A SENSOR WEB SERVICE FRAMEWORK TO ENABLE REALTIME INFORMATION SHARING

By

HUNG ANH MA

A thesis submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WASHINGTON STATE UNIVERSITY VANCOUVER
School of Engineering and Computer Science

AUGUST 2009

To the Faculty of Washington State University Vancouver:

The members of the Committee appointed to examine the thesis of HUNG ANH MA find it satisfactory and recommend that it be accepted.

_____
Chair

_____

_____

_____

ACKNOWLEDGEMENT

I would like to express my gratitude to my advisor, Professor WenZhan Song, for his guidance, inspiration and supervision of this thesis. Thanks for his useful comments to improve my thesis.

I would also express sincere appreciation to all the members on our team, who gave me so many good suggestions when I did the implementation part for this thesis.

Many thanks to all my friends at WSU. Without their generous help, continuous encouragement and moral support, this work could not have been completed.

Hung Anh Ma

A SENSOR WEB SERVICE FRAMEWORK TO ENABLE REALTIME INFORMATION SHARING

Abstract

by Hung Anh Ma, M.S.
Washington State University Vancouver
August 2009

Chair: WenZhan Song

A Sensor Web is a coordinated observation infrastructure composed of a distributed collection of resources, e.g. sensors, platforms, models, communications infrastructure, that can collectively behave as a single, autonomous, task-able, dynamically adaptive and reconfigurable observing system that provides raw and processed data, along with associated metadata, via a set of standards-based service-oriented interfaces. The definition of the term, sensor, is intentionally broad and abstract to include a wide range of data and/or information providers. This definition, for example, includes models and not just physical instruments capable of sensing a phenomenon. Human reports, radar, satellite feeds, models, and thermometers are all examples of sensors within the context of a sensor web.

This thesis proposes a Sensor Web Service Framework to enable information sharing program in peer-to-peer (P2P) manner. It provides the interface for the data providers to publish their data, also provides the interface for the clients to set alert on their desired data and to query/view those data. As the proof of concept, I will use it for managing two types of data: the online merchandise database and the volcanic datastream including seismic and infrasonic. This work is a first step to providing an intelligent backbone as the core of the next Internet. An implicit contribution of this project is to provide realtime and situation-aware information sharing services over the Internet in the P2P manner.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER ONE

INTRODUCTION

Wireless Sensor networks (WSNs) are a special category of ad-hoc wireless networks. They are highly distributed networks of small, lightweight wireless nodes, deployed in large numbers to monitor the environment or system by the measurement of physical parameters such as temperature, pressure, or relative humidity. Originally, the development of wireless sensor networks was motivated by military applications such as battlefield surveillance and monitoring and detection of attack by chemical, biological, or nuclear weapons.

Due to the rapid development of sensor technology, current sensor nodes [18] are much more sophisticated in terms of CPU, memory, and wireless transceiver. As the result, wireless sensor networks are now used in many industrial and civilian application areas, including industrial process monitoring and control, machine health monitoring, environment and habitat monitoring [13] [24] [19] [14], healthcare applications [23], home automation [21], and traffic control (Figure 1.1).

1.1 Motivation

1.1.1 Necessity of a Sensorweb Service Framework

Many sensor network applications have been successfully developed and deployed around the world. Some concrete examples include:

- Great Duck Island Application [24]: as Mainwaring et al., 2002 stated, 32 motes are placed in the areas of interest, and they are grouped into sensor patches to transmit sensor data to a gateway, which is responsible for forwarding the data from the sensor patch to a remote base station. The base station then provides data logging and replicates the data every 15 minutes to a Postgress database in Berkeley over a satellite link.

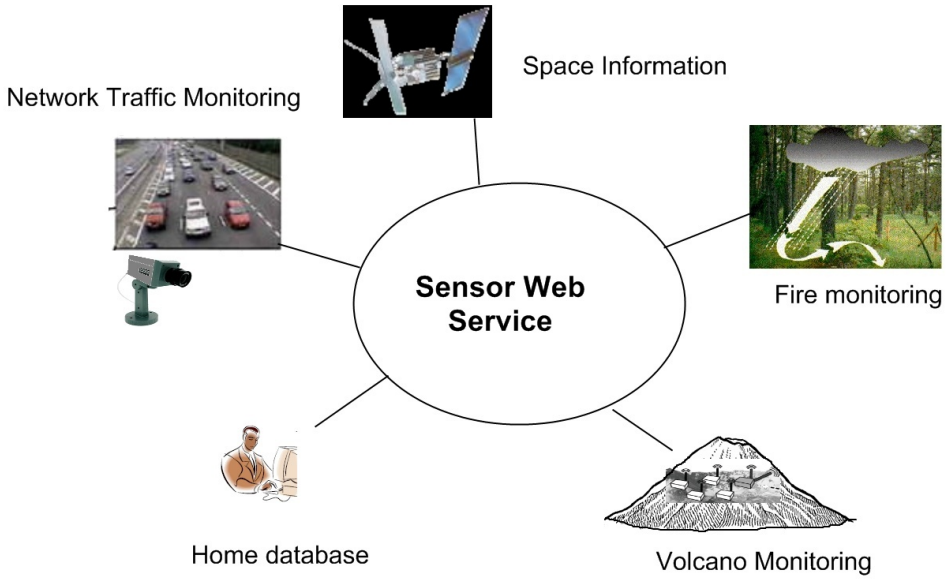

Figure 1.1: Conceptual View Of Sensorweb Service.

- Cane-toad Monitoring Application [8] [16]: two prototypes of wireless sensor networks have been set up, which can recognize vocalizations of at maximum 9 frog species in Northern Australia. Besides monitoring the frogs, the researchers also plan to monitor breeding populations of endangered birds, according to Hu et al., 2003.

- Soil Moisture Monitoring Application [15] [9]: Cardell-Oliver et al., 2004 presents a prototype sensor network for soil moisture monitoring that has been deployed in Pinjar, located in north of Perth, WA. The data is gathered by their reactive sensor network in Pinjar, and sent back to a database in real time using a SOAP Web Services.

However, none of these applications address the ability for interoperability which means that users cannot easily integrate the information into their own applications. Moreover, the lack of semantics for the sensors that they have used makes it impossible to build a uniform Web registry to discover and access those sensors. In addition, the internal information is tightly coupled with the specific application rather than making use of standard data representations, which may restrict

the ability of mining and analyzing the useful information. Therefore, it is important to develop a Sensor Web Service Framework to unify the information from different sensor network application.

1.1.2 Challenges for developing of Sensorweb Service Framework for WSNs

The ability of the sensor networks to collect information accurately and reliably enables building both real-time detection and early warning systems. In addition, it allows rapid coordinate responses to threats such as bushfires, tsunamis, earthquakes, and other crisis situations [10]. However, the heterogeneous features of sensors and sensor networks turn the efficient collection and analysis of the information generated by various sensor nodes into a rather challenging task due to the following reasons:

- There is no uniform operations and standard representations for sensor data that can be used by diverse sensor applications.
- There exists no means to achieve resource reallocation and resource sharing among applications as the deployment and usage of the resources has been tightly coupled with the specific location, sensor application, and devices used.

Beside the two main challenges of dealing with the heterogeneous features of sensors and sensor networks, the wireless sensor network (WSN) itself also poses many scientific challenges due to the following [30]:

- Sensor networks are infrastructure-less. Therefore, all routing and maintenance algorithms need to be distributed. Sensor nodes should be able to synchronize with each other in a completely distributed manner.
- An important bottleneck in the operation of sensor nodes is the available energy. Hardware design for sensor nodes should also consider energy efficiency as a primary requirement.

- A sensor network should also be capable of adapting to changing connectivity due to the failure of nodes, or new nodes deployed.
- Real-time communication over sensor networks must be supported through provision of guarantees on maximum delay, minimum bandwidth, or other QoS parameters.
- Sensor nodes are randomly deployed and hence do not fit into any regular topology. Once deployed, they usually do not require any human intervention. Hence, the setup and maintenance of the network should be entirely autonomous.

1.2 Research Background

The research topic of this thesis occurred when I was working on two different sensor network projects. The first one was the Optimized Autonomous Space-In-situ Sensorweb (OASIS) project funded by NASA. The goal of the OASIS project was to develop a prototype dynamic and scalable hazard monitoring Sensorweb (Figure 1.2) [32] to monitor Mount St. Helens. Mount St. Helens is an active stratovolcano located in Skamania County, Washington. It is about 50 miles north of Washington State University Vancouver. Mount St. Helens is most famous for its catastrophic eruption on May 18, 1980, at 8:32 am PDT which was the deadliest and most economically destructive volcanic event in the history of the United States. Fifty-seven people were killed; 250 homes, 47 bridges, 15 miles of railways, and 185 miles of highway were destroyed. The eruption caused a massive debris avalanche, reducing the elevation of the mountain's summit from 9,677 feet to 8,365 feet and replacing it with a 1 mile (1.6 km) wide horseshoe-shaped crater. The debris avalanche was up to 0.7 cubic miles in volume. The Mount St. Helens National Volcanic Monument was created to preserve the volcano and allow for its aftermath to be scientifically studied. In this OASIS project, earth scientists wanted to monitor real-time seismic, infrasonic and lightning data to enable emergent alarm generations. Various geophysical and geochemical sensors such as seismic sensors, infrasonic sensors and lightning sensors were equipped to collect continuous

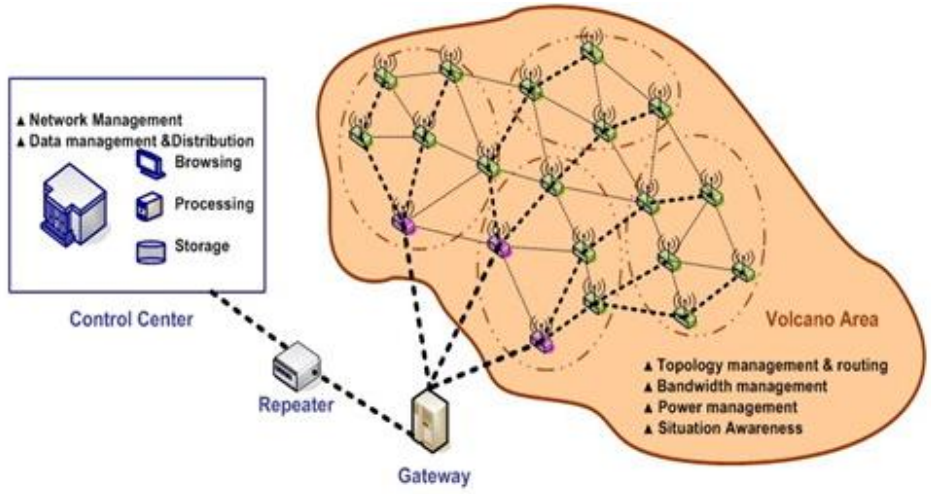

Figure 1.2: Wireless Sensor Network in OASIS Project

data representing different aspects of volcano activities. Then, these sensors were deployed in the crater of Mount St. Helens, which is a dynamic three-dimensional communication environment, to collect real-time volcano information and send them back to the control center. At the control center, we developed a java-based application, called the Monitor, to collect real-time volcano data. This Monitor application, however, can only work with our wireless sensor network since it only interprets our data format. Vice versa, our data can only be visualized by this Monitor program. Therefore, other scientists must install our Monitor so that they can receive the volcano information of our sensor network.

