SPECIFYING SENSOR/ACTUATOR NETWORK EXECUTION: A RELATIONAL DATABASE APPROACH

By Michael Middleton

A Thesis

Submitted in Partial Fulfillment

of the Requirements for the Degree of

Master of Science

in Engineering

Northern Arizona University

May 2016

Approved:

Paul G. Flikkema, Ph.D., Chair

James Palmer, Ph.D.

Fatemeh Afghah, Ph.D.

ABSTRACT

SPECIFYING SENSOR/ACTUATOR NETWORK EXECUTION: A RELATIONAL DATABASE APPROACH

MICHAEL MIDDLETON

Wireless sensor networks (WSN) provide a robust and versatile solution to distributed low-power computing over a range of diverse network topographies. One-way data aggregation from network to data center is traditionally the fundamental data acquisition feature for many wireless sensor networks. With the increasing prevalence of IoT devices and the large number of WSN application possibilities, network and nodal reconfiguration to suit changing environmental conditions and sensing needs via outbound commands is not only an attractive feature, but in many cases a growing necessity. The work of this thesis comprises the implementation of a web-based application allowing end users to identify and reconfigure sets or subsets of WSN nodes based on real-time network state information. This end-to-end network reconfiguration procedure relies upon an existing two-way cyber-infrastructure and a relational database to store and track network state information. Additionally, the streaming middleware solution, integral to this cyber-infrastructure, provides a tracking mechanism for network reconfiguration commands. This tracking ability along with the reconfiguration mechanisms

will not only allow end-users to enact behavioral network changes, but also provide the foundation for virtualized users to autonomously monitor and enact behavioral changes to the network in response to changing network or environmental conditions. The specific aim of this work is the design and implementation of the mechanisms upon which these features will be built.

Acknowledgements

List acknowledgments here

Contents

List of Tables

List of Figures

List dedication here

Chapter 1

Introduction

1.1 Wireless Sensor Networks

Transducers can be used to sense radiation and measure soil moisture, temperature, and other physical phenomena. Transducers are often built into devices referred to as wireless sensors, with wireless networking capabilities. Wireless sensor networks (WSNs) use wireless networking technologies to enable groups of wireless sensors to communicate with each other and connect to the Internet. Low power distributed computing, intelligent wireless networking, and sophisticated data management form the core of WSN technology. The integration of these technologies are enabling the development of modern applications like the Internet of Things (IoT).

Some collections of wirelessly networked sensing devices form a cyber-physical system. A cyber-physical system is one where computation and networking processes operate in conjunction with a physical system, such as a smart garden or an advanced industrial process. While there are a broad range of implementations and platforms that fall under the umbrella of WSN, they all have a particular set of design restrictions and offer fundamental baseline features. WSN devices generally lack sophisticated processing power, run on a finite power supply (typically batteries), and use a diverse range of network topologies. Due to the power constraints of many wireless sensor nodes, transmission ranges are often quite short, and networks require a base station with greater networking capabilities, enabling access to the rest of the Internet; this is typically accomplished via a cellular or satellite connection.

WSNs allow researchers, scientists, and analysts to embed transducers into the environment which manage and perform sampling duties on a diverse range of sensors. Additionally, actuators allow for changes to be imposed upon the environment at given scheduled times or in reaction to certain events. In a WSN, the data collected from the various devices is aggregated and routed through the network to a base station where it is accessible to applications or archived at a data center. Depending on the diversity of the data types, the sophistication of the sensors, or the high-level data management schema, there is a diverse range of operations such as data conversions, checksum error detection, and stream creation that must be performed. These operations, data conversion for instance, might be

performed at the sensor node, at the base station, or even at the data center.

1.1.1 Difficulties of Network Reconfiguration

The continued persistent functionality of a WSN requires that it have a certain level of autonomy. Basic WSN operation requires data aggregation from the nodes at the data center by whatever infrastructure is in place. There is also, however, a need for direct network interaction allowing a user to observe and enact behavioral changes within subsets of a network or the network as a whole. Researchers rely upon WSN to provide the means for extended monitoring or experimentation that span lengthy periods of time; changing circumstances may require for users to change the configuration of WSN devices based upon changing experimental needs or environmental conditions. For example, if a researcher studying the effects of soil moisture at a particular geographic location predicts a rain event, he or she might want to reconfigure the behavior of the WSN devices to gather soil moisture readings at an increased sampling rate to achieve greater data resolution.

There are a number of different design implementations for heterogeneous wireless sensor network control and reorganization, such as TinyOS and MagnetOS. These employ systems that allow modification of the sensors' behavior. It can be difficult to perform reconfigurations at the device level when WSNs are in remote or difficult to access locations such as rural areas or dangerous industrial facilities. Providing a user interface for researchers which accommodates making

such modifications would expand the functionality of the network in potentially new and unique ways.

1.2 Statement of the Problem

When physical access to a device is needed to perform device or network modification, this can place a significant burden on researchers by necessitating a large number of hours of travel time to remote locations or the traversal of dangerous environments to access the devices, such as in an industrial setting. The wireless nature of WSN systems offers a natural mechanism for users to communicate with the devices remotely.

Enabling users to remotely change the behavior of WSN nodes would allow researchers to rapidly tailor the behavior of the network to changing needs or environmental conditions. The creation of a web-based application which provides users with real-time network state information and the ability to enact changes would significantly reduce the amount of travel time, increase the amount of control a user has to rapidly reconfigure WSN nodes, and allow the mechanisms necessary to automate the reconfiguration process.

An agent is a software entity that can utilize the same network device reconfiguration capabilities that a person would have, though it's actions can be defined by software that can respond to changes in network conditions. An example is an

agent which monitors the power consumption of WSN devices in the network and reconfigures the devices to conserve energy. For example, if a specific WSN node is low on battery voltage, an agent might send the node commands that lower a transducer sampling rate.

The specific objectives of this work are as follows:

- Implementation of an application which allows users to remotely identify and select sets of WSN nodes based on static and dynamic search criteria;
- Implementation of an interface allowing users to enact behavioral changes in individual or groups of nodes; and
- Generalization of the reconfiguration software such that agents can monitor and reconfigure network nodes.

This thesis presents a system for both users and automated agents to reconfigure arbitrary subsets of nodes in large, complex collections of WSNs using (1) a database paradigm for selection of node subsets based on both static and dynamic data, and (2) an interface that allows both users and automated agents to identify subsets and issue valid reconfiguration commands.

1.3 Summary of Contents

Chapter 2 summarizes the academic literature discussing reconfiguration of sensor networks and the various platforms where reconfiguration has been implemented.

This chapter also discusses the challenges associated with reconfiguring wireless sensor networks and the design features that researchers have utilized to overcome these challenges.

Chapter 3 discusses the WSN platform and cyber-infrastructure named WiSARDNet.

Chapter 4 explains the end to end network reconfiguration software and execution via a relational database to track network state information.

In chapter 5, node reconfiguration use cases, software validation, and user access control are discussed in detail.

In chapter 6, node reconfiguration procedures and automated users are discussed in detail.

The final chapter summarizes the results of the research and software implementation. Additionally, opportunities for future work are discussed, followed by conclusions.

Chapter 2

Literature Review

2.1 Introduction

WSN technology has matured such that it is a core component of the developing Internet of Things (IoT). The IoT describes the way in which networking capabilities formerly restricted to more traditional computers are being extended to physically-embedded devices, allowing for the acquisition and transfer of data on a much larger scale. WSN are geographically clustered networked devices equipped with one or more transducers which can range from a simple temperature probe to an implanted medical device. In general, these devices have limited computational resources and operate on supplies with limited power and energy. Due to these limitations, developers need to consider the design constraints of WSN applications. Many WSN devices are constrained such that their computational resources may not support a complete operating system (OS), which have been

fundamental to traditional computing systems. The lack of a traditional OS may complicate the development or limit the functionality of an application.

