                      constChar_ptr function,
                      uint8_t isFatal)
  {
    //handle the error
debug("TEST(\"%s\"):",(char*)test_ptr->name);
debug("%s:%s:%s:%u:",(char*)condition,(char*)fileName,(char*)function,lineNum);
//success
if (expression)
{
debug("SUCCESS");
}else//fail
{
debug("FAIL");
if(isFatal)
{
debug(" **FATAL**\n");
}
}
debug("\n\n");
  }

  struct avr_tm *ptr;
  avrtime_t time = 0;
  /**
   * Initialize the component.
   *
   * @return Always returns <code>SUCCESS</code>
   **/
  command result_t StdControl.init() {
//initialize the test suite
    init_eu_test_suite(&suite,"Blink Test Suite");
//set our custom handler
    EU_setCustomHandler(&suite,&customHandler);
//initialize the test cases
    init_eu_test_case(&test1,"TEST INITIAL TIME");
```

```
    init_eu_test_case(&test2,"TEST TIME UPDATED");
//add them to the test suite
    EU_add_test_case(&suite,&test1);
    EU_add_test_case(&suite,&test2);

    call Leds.init();

    return SUCCESS;
  }


  /**
   * Start things up.  This just sets the rate for the clock component.
   *
   * @return Always returns <code>SUCCESS</code>
   **/
  command result_t StdControl.start() {
    time = 0;
    //verify the initial time is 0
EU_ASSERT(&test1,time == 0);
//completed with test 1
    EU_testComplete(&suite,&test1);
//make sure other test starts
    return call Timer.start(TIMER_REPEAT, 1024);
  }

  /**
   * Halt execution of the application.
   * This just disables the clock component.
   *
   * @return Always returns <code>SUCCESS</code>
   **/
  command result_t StdControl.stop() {
    return call Timer.stop();
  }

  /**
```

```
   * Toggle the red LED in response to the <code>Timer.fired</code> event.
   *
   * @return Always returns <code>SUCCESS</code>
   **/
  event result_t Timer.fired()
  {
    avrtime_t oldtime;
    oldtime = time;
    time = time + 1024;
//verify with failure test
    EU_ASSERT_FALSE(&test2, oldtime >= time);

//completed with test 2
    EU_testComplete(&suite,&test2);
    call Leds.redToggle();
    return SUCCESS;
  }
}
```

In this example, the programmer appends an include directive to the program and adds the EUnit include path to the compiler path. In addition, the programmer must add a link flag to statically link against EUnit during compilation. EUnit requires two structures to maintain test state, the EU_TestSuite and EU_TestCase object. The EU_TestSuite maintains a set of test cases that exist for a EUnit, including accounting information regarding the specific test suite. EU_Test is the object for managing test case state. This allows the programmer to add multiple suites to different modules, enabling suites to run simultaneously. If there is intention to share suites across modules, the programmer must declare the EU_TestSuite out of the TinyOS module scope, making it global. The test suites are initialized with the *init_eu_test_suite* function. The test cases are initialized with the *init_eu_test_case* function. Instead of maintaining a dynamic registry, the byte array (string) that is passed into the initialization functions is used to identify the test suites and test cases, respectively. If it is desired to perform an action for every test, the

programmer must set the custom handler function, as seen in the previous example by invoking the *EU_setCustomHandler* function. This function will be invoked after EUnit acts upon a test assertion.

After initialization, the programmer must add the test cases to a test suite so they can be executed. Once this completed, all the assertions must be invoked with a pointer to their respective test case. This is necessary since the test case utilize this to update test case state. In addition, it allows EUnit to trap the program when the program enters a fatal state defined by the programmer.

Since the software written for WSNs are continuous, loop and event based, block execution cannot be determined at compile time. It is necessary for the programmer to notify EUnit is completed with a test. Otherwise, completion of the tests can never be determined. It is the responsibility of the programmer to ensure all events are invoked. In the previous example, this is accomplished by starting the timer that will lead to the second test completion.

After the program is built, it is linked against the EUnit static library. This eliminates declaring functions with the "__attribute__((C))" attribute, since the EUnit source will not be compiled by ncc. ncc is the nesC compiler used by TinyOS. Utilizing the UIF and EUnit, the programmer would see the resulting output from the program after connecting the test bridge to their computer and uploading the program to the target device:

```
TEST("TEST INITIAL TIME"):time == 0:BlinkM.nc:BlinkM$StdControl$init:84:SUCCESS

TEST("TEST TIME UPDATED"):EU_ASSERT_FALSE(oldtime >= time):BlinkM.nc:BlinkM$Timer$fired:121:SUCCESS