In addition, I also worked on some different sensor network projects. They also needed a control center, such as the Monitor, to communicate with the sensor network. For example, MineNet was a wireless sensor network to monitor the oxygen level of an underground coal mine. If the oxygen level of the coal mine was dangerous, MineNet can provide the emergency escape guidance to the miners. In this project, I needed a control center to monitor the oxygen level of the coal mine so I modified the Monitor to work with the MineNet message format. However, this process took a lot of time because the sensor data of MineNet and OASIS were very different (Figure 1.3) [25].

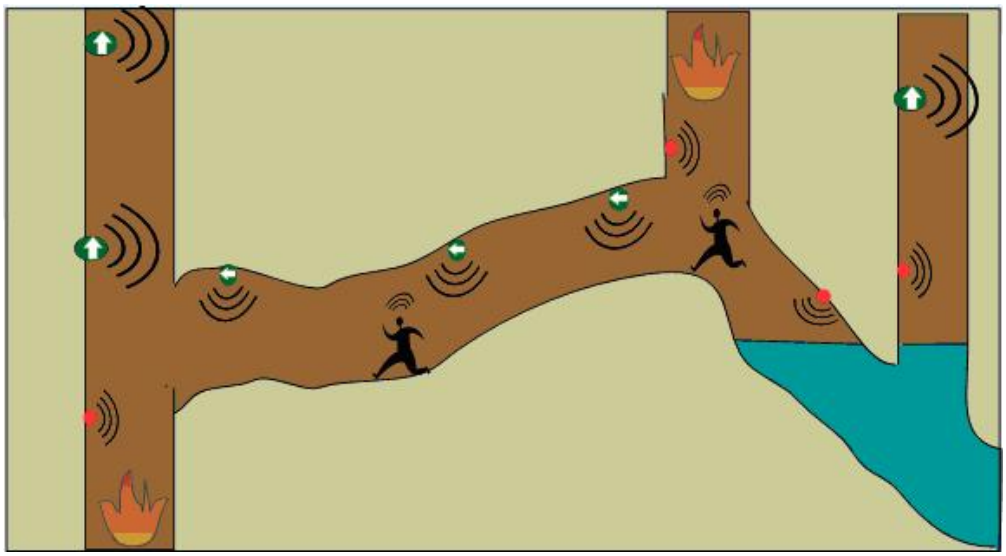

Figure 1.3: Wireless Sensor Network in MineNet Project

In addition, if we developed different standards to interpret the data of different sensor networks, it will be difficult to perform data mining or data integration on multiple sensor network at the same time. Imagine that Washington State University Vancouver (WSUV) develops a sensor network to monitor the temperature in Vancouver, and Portland State University (PSU) develops a different sensor network to monitor the temperature in Portland. The first sensor network is using CSMA protocol for their MAC Layer and the second sensor network is using TDMA protocol for their MAC Layer. Although both network can provide the temperature data, we would need to interpret their data messages in different ways. WSUV and PSU can develop their own application to monitor the temperature in their respective area. However, it would be difficult to determine the average temperature in Portland-Vancouver area. The main goal of this thesis is to investigate this problem when designing a Sensorweb Service Framework and we seek to answer the following questions:

- How to design a middleware component to translate each sensor message to human readable format?
- How to design uniform operations and a standard representation for sensor data that can be

used by diverse sensor applications?

  - Should I make a more sophisticated standard which require the users to have more knowledge about the WSN, or should I make a simpler standard which the users can learn easier? However, the latter standard will provide less detail about the WSN.

- How to allocate and share resources efficiently through the Internet?

1.3 Thesis Outline

The remainder of this thesis is organized as follows.

Chapter 2 introduces the related works about the sensor web service applications and standards.

Chapter 3 first give the overview of the Sensor Web Service Framework and explains the design goal of the system. Then, this chapter will describe the key components and their respective roles in the system.

Chapter 4 discusses the design and implementation of our Sensor Interpretation Standard (SIS). SIS provides standard models and an XML encoding for describing sensor data and services.

Chapter 5 discusses the design and implementation of our overlay network protocol named RTSC Cloud. Such a protocol is very important for the Sensor Web Service Framework to increase the system robustness and large scale deployment.

Chapter 6 describes how we designed and implemented the data collection service. The users can use this service to request, filter, and retrieve the sensor network information in real-time. In addition, this chapter also describes how we designed and implemented the event alert service. The users can use this service to plan an alert on a sensor network, and receive the sensor network information based on an alert condition.

Chapter 7 describes how we designed and implemented a protocol for the data publishing service. The users can use this service to publish their data to a central server or they can also publish a connection to their local database server or their local sensor network.

Chapter 8 describes how we designed and implemented the remote control service. The users can use this service to invoke a service within the sensor nodes. For example, a service can be to change the data rate of the node, to change the channel of the node, and to restart the node.

Chapter 9 first explains the implementation our prototype Sensor Web Framework. Evaluation is made by applying the system with a simple volcano monitoring sensor network.

Chapter 10 concludes the thesis. It discusses the advantages and shortcomings of current design and implementation, and it also notes several areas of future work.

CHAPTER TWO

RELATED WORK

This section briefly reviews existing work on systems that enable real-time and situation-aware information sharing over the Internet. In a later chapter, I will compare my Sensorweb Service Framework and the following works.

2.1 Open Geospatial Consortium Standard

Founded in 1994, the Open Geospatial Consortium (OGC) [2] is an international voluntary consensus standards organization. In the OGC, many commercial, governmental, nonprofit and research organizations worldwide collaborate in an open consensus process encouraging development and implementation of standards for geospatial content and services. OGC develops Sensor Web Enablement (SWE), which consists of seven sub specifications including Sensor Model Language, Observation and Measurement, Transducer Markup Language , Sensor Observations Service, Sensor Planning Service, Sensor Alert Service and Web Notification Service.

- Observations & Measurements Schema (O&M) - Standard models and XML Schema for encoding observations and measurements from a sensor, both archived and real-time.

- Sensor Model Language (SensorML) - Standard models and XML Schema for describing sensors systems and processes; provides information needed for discovery of sensors, location of sensor observations, processing of low-level sensor observations, and listing of taskable properties.

- Transducer Markup Language (TransducerML or TML) - The conceptual model and XML Schema for describing transducers and supporting real-time streaming of data to and from sensor systems.

- Sensor Observations Service (SOS) - Standard web service interface for requesting, filtering, and retrieving observations and sensor system information. This is the intermediary between a client and an observation repository or near real-time sensor channel.

- Sensor Planning Service (SPS) - Standard web service interface for requesting user-driven acquisitions and observations. This is the intermediary between a client and a sensor collection management environment.

- Sensor Alert Service (SAS) - Standard web service interface for publishing and subscribing to alerts from sensors.

- Web Notification Services (WNS) - Standard web service interface for asynchronous delivery of messages or alerts from SAS and SPS web services and other elements of service workflows.

As [6] states, the purpose of SWE is to make all types of web-resident sensors, instruments and imaging devices, as well as repositories of sensor data, discoverable, and accessible, where applicable and controllable via the World Wide Web. In other words, the goal is to enable the creation of web-based sensor networks.

2.2 Microsoft Sensormap

Microsoft is also developing SensorMap [27] as part of its SenseWeb project to provide an online GUI application enabling the querying and visualizing of captured physical data. The Microsoft SensorMap project allows WSN owners to register their sensing devices and publish their captured physical data to a centralized Web portal consisting of four components:

- GeoDB is a database housing sensor metadata including the publisher's name; the sensor's location, name, and type; the data type; and free text descriptions. We envision users basing their queries on sensor types, descriptive keywords, and geographic locations, such as the

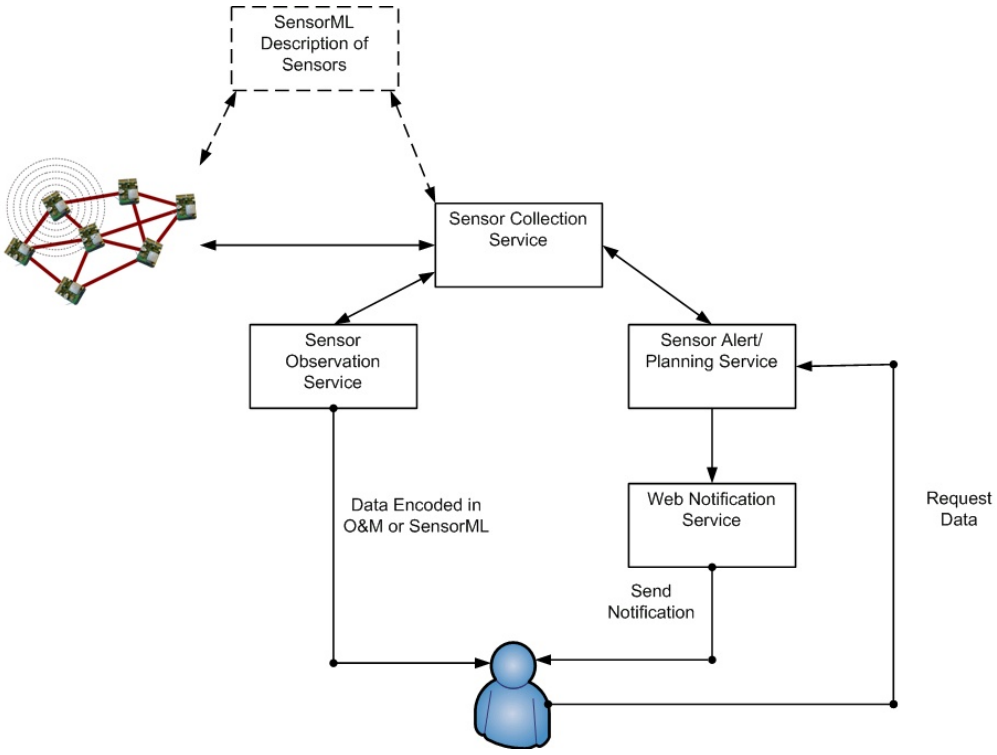

Figure 2.1: OGC Framework

list of cameras along a route or the average temperature that thermometers report inside a geographic region.