The range of devices being added to the IoT which generate data grows daily. Even though extending the operational lifespan and overcoming computational limitations remains at the forefront of the challenges facing WSN technology, numerous developments are enabling greater WSN functionality. Some of the features being developed within the domain of WSN are intelligent routing and communication protocols, fault tolerance and network health monitoring, service oriented design, and application execution frameworks. In recent years, developers and researchers have streamlined many of these features and have created new abstractions for WSNs. These features allow developers to overcome many of the hurdles facing modern WSNs.

2.2 WSN Execution

The functionality necessary for WSN devices includes two critical features: distributed sensing and wireless data aggregation. WSN nodes have the ability to sample data from transducers and the ability to enact changes through the use of actuators. Ideally, researchers would be able to automate their experiments through an application program that interacts with individual or groups of WSN nodes. Clever use of abstraction to manage the complexities of the WSN will dictate how an application is executed; an application might run as an executable binary directly on a sensor node's hardware or perhaps within an abstraction of

the network at a higher layer in the form of scripted events or command sequences. WSN hardware is often extremely restrictive in its computational resources, and therefore application deployment and execution must be carefully designed.

Ivester et al. in [1] describe ISEE, an interactive execution framework for monitoring network services and specifying operation of applications executing on a WSN node using a design paradigm known as service oriented architecture (SOA). Network monitoring functionality is an important set of features, especially as experiments increase in complexity. The generalization of these features into modular services allows them to be used with greater versatility and independent of specific application code. Researchers who have developed other modern WSN platforms have also adopted the SOA paradigm of implementing WSN functionality. Hammoudeh et al. in [2] describe a SOA approach to WSN fault tolerance and inter-node cooperation.

Another category of WSN execution paradigms includes those that use a virtual machine (VM). A virtual machine emulates a particular hardware architecture by translating instructions to another hardware architecture, thus allowing software designed for a particular platform to execute atop a different platform. Two of the more prominent WSN reconfiguration approaches that utilize virtual machines are Maté [3] and MagnetOS [4]. Maté is a virtual machine that is created to overcome many of the challenges of implementing heterogeneous network execution on

resource-constrained wireless nodes. The fundamental feature of Maté is a byte-code interpreter. Programs and applications, even those with minimal complexity, can be hundreds of kilobytes in size. The virtual machine takes high-level operational instructions and organizes them into capsules that can be smaller than 100 bytes in size. The Maté byte code interpreter running on individual nodes allows for the execution of capsules that are disseminated throughout the network. This allows for a scalable approach for specifying network execution on a distributed system of resource constrained nodes. Similarly, the creators of MagnetOS use a virtual machine to specify network-level execution. These researchers use a version of the Java Virtual Machine (JVM) designed to appear as though it is operating on a single computer; however, the computer is actually a WSN. This abstraction of the WSN as a platform which can run the JVM allows for Java application code to be written by developers. The Java objects produced by the JVM can be sent throughout the network to different physical hosts, enabling the distributed execution of a single Java application. This is performed by a byte-code level translation into instructions that the nodes can execute. Maté and MagnetOS can compress large programs into small device operations, due to the abstractions which they were able to make in their design. Though a complete virtual machine abstraction of a WSN is not utilized in the work of this thesis, the core principles of energy and resource aware network execution through abstraction is one of the primary design goals.

Flikkema et al. [5] describe the design and implementation of a cyber-physical system which utilizes an ultra low power sensing platform, streaming data middleware, and a control-oriented approach to performing complex ecological experiments. Several core principles from the previous works such as energy-aware network execution and abstracted application development through high-level programming languages have been designed into this platform. An important distinction between previously described works and the platform used in this thesis is that the WSN cyber-infrastructure is a network of networks. An individual cluster of nodes at a geographic location form a WSN, but behavior needs to be tracked and controlled across multiple WSN locations. This platform forms the foundation of the work of this thesis and will be described in greater detail in Chapter 3.

As the complexity of WSNs and diversity of applications increase, so increases researchers' need to reconfigure specific hardware components or WSN properties. A simple sensing network might only support unidirectional communication where the nodes pass their data to a base station. More sophisticated sensing networks support bi-directional communication where nodes both send and receive information. Bidirectional communication is a critical feature for any WSN system to support remote device reconfiguration and reprogramming. Depending on the size of a WSN supporting a particular application, a change could affect one node or even an entire cluster of nodes, depending on the type or magnitude of the change. For example, a researcher monitoring temperature fluctuations over a large geographic area with a cluster of WSN nodes might want to increase the sampling rate

to observe fine-scale environmental responses to a weather event. The difficulty of this task depends on how device reconfiguration was designed into the WSN platform and accompanying cyber-infrastructure.

Reprogramming and reconfiguration are often used interchangeably, as they might have the same effect on a researcher's experiment, but there is an important distinction between them. According to some interpretations, like e.g. [6], if changes are made exclusively to the software components, then the device has been reprogrammed. Alternatively, if there is some change in the device hardware, the device is said to have been reconfigured. For the work of this thesis, we will define reprogramming as involving transfer of new source code or application software to one or more devices, whereas reconfiguration implies that software parameters or hardware functionality can be altered without the addition of new program code. Ivester et al. [1] describe the role of control information in a WSN as a means for reconfiguration. Control information describes commands or network state information that are injected into a WSN, causing change in one or more nodes. Software called Ismanage used in [1] accompanies ISEE by providing several features, the most important of which is the ability to disseminate control information into the WSN for the purpose of reprogramming and reconfiguration. This approach is similar to the reconfiguration methodology used in WiSARDNet, the platform used for this thesis.

Eronu et al. [6] summarize many of the issues and challenges surrounding WSN reconfiguration. Given that WSN hardware is often limited in power availability, the energy overhead involved in WSN reconfiguration is a major concern, especially when the number of nodes to be reconfigured increases [6]. The authors of [7] describe a wireless data collection protocol named ORiNoCo, designed specifically to utilize the existing bidirectional messaging capabilities of a WSN for the purpose of energy efficient network reconfiguration. Efficient network reconfiguration is also a core feature of the Maté and MagnetOS platforms discussed in the previous section [3], [4]. Nodes that run the Maté byte-code converter execute capsules which contain small control sequences. In this way, new configurations can be encoded and transmitted to a WSN node. The target node, upon receiving the encoded configuration, can decode the capsule with an intermediate software layer that translates the capsule's sequence into specific values to be stored in the device's memory. The reconfiguration of nodes in the MagnetOS platform allows for software changes to be instigated within an image that runs on the Java Virtual Machine. The virtual machine then interprets the image and sends the appropriate byte-code values to each hardware device in the network. Reconfiguration of these nodes then becomes no more difficult than executing a new application. We can see that when the sensing hardware is made available as a service to higher layers as in [1], and bidirectional communication protocols are leveraged effectively, network-wide reconfiguration of many nodes can be achieved with low energy overhead.

2.3 Relational Databases for WSN Applications

Abstractions and unique design paradigms can guide the development of complex systems in ways that are simple to understand. Storing information in a database enables access to specific data via query statements. A database abstraction can be used to simplify the complexities of data collection from a WSN [8], [9], [10]. [8] uses a WSN design paradigm where the data from each node can be aggregated at a data sink, and the sink acquires the data from each node through SQL-like queries. The nodes act like individual databases, and therefore the complexity of distributed data collection can be managed through simple queries. In this way, data is acquired on demand rather than continuously streaming towards the sink, effectively reducing the amount of data transmitted to only what is desired as opposed to all available samples. The authors in [10] utilize this approach for data acquisition as well.

Database abstractions provide data archival, organization, and access features that support a variety of applications. These features can greatly improve the efficiency and usefulness of WSN systems by providing scalable storage solutions across a variety of platforms. The division of data into multiple tables which can be managed separately and linked via foreign key constraints allows complex yet versatile access by experimenters and analysts. These ideas also pair well with modern software design paradigms such as behavioral parameterization [11], where

methods can be written in such a way that they define different behaviors to be decided at runtime. Behavioral parameterization is a powerful design paradigm that will allow WSN data to be easily and accessed and utilized in the ever-changing landscape of IoT applications.