==SUMMARY FOR TEST SUITE ("Blink Test Suite")==
TEST("TEST INITIAL TIME"):PASS
TEST("TEST TIME UPDATED"):PASS
==Out of 2 test cases, 0 failed 2 passed==
```

When the prior example executes, EUnit notifies the programmer the name of the test, the assertion, the filename of the test, the function name, the line number and whether the assertion passed. This is a result of the programmer implementing the "customHandler" function. On the event that the test is completed by invoking all EU_testComplete functions, the summary for the test suite is printed out. The summary lists the tests of the suite, whether they passed, and a total quantity of successful and unsuccessful tests. This is emitted regardless of a custom handler set by the programmer. After the suite is completely executed once, the results will not be emitted onward. However, if any fatal assertions are reached, this will force the program into a program trap. Upon a program trap event the programmer can use JTAG-ICE to inspect the state of the MCU during a defined fatal assertion.

A.3.4 Runtime Verification

Runtime verification is the process of monitoring a program during runtime, and verifying that it conforms to a specification [17]. The specification is created based on observable properties and behaviors, such as those defined in section A.1. In this work, two run-time verification techniques, log-file analysis and aspect oriented programming (AOP), are utilized to verify program correctness of common Wireless Sensor Networks properties. Each of which are novel in their usage towards WSN testing [9].

A.3.4.1 Log-File Analysis

Log file analysis is a commonly employed method to verify desired program behavior. As the program executes, messages are emitted from the system. In this work, the messages are emitted regarding the machine state. The messages contain symbols that are used to map to a defined state during processing. Some examples of states include

sleeping and reading specific sensors of the system. The order at which these states occur are usually defined by a state-machine and then checked with an alternative program to determine of the runtime behavior is rejected or accepted [9].

In this work, log-file analysis is used in both simulation and in the real system during testing. The messages are emitted utilizing UIF (Section A.3). In the final deployed program, log-file analysis is used to verify the machine functions properly. The states that the deployed program visit include sleep, check voltage, sense GPS, sense compass, write to flash, read from flash, short-range communication and long-range communication. For the special cases, each state feature an edge to states that are accepted but not desirable behavior. For example, in the check voltage state when the voltage is not high enough, the machine will traverse into a sleep state instead of go into the sample GPS state. This is not desirable, since the tracking device will not be able to acquire a fix upon when the voltage is tool low [9].

Log-file analysis is sufficient but incapable of being expressive as aspect-oriented programming [9]. However, the runtime overhead of log-file analysis is minimal when compared to that required of aspect-oriented programming. For this reason, the final software is verified with log-file analysis.

A.3.4.2 Aspect Oriented Programming

The testing framework also features aspect-oriented programming capabilities. In collaboration with Dave, ACC is incorporated into the TinyOS build process so aspect-oriented programming capabilities can be utilized for run-time verification. The details regarding how this accomplished is available in Dave's masters thesis, and will be left out of discussion [10]. Following the collaborative efforts, I made modifications to the process to enable the aspects to execute on real-hardware. Where the work with Dave only allowed aspects to be weaved into a simulation. The result of this is explored in Section A.4.

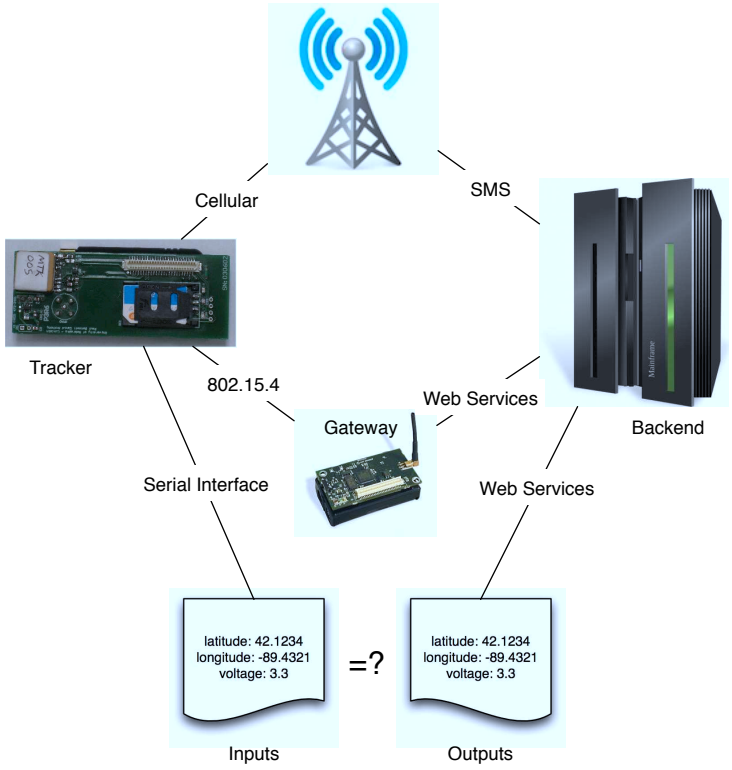


Figure A.1: Illustration of system integration testing.

A.3.5 System-Integration Testing

The crane tracking software is composed of many moving parts. Mainly, the tracking device software, back-end services, and the visualization tools. Independently, these components can be tested utilizing unit tests and run-time verification techniques. However, it is difficult to test all components when integrated together. This section discusses the process used to test the system in the lab, before deployed on proxy species covered in Chapter 6.

During system-integration testing, the system is black-boxed tested [91] to verify system wide correctness. This is accomplished by creating inputs and expected outputs for the tracking device, and checking the results from the back-end services. The process is depicted in Figure A.1. Initially, the test input is sent through the serial interface to

the test bridge device. The test bridge device utilizes serial communication to transmit the test input to the target device where it is processed and eventually disseminated to the back-end. The target utilizes both the radio interface and the cellular network to communicate with the back-end. The back-end processes the data, and the test script accesses the results through the REST interfaces.

The process is automated with a script, where a file consisting of expected inputs and outputs are provided by the tester. The script retains knowledge of the REST interface provided by the back-end of the system. In the occurrence of a failure or unexpected output, the test script reports the input and actual output. The programmer then evaluates the cause of the failure, where it can either be an issue with the input or a bug in the program.

Before being deployed, the tracking device is activated and left operational for some time. Since the expected values are not exactly known, the end-user must closely evaluate whether or not that the data is logical.

A.4 Hardware Analysis

In this section, the testing framework is analyzed with the intention to exploit the performance behaviors of run-time aspects in hardware. In this process, the properties described in section A.1 are used to develop aspects for evaluation of the device software. The results of the initial aspects in comparison to log-file analysis is published in Dave's thesis [10]. In this section, the evaluation of the aspects executed on real hardware.

The program used for simulation was incapable of being executed on the mote due to resource limitations. To study the behavior aspects have on resource constrained hardware, a simpler program was used. The test program was designed to start a timer at a fixed rate and toggle an LED. The following aspects were created to observe different