- DataHub provides two ways to make real-time data available on SensorMap. Sensors with public Web interfaces can register their URL directly to GeoDB. The SensorMap client uses these URLs to fetch realtime data. For sensors with an Internet connection but no URL (such as those in mobile phones or behind firewalls), DataHub provides a simple interface to cache sensor data. The sensors are clients for DataHub and can send data in real time using standard Web service calls. The Aggregator or the SensorMap GUI directly retrieves these cached data from DataHub rather than try to contact the sensors.

- The Aggregator creates icons representing sensor data that users can mash up with maps. It accepts queries from the client and redirects the geographic components of the queries to

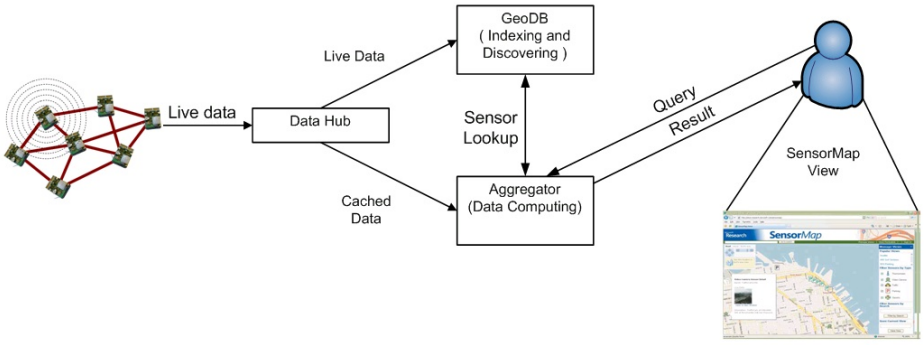

Figure 2.2: SensorMap architecture consisting of four components

GeoDB. After obtaining the metadata of a set of sensors that satisfies a client query, the Aggregator contacts the sensors and DataHub for their realtime data. It then aggregates the data accordingly (depending on sensor type and the underlying map's zoom level). For example, for data collected from thermometers, the Aggregator displays average and standard temperature deviations reported in a neighborhood. By doing so, SensorMap usefully summarizes data to the client without clogging the map with overlapping icons.

- The SensorMap is the GUI, in which end-user can send query and receive result. In this project, the GeoDB and the Aggregator are transparent to both the data publishers and users.

Although OGC has provided a popular XML standard to specify how the information can be shared over the Internet, it does not provide the underlying infrastruc- ture to share that data. In addition, the SensorMap uses the centralized Web portal to relay the data for several WSNs. We will use XML standard for data sharing as in the OGC project. In addition, we will design the P2P architecture so that the data is hosted distributively as opposed to the SensorMap, which uses the centralized Web portal. Essentially, our work will focus on solving the previous specified problem so that we can enable the P2P architecture for large scale real-time and situation-aware information

sharing over the Internet. In this architecture, we would allow any individual to share their data across our Sensor Web Service Framework. As a proof of concept, we will demonstrate its great benefit with an online bookstore database and a WSN datastream such as seismic data or infrasonic data

CHAPTER THREE

SENSORWEB SERVICE FRAMEWORK DESIGN

3.1 Overview

Sensorweb Service Framework is an online system that can enable real-time and situation-aware (alarm setting) information sharing. The Sensor Web Service Framework provides the back-end middleware for the data providers to publish their data, which could be an online database or could be a wireless sensor networks datastream. It also provides the front-end GUI for the users to query/view the information or to set the alarm for their desired information. The core of this system is the Peer-to-peer (P2P) architecture that allows the back-end middleware to communicate with the front-end GUI effectively. P2P networking is the technique for organizing distributed applications, which takes advantage of resources available at the Internet edges. Since the Sensor Web Service Framework provides the real-time data sharing features, it must support hundreds of clients machines communicating simultaneously. Therefore, P2P is our preferred architecture for client/server architecture.

By using the Sensorweb Service Framework, individual sensor networks can be linked together as services, which can be register, discover and access by different clients using a uniform protocol. Moreover, the data providers can provide advanced services for each WSNs by configuring scenario-specific operators at runtime. For example, the earth scientist can command the network to deliver seismic and infrasonic data at higher sampling rate when the earthquake is detected.

3.2 Design Goal

3.2.1 Support the heterogeneous sensor networks

The most important feature of the Sensor Web Service Framework is the ability to adapt to various WSN applications. The Sensor Web Service Framework must be able to collect sensor data from different WSNs and serve them to the end-users. In addition, the end-users should be able to

control the WSNs by sending out control commands remotely. As the result, the Sensor Web Service Framework should be able to send out these commands for each specific type of WSNs.

Thus, the Sensor Web Service Framework will define a standard for describing the WSNs' messages. In addition, it will provide a middleware that can translate data from WSNs' specific format to our Sensor Web Service Framework's readable format; and vice versa. This standard is called the Sensor Interpretation Standard (SIS). Using this standard, each data provider can define their own way of interpreting their WSNs' messages in one self-contained XML file. The chapter Sensor Interpretation Standard will explain how to use SIS in detail.

3.2.2 Provide the Data Collection, Event Alerting and Remote Control Services

The Sensor Web Service Framework not only provides the information model and encoding (e.g.; the Sensor Interpretation Standard) but also defines several useful services that can be used to interact with the sensor networks.

One of the most important service is the Data Collection Service which collects sensor data from the sensor networks. This service allows the end-users to collect and process the real-time sensor data. Each client who intends to invoke the Data Collection Service must strictly follow the Sensor Interpretation Standard.

Secondly, the users can use the Event Alerting Service specify how alert conditions are defined, detected and made available to interested users. Each alert should contain a list of requests from the users such as the data type; and the data range. When the sensor data satisfies the user's requests, the Event Alerting Service will send the alert to the users.

Finally, the users can control the sensor network's behavior by using the Remote Control Service. This service allows the user to send the remote command to change the parameters of the sensor nodes. For example, if the event is detected at some location in the volcano, the scientist can send one remote command to increase the data rate at that particular location so that they can acquire more data for their analysis. In addition, if the sensor node gets into an unstable state, the

sensor network operators can also send a remote command to restart the mote.

As the sensor network community is still evolving, new services will appear to satisfy other requirements of the users. As the result, we will try to define our service in a generic way so that it will be easy to add more services in the future. The standard to define our services is also a part of the Sensor Interpretation Standard.

3.2.3 Allocate and share resource effectively

Our Sensor Web Service want to make various types of web-resident sensors, instruments, image devices, and repositories of sensor data, discoverable, accessible, and controllable via the World Wide Web. Therefore, there will be many computers, who will be connecting to our system simultaneously. As the result, our system might suffer a lot of communication and management overhead.

Our most important goal is to design an overlay network protocol to share the sensor data effectively. In this project, we want to take advantage of the resources of the clients, including bandwidth, storage space, and computing power. We prefer the P2P architecture over the client/server architecture. As the demand on the system increases, more peers will be added to our system to increase the total capacity of the system. This is not true of a client-server architecture with a fixed set of servers, in which adding more clients could mean slower data transfer for all users.

In addition, the distributed nature of P2P networks also increases robustness in case of failures by replicating data over multiple peers. In P2P systems, each peer can find the data without relying on a centralized index server. Therefore, there is no single point of failure in the system.

3.2.4 Lightweight Framework

Unlike the traditional wired networks, sensor nodes in WSNs generally operate with very tight resources, such as limited battery power, limited storage capacity, and constrained wireless communication. This is one of the unique attributes of WSNs. Therefore, it is an extremely critical concern for the Sensor Web Service Framework to limit the management overhead and maximized

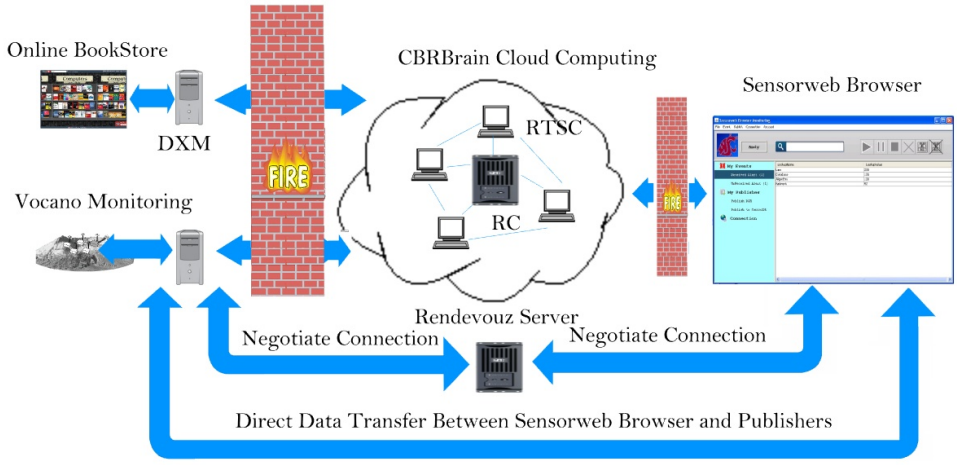

Figure 3.1: SensorMap architecture consisting of four components

the overall performance. Our goal is to try to design a lightweight framework, which does not add any communication overhead to the WSNs.

3.3 Main Components in Sensor Web Service Framework

The Sensor Web Service Framework would be composed of a number of components, each of which would offer defined roles within the system. This system could be distributed over the Internet with components being held on accessible servers as required. Sensor Web Service Framework has 5 components: Data Exchange Middleware (DXM), RealTime Sharing Center (RTSC), Routing-Center (RC), Sensorweb Browser (SB) and Rendezvous Server. Among these components, the DXM and SB can reside on private networks, while the RTSC, RC and Rendezvous Server are hosted on public computers (Figure 3.1).