The aim of this thesis is the use of a database abstraction to simplify the complexity of network reconfiguration by storing network state information and other parameters that specify WSN execution, and utilizing this information to generate control messages for dissemination into the network for reconfiguration. Modular software design principles such as behavioral parameterization are used extensively in this thesis to accommodate rapid network reconfiguration as well as changing design requirements of future applications. This approach is described in further detail in Chapter 4.

Chapter 3

WiSARDNet Cyberinfrastructure

3.1 Overview

This chapter describes the platform on which the work of this thesis is built. WiSARDNet is a WSN platform which consists of hardware and software that connect sensor/actuator nodes to users. The National Science Foundation has adopted the term cyberinfrastructure to describe computational systems with advanced data acquisition, processing, and management capabilities. According to this definition, WiSARDNet can be thought of as cyberinfrastructure. Each network component is described in this chapter, as well as the data management and archival processes. The cyberinfrastructure hardware and software components can be divided into four categories: WSN, base station, real-time data center (RTDC), and end-user. Figure 3.1 shows how the hardware and software components for each of these

categories fit together. The core component which links the WSN, the base station, and the real-time data center is a TCP/IP based message broker that uses the MQTT protocol. MQTT will be described in further detail in this chapter. Additionally, database archival components will be described in terms of their role in the cyberinfrastructure in this chapter. The technical specifics of the database from a software perspective will be described in greater detail in Chapter 4.

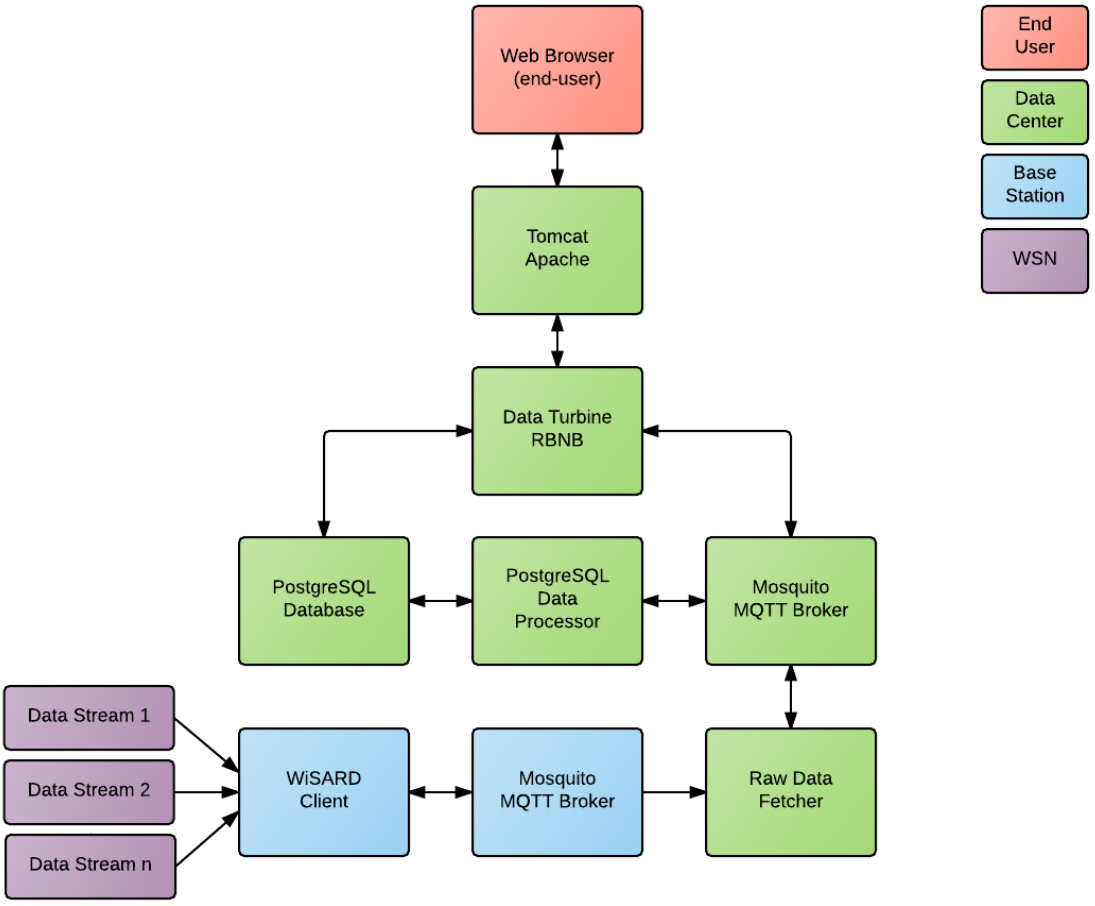

FIGURE 3.1: Cyberinfrastructure hardware and software component hierarchy

3.2 WiSARD Devices

The Wireless Sensing and Relay Device (WiSARD) is a sensor/actuator (S/A) device designed to be modular, adapting easily to the different sensing and actuation needs of users. Flexibility and energy awareness are the two driving design motivations in the WiSARD development. At the heart of each WiSARD device is a central processor (CP) which governs the operation of the device and its peripherals. The CP dictates when and how tasks will be scheduled and executed, establishes and manages wireless links to other WiSARDs, facilitates the storage of sensor readings, and offloads the sensing and actuation tasks to daughterboards called satellite processors (SP). A WiSARD can accommodate up to four different SPs. This enables each WiSARD to perform a variety of tasks specifically tailored to meet the needs of researchers and experimenters. The modular design allows the CP to offload sensing operations to its SPs which can execute tasks in parallel with CP operation.

Both CP and SP boards use an MSP430 16-bit ultra-low-power microcontroller by Texas Instruments. This microcontroller has a variety of features and also allows for the device to be placed in various low-power modes. The low-power modes of operation accompanied by energy conscious software design allow for a WiSARD to operate on a single pack of AAA batteries for many weeks or even months, depending on the rate at which the devices dispatch their sampling operations. The CP also connects to a radio board which uses an Analog Devices ADF7020 RF transceiver module that operates in the 902-928 MHz license-free

ISM band. When WiSARDs are powered up, they autonomously form a self-organizing and self-healing multi-hop wireless ad-hoc network. A special WiSARD which is assigned the software role of Hub acts as the base station, and is connected to an embedded Linux computer referred to as a garden server.

3.3 Middleware

Middleware is a term which describes a software abstraction of the transfer of information from point A to point B. By hiding the complexities of data transport behind a simple interface that connects two pieces of software, the development of powerful data processing and management tools can be accelerated. WiSARDNet relies heavily upon the use of middlware to make the WSN data available to the rest of the world.

The central architectural component that connects the garden servers to the RTDC is MQTT. MQTT is a data transfer protocol created by IBM that utilizes the publish/subscribe middleware paradigm. An MQTT broker can be installed on a computer or server and will then listen for TCP/IP connection requests. A data source on another device can publish data by connecting to the broker and sending data messages. WiSARDNet uses Mosquitto [12], an open source implementation of a message broker that uses the MQTT protocol.

A message broker installed on each garden server keeps all of the data from that site in persistent storage, which is a collection of archived data files on the hard disk. To transfer the data from the WiSARDs to the broker, a Java application referred to in WiSARDNet as a WiSARD client uses MQTT messages to publish data. The RTDC can then retrieve all of the data with a subscription client by connecting to the garden server's message brokers over cellular or satellite connections.

An MQTT subscription client for each garden site run at the RTDC and is responsible for subscribing to all the data published to each broker. These subscription clients are what is referred to in WiSARDNet as a raw data fetcher (RDF). The purpose of the RDFs is to retrieve all of the data from all of the gardens so that the data is accessible for processing, storage, or viewing. These processes are discussed in detail in the following section.