areas of embedded software:

- *timers* instrument a message before every timer is fired.
- *trace* instrument a message at every method invocation.
- *messageEvery30Seconds* monitor system time and instrument a message every 30 seconds.
- *haltAt5minutes* monitor the operational time to change behavior of execution.

After compilation, each of the aspects were able to be fit in the available resources of the system. The size of the executable and consumed static memory fit for all aspects (except the *trace*). More importantly, each aspect completed task they were designed for.

There were interesting behaviors that should be considered by developers when constructing aspects for real embedded devices. First, the monitoring capability decreases as the interrupt service routine occurrences increase. For example, when the timer rate was increased from 1Hz to 20Hz, the instrumentation generated by the aspects would never complete before the next interrupt occurred. Secondly, the observing method invocations should be minimal since the stack quickly overflows. This was observed using the testing framework JTAG-ICE capabilities, where the device would reset after the stack exploded in depth size.

During development of the final tracking software, log file analysis was the only technique able to be used to verify the system functioned as desired. This was a result of resource limitations of the tracking device and the simulators inability to model all components of the real device. Thus, the developed tracking application consumed too many resources, preventing aspects to be used for monitoring runtime behavior of the actual application. Unfortunately, this work displayed that it was impractical to utilize aspects for complex software applications on real devices.

Appendix B

Embedded Software

B.1 Hayes Command Protocol

The command is composed of a prefix (”AT”), body and termToken (carriage-return plus line-feed). The body of the command is either a read command, test command, or action command. Read commands are for retrieving device properties such as signal quality of the GSM unit. Test commands are for querying the interpreter for appropriate parameter fields. The action commands are used for performing an action such as setting the baud-rate of the device. The inputs accepted as responses and inputs utilized by the platform include smsResponse, informationResponse, and errorResponse. The smsResponse node denotes when the device is ready to accept SMS data that is desired to be transmitted by the unit. The information response token represents the result usually received after sending a read command. In other cases, such as verbose status updates are accepted by this node in the parser as well. In order to reduce program code size and overhead of the responses returned by the GSM unit, it is the responsibility of the software using the parser to handle the information string at a higher level. Otherwise, it was thought at the time that handling every single information response in the parser, especially those

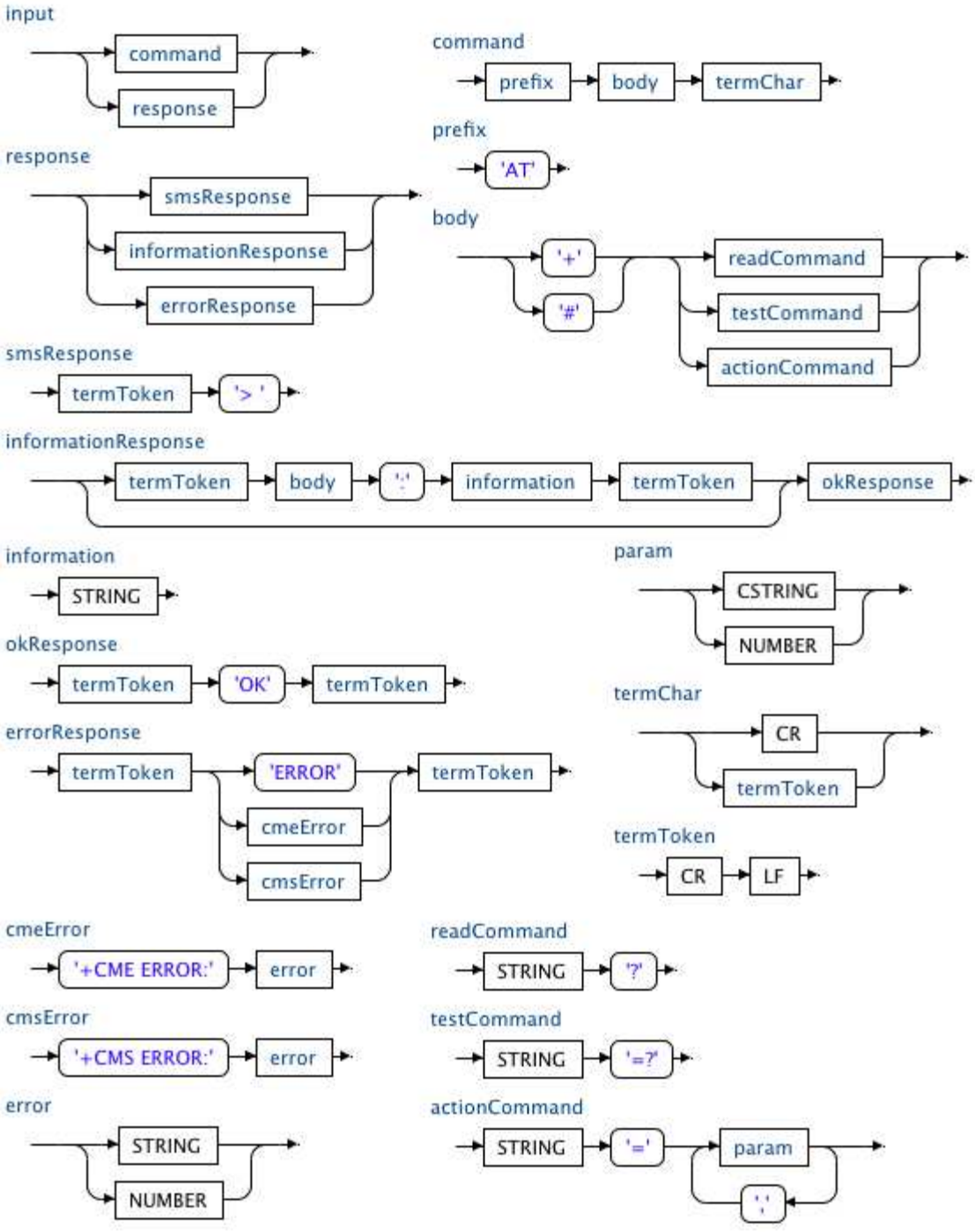


Figure B.1: The Hayes command and response tokens that are scanned and parsed by the GSM Driver.

not used by the higher-levels, would explode the driver footprint. The error response accepted by the parser, handles both numerical and verbose errors. Only equipment errors and message service errors are supported by the driver, otherwise a general error is accepted and left for higher-levels to handle. The error codes are utilized by upper-levels to determine device health and communication errors.

The supported initialization operations are provided to higher-levels through the following functions; ”init”, ”start”, ”stop”, ”powerOn”, ”powerOff”, ”setBaud”, ”verboseErrors”, and ”setSmsMode”. The ”init” function handles configuration of GPIO and command states of the driver. The ”powerOn”/”powerOff” functions activate and deactivate the GPIO line connected to the BJT. This is feature is required for the GSM unit to power on correctly. The ”powerOn” command must be invoked before any function can interact with the GSM unit. The ”start”/”stop” functions start and stop the underlying TinyOS modules utilized by the driver such as the ByteControl module. The ”setBaud” function configures the UART baud rate on the GSM unit. The ”verboseErrors” function enables the GSM unit to emit error responses in verbose format to enable human readable debugging. Lastly, the ”setSmsModoe” configures the SMS mode of the GSM, since it is able to support both PDU and operational modes. The driver currently only supports and utilizes the text mode of SMS.

The GSM supports both GPRS and SMS communication. However, the prototype only supports SMS communication for the proof-of-concept. This was also decided in efforts to conserve energy, since the GSM unit consumes less current while transmitting data in SMS, when compared to GPRS. In the future, it is desired to extend the software to support GPRS. With GPRS support, the unit will be able to offload more data in a shorter period of time, actually reducing the energy consumption of the communication process. The ”sendSMS” function manages sending an SMS and when this process is completed the ”sendSmsDone” event is fired to notify completion of a message being

sent.

The GM862-GPS unit is capable of sensing GPS and cellular signal qualities. The GSM-GPS interface provides each of these sensing functionalities and they are accessible utilizing the ”getAqrdGpsPos” and ”getSignalQuality” commands. In addition, the GPS can be disabled and enabled using the ”startGps” and ”stopGps” respectively. Unlike the GPS driver for the MTS-420, the GM862-GPS provides position information through the AT commands. Unfortunately, Dave's driver extension could not be utilized since this protocol differs from the NMEA0183 messages emitted by a standard GPS. The position information is encompassed in a ”information” node, a child of a ”informationResponse” node, included in the parser depicted in Figure B.1. For simplicity, the string representation of the position is returned by the ”getAqrdGpsPosDone” event. In addition to the position information, the driver also capable of providing cellular base-station and signal quality information. The cellular base-station information was not supported by the prototype driver, only the signal quality was supported. This design decision was due to the unknown uses of the base-station data, but the known uses of the signal quality. In any case, the base-station information is utilized in the final software, which will be discussed in the following sections.

Appendix C

Back-end

C.1 CPN Query Language