3.3.1 Data Exchange Middleware

The `Data Exchange Middleware` component (DXM), written in Java, bridges the datasources of the providers and our Sensor Web Service. While the focus of the Sensor Web Service Framework is to provide a technology to share Wireless Sensor Networks (WSN) data through the Internet. The developed technology can be applied to share the data from online databases such as

an online bookstore or an online WSN database . In figure 3.1 above, you can see two type of DXM: a database DXM and a WSN datastream DXM. In the nutshell, both the DXM database and the DXM datastream are working in the same manner. They receive the upstream data from the datasources, reformat the data using the lightweight Sensor Interpretation Standard (SIS), and send the reformatted data to the end-user's clients. They can also receive the downstream remote control command from the end-users' clients, reformat the command using the lightweight Sensor Interpretation Standard, and send the reformatted command to the datasources. Note that the remote procedure call (RPC) /citeMDDH-PCAC2006 features are developed for WSN community. Hence, our Sensor Web Service Framework will only support RPC for the WSN DXM datastream.

Figure 3.2 describes how the DXM Database can connect the database to our Sensor Web Service Framework. The MS Access Database connects to our DXM Database by using the ODBC Driver for MS Access. When the DXM Database receives the request from the end-users clients, it can send the query to the database and receive the result dataset. If the result dataset contains at least one record, the DXM will forward this dataset to the end-users clients through the RTSC Cloud. The RTSC Cloud detail will be discuss in the next section.

When the database is an independent database such as an online bookstore, the data provider can input their data by using their own data manipulation application. On the other hand, the database can also be used to store the data from a sensor network. In this case, the datastream comes directly from the sensor network to the Serial Forwarder to the MS Access Database. The Serial Forwarder program is used to read packet data from a serial port and forward it over an Internet connection, so that other programs can be written to communicate with the sensor network over the Internet. Moreover, the Serial Forwarder program can also receive a command from the Internet Connection and forward this command to the sensor network.

Figure 3.3 describes how the DXM DataStream can connect the sensor network to our Sensor Web Service Framework. In contrast with the DXM Database which only queries the history data from the database, the DXM DataStream can receive the real-time sensor data from the Serial

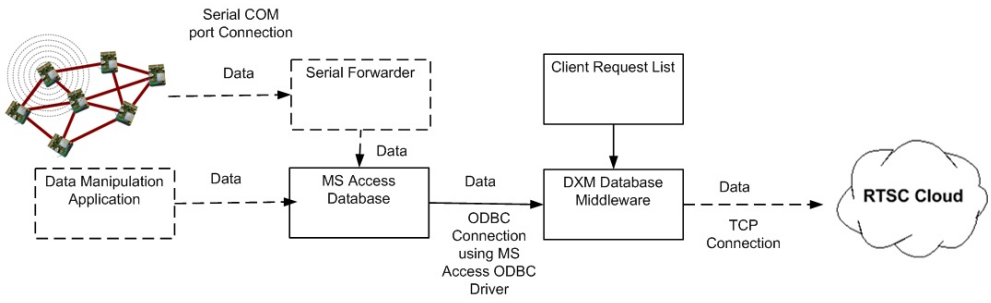

Figure 3.2: Connection Between Database and DXM

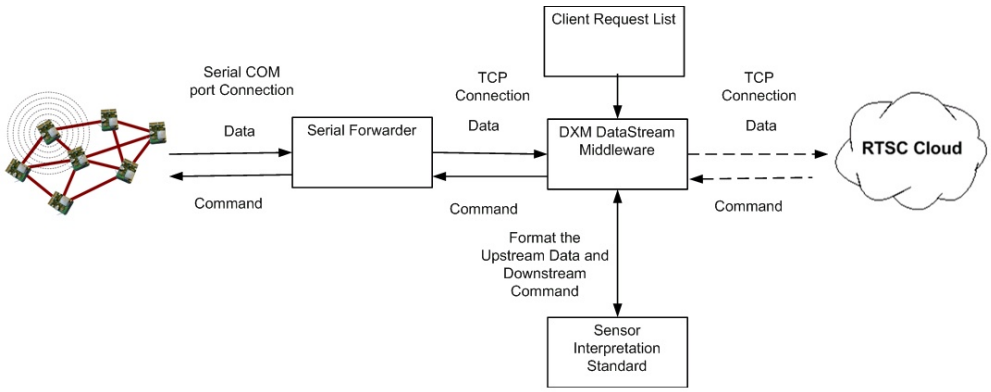

Figure 3.3: Connection Between Sensor Network and DXM

Forwarder through the TCP connection. After receiving the real-time data, it will read from the client request list to see whether any client is waiting for this real-time datastream. When the real-time datastream is requested by a client, the DXM DataStream will format the data by using Sensor Interpretation Standard (SIS), and send out the formatted data to the RTSC Cloud. In addition, the end-users clients can send RPC /citeMDDH-PCAC2006 command to the sensor network through the DXM DataStream. Upon receiving the RPC command, the DXM DataStream will verify the command to see whether this command is supported by the sensor network. If the command is supported, the DXM DataStream will format the command using the SIS and send out the formatted command to the Serial Forwarder, which will further forward the command to the sensor network.

In this project, I connected the DXM Database to the MS Access database and connected

the DXM DataStream to the Serial Forwarder to demonstrate the concepts of DXM. If the data providers are not using the MS Access database or not using the Serial Forwarder to collect the data from the data sources, they should be able to modify the DXM to work with different data source such as MySQL database.

3.3.2 RealTime Sharing Center Cloud

The RealTime Sharing Center Cloud consists of the Routing Center and many RealTime Sharing Center Peers. Together, the RC and RTSC will form an overlay network to perform the P2P communication protocol. P2P communication protocol is necessary for the Sensor Web Browser to reallocate resource effectively. In this architecture, the RTSC and RC servers must have the public IP address and must be accessible from other components.

Routing Center

The RoutingCenter (RC) is the superpeer among all of the RTSC peers. A superpeer is a necessary entry point for boot-strapping a RTSC peer willing to join the overlay network. Once a new peer joins the network, the superpeer will save the connection information of this peer (such as the IP address and TCP port) to the active-peer list, and increment the total number of peers by one. This information is important because it lets the RTSC peers communicate with each other.

In addition, the RC will also run the central DXM Database. The central DXM provides a database for the users who want to publish their data without having to maintain one online database. For example, a graduated student just wants to sell his old books so he wants to use our system to publish the book information and prices. That said, this student does not want to maintain an online database, and does not want to dedicate his personal computer to run the DXM Database just to sell a few book. He could, however, publish his book information to the central database, which is maintained by Sensor Web Service Operators.

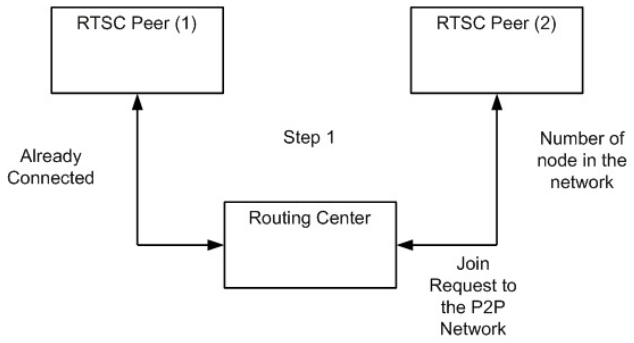

Figure 3.4: RTSC Cloud Setup

RealTime Sharing Center Peer

In our Sensor Web Service Framework, the RTSC cloud acts as the P2P search engine for the client applications. Each RTSC peer is a public computer somewhere on the earth and multiple RTSCs form a P2P network. When a RTSC joins the network, it will request the total number of peers from the superpeer, which is Routing Center (Figure 3.4). By using the total number of peers, this new peer can calculate its own ID and calculate the list of neighbor IDs.

In addition, the superpeer will use the same procedure to calculate the list of neighbor IDs of this new node, and send out the notification message to these neighbor peers. When the neighbor nodes receive the notification message, they will recalculate their ID and their neighbor table. This calculation is using the de Bruijn Graph algorithm [7]. Then, it will contact the superpeer to retrieve the the neighbor peer's connection information (e.g.; IP address and Port number). This information is maintained in the superpeer's internal cache.

Finally, the new peer will attempt to connect to each of its neighbor peer. Once a RTSC peer connects to the superpeer, it must periodically ping the superpeer after every 5 minutes, so that the super peer can confirm its status. If the RTSC peer fails to ping the super peer every 5 minutes, the superpeer will shutdown the connection to this RTSC peer and remove it from the active-peer list. In addition, the superpeer will send out a notification message to the neighbor peers of this failure peer.

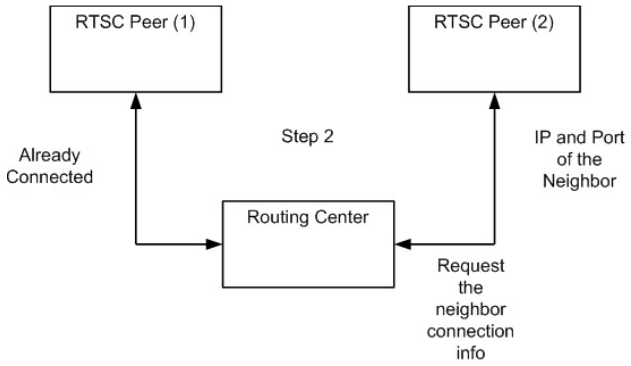

Figure 3.5: RTSC Cloud Setup

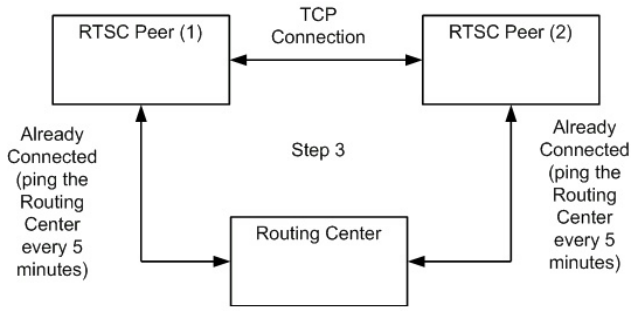

Figure 3.6: RTSC Cloud Setup

3.3.3 Sensorweb Browser

The next component is the Sensorweb Browser (SB). This component is installed in the client's machine and it provides the GUI for the end-user to interact with our system. The SBs roles are mainly:

- Provide a user interface for the data provider to register their dataset to the Routing Center (RC).
- Provide a user interface for the data provider to register their DXM connection to the Routing Center (RC) .
- Provide a user interface for the user to search and register their request to the data providers.
- Provide a user interface for the user to manage their alarm properties such as set alarm type, set alarm time, set email address to send alert.
- Provide a user interface for the user to view their alert details. For example, when the book event is received, the users can see the name and the price of the book.

When the SB sends a request to the RTSC cloud, the cloud will find the appropriate DXM machine by using the CBRBrain algorithm /citeSL-ICCCN2004, and send this request to this DXM machine. When the request is received by the DXM machine and the response is returned to the SB, the DXM and SB can communicate with each other directly without maintaining a persistent connection with RTSC cloud.