3.4 Real-time Data Center

The RTDC is a collection of servers which host a Tomcat Apache web server, a Mosquitto MQTT subscription client, an instance of an open-source data streaming middleware called Data Turbine, a PostgreSQL relational database, and all of the data management processes and applications which interact with the WSN. The RTDC has several functions, the first of which is to retrieve all of the WSN

data from each of the gardens. The RTDC is a centralized destination for all of the network's data streams. From this location, applications can access, store, and modify data for their various needs, as opposed to having to interact with multiple networks individually. For instance, a process which reads in raw data streams from temperature sensors might need to calculate human-readable values from the raw sensor readings, such as degrees Celsius. A single data converting processor which moves data from raw transducer values to human-readable values, can easily acquire all of the raw data streams for temperature sensors at all network locations with a single fetch, rather than requiring that the process fetch the raw data from each specific garden individually. Additionally, aggregating all data at the RTDC greatly simplifies archival and backup procedures.

3.4.1 Relational Database

Relational databases provide a flexible way to store and access large amounts of data. A relational database is composed of one or more tables; tables are 2 dimensional matrices of rows and columns. According to Rockoff [13], rows and columns are referred to as records and fields, respectively. An entry in a database table occupies a single row and each field corresponding to a different attribute of that entry constitutes a new column. For example, a table which stores people might have columns for an identification number, a name, an address, and a telephone number. An entry of a new person into this table would occupy a single row,

with the data matching each of the attributes would occupy the rows' intersections with each column. Formally, these columns are referred to as primary keys, as they map individual fields to the entries which they belong. The database at the RTDC has numerous tables which store information about every device, every garden location, and the sensor readings themselves.

In WiSARDNet, data streams are archived in a PostgreSQL table. Additional tables in this database store all of the meta-data and logistical information regarding all of the WSN devices, as well as every data point sampled from any of the gardens. Additionally, there are many data streams that also need to be tracked and archived such as diagnostic control data, error reporting, and other useful information that help with network upkeep. An example of diagnostic control data might be the logging of a device restart event; such events can be crucial in detecting and analyzing network performance issues.

At the RTDC, the PostgreSQL database is physically located on its own server, separate from the other data management and processing clients. By placing the database on a separate server, the database has exclusive access to dedicated processing resources to allow for the quickest possible query and insert response times for the the data schema and the queries used. Another function that the RTDC performs is the execution of applications and processes which interact with the WSNs and their data. One example of an application running on a server

at the RTDC might be a researcher performing an experiment which attempts to maintain equal soil moisture readings at two different geographic locations. The RTDC servers possess high-performance computing hardware such that many applications and processes can use to interact with the networks in real-time. By executing these applications and processes at the RTDC, they will operate at optimal performance, minimizing latency from performing queries and calculations to preserve as close to a real-time experience as possible.

3.4.2 Data Schema

WiSARD hardware components are subject to a design hierarchy which specifies how the different pieces organize and interact in operation. Figure 3.2 shows the hierarchy of the WiSARD hardware components.

In a similar manner, the data gathered in WiSARDNet follows a design hierarchy that specifies how data is organized. The organizational structure of the the data tables and their relations, or the data schema as it is commonly referred to, was specifically designed to best accommodate the ever-changing configuration status of WiSARDNet devices. Every piece of hardware is referred to in the data schema as a device, and therefore the primary table in the data schema is the Device table. A record in the device table will describe any device of any type. For instance, a radio, a soil-moisture transducer, a satellite processor, or a garden server are all devices that have a record in the device table. The configuration of these devices, however, might change at some point in time; it might be moved

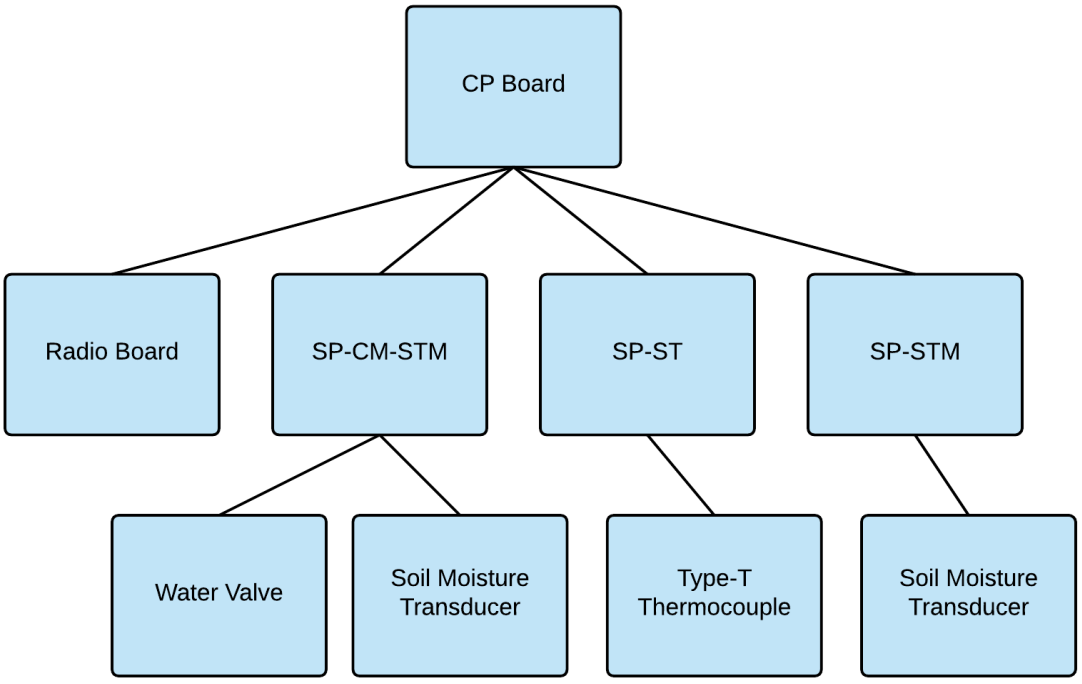

FIGURE 3.2: Device Relationship Structure

to another location, it might be upgraded or replaced, or it might receive a new firmware version. In this schema, the device's software parameters, hardware configuration, and state of operation are all encapsulated in a Deployment. Using the hierarchical structure of the WiSARD hardware, we use a tree structure in grouping devices together through their deployment records. Figure 3.3 shows how devices and deployments are structured in the database. Each deployment record has a field for a parent device deployment.

If a particular transducer is attached to a satellite processor, then its active deployment record references the satellite processor's deployment record in the parent deployment field. These are recorded and tracked through a deployment table where each record is a particular deployment that references a device record