```
query:
  andExpression (andExpression)*;

andExpression:
  orExpression  ( '&' orExpression)*;

orExpression:
  expression  ( '|' expression)*;

expression:
  fieldExpression | limitExpression;

fieldExpression:
  field '@' ( singleParam | rangeParam | orderingParam);
```

```
limitExpression:
  Limit '@' NUMBER;

pageExpression:
  Page '@' NUMBER;

field:
  STRING;

singleParam:
  singleParamOperator '=' type ';' value;

rangeParam:
  rangeParamOperator '=' type ';' value '?' type ';' value;

orderingParam:
  orderingParamOperator '=' 'asc' | 'desc';

singleParamOperator:
  GreaterThan
  | GreaterThanEqual
  | LessThan
  | LessThanEqual
  | OrderBy
```

```
  | Like
  | Is
  | NotEqual
  | Equal;

rangeParamOperator:
  Between;

orderingParamOperator:
  OrderBy;

type:
  intType | floatType | dateType | nullType;

intType:
  'i' | 'integer';
floatType:
  'f' | 'float';
dateType:
  'd' | 'date';
nullType:
  'null';
value:
  NUMBER | STRING;
NotEqual: 'ne';
```

```
Equal: 'eq';
GreaterThan: 'gt';
GreaterThanEqual: 'gte';
LessThan: 'lt';
LessThanEqual: 'lte';
OrderBy: 'orderby';
Like: 'like';
Is: 'is';
Between: 'btw';
Limit: 'limit';
Page: 'page';

NUMBER: SIGN? INT DEC? INT (EXP SIGN? INT)? ;

INT: ('0'..'9')+;
DEC: '.';
SIGN: '-' | '+';
EXP: 'e' | 'E';