3.3.4 Rendezvous Server

The final component is the Rendezvous Server. This component is developed and maintained by Cornell University. It helps two computers from different NAT to setup the direct TCP connection. We used the Rendezvous Server to setup the TCP connection between the DXM and SB machines, which reside on different NATs. Upon receiving the request from the client, the DXM machine will

generate a unique connection identification, and return this identification back to the SB through the RTSC cloud. When the SB receives this connection identification, both the DXM and the SB can close the connection with the RTSC cloud. Then DXM will register this connection identification into the Rendezvous Server. Next, the SB will request the connection to the DXM using the same connection identification. When the Rendezvous Server receives the request, it will reconcile the connection between the DXM and the SB so that they can create the direct TCP connection. The details of this technique will be discussed in chapter 5.

CHAPTER FOUR

SENSOR INTERPRETATION STANDARD

4.0.5 The difficulty of interpreting the wireless sensor network format

Similar to the Internet, wireless sensor network uses a communication stack method to generate the communication messages. Individual protocols within a suite are often designed with a single purpose in mind. This modularization makes design and evaluation easier. Because each protocol module usually communicates with two others, they are commonly imagined as layers in a stack of protocols. The lowest protocol always deals with low-level, physical interaction of the hardware. Every higher layer adds more features.

For example, one simple protocol stack can have four layers: the application layer, the transport layer, the network layer and the physical layer. In this communication message, the application layer will implement the sensing protocol which acquires data from the ADC channels and encodes them into the messages. The Transport Layer is responsible for delivering data to the appropriate application process on the host computers reliably. The network layer will implement the routing protocol to route the message from the wireless sensor node to the base station. Finally, the Physical Layer is responsible for bit-level transmission between wireless sensor nodes.

Since the wireless sensor network community is not yet mature, there is not a strict communication stack for every network. Each network has its own communication stack. For example, the Internet uses TCP/IP for the Transport Layer, however, due to the application specific nature of sensor networks, it is difficult to design a single monolithic transport system that can be optimized for every application.

The problem of achieving reliable transmission between remote nodes over multiple hops despite channel errors, collisions or congestion has at least the following dimensions:

- Single packet vs. block of packets vs. stream of packets: the cases of delivering only a single packet on the one hand and of delivering a number or even an infinite stream of packets on

the other hand differ substantially in the protocol mechanisms usable in either case. Reliable delivery of single packets can be important for example for highly aggregated data, reliable delivery of blocks is required for applications like disseminating new code or new queries into the network. Periodic data reporting is the primary example for streams of packets.

- Guaranteed vs. stochastic delivery: Some applications require guaranteed delivery. Examples are:
  - Reporting of very important events from sensors to a sink node.
  - The distribution of new code or queries from the sink node to sensors.
  - Handing over the target state in a tracking application between nodes close to the target trajectory. Other situations might well tolerate a certain degree of losses. For example, when many sensors transmit strongly correlated sensor readings, occasional loss is tolerable. One way to specify the tolerable amount of losses is to prescribe a delivery probability. In general, the higher the desired delivery probability, the higher are the energy costs needed to achieve this.
- Sensors-to-sink vs. sink-to-sensors vs. local sensor-to-sensor: as opposed to other types of networks communication in sensor networks does not take place between arbitrary nodes, but is either from (groups of) sensors to a single or a few sink nodes, from a sink to (groups of) sensors or locally between (groups of) sensors when these run collaborative signal processing algorithms.

4.0.6 Sensor Interpretation Standard Implementation

The Sensor Interpretation Standard consists of the following attribute:

I) It provides a lightweight format to interpret the communication message from different sensor networks.

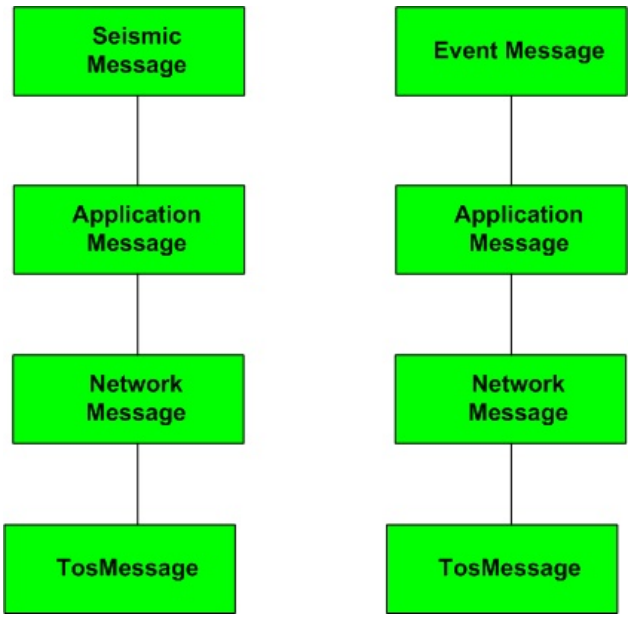

Figure 4.1: Communication Stack In Wireless Sensor Network for two different type of data

In the following example, there are two different communication stacks (Figure 4.1), which are used in the sensor network to monitor the St. Helens volcano. The first communication stack is used for the data messages. The sensor node will periodically collect the seismic, and then it will packet this data into this format and will deliver the data to the base station. The second communication stack is used for event messages. The event message is used to monitor the healthiness of the sensor node. When the battery level of a sensor node is going low, the node will automatically send out and event message to the base station. By monitoring this event message, the operator can adjust the workload of the sensor node by sending the RPC /citeMDDH-PCAC2006 command. Apparently, the sensor node with less battery power should have lower data rate than the others so that our monitor coverage is still guaranteed.

In order to parse the communication stack, the Sensor Interpretation Standard is using two attributes:

- First, it will use the `<MessageHierarchy>` component to determine the communication stack. In Figure 4.1, left side represent the communication stack of the Seismic Message and

```
<MessageHierarchy>
<TOSMessage type="129">
<NetworkMessage type="1">
<ApplicationMessage type="0">
<SeismicMsg type="-1" />
</ApplicationMessage>
</NetworkMessage>
</TOSMessage>
</MessageHierarchy>
```

Figure 4.2: Hierarchy of The Seismic Message

```
<MessageHierarchy>
<TOSMessage type="129">
<NetworkMessage type="1">
<ApplicationMessage type="0">
<EventMessage type="-1" />
</ApplicationMessage>
</NetworkMessage>
</TOSMessage>
</MessageHierarchy>
```

Figure 4.3: Hierarchy of The Event Message

the right side represent the communication stack of the Event Message. In Figure 4.2, the MessageHierarchy component tells the parser that the communication stack of the seismic message is TOSMessage, NetworkMessage, ApplicationMessage and SeismicMessage in this particular order. Similarly, the communication stack of the event message is TOSMessage, NetworkMessage, ApplicationMessage and EventMessage (Figure 4.3). The type attribute is used to determine the next block in the communication stack. For example, the block `<TOSMessage type="129">` determines that the NetworkMessage type is 129, and the block `<NetworkMessage type="1">` determines that the ApplicationMessage type is 1. Finally, the `<SeismicMsg type="-1" />` determines that this is the final block of the

communication stack.

- After the parser understands the message hierarchy, it will use the <MessageFormat> component to determine the message format details. For example, Figure 4.4 shows the message details of the TOSMessage, the NetworkMessage, the ApplicationMessage, and the EventMessage. Let's take a look at the TOSMessage block. The first line "<TOSMessage bit-offset="I:0" size="I:80" name="TOSMessage">" tells the parser the general information of the TOSMessage such as the bit-offset is 0 and the size is 80. This means that the TOSMessage would start at bit zero and end at bit 80 of the receiving message. The rest of the block tells the parser the detail information of the TOSMessage. For example, let's look at this line <field bit-offset="I:72" name="group" size="I:8" repeat="1" />. If the end-user would request the group of the sensor network, the parser will deliver the last eight bits of the TOSMessage to the end-users.

II) It provides method for the user to receive data or send command to the sensor network.

In the Sensor Web Service Framework, the data provider specifies certain methods on how the end-users can request the data from the sensor network, and how the end-users can send the command to the sensor network. The data providers use the <Read> keyword to define which data the end-user can request and how they can request them. In addition, the data providers use the <Write>keyword to define how the end-users can send a remote control command to the sensor network.

Moreover, the data providers need to define how the data is extracted from the receiving message or how the RPC command is packaged and send to the sensor network (Figure 4.5). In order to do this, the data providers should give a dataname for each type of data reading. For example, the <data dataName="seismic" excuteType="oscope_table" /> identifies that the end-users can receive the seismic data from the sensor networks, and the seismic data can be displayed in a table view or in an oscilloscope drawing. Another example is <data

`dataName="setReportLevel" excuteType="write" />`. This line indicates that the end-users can send out the RPC command setReportLevel. In these two example, the execute type identifies the method in which the end-users can interact with the sensor network. Currently, I have defined three execute types: table, oscilloscope, and write.