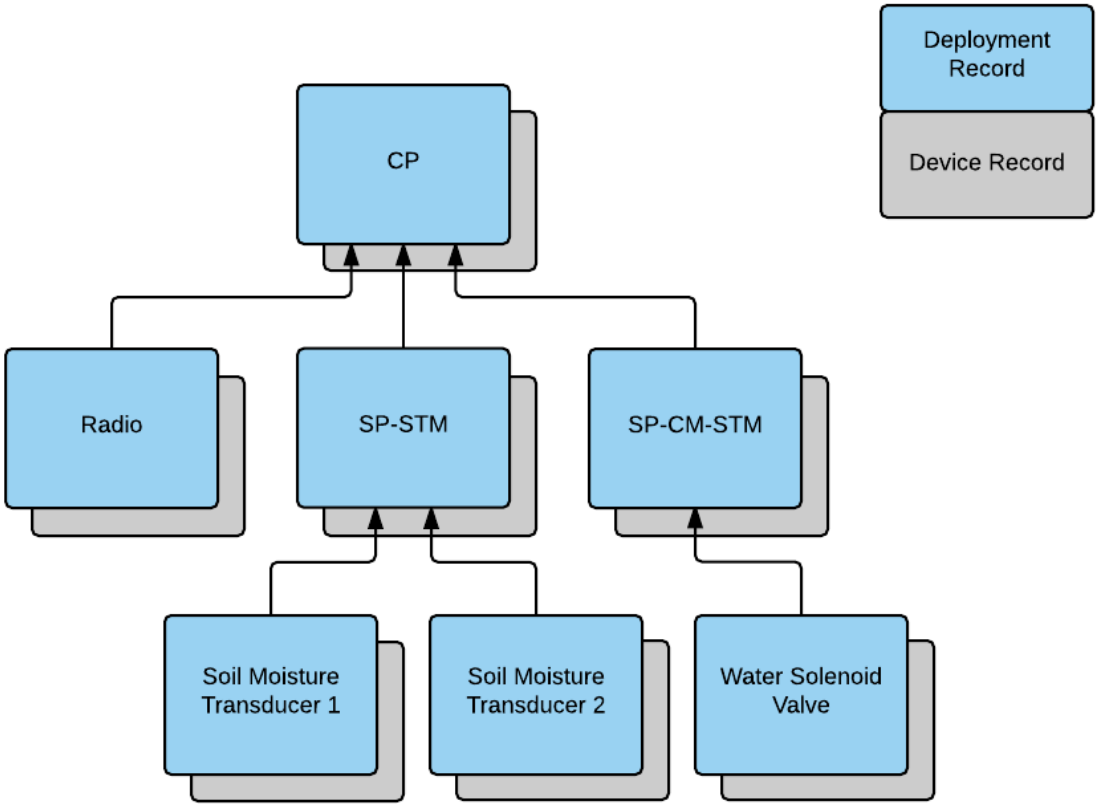

FIGURE 3.3: Deployment Relationship Structure

in the device table. When a change is made to any of the parameters tracked in the deployment table, the device is thought of as having a new deployment, and therefore a new record is inserted into the deployment table, and the device record is referenced by the new deployment. The key field in the deployment table is a binary field which stores a true/false value of whether or not the deployment is active. Once a deployment is changed, the old deployment becomes inactive and the new deployment becomes active. Over time, many deployment records may accumulate for a particular device as changes are made; the entire history of the device is stored so that for any given data point, the deployment configuration of the device associated with that data sample is accessible.

Having the ability to store and track each device's configuration and related meta data in the database is of great value in the management of the networks and the various experiments. When a WiSARD is ready for deployment at a garden site, a deployment record is created with all of its configuration data, the deployment record is set to active, and the start date and time is set. As long as the device remains in this configuration, the deployment object pointing to that device will remain active. If a setting changes or if a sensor fails and is replaced for instance, the following actions are taken:

1. The deployment record to which the failed sensor device points is set to inactive.
2. The current time is inserted into the stop-time field of the deployment record
3. The replacement sensor is added as a record to the device table
4. A new deployment record is inserted into the deployment table
5. The parent field of the new deployment record references the deployment record of the SP device to which it is installed
6. The active status field of the new deployment record is set to active
7. The current time is inserted into the start-time field of the new deployment record

By following this procedure, previous device configurations are current configurations are easily accessible from the database as well as any previous configurations and the the time period which the were active. Active and inactive deployment tracking through a single record addition for each instance of a configuration change is trivial with regards to computational complexity and storage resources. This approach is simple, intuitive, and scales well as the number of devices deployed in the field increase. With this system in place for accessing up-to-date configurations of any registered device, there is now a foundation in place for fully describing every single data point which is sampled from any transducer or device. With this paradigm, a data sample is not merely a sample from a device, it is a sample from a specific deployment of a device. When a device's configuration is altered, producing a new deployment for that device, the data generated by that device becomes a new stream.

When new data from WiSARDs arrive at the RTDC from the MQTT broker, they need to be accessible for users and other services to request. Data Turbine's network accessible ring buffer data structure works well for this purpose. At the RTDC, the data streams arriving via MQTT are placed into Data Turbine's ring buffer in accordance with the stream name which identifies the device from which the sample was taken. Since the complexities of WiSARD configurations are handled via the device, devicetype, and deployment tables, archival of sampled data in the database becomes a simple procedure. All samples from all streams

are placed as individual records in a table named data. In addition to the sampled value and the time at which it was sampled, the sample via its Data Turbine stream name can be mapped to the specific device and deployment records for which the deployment was active. In this way, data archival of acquired samples is extremely simple and all information regarding a device and its samples is easily accessible and intuitively obtained through the thoughtful design of the data schema.

3.5 Summary

WiSARDNet is a WSN CI which was designed from the ground up to be modular, scalable, and accessible to users. The Mosquitto MQTT broker, the data streaming middleware Data Turbine, and the RTDC enable data sampled from sensors to be retrieved, managed, processed, and archived into a PostgreSQL database. The way in which WiSARD meta-data and configuration information is stored and accessed is critical for the development of a network configuration software which comprises the work of this thesis. How these features are utilized in the network management software is described in further detail in Chapter 4.

Chapter 4

Network Reconfiguration Software

4.1 Overview

This chapter discusses the software components developed in this thesis. First, this chapter discusses the procedures involved in network reconfiguration. The rest of the chapter explains how those procedures are implemented in software. The software discussion is broken into two sections. The first section describes the different architectural components which perform each operation in the reconfiguration sequence of events. The other section discusses the specific software architecture and modules which achieve the system operations.

4.2 Network Reconfiguration Stages

The reconfiguration of a WSN on this platform is accomplished through a sequence of operations:

- Specify a set of WSN nodes to reconfigure
- Specify the details of the desired changes to be made
- Validate that the desired changes are allowed and do not create conflicts
- Signal the network to perform the specified changes
- Alert the user to the status of the changes

A practical example might be the scenario where a researcher would like all soil moisture transducers attached to WSN nodes in a specific region to be sampled at twice their current sampling rate. Performing this procedure would involve the user specifying the region and sensor types needed to produce a matching set of WSN nodes and that they would like the sampling rate of those transducers to be changed. The software would need to validate that there are no conflicts between that user's requested changes and the existing operational configurations of other WSN nodes. Finally, the nodes would each be signaled in their respective WSNs via command packets and would then alert the user of the changes made.

4.2.1 Identification of WSN Nodes

The first stage in reconfiguring a WSN is the specification of the node or nodes whose configurations will be modified. This operation can be completed by querying the network state information stored in PostgreSQL database tables. Transducer samples, sensor operational data, software profile information, and hardware meta-data are all stored within the database and are necessary to determine which nodes will need to be reconfigured.

4.2.2 Describing the New Configuration

A WiSARD's task execution is governed by a set of configuration parameters stored within the device's non-volatile memory; this area of memory is called the task control block. Changing a device's behavior is achieved by overwriting specific values within the task control block, with values that represent different behaviors. The user must describe the changes to the system so that the correct fields in the task control block can be overwritten with appropriate values corresponding to the new behaviors. This is achieved by encoding different behavior or behavior profiles into control sequences which the WiSARDs are able to decipher.

4.2.3 Validating the New Configuration

As the number of configuration changes and the number of affected WSN nodes increases, the opportunity for error or conflicts of interest between researchers and their assets also increases. To prevent conflicts, there is a need to validate that

the intended changes will not interfere with the execution of other nodes in the network. This is achieved with user access control and command validation. User access control is the process of restricting the access of users to specific pieces of hardware or software based on a set of rules and permissions that govern the extent to which they can interact with the WSN. For example, a researcher should not be allowed to reconfigure another researcher's hardware or reconfigure the WSN in a way that will adversely affect other experiments. Command validation is the process of analyzing the intended changes and the state of the network, determining whether or not the the command is feasible as well as making sure the user or process issuing the command has the appropriate permissions or membership access to the affected hardware.

4.2.4 Signaling the WSN

Once the commands are created, the specified nodes within the WSN need to be made aware of the desired changes. The command packets are added to MQTT messages and flushed from the MQTT broker at the data center to the destination garden server or servers. Each garden server has a WiSARD client which will retrieve the command packet from the local MQTT instance and send it into the WSN via the hub node.

4.2.5 Executing the Reconfiguration

When a WiSARD receives a command message from a parent node, the WiSARD parses the command message and schedules a task to execute the command. Each WiSARD schedules tasks and executes them at the appropriate time. When a WiSARD executes a command, a report message is created and sent back through the network to the RTDC. The report message will confirm that the task was successfully executed or report that it failed, if the task did not execute successfully.