STRING: '\'' ( CH | ~('\\'|'\'') )* '\'';
CH: ('0'..'9'|'a'..'z'|'A'..'Z');
```

Bibliography

[1] C. A. Bahon and V. Sole. A jerk threshold-based involuntary lateral movement algorithm. In *Emerging Technologies Factory Automation, 2009. ETFA 2009. IEEE Conference on*, pages 1 –4, sept. 2009. 4.2.2

[2] G.D. Abowd and J.P.G. Sterbenz. Final report on the inter-agency workshop on research issues for smart environments. *IEEE, Personal Communications*, 7(5):36 –40, oct 2000. 2.2, 2.2.1

[3] Jon Agre and Loren Clare. An integrated architecture for cooperative sensing networks. *Computer*, 33(5):106–108, May 2000. 2.2

[4] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A Survey on Sensor Networks. *IEEE Communications Magazine*, 40(8):102 – 114, August 2002. 2.2, 2.2.1, 4.1, 4.2, 5, 6

[5] I.F. Akyildiz and M.C. Vuran. *Wireless Sensor Networks*. John Wiley & Sons Ltd., 2010. 1.1

[6] I. Amundson and X. Koutsoukos. A survey on localization for mobile wireless sensor networks. In *Proceedings of the 2nd international conference on Mobile entity localization and tracking in GPS-less environments*, MELT'09, pages 235–254, Berlin, Heidelberg, 2009. Springer-Verlag. 2.2.1

[7] I. Amundson, X. Koutsoukos, and J. Sallai. Mobile sensor localization and navigation using rf doppler shifts. In *Proceedings of the first ACM international workshop on Mobile entity localization and tracking in GPS-less environments*, MELT '08, pages 97–102, 2008. 2.2.1

[8] J.H. Andrews and Y. Zhang. Broad-spectrum studies of log file analysis. In *Proceedings of the 22nd International Conference On Software Engineering (ICSE'00)*, pages 105–114, Limerick, Ireland, June 2000. A.2

[9] D. Anthony, W.P. Bennett, M.C. Vuran, M. Dwyer, S. Elbaum, and F. Chavez-Ramirez. Simulating and Testing Mobile Wireless Sensor Networks. In *Proceedings of the 13th ACM International Conference on Modeling, Analysis, and Simulation of Wireless and Mobile Systems (MSWIM'10)*, pages 49–58, Bodrum, Turkey, October 2010. 5, 5.3.1, A, A.1, A.2, A.3.2, A.3.4, A.3.4.1

[10] David Anthony. A multi-modal sensing and communication platform for continental-scale migratory bird tracking. Master's thesis, University of Nebraska-Lincoln, 2012. 1.2, 1.3, 2.2.1, 3, 3.3.2, 3.3.4, 6.2, 6.2, A.2, A.3.4.2, A.4

[11] David Anthony, William P. Bennett, Mehmet C. Vuran, Matthew B. Dwyer, Sebastian Elbaum, Anne Lacy, Mike Engels, and Walter Wehtje. Sensing through the continent: towards monitoring migratory birds using cellular sensor networks. In *Proceedings of the 11th international conference on Information Processing in Sensor Networks*, IPSN '12, pages 329–340, Beijing, China, 2012. 1.1, 1.3, 2.1, 2.2, 2.2.1, 2.2.2, 2.3, 2.4, 3, 3.2.2.1, 3.2.2.1, 4.1, 4.2.1, 5.3.6.2, 6.1, 6.2, 6.3, 6.4, 6.4, 6.4, 6.5, 6.5, 6.5, 6.6, 6.6, 6.6, 6.6, 6.6, 6.6, 6.6, 6.6, 6.6

[12] Apple. Xcode. Electronic, June 2012. https://developer.apple.com/xcode/. 3.4.2.1

[13] Arduino. Api reference. Electronic, Feb. 2009. http://arduino.cc/it/Reference/HomePage. 3.4.1.1

[14] Atmel Corporation. ATmega 1281 Datasheet. http://www.atmel.com, April 2012. 2.2

[15] AT&T. AT&T global smart messaging suite. electronic, June 2012. http://www.wireless.att.com/businesscenter/solutions/email-messaging/smart-messaging-suite.jsp. 4.1.5.1

[16] Edgardo Avilés-López and J. García-Macías. TinySOA: a service-oriented architecture for wireless sensor networks. *Service Oriented Computing and Applications*, 3(2):99–108, June 2009. 2.3.1

[17] H. Barringer, A. Goldberg, K. Havelund, and K. Sen. Rule-based runtime verification. In Bernhard Steffen and Giorgio Levi, editors, *Verification, Model Checking, and Abstract Interpretation*, volume 2937 of *Lecture Notes in Computer Science*, pages 277–306. Springer Berlin / Heidelberg, 2004. 10.1007/978-3-540-24622-0_5. A.3.4

[18] Bell Labs. *Unix Programmer's Manual*, 11 1971. 3.4.2.1

[19] W.P. Bennett, M. Fitzpatrick, D. Anthony, M.C. Vuran, and A. Lacy. Poster Abstract: Crane Charades: Behavior Identification via Backpack Mounted Sensor Platforms. Proceedings of the 11th ACM/IEEE Conference on Information Processing in Sensor Networks (IPSN'12), Beijing, China, April 2012. (Best Poster Award). 2.1, 4.2.2, 6.5, 6.5, 6.5, 7.3

[20] R. Biswas and E. Ort. The java persistence api - a simpler programming model for entity persistence. Electronic, May 2006. 4.1.3

[21] N. Bulusu, D. Estrin, L. Girod, and J. Heidemann. Scalable coordination for wireless sensor networks: Self-configuring localization systems. In *Proceedings of the Sixth International Symposium on Communication Theory and Applications (ISCTA '01)*, July 15-20th 2001. 2.2, 2.2.1

[22] Eduardo Cañete, Jaime Chen, Manuel Díaz, Luis Llopis, and Bartolomé Rubio. Useme: A service-oriented framework for wireless sensor and actor networks. In *Proceedings of the 2008 Eighth International Workshop on Applications and Services in Wireless Networks*, ASWN '08, pages 47–53, Washington, DC, USA, 2008. IEEE Computer Society. 2.3.1

[23] Tracy Camp, Jeff Boleng, and Vanessa Davies. A survey of mobility models for ad hoc network research. *WIRELESS COMMUNICATIONS & MOBILE COMPUTING (WCMC): SPECIAL ISSUE ON MOBILE AD HOC NETWORKING: RESEARCH, TRENDS AND APPLICATIONS*, 2:483–502, 2002. 5.3.5