Finally, each `dataname` is associated with a data structure (Figure 4.6 and Figure 4.7). The data structure explains the detail of each `dataname`. For example, figure 4.6 shows the data structure of `seismic`, which consists of the source, the timestamp and the reading fields. Each field consists of the following properties:

- Message Property: this property tells the client where can they extract the messageField data.
- MessageField Property: this property tell the client which data should be extracted from the messages.
- Encode Property: this property tell the client which format is used to display the receiving data. The system can support the following format: decimal, hexadecimal and string format.
- Value Property: the value property can only be:
  - Input: input is a reserved keywork for RPC command. It means that the `messageField` is entered by the end-users.
  - Output: output is a reserved keywork for Data Message. It means that the `messageField` is read from the received messages.
  - Any other value which is not input or output is treated at some fixed value for that `messageField`. For example, every sensor network is identify by a fixed user_hash and unix_time. Therefore, every exchange messages between the sensor network and the clients should use the same user_hash and unix_time values (Figure 4.7).

```
<MessageFormat>
<TOSMessage bit-offset="I:0" size="I:80" name="TOSMessage">
<field bit-offset="I:0" name="length" size="I:8" repeat="1"/>
<field bit-offset="I:8" name="fcfhi" size="I:8" repeat="1"/>
<field bit-offset="I:16" name="fcflo" size="I:8" repeat="1"/>
<field bit-offset="I:24" name="dsn" size="I:8" repeat="1"/>
<field bit-offset="I:32" name="destpan" size="I:16" repeat="1"/>
<field bit-offset="I:48" name="addr" size="I:16" repeat="1"/>
<field bit-offset="I:64" name="type" size="I:8" repeat="1"/>
<field bit-offset="I:72" name="group" size="I:8" repeat="1"/>
</TOSMessage>
<NetworkMessage bit-offset="I:80" size="I:80" name="NetworkMessage">
<field bit-offset="I:0" name="linksource" size="I:16" repeat="1" />
<field bit-offset="I:16" name="type" size="I:8" repeat="1" />
<field bit-offset="I:24" name="ttl" size="I:5" repeat="1" />
<field bit-offset="I:29" name="qos" size="I:3" repeat="1" />
<field bit-offset="I:32" name="dest" size="I:16" repeat="1" />
<field bit-offset="I:48" name="source" size="I:16" repeat="1" />
<field bit-offset="I:32" name="crc" size="I:32" repeat="1" />
<field bit-offset="I:64" name="seqno" size="I:16" repeat="1" />
</NetworkMessage>
<ApplicationMessage bit-offset="I:160" size="I:32"
name="ApplicationMessage">
<field bit-offset="I:0" name="type" size="I:8" repeat="1" />
<field bit-offset="I:8" name="length" size="I:8" repeat="1" />
<field bit-offset="I:16" name="seqno" size="I:16" repeat="1" />
</ApplicationMessage>
<EventMessage bit-offset="I:192" size="I:24" name="EventMessage">
<field bit-offset="I:0" name="type" size="I:8" repeat="1" />
<field bit-offset="I:8" name="level" size="I:8" repeat="1" />
<field bit-offset="I:16" name="length" size="I:8" repeat="1" />
<field bit-offset="I:16" name="data" size="I:8" repeat="0" />
</EventMessage>
<SeismicMsg bit-offset="I:192" size="I:0" name="SeismicMsg">
<field bit-offset="I:0" name="timeStamp" size="I:32" repeat="1" />
<field bit-offset="I:32" name="reading" size="I:16" repeat="0" />
</SeismicMsg>
</MessageFormat>
```

Figure 4.4: Data Structure

```
<Read>
<data dataName="seismic" excuteType="oscope_table" />
<data dataName="infrasonic" excuteType="oscope_table" />
<data dataName="response" excuteType="table" />
</Read>
<Write>
<data dataName="setReportLevel" excuteType="write" />
<data dataName="getReportLevel" excuteType="write" />
<data dataName="getRealtimeSynMode" excuteType="write" />
<data dataName="setRealtimeSynMode" excuteType="write" />
</Write>
```

Figure 4.5: Read And Write Component of SIS

```
<struct structName="seismic">
<field message="NetworkMessage" messageField="source" encode="dec"
value="output" />
<field message="SeismicMsg" messageField="timeStamp" encode="hex"
value="output" />
<field message="SeismicMsg" messageField="reading" encode="dec"
value="output" />
</struct>
```

Figure 4.6: DataFormat of the Seismic Message

```
<struct structName="setReportLevel">
<field message="TOSMessage" messageField="addr" encode="dec"
value="65535" />
<field message="TOSMessage" messageField="group" encode="dec"
value="125" />
<field message="RpcCommandMessage" messageField="unix_time"
encode="hex" value="4a122482" />
<field message="RpcCommandMessage" messageField="user_hash"
encode="hex" value="4c6e9f74" />
<field message="RpcCommandMessage" messageField="returnAddress"
encode="dec" value="0" />
<field message="RpcCommandMessage" messageField="responseDesired"
encode="dec" value="1" />
<field message="RpcCommandMessage" messageField="commandID"
encode="dec" value="1" />
<field message="RpcCommandMessage" messageField="dataLength"
encode="dec" value="2" />
<field message="RpcCommandMessage" messageField="address"
encode="dec" value="65535" />
<field message="SetReportLevelMsg" messageField="type" encode="dec"
value="input" />
<field message="SetReportLevelMsg" messageField="level"
encode="dec" value="input" />
</struct>
```

Figure 4.7: DataFormat of the RPC Command SetReportLevel

CHAPTER FIVE

RTSC CLOUD MECHANISM

5.0.7 The Sensorweb Service Framework uses the CBRBrain as the Overlay Network Protocol

The Sensor Web Service Framework implements the CBRBrain [33] as the overlay network for our P2P communication. It uses the RTSCs to construct the CBRBrain backbone. Each RTSC is acting as the IP router in the CBRBrain algorithm. When the RTSC starts up, it will connect to the RC to get the number of active RTSC peers. By using this number, this RTSC can calculate its own ID, as well as, its neighbors' IDs by using De Bruijin graph algorithm. Next, this RTSC will request the IPAddress and port of its neighbors from the RC, and open a TCP connection to each RTSC neighbor. Each RTSC is acting as the gateway for some DXM machines. Each DXM machine is hosting a database or a datastream source, which is identified by its source name. Therefore, each RTSC can create its key (as explained in the CBRBrain algorithm) by hashing its DXM source names. Then the RTSC can publish this key to its neighbor. When the SB client requests some information, it will send the search keyword to some random RTSC server. Since this random RTSC server is inside the CBRBrain backbone, it can perform the content based routing and find the target router who has the search keyword in its lookup table. This router then delivers the request to the target DXM machine.

5.0.8 CBRBrain

The backbone of the CBRBrain [33] system is a content addressable network, which can be described by a pair (K,G) where K is a set of keys and G = (V;E) is a logical graph or topology. The set K is generated by hosts who hash each shared content into a value, hereafter called key, and publish it to the backbone. Each node u in G is assigned a subset of keys K_u such that $\cup_{u \in V} K_u = K$. In practice, node u needs to store a lookup table which contains necessary information related to each key $k \in K_u$, such as the address of the host who published the key and owns the content. The

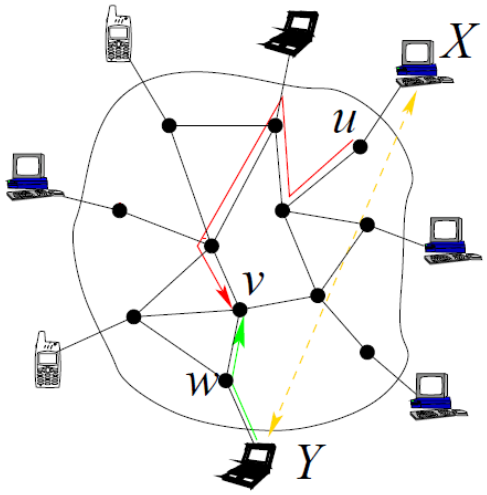

Figure 5.1: CBRBrain Architecture

assignment of key to node is performed by mapping both keys and nodes labels to a real domain, then the key/value pairs are assigned to the closest server. As opposed to other networks, routing in a content addressable network is not performed according to the destination address, but according to the content key. More precisely, no one can know the address of the closest server in advance. It is eventually found out by content routing and key matching.

In the CBRBrain system, IP routers act as content ROUTERS in finding the best route from one point to another, and the user host will not participate in any intermediate routing and forwarding. Figure 5.1 illustrates such an architecture. The region inside the cloud represents the CBRBrain backbone which overlays the backbone of Internet. The set of routers construct a self-routing topology. The end hosts connect to the network through those gateway routers inside the backbone. Notice that, we do not force all routers to join into the CBRBrain network, as we will see later, the uninvolved routers are transparent to the system like network cables; and any dedicated host can also act as a router in the system if it is authorized by the system coordinator. For simplicity of presentation, here before and after, the router always represents the backbone router or dedicated host who has joined the routing chain by authorization.

From the users viewpoint, the CBRBrain network works like a central server, where a user

could query to and get a response from another computer, although the backbone is formed by many routers and the content location is actually decentralized to individual user hosts. From the viewpoint of the Internet, the CBRBrain backbone is an overlayed logical network over Internet backbone, which provides additional service, Content Based Routing, to sustain various P2P applications and other intelligent services in the future. For illustration, we briefly discuss how to retrieve a file in a P2P file sharing application under the CBRBrain system. Notice that the CBRBrain architecture itself is not restricted to the file sharing. Figure 5.1 illustrates an example that follows:

1. A host, X, inquires of the CBRBrain system about a file stored in Internet. Host X first uses the globally predefined DHT function to map to a key k, then sends it to the gateway router u.

2. The CBRBrain backbone performs the content based routing service and finds the target router v who has the key k in its lookup table. The router v then finds the corresponding IP address(es) of the target host Y if it exists. There are two options here: 1) the router could then retrieve the content and feed it to the requesting host, or 2) the router gives the IP address of Y to the requesting host and lets it retrieve the content. The first approach makes the targeting host anonymous to the requesting host, while the second approach alleviates the burden of the router.

However, our P2P architecture is facing a severe Network Address Translation (NAT) problem on large scale deployment. NAT causes well-known difficulties for peer-to-peer (P2P) communication, since the peers involved may not be reachable at any globally valid IP address. Recent work has proposed work-arounds that establish a TCP connection without the use of proxies or tunnels as in NATBLASTER [5], STUNT [28], and NUTSS [12]. This is accomplished by setting up the necessary connection-state on the NAT through a carefully crafted exchange of TCP packets. However, not all NATs in the wild react the same way, causing these approaches to fail in various

cases. This paper will combine the most reliable but least efficient relaying method, and the less reliable but more efficient NUTSS method [12] to provide a practical NAT traversal solution. In this paper, we will discuss the core architecture of our system, which implements both CNRBrain and P2P NAT traversal technique to enable P2P communication.

5.0.9 NAT Traversal

Section 1 and 2 will discuss the Relaying Method and TCP Hole Punching method, which are used for NAT Traversal. These methods have been presented in [11] and [12]. In addition, Cornell University has implemented the NUTSS service to connect two computers behinds NAT. Section 3 will describe how we will implement the Relaying Method and how we will use NUTSS service to solve the NAT Traversal problem in our Sensorweb Service Framework. The following text describing the relaying and TCP hole punching mechanisms, and the associated Figure 5.2 and 5.3 are reproduced with permission from [11].

Relaying Method

> The most reliable, but least efficient method of P2P communication across NAT is simply to make the communication to the network look like standard client/server communication, through relaying. Suppose two client hosts A and B have each initiated TCP or UDP connections to a well-known server S, at Ss global IP address 18.181.0.31 and port number 1234. As shown in Figure 5.2, the clients reside on separate private networks, and their respective NATs prevent either client from directly initiating a connection to the other. Instead of attempting a direct connection, the two clients can simply use the server S to relay messages between them. For example, to send a message to client B, client A simply sends the message to server S along its already-established client/server connection, and server S forwards the message on to client B using its existing client/server connection with B.