4.2.6 Verifying the New Configuration

Report messages that arrive at the RTDC are treated as data streams, the same way sensor readings or diagnostic data are. Reports are archived and made accessible

After a change has been made and the report message has been sent back to the RTDC, these reports can be pulled and checked to verify that the changes specified by the user have been applied. After making a change, a WiSARD will report its status which then be reflected in the user's view of the network within their session once the data arrives back at the data center.

4.3 Software System

The WiSARDNet cyberinfrastructure was designed to support end-to-end full-duplex communication, allowing for complex features and flexibility to be provided to users and other system components. This however adds complexity to the set

of procedures required for network reconfiguration. This section gives a high-level systems-oriented overview of two subsystems that directly relate to network reconfiguration: user access control and the admin control subsystem.

4.3.1 User Access Control

The user access control subsystem is a software component which regulates the ability of different users to perform configuration changes on the WSNs within the WiSARDNet cyberinfrastructure. This subsystem acts a service which can provide validation of a specific user's requests, regulating their ability to proceed with actions they do not have permission to execute. It also interfaces with multiple other sybsystems. First, it needs to interface with the user interface subsystem; it filters a user's view of a WSN to only include nodes which a user has at least 'read' access for. Additionally, it needs to interface with the web portal admin control subsystem, which provides a suite of administrative tools that allow a user to view a WSN and its current configuration, signal changes to the WSN, and modify a WSN. By building this subsystem in such a way that it acts as a service to the other subsystems, it provides a generalized and scalable approach to controlling and regulating new subsystems and features which may be developed in the future.

4.3.2 Admin Control

A core design philosophy of WiSARDNet is that it is a resource to enable researchers to effectively utilize and manage complex experimentation. The admin

control subsystem describes the set of tools a user has to view and manipulate WSNs and experiments to directly enable this design philosophy. This set of tools is governed by the user access control subsystem and therefore can provide a different user experience depending of the individual logged into the system. Each tool presents the user a set of features that enable WSN reconfiguration.

The first tool is the WiSARD browser. This allows a user to query the network information tables within the database and see the current configuration of every WiSARD that they have access to view. WiSARDs are identifiable based on many different search criteria such as hardware identification numbers, SP board types, or deployment locations. It is important for a user to be able to filter lists of WiSARDs according to these criteria because the scope of desired changes may depend entirely upon a network's state with respect to that variable. For example, if a researcher wants to modify nodes between multiple sites equipped with a specific type of sensor then they need to be able to identify the set of nodes which match that criteria quickly to make the desired change.

The second tool provided within the admin control subsystem is the WSN reconfiguration tool. This tool integrates with the WiSARD browser by taking a set of WiSARDs which the user specified and taking action to reconfigure them. This tool integrates with the user access control subsystem as well, because the WSN needs to be protected from changes that could adversely affect nodes and

experiments outside of a user's ownership. The user access control subsystem filters which actions that the user is allowed to take in reconfiguring the sets of WiSARDs they retrieved from the WiSARD browser. Once a user signals an authorized action to be taken towards reconfiguring a WSN, this tool will build the messages that need to be sent to the affected WSNs where the commands can be scheduled and executed. At a high level, this process seems fairly straight forward, but actually accomplishing this within WiSARDNet requires a complicated set of interactions within the cyberinfrastructure. These interactions are described in detail in the following section.

4.4 Software Modules

The user access control and admin control subsystems provide users with the ability to observe and reconfigure WSNs but the software which implements these features relies heavily on a set of other software modules and services. This section discusses each software module in the system and how the features described in the previous section operate in practice. Figure 4.1 shows the software architecture of WiSARDNet that represents the work of this thesis.

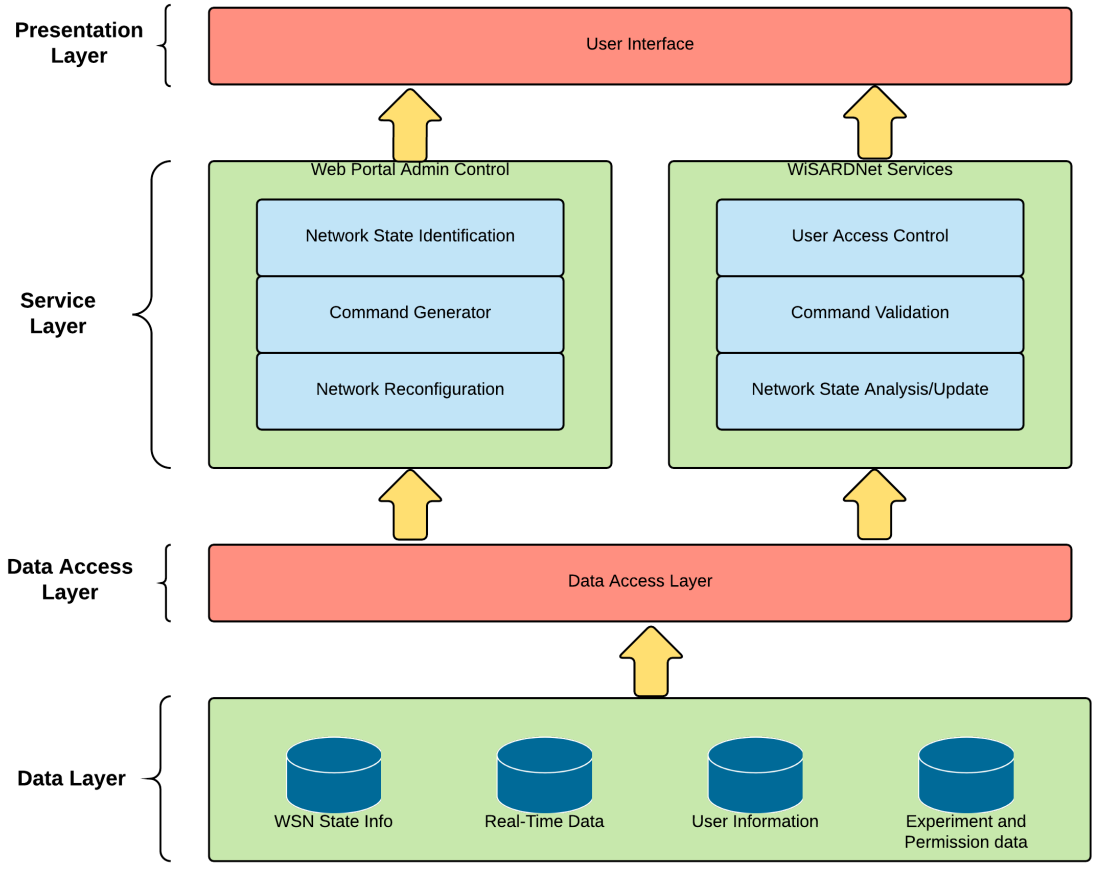

FIGURE 4.1: A visual representation of the software architecture that organizes the different software modules

4.4.1 User Interface

A user interface for the WiSARD browser and the configuration tool exists on a dynamic web page within the WiSARDNet web portal. This user interface is built with JavaServer Pages (JSP) and Java servlet technology which Apache Tomcat facilitates. JSP was chosen as the development platform for these tools because a large portion of the back-end cyberinfrastructe code was written in Java. Integration of new modules and application features into the existing system is significantly less difficult in this instance if they are written in Java. A jsp page

with some javascript defines a single user interface for the user to use the WiSARD browser and network configuration tool. Though the graphical user interface (GUI) for these tools exists within a single JSP page which defines the HTML code displayed by the user's browser, it interfaces with several servlets which form the back-end modules.

A user that uses the GUI will begin by entering values to populate a series of forms in a JSP page with the search criteria that will help filter a set of WiSARDs from all of the WSNs. The user then submits their form to a servlet which will return the list of WiSARDs to the JSP page. The procedure for specifying commands to execute and performing the reconfiguration follow the same pattern; the user fills out a form, sends the form to the servlet, and the servlet responds with a set of results which update the JSP page. The servlet directly handles getting information to and from the user via the JSP page, as well as delegating all of the complex computation operations to other modules. Figure 4.2 demonstrates how Apache Tomcat facilitates the interaction between the JSP page and the different servlets.