[24] M. Caporuscio, P. Raverdy, H. Moungla, and Valerie Issarny. ubisoap: A service oriented middleware for seamless networking. In *Proceedings of the 6th International Conference on Service-Oriented Computing*, ICSOC '08, pages 195–209, Berlin, Heidelberg, 2008. 2.3.1

[25] G. Chen, J. Branch, M. J. Pflug, L. Zhu, and B. K. Szymanski. Sense: A sensor network simulator. Electronic, Febuary 2004. 5

[26] Atmel Corp. *AVR JTAGICE mkII Quick Start Guide*. Atmel, c edition, 10 2010. http://www.atmel.com/Images/doc2562.pdf. A.2, A.3.1

[27] Atmel Corp. AVR155: Accessing an I2C LCD display using the AVR 2-wire serial interface. electronic, June 2012. http://www.atmel.com/Images/doc1981.pdf. 3.4.1.1

[28] Memsic Corp. Moteworks getting started guide. Electronic, July 2007. http://www.memsic.com/support/documentation/wireless-sensor-networks/category/6-user-manuals.html?download=603.2.1

[29] cunit. Cunit: A unit testing framework for c. Electronic, June 2012. http://cunit.sourceforge.net/. 3.4.2.1, A.3.1, A.3.3

[30] J. G. Dickson. *The Wild Turkey: Biology and Management*. National Wild Turkey Federation (U.S.), United States. Forest Service, 1992. 6.4

[31] X. Dong and M. C. Vuran. Spatio-temporal soil moisture measurement with wireless underground sensor networks. In *Ad Hoc Networking Workshop (Med-Hoc-Net), 2010 The 9th IFIP Annual Mediterranean, Juan Les Pins, France*, pages 1 –8, june 2010. 2.2, 2.2.1

[32] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, LCN '04, pages 455–462, Washington, DC, 2004. IEEE Computer Society. 5

[33] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005. 4.1

[34] R. Estrin, D.and Govindan, J. Heidemann, and S. Kumar. Next century challenges: scalable coordination in sensor networks. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, MobiCom '99, pages 263–270, Seattle, Washington, August 1999. 2.2, 2.2.1

[35] Levis. et. al. TOSSIM: accurate and scalable simulation of entire TinyOS applications. In *Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems (SenSys'03)*, pages 126–137, Los Angeles, CA, November 2003. 5, 5.1, 5.2

[36] P Barnaghi et. al. A service oriented middleware architecture for wireless sensor networks. *Networks*, 02:1–10, Sept. 2010. 2.3.1

[37] P. Luschi et. al. Satellite tracking of captive-reared juvenile loggerhead sea turtles (caretta caretta) released in southern new england. *Marine Biology*, 143:793–801, March 2006. 6.2

[38] Sundani et al. Wireless sensor network simulators: A survey and comparisons. *International Journal of Computer Networks (IJCN)*, 2:249–265, August 2011. 5

[39] William Bennett et al. Cranetracker: A multi-modal platform for monitoring migratory birds on a continental scale. Inprocedings, September 2011. Demo. 4.2.2

[40] C.K. Eun, Tze-Ching Fung, B. Mitra, and Y.B. Gianchandani. A magnetically enhanced 3-electrode wireless micro-geiger counter. In *IEEE 20th International Conference on Micro Electro Mechanical Systems, 2007. MEMS.*, pages 599 –602, Jan. 2007. 2.2

[41] L. Fang, P. J. Antsaklis, L. Montestruque, M. B. Mcmickell, M. Lemmon, Y. Sun, H. Fang, I. Koutroulis, M. Haenggi, M. Xie, and X. Xie. Design of a wireless assisted pedestrian dead reckoning system - the navmote experience. *Instrumentation and Measurement, IEEE Transactions on*, 54(6):2342 – 2358, dec. 2005. 2.2.1

[42] A. Ferscha and S. K. Tripathi. Parallel and distributed simulation of discrete event systems. Technical report, College Park, MD, 1994. 5.1, 5.1

[43] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. 4.1.7

[44] U.S. FIsh and Wildlife Service. Texas parks and wildlife fact sheet. Electronic, June 2012. http://www.tpwd.state.tx.us/publications/pwdpubs/media/pwd_lf_k0700_0849.pdf. 6.2

[45] International Crane Foundation. International Crane Foundation. http://www.savingcranes.org, March 2012. 1.3

[46] Pascal Fradet and Mario Sdholt. AOP: towards a generic framework using program transformation and analysis. Electronic, June 2012. A.2

[47] M. Francesco, S. K. Das, and G. Anastasi. Data collection in wireless sensor networks with mobile elements: A survey. *ACM Trans. Sen. Netw.*, 8(1):7:1–7:31, August 2011. 2.2

[48] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. 4.1.3, 5.1.1

[49] Adrian Gleiss and et. al. Making overall dynamic body acceleration work: on the theory of acceleration as a proxy for energy expenditure. *British Ecological Society, Methods in Ecology and Evolution*, 2:23–33, August 2011. 2.1

[50] GlobalTop. GPS-PA6B. http://www.gtop-tech.com/, March 2012. 3.3.2

[51] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, David Moss, and Philip Levis. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded*

Networked Sensor Systems, SenSys '09, pages 1–14, Berkeley, California, November 2009. 2.2

[52] Google. Google voice about. Electronic, June 2012. http://www.google.com/googlevoice/about.html. 4.1.5.1

[53] W. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, MobiCom '99, pages 174–185, Seattle, Washington, August 1999. 2.2

[54] Joanne Herfel. Experience the ultra(light) in migration. Newsletter of the Madison Audubon Society, March 2002. 2.1