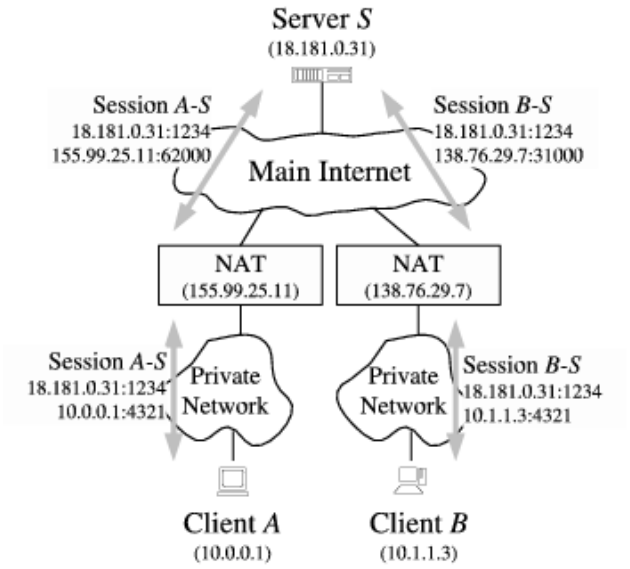

Figure 5.2: NAT Traversal using relaying method. Reproduced with permission from [11]

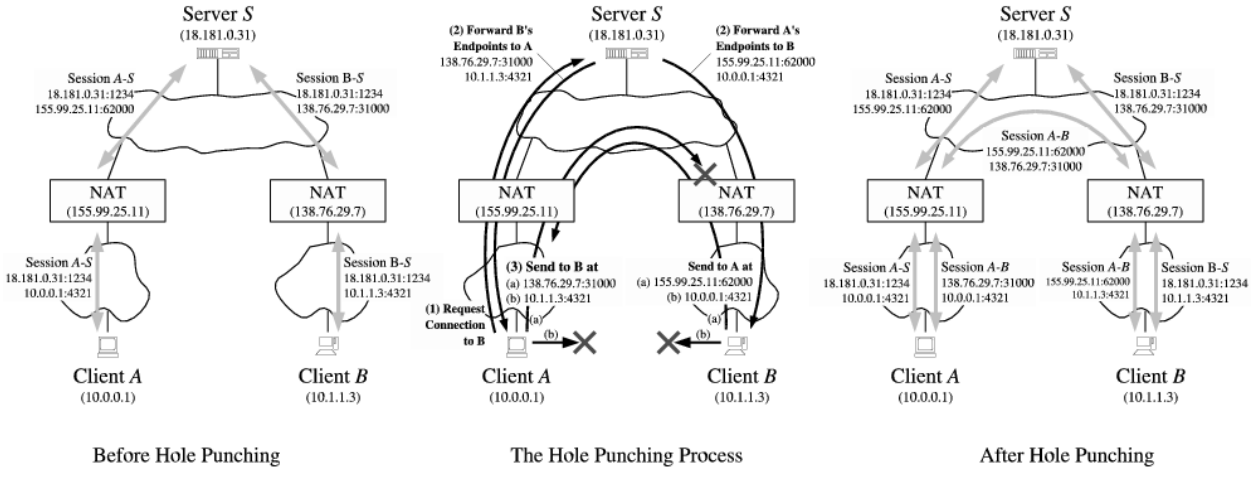

Figure 5.3: NAT Traversal with TCP Hole Punching. Reproduced with permission from [11]

TCP Hole Punching Method

Suppose that client A wishes to set up a TCP connection with client B. We assume as usual that both A and B already have active TCP connections with a well-known rendezvous server S. The server records each registered client's public and private endpoints.

1. Client A uses its active TCP session with S to ask S for help connecting to B.
2. S replies to A with B's public and private TCP endpoints, and at the same time sends A's public and private endpoints to B.
3. From the same local TCP ports that A and B used to register with S, A and B each asynchronously make outgoing connection attempts to the other's public and private endpoints as reported by S, while simultaneously listening for incoming connections on their respective local TCP ports.
4. A and B wait for outgoing connection attempts to succeed, and/or for incoming connections to appear. If one of the outgoing connection attempts fails due to a network error such as connection reset or host unreachable, the host simply re-tries that connection attempt after a short delay (e.g., one second), up to an application-defind maximum timeout period.
5. When a TCP connection is made, the hosts authenticate each other to verify that they have connected to the intended host. If authentication fails, the clients close that connection and continue waiting for others to succeed. The clients use the first successfully authenticated TCP stream resulting from this process.

Consider the common-case scenario in which the clients A and B are behind different NATs, as shown in Figure 5.3. The outgoing connection attempts A and B make to each other's private endpoints will fail. However, the clients' outgoing connection

attempts to each other's public endpoints cause the respective NATs to open up new holes enabling direct TCP communication between A and B. If the NATs are well-behaved, then a new peer-to-peer TCP stream automatically forms between them. If A's first SYN packet to B reaches B's NAT before B's first SYN packet to A reaches B's NAT, for example, then B's NAT may interpret A's SYN as an unsolicited incoming connection attempt and drop it. B's first SYN packet to A should subsequently get through, however, because A's NAT sees this SYN as being part of the outbound session to B that A's first SYN had already initiated.

NAT Traversal In Sensorweb Browser

In the current version, our Sensor Web Service Framework implements both the relaying method and the TCP Hole Punching method for NAT Traversal. We use the TCP Hole Punching algorithm as the preferred method and the relaying algorithm as the backup method. According to the test result as in [12], the TCP Hole Punching algorithm fails on the NAT, which does not support hairpin translation. On the other hand, the relaying method is 100% reliable. In Sensor Web Service Framework, our NAT Traversal works in the following steps.

1. When two random peers try to communicate with each other, they will first exchange a unique connection identification. This exchange will need to go through a RTSC cloud.

2. When both machines receive the connection identification, they can close the connection to the RTSC cloud.

3. The machine which creates the connection identification will register its connection to the rendezvous server. The other machine will request the direct TCP socket using the received connection identification. The rendezvous server is a public machine which run the NUTSS program developed Cornell University. This NUTSS program implemented the hole punching technique.

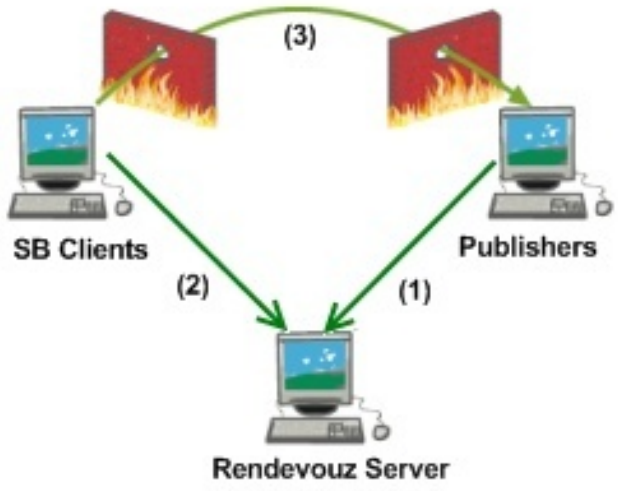

Figure 5.4: NAT Traversal using Hole Punching Method In Sensorweb Browser

4. If the connection is setup successfully, the process is done (figure 5.4).

5. If the connection fails three times, both machines will switch to the relaying method for further communication (Figure 5.5).

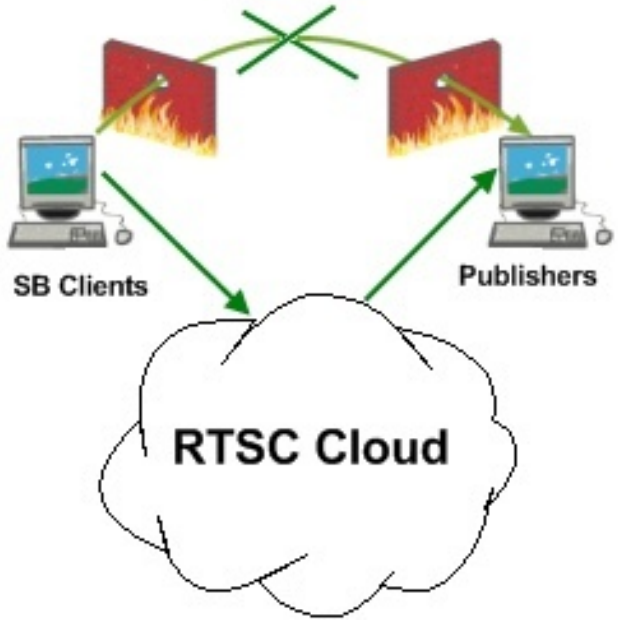

Figure 5.5: NAT Traversal using Relaying Method In Sensorweb Browser

CHAPTER SIX

DATA COLLECTION SERVICE AND EVENT ALERTING SERVICE

Each RTSC peers maintains a complete or partial knowledge of the sensorweb network. The RC maintains the registration of peers through a list called routingtable.xml. Each element simply is the pair of the host IP and services port of the RTSC machine and the host IP and service port of each DXM connected to this RTSC. Each RTSC maintains a local cache of the RC routingtable.xml, called cached list. It periodically contacts its RC to update its cached list.

When the client sends out a query to its RTSC, this peer will forward the request to the peer that contains the keyword search in its hash key. By using this method, the RTSC cloud will distribute this query to every DXM that contains the search keyword. The DXM will continuously compare the inputstream with the event criteria. When the incoming data satisfies the criteria, the DXM will deliver the data to the SB client using the proposed NAT traversal method.

If the data is not available, the client can use the Event Alerting Service to watch the event. Whenever the data is available, the client will receive the alert immediately. The Event Alerting Service is using the same procedure as the Data Collection Service. However, the request will be kept within the DXM until the data is received by the client.