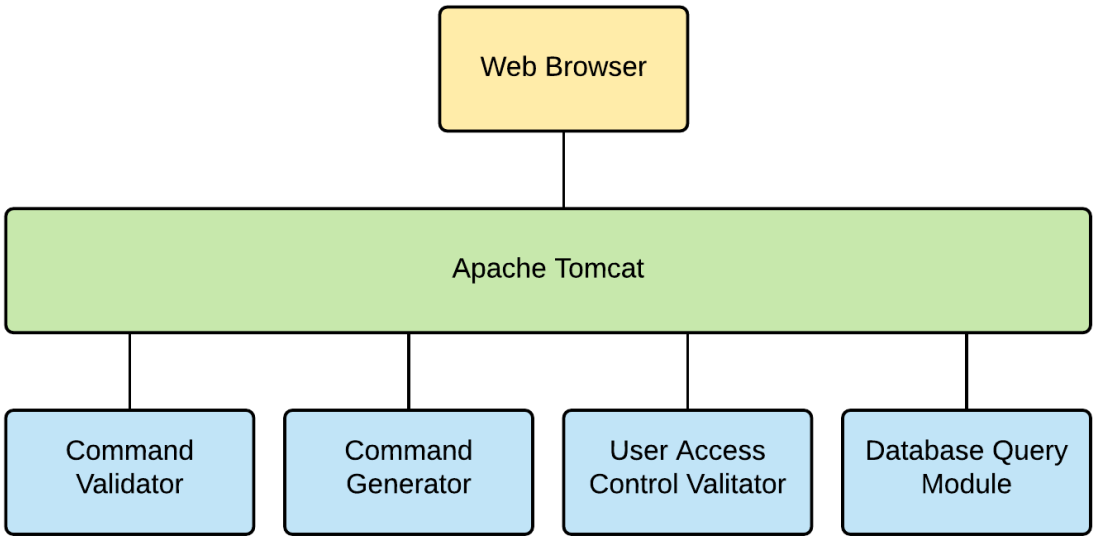

FIGURE 4.2: The organizational structure of the manner in which Apache Tomcat facilitates the interaction between the web browser and the servlet classes which provide critical application features

4.4.2 Database module

The database module at its most basic level is responsible for executing queries to the PSQL WiSARDNet database and returning data to the modules that use it. The database contains many different tables that each serve a vital purpose outside of simply storing transducer samples. All of the network state information that describes the network, user access and permission data, and CI networking information is stored in the database as well, making this a very robust and versatile module.

4.4.2.1 User Access Control

A large part of the user access control subsystem is managed by several tables in the PSQL database. The data schema for permissions was designed to be generic such that permissions can be applied to hardware, experiments, and software tools within the WiSARDNet web portal, as well as any future pieces of hardware and software that may be incorporated into WiSARDNet in the future. This was achieved by creating multiple abstractions that generalize users and WiSARDNet assets into two categories: permission entities and permission resources. A permission entity is a concept which describes any user, group, or organization that can be given permission to perform a task. A permission resource is a concept which describes any piece of hardware or software that an entity would need permission to use. With permission entities and permission resources clearly defined, a permission table with a single record maps an entity to a resource with a permission level. By implementing user access control in this manner, there is now a generalized way to grant permissions from a variety of entities to any kind of resource. Figure 4.3 shows the references that connect each database table through foreign key constraints.

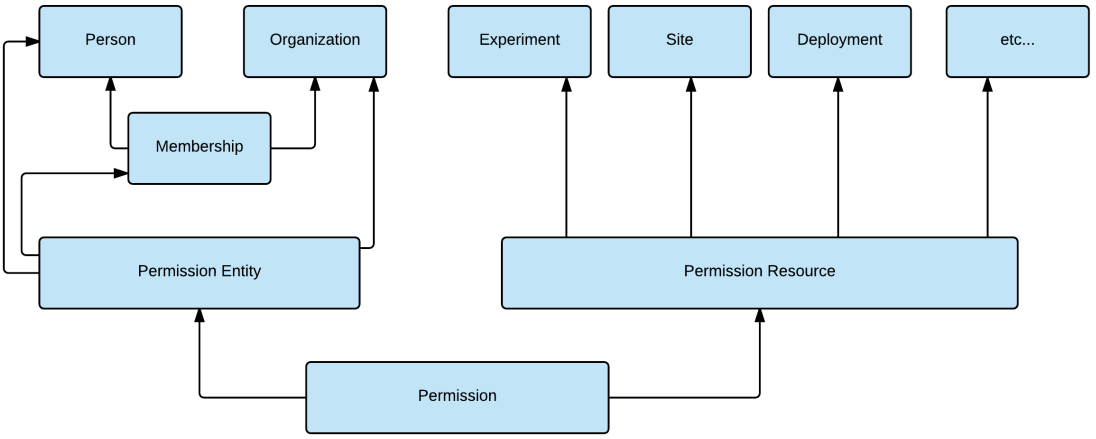

FIGURE 4.3: A visual representation of the PSQL tables and their references which support the user access control paradigm

4.4.2.2 WiSARD Data Objects

Chapter 3 discussed how device and deployment tables managed the specific implementation of a WSN node or a piece of hardware deployed into the field. A deployment record exists to record a specific instance of the device which it references. Data collected from one deployment of a device is a conceptually different piece of data than samples from another another deployment of that same device. WSN reconfiguration requires altering WiSARDs at the node level as individual transducers cannot be accessed directly. For this reason, the database module needs a way to select all of the relevant WSN node information as a WiSARD data object from the different PSQL tables. From the perspective of this data schema, a WiSARD can be thought of as a collection of active device deployments. Figure 4.4 shows how a WiSARD data object is created from the information stored in other tables.

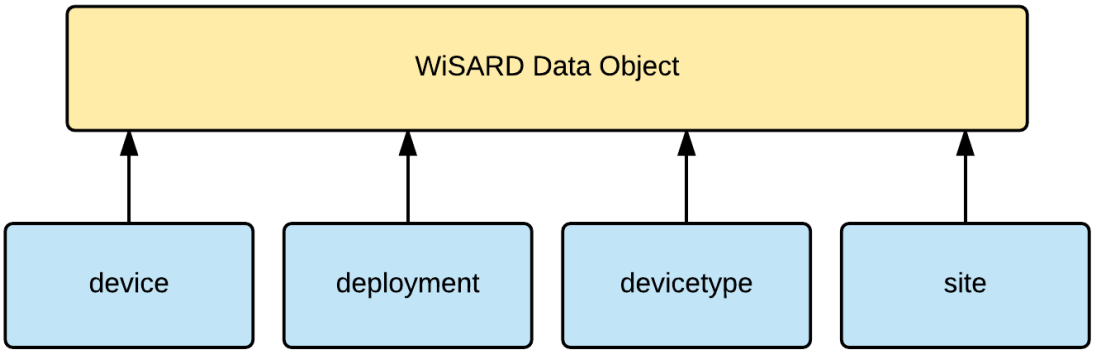

FIGURE 4.4: A visual of the database abstraction which simplifies the creation of WiSARD data objects

4.4.3 Command generation Module

Once a set of WiSARD data objects has been identified and the user has chosen which changes they would like to be made to the WSN, the command generation module needs to assemble command packets with information and task parameters that will allow WiSARDs to interpret. This is the software module which allows WSNs to be thought of as databases of heterogeneous behaviors. Different profile information describing new tasks can be queried from the database, updated with network state information, and then packetized.

4.4.4 Validation Module

Following the creation of the commands but prior to sending them to the WSN, each command needs to be validated to ensure the user has adequate permissions to modify the affected nodes and that the commands are correct for the given set of WiSARDs. A user should not be able to send commands to modify nodes

in ways they aren't capable of supporting or will negatively affect the WiSARDs without appropriate privileges. For example, a user should not be allowed to send a command that will change the WiSARD's software role to be a hub node, since hub nodes require special hardware connectivity and setup. Sending a command to perform that action would cause that node to stop sampling from its transducers and would orphan the node from the network such that no other commands can be sent to it. The validation module checks for potentially harmful commands such as this are filtered and prevented by the validation module before they are sent out to the WSN nodes.