[55] Honeywell International. HMC6343 3-Axis Compass with Algorithms. http://www.honeywell.com, April 2012. 3.3.1, 3.3.1.1, 3.3.1.3, 3.3.3.1, 3.4.1.2, 3.4.2.1, 5.4

[56] Marshall A. Howe. Migration of radio-marked whooping cranes from the Aransas-Wood Buffalo population : patterns of habitat use, behavior, and survival. Technical report, U.S. Dept. Interior, Fish and Wildlife Service, 1989. 1.1

[57] Google Inc. Google earth. Electronic, June 2012. http://www.google.com/earth/index.html. 5.3, 5.3.5

[58] P. Juang, H. Oki, Y. Wang, M. Martonosi, Li Shiuan Peh, and Daniel Rubenstein. Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. *SIGPLAN Not.*, 37:96–107, October 2002. 2.4

[59] Xenofon Koutsoukos, Manish Kushwaha, Isaac Amundson, Eep Neema, and Janos Sztipanovits. Oasis: A service-oriented architecture for ambient-aware sensor networks . *SOCA*, 5:71–85, April 2011. 2.3.1

[60] H. Kung, C. Huang, and H. Ku. Efficient sensor deployment control schemes and performance evaluation for obstacle and unknown environments. *Wirel. Pers. Commun.*, 45(2):231–263, April 2008. 2.2

[61] E. Kuyt. Banding of juvenile Whooping Cranes and discovery of the summer habitat used by nonbreeders. In *Proceedings of the 1978 Crane Workshop*, Rockport, TX, December 1979. 2.1, 2.3, 2.4, 3.2

[62] E. Kuyt. Aerial radio-tracking of Whooping Cranes migrating between Wood Buffalo National Park and Aransas National Wildlife Refuge, 1981-1984. Technical report, Canadian Wildlife Service, 1992. 2.1, 2.3, 2.4, 3.2, 6.2

[63] Olaf Landsiedel and Klaus Wehrle. Aeon: Accurate prediction of power consumption in sensor networks. In *In Proceedings of The Second IEEE Workshop on Embedded Networked Sensors (EmNetS-II*, 2004. 5

[64] Ramon Lawrence. The space efficiency of XML. *Information and Software Technology*, 46:753–759, March 2004. 4.1.6

[65] Edward A. Lee. Embedded software. *to appear in Advances in Computers*, 56, 2002. 3.4.2.5

[66] M. Lehtinen, A. Happonen, and J. Ikonen. Accuracy and time to first fix using consumer-grade GPS receivers. In *Proc. IEEE SoftCom '08*, pages 334 –340, Dubrovnik, Croatia, Sep. 2008. 5.3.2, 5.3.3, 5.3.6.3, 6.4

[67] P. Levis, S. Madden, J. Polastre, R. Szewczyk, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. TinyOS: An operating system for sensor networks. In *Ambient Intelligence*. Springer Verlag, 2004. 1.2, A.3.1

[68] A. Lindgren, C. Mascolo, M. Lonergan, and B. McConnell. Seal-2-Seal: A delay-tolerant protocol for contact logging in wildlife monitoring sensor networks. In *Proceedings of the 5th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS'08)*, pages 321–327, Atlanta, GA, September 2008. 2.2.2

[69] T. Liu, C. Sadler, P. Zhang, and M. Martonosi. Implementing software on resource-constrained mobile sensors: experiences with Impala and ZebraNet. In *Proceedings of the 2nd ACM International Conference on Mobile Systems, Applications, and Services (MobiSys'04)*, pages 256–269, Boston, MA, June 2004. 2.2.2

[70] Y. Liu, H. Ngan, and L. M. Ni. Power-aware node deployment in wireless sensor networks. In *Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing -Vol 1 (SUTC'06) - Volume 01*, SUTC '06, pages 128–135, Taichung, Taiwan, June 2006. 2.2

[71] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TinyDB: An acquisitional query processing system for sensor networks. *ACM Trans. Database Syst*, 30:122–173, March 2005. 2.3.1

[72] A. Markham, N. Trigoni, S. Ellwood, and D. Macdonald. Revealing the hidden lives of underground animals using magneto-inductive tracking. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys'10)*, pages 281–294, Zurich, Switzerland, November 2010. 1.1, 2.2.2, 2.4, 6

[73] B. McConnell, R. Beaton, E. Bryant, C. Hunter, P. Lovell, and A. Hall. Phoning Home- A New GSM Mobile Phone Telemetry System To Collect-Mark Recapture Data. *Marine Mammal Science*, 20(2):274–283, April 2004. 2.2.2, 6

[74] Colin McDonald. Whooping cranes spark a water war. Electronic, December 2011. http://www.mysanantonio.com/news/environment/article/Water-for-whooping-cranes-2342852.php. 5.3.3

[75] Memsic Inc. Iris Datasheet. http://www.memsic.com/, March 2012. 2.2

[76] Memsic Inc. MTS420 Datasheet. http://www.memsic.com/, March 2012. 3.2.2.2

[77] microvideox.com. Ctt-100 real time tracker. Electronic. 2.2.2

[78] Nader Mohamed and Jameela Al-Jaroodi. A survey on service-oriented middleware for wireless sensor networks. *Serv. Oriented Comput. Appl.*, 5(2):71–85, June 2001. 2.3.1, 2.3.1

[79] NS. The ns-3 simulatior. Electronic, June 2012. http://www.nsnam.org/. 5

[80] Michael Okola and Kamin Whitehouse. Unit testing for wireless sensor networks. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Sensor Network Applications*, SESENA '10, pages 38–43, Zurich, Switzerland, June 2010. A.2

[81] Oracle. Java architecture for xml binding (jaxb). Electronic, July 2012. http://www.oracle.com/technetwork/articles/javase/index-140168.html. 4.1.3

[82] Oracle. Java ee at a glance. Electronic, June 2012. http://www.oracle.com/technetwork/java/javaee/overview/index.html. 4.1

[83] Oracle. What's new in java servlet api 2.2? Electronic, June 2012. 4.1.2

[84] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 641 –648, nov. 2006. 5

[85] Bhawana Parbat, A. K. Dwivedi, and O. P. Vyas. Data visualization tools for wsns: A glimpse. 4.2, 4.2.3