4.4.5 Network Configuration Module

The final module that completes the admin tool subsystem is the network configuration module. When a set of commands has been created by the command generation module and verified by the validation module, the network configuration model determines which garden servers commands need to be sent to and prepares them to be sent. Each garden server is accessible either via cellular or satellite modem, and therefore a separate TCP/IP connection must be made to each garden server separately. Getting the commands from the data center to the gardens servers relies heavily upon the Moquito MQTT broker. The software creates a new publisher client at the data center and encapsulates each command within an MQTT data message. The publisher client then publishes those messages to the MQTT broker of the garden server which the affected WSN resides.

This process is then repeated for each garden server until all WiSARDs have been reconfigured.

4.5 Summary

Remote network reconfiguration is a complicated procedure and the WiSARDNet CI has many interacting software modules which make this possible. Efficient use of component abstractions, such as treating a WSN as a database of heterogeneous behaviors, helps to manage the complexity of complicated reconfiguration procedures.The different subsystems and software modules all interact to form a robust system that allows for a variety of control procedures which enable researchers to engage in complex remote experimentation.

Appendix

Appendix content goes here

Bibliography

[1] M. Ivester and A. Lim, "Interactive and extensible framework for execution and monitoring of wireless sensor networks," in *2006 1st International Conference on Communication Systems Software & Middleware*, pp. 1–10, IEEE, 2006.

[2] M. Hammoudeh, S. Mount, O. Aldabbas, and M. Stanton, "Clinic: A service oriented approach for fault tolerance in wireless sensor networks," in *Sensor Technologies and Applications (SENSORCOMM), 2010 Fourth International Conference on*, pp. 625–631, IEEE, 2010.

[3] P. Levis and D. Culler, "Mat: A tiny virtual machine for sensor networks," *ACM Sigplan Notices*, vol. 37, no. 10, pp. 85–95, 2002.

[4] R. Barr, J. C. Bicket, D. S. Dantas, B. Du, T. Kim, B. Zhou, and E. G. Sirer, "On the need for system-level support for ad hoc and sensor networks," *ACM SIGOPS Operating Systems Review*, vol. 36, no. 2, pp. 1–5, 2002.

[5] P. G. Flikkema, K. R. Yamamoto, S. Boegli, C. Porter, and P. Heinrich, "Towards cyber-eco systems: Networked sensing, inference and control for distributed ecological experiments," in *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on*, pp. 372–381, IEEE, 2012.

[6] E. Eronu, S. Misra, and M. Aibinu, "Reconfiguration approaches in wireless sensor network: Issues and challenges," in *2013 IEEE International Conference on Emerging & Sustainable Technologies for Power & ICT in a Developing Society (NIGERCON)*, pp. 143–142, IEEE, 2013.

[7] A. Reinhardt and C. Renner, "Remote node reconfiguration in opportunistic data collection wireless sensor networks," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, pp. 1–3, IEEE, 2014.

[8] S. Madden, "Database abstractions for managing sensor network data," *Proceedings of the IEEE*, vol. 98, no. 11, pp. 1879–1886, 2010.

[9] O. Diallo, J. J. Rodrigues, M. Sene, and J. Lloret, "Distributed database management techniques for wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 2, pp. 604–620, 2015.

[10] L. D. Chagas, E. P. Lima, and P. F. R. Neto, "Real-time databases techniques in wireless sensor networks," in *Networking and Services (ICNS), 2010 Sixth International Conference on*, pp. 182–187, IEEE, 2010.

[11] R.-G. Urma, M. Fusco, and A. Mycroft, "Java 8 in action," 2014.

[12] "Mosquitto: An open source mqtt broker." https://mosquitto.org/, 2012.

[13] L. Rockoff, *The language of SQL*. Nelson Education, 2010.

[14] T. AbuHmed, N. Nyamaa, and D. Nyang, "Software-based remote code attestation in wireless sensor network," in *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, pp. 1–8, IEEE, 2009.

[15] A. Antola, A. L. Mezzalira, and M. Roveri, "Ginger: a minimizing-effects reprogramming paradigm for distributed sensor networks," in *Robotic and Sensors Environments (ROSE), 2014 IEEE International Symposium on*, pp. 88–93, IEEE, 2014.

[16] A. Caracas, T. Kramp, M. Baentsch, M. Oestreicher, T. Eirich, and I. Romanov, "Mote runner: A multi-language virtual machine for small embedded devices," in *Sensor Technologies and Applications, 2009. SENSORCOMM'09. Third International Conference on*, pp. 117–125, IEEE, 2009.

[17] P. D. Felice, M. Ianni, and L. Pomante, "A spatial extension of tinydb for wireless sensor networks," in *Computers and Communications, 2008. ISCC 2008. IEEE Symposium on*, pp. 1076–1082, IEEE, 2008.

[18] D. He, C. Chen, S. Chan, and J. Bu, "Sdrp: A secure and distributed reprogramming protocol for wireless sensor networks," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 11, pp. 4155–4163, 2012.

[19] C.-M. Hsieh, Z. Wang, and J. Henkel, "Eco/ee: Energy-aware collaborative organic execution environment for wireless sensor networks," in *2012 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1998–2002, IEEE, 2012.

[20] C.-H. Hsueh, Y.-H. Tu, Y.-C. Li, and P. H. Chou, "Ecoexec: An interactive execution framework for ultra compact wireless sensor nodes," in *2010 7th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pp. 1–9, IEEE, 2010.

[21] H. Hu, J. He, and J. Wu, "Distributed processing of approximate range queries in wireless sensor networks," in *Parallel Architectures, Algorithms and Programming (PAAP), 2015 Seventh International Symposium on*, pp. 45–51, IEEE, 2015.

[22] C. Jardak, P. Mhnen, and J. Riihijrvi, "Spatial big data and wireless networks: experiences, applications, and research challenges," *IEEE Network*, vol. 28, no. 4, pp. 26–31, 2014.

[23] L. Jiang, L. D. Xu, H. Cai, Z. Jiang, F. Bu, and B. Xu, "An iot-oriented data storage framework in cloud computing platform," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1443–1451, 2014.

[24] Y.-S. Kang, I.-H. Park, J. Rhee, and Y.-H. Lee, "Mongodb-based repository design for iot-generated rfid/sensor big data," *IEEE Sensors Journal*, vol. 16, no. 2, pp. 485–497, 2016.

[25] S. Sarangi and S. Kar, "A scriptable rapid application deployment framework for sensor networks," in *Advances in Computing, Communications and Informatics (ICACCI), 2013 International Conference on*, pp. 1035–1040, IEEE, 2013.

[26] M. Szczodrak, O. Gnawali, and L. P. Carloni, "Dynamic reconfiguration of wireless sensor networks to support heterogeneous applications," in *2013 IEEE International Conference on Distributed Computing in Sensor Systems*, pp. 52–61, IEEE, 2013.

[27] R. Tompkins, T. B. Jones, R. E. Nertney, C. E. Smith, and P. Gilfeather-Crowley, "Reconfiguration and management in wireless sensor networks," in *Sensors Applications Symposium (SAS), 2011 IEEE*, pp. 39–44, IEEE, 2011.

[28] V. Vaidehi and D. S. Devi, "Distributed database management and join of multiple data streams in wireless sensor network using querying techniques," in *Recent Trends in Information Technology (ICRTIT), 2011 International Conference on*, pp. 583–588, IEEE, 2011.

[29] J. L. Wilder, V. Uzelac, A. Milenkovic, and E. Jovanov, "Runtime hardware reconfiguration in wireless sensor networks," in *2008 40th Southeastern Symposium on System Theory (SSST)*, pp. 154–158, IEEE, 2008.