[86] Terence J. Parr and Russell W. Quong. Antlr: A predicated-ll(k) parser generator. *Software Practice and Experience*, 25:789–810, 1994. 3.4.2.1, 4.1.7

[87] Enrico Perla, Art Ó Catháin, Ricardo Simon Carbajo, Meriel Huggard, and Ciarán Mc Goldrick. PowerTOSSIM-Z: Realistic energy modelling for wireless sensor network environments. In *Proceedings of the 3nd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, PM2HW2N '08, pages 35–42, Vancouver, Canada, October 2008. 5

[88] Java Community Process. Jsr-000914 javatm message service (jms) api. Electronic, June 2012. 5.1.1

[89] W.K.G. Seah, Zhi Ang Eu, and Hwee-Pink Tan. Wireless sensor networks powered by ambient energy harvesting (wsn-heap) - survey and challenges. In *1st International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology, 2009. Wireless VITAE 2009.*, pages 1 –5, Princeton, New Jersey, June 2009. 2.2

[90] Emily Shepard and et al. Identification of animal movement patterns using tri-axial accelerometry. *Endangered Species Research*, 10:47–60, March 2008. 2.1

[91] British Computer Society. Standard for software component testing. Electronic, June 2012. A.3.5

[92] O. Sokolsky, U. Sammapun, J. Regehr, and I. Lee. Runtime Verification for Wireless Sensor Network Applications. In *Dagstuhl Seminar Proceedings 07011*, Dagstuhl, Germany, August 2008. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. 5.3.3

[93] Dassault Systmes. Solid works design tool. Electronic, June 2012. 6.2

[94] Robert Szewczyk, Alan Mainwaring, Joseph Polastre, John Anderson, and David Culler. An analysis of a large scale habitat monitoring application. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, pages 214–226, New York, NY, USA, 2004. ACM. 1.1, 2.2.2

[95] Cellular Tracking Technologies. Ctt-1000 series avian tracker. http://celltracktech.com/index.php/products/wildlife-telemetry-solutions/ctt-1000-series/, June 2012. 2.2.2

[96] Telit Communications S.p.A. At commands reference guide. Electronic, June 2012. www.telit.com/module/infopool/download.php?id=542. 3.3.1.2, 3.3.3.3

[97] Telit Communications S.p.A. GE865 Datasheet. Electronic, March 2012. www.telit.com/module/infopool/download.php?id=1484. 3.3.2, 3.3.3.3, 3.3.3.3, 3.4.2.1

[98] Telit Communications S.p.A. GM862 Datasheet. Electronic, March 2012. www.telit.com/module/infopool/download.php?id=165. 3.3.1, 3.3.1.2, 3.3.2, 3.3.3.2, 3.3.3.3, 3.3.3.3

[99] Inc. The Crane Trust. The Crane Trust, Inc. http://www.cranetrust.org, March 2012. 1.3

[100] TinyOS. Tunit. Electronic, 06 2012. http://docs.tinyos.net/tinywiki/index.php/TUnit. A.2, A.3.1

[101] B.L. Titzer, D.K. Lee, and J. Palsberg. Avrora: scalable sensor network simulation with precise timing. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN'05)*, pages 477–482, Los Angeles, CA, April 2005. 5

[102] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A macroscope in the redwoods. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, pages 51–63, San Diego, CA, November 2005. 2.2.1

[103] J. Tooker, X. Dong, M. C. Vuran, and S. Irmak. Connecting soil to the cloud: A wireless underground sensor network testbed. IEEE Secon Demo, June 2012. 1.2, 2.2.1, 4.1, 7.2, 7.3

[104] M. Turon. Mote-view: a sensor network monitoring and management tool. In *Proceedings of the 2nd IEEE workshop on Embedded Networked Sensors*, EmNets '05, pages 11–17, Sydney, Australia, 2005. 2.3.1, 4.2, 4.2.3

[105] Typesafe. Typesafe stack. Electronic, May 2012. 4.1, 4.1.2, 4.1.3

[106] u-blox. u-blox ANTARIS 4 GPS Module. http://www.u-blox.com/, August 2010. 5.3.2, 5.3.3

[107] V. Vipindeep and Pankaj J. List of common bugs and programming practices to avoid them. Electronic, March 2005. A.1.1

[108] W. Wehtje. Aransas Wood Buffalo Population Radio-Marked Whooping Crane Fall 2010 Migration Report. April 2011. 2.1, 2.3, 3.2, 3.3.1.3, 6.6

[109] M. West. *Developing High Quality Data Models*. Shell International Limited, ISCL/4, Shell Centre, London, SE1 7NA, UK, 2.0 edition, June 2012. A.1

[110] B. A. Wichmann and I. D. Hill. Generating good pseudo-random numbers. *Comput. Stat. Data Anal.*, 51(3):1614–1622, December 2006. 5.3.2

[111] R. Wilson. Moving towards acceleration for estimates of activity-specific metabolic rate in free-living animals: the case of the cormorant. *British Ecological Society, Journal of Animal Ecology*, 75:1081–1090, Sept. 2006. 2.1

[112] R. Wilson. Prying into the intimate details of animal lives: use of a daily diary on animals. *Endangered Species Research*, 4:123–137, Jan. 2008. 2.1

[113] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M. Das. Perracotta: Mining temporal API rules from imperfect traces. In *Proceedings of the 28th International Conference on Software Engineering (ICSE'06)*, pages 282–291, Shanghai, China, May 2006. A.2

[114] Y. Yao and J. Gehrke. The COUGAR approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31(3):9–18, September 2002. 2.3.1, 4.2.3

[115] L. Yu, N. Wang, and X. Meng. Real-time forest fire detection with wireless sensor networks. In *Wireless Communications, Networking and Mobile Computing, 2005. Proceedings. 2005 International Conference on*, volume 2, pages 1214 – 1217, Sept. 2005. 2.2.1