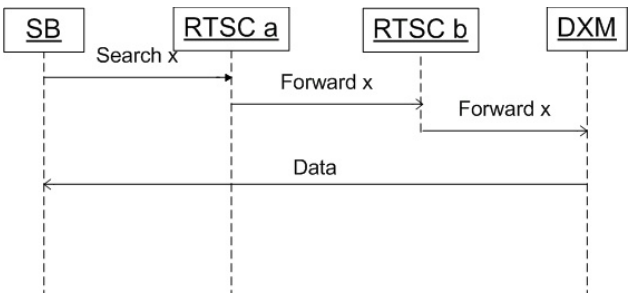

Figure 6.1: Data Collection and Event Alerting Service

CHAPTER SEVEN

DATA PUBLISHING SERVICE

1. Using DXM: If the users have a big database, they can share it using the DXM. First, they should make sure that their database can connect to the DXM. Then, they can register their DXM connection to the superpeer so that the superpeer can tell the other peer to locate their DXM. Currently, the DXM only support the MSAccess database connection. However, it should be easy for the developers to add other database connection, such as MySQL, to their DXM . The DXM owner must verify that the database connection defined in the dxmConfig xmlfile is correctly pointed to their database. Once the connection is setup, they should be able to start the DXM component and see the connected status. Then, the data provider can register their DXM to the RC by using the service provided by SB. They have to submit their public IP address, their DXM port number as defined in the dxmconfig filexml (this port must be open for RTSC and SB to setup their connection), and their DXM name. The request will be submitted to the RC to verify. Once the request is accepted, the RC will add this DXM connection to the routingtable.xml list and announce its existence to RTSC peer. RTSC peer will then update their local cache and open the connection to the new DXM.

2. Submit the data to the central database (Figure 7.1 and 7.2): If the users just want to share

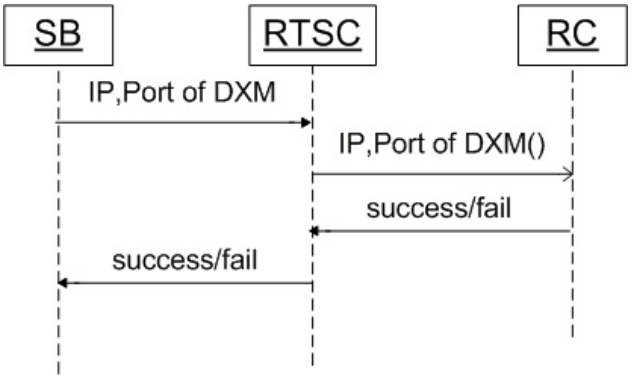

Figure 7.1: Data Publishing Service

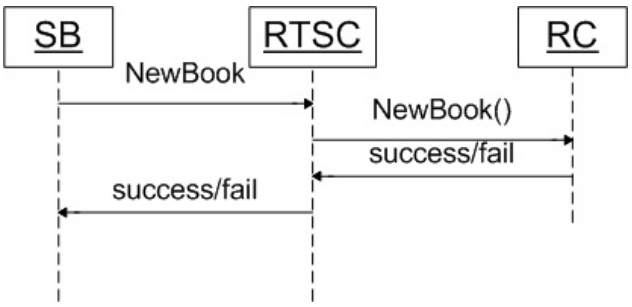

Figure 7.2: Data Publishing Service

a single piece of data instead of a big database, they can upload their data into the central database by using the SB service. The users can use the SB service to create and submit their dataset to the RC. After the RC verifies this dataset, the dataset is saved into the central database and the RC will send the new dataset ID back to the client. The client can later view and modify this dataset using the dataset ID.

CHAPTER EIGHT

REMOTE CONTROL SERVICE

The Remote Control Service is one of the most important properties of the Wireless Network Management. Therefore, I will introduce the definition and architecture details of the Wireless Network Management. My discussion is based on material available in [4]

8.1 Definition

According to the definition by ISO, Network management includes the deployment, integration, and coordination of the hardware, software, and human elements to monitor, test, poll, configure, analyze, evaluate, and control the network and element resources to meet the real-time, operational performance, and Quality of Service requirements at a reasonable cost. The ISO has also created a network management model that is useful for placing all scenarios in a more structured framework. Five areas of network management are defined in that model:

- Performance Management: Quantify, measure, report, analyze, and control the performance of different network components.Protocol standards such as the Simple Network Management Protocol (SNMP) [RFC3410] play a central role in Internet performance management.

- Fault Management: Log, detect, and respond to fault conditions in the network. Fault management is the immediate handling of transient network failures, while performance management is the longer term of performance. SNMP plays a central role in fault management too.

- Configuration Management: Allows a network manager to track which devices are on the managed network and the hardware and software configurations of these devices.

- Accounting Management: Allows specifying, logging and controlling user and device access

to network resources. Usage quotas, usage-based charging, and the allocation of resource-access privileges all fall under accounting management.

- Security Management: Control access to network resources according to some well-defined policy. The main components of security management includes the key distribution centers and certification authorities, and the use of firewalls to monitor and control external access points to ones network.

First, the management should initialize the network system (configuration management). If no errors occur, the network comes into service and the operational phase starts. During this phase, the management monitors the network system to check errors. In case of failures, the malfunctioning system will be identified, isolated and repaired. If the system can not be repaired, it will be replaced by a new system, which also must be initialized (fault management). New systems may also be added to allow the connection of new users, to increase performance or to add new functionality. The addition of the new system usually implies reconfiguration. Monitoring the network is also useful to detect changes in the traffic flow. Once such changes are detected, network parameters may be modified to optimize the networks performance (performance management).

8.2 Architecture

Based upon the information collection and communication strategy, there are three types of network management architectures: Centralized, Distributed, and Hierarchical

In a centralized network management system, management decisions will be taken from the limited number of central locations. The management functionality that takes these decisions is called the manager. To manage the operation of the primary functions, agents should be added to the systems that perform primary functions. Such agents represent the management support functionality through which manager(s) initialize, monitor and modify the behavior of the primary

functions. To allow managers to communicate with their agents, a management information protocol is necessary. Examples of such protocols are Common Management Information Protocol (CMIP) and Simple Network Management Protocol (SNMP).

With distributed management, there are no central systems from which management decisions are taken. Instead, functions that take such decisions will be added to the systems that already perform the primary functions. Such addition will usually be performed on a proportional scale. A distributed management system has multiple manager stations; each manages a subnetwork and communicates with other manager stations in a peer-to-peer manner. This approach has been adopted by the emphTelecommunication Management Network (TMN) and management model for emphAsynchronous Transfer Mode (ATM) networks.

Hierarchical network management systems use intermediate managers to distribute the manager tasks. Each intermediate manager has its domain; it collects and processes node information of its domain and passes the information to the upper level manager if necessary. It also distributes the messages from the upper level manager to nodes in its domain. There is no direct communication between intermediate managers.

A disadvantage of distributed management is that it will be difficult to change after the operational phase has started the functionality that makes the management decisions, because such changes require the modification of a large number of network systems, which will be expensive. It would be better to use the centralized management approach and concentrate the management functionality that makes the decisions within a single system. It is also easier to introduce Intelligent Networks when using centralized management.

A disadvantage of centralized management is that the entire network may get out of control after the failure of a single manager. Compared to distributed management, centralized management may also be less efficient: it is likely that more management information needs to be exchanged and the central managers may become performance bottlenecks.

8.3 Traditional Network Management

The Simple Network Management Protocol (SNMP) was developed in the late 1980s to provide network operators with a simple tool they could use to manage their networks. It has gained widespread acceptance since 1993, making it a standard to manage TCP/IP networks.

SNMP is based on the client-server centralized paradigm, where a central station collects and analyzes data retrieved from physically distributed network elements. The SNMP Manager makes the connections to an SNMP Agent which runs on a remote network device, and serves information to the manager regarding the devices status. The database, controlled by the SNMP agent, is referred to as the SNMP Management Information Base (MIB), and is a standard set of statistical and control values. Directives, issued by the network manager to an SNMP agent, consist of the identifiers of SNMP variables (referred to as MIB object identifiers or MIB variables) along with instructions to either GET the value for identifier, or SET the identifier to a new value. Through the use of private MIB variables, SNMP agents can be tailored for a lot of specific devices, such as network bridges, gateways, and routers. The definitions of MIB variables supported by a particular agent are incorporated in descriptor files.

The popularity of SNMP is due to a number of features. It can cover a large range of devices to be managed, and it is a very flexible and extensible management protocol. It is also proved to be good under poor network conditions. However, SNMP is not a particularly efficient protocol. Bandwidth is wasted with needless information, such as the SNMP version (transmitted in every SNMP message) and multiple length and data descriptors scattered throughout each message.

The network management systems based on Client/Server paradigm normally requires transferring large amounts of management data between the manager and agents. The large amount of data not only requires considerable bandwidth, but also can cause a processing bottleneck for management. As current networks grow larger and more complicated, the problem becomes more severe.

The Management by Delegation (MbD) model was proposed in 1991 to address the difficult to manage centralized systems. The key idea of the MbD approach is to delegate management functions to remote devices in order to reduce communication costs, to avoid a single point of failure, and to increase the scalability of management applications. The management architecture of MbD still includes a management protocol and agents, yet an elastic process run-time support is assumed on each device. Instead of exchanging simple messages, the management station can specify a task by packing a set of commands to agents into a program and send it to the devices involved, thus delegating the actual execution of the task to them. This execution is completely asynchronous, enabling the management station to perform other tasks in the meantime and introducing a higher degree of parallelism in the management architecture. Moreover, since the code fragments are not statically bound to devices, they can be changed and re-sent by the management station at any time. This enables more flexibility, because the management station can customize and enhance dynamically the services provided by the agents on the devices.

Remote Monitoring (RMON) assumes the availability of network monitoring devices called monitors or probes. By monitoring packet traffic and analyzing the headers, probes provide information about links, connections among stations, traffic patterns, and status of network nodes. Hence, RMON can be regarded as traffic-oriented approach because the status of the network is determined by direct inspection of the packets flowing in it, rather than inspection of the status of each device. A probe in RMON can detect failures, misbehaviors, and identify complex relevant events even when not in contact with the management station, which is likely to happen when the network is overloaded or in critical conditions. In addition, the agent on the probe can also do periodic checking and semantic compression, which further increases decentralization.

Another solution for the problem of centralized management is the use of Mobile Agent (MA) technology to distribute and delegate management tasks. The emergence of mobile agent frameworks has led many researchers to examine their applicability to network management and control environments. It is believed that mobile agents can provide better solutions to performance and

fault management problems, given the large amount of data that needs to be transferred in respective solutions based on traditional approaches. Ten Mobile agent frameworks are currently addressed by two standards bodies. The Federation of Intelligent Physical Agent (FIPA) looks at high-level semantically rich interactions between software agents that deploy some form of intelligent adaptability. It has its roots in Distributed Artificial Intelligence (DAI). OMG looks mostly at the issue of mobility according to a standard interoperable framework through its Mobile Agent System Interoperability Facility (MASIF). In the latter, the agent systems model the execution environment able to host mobile agents.

8.4 Sensor Network Management

Network management becomes more and more necessary with the development of applications running on WSNs. When the requirement of management in WSNs first arose, the most natural way to do it was to try to apply what we have already had for traditional wired networks into WSNs, such as the traditional standard network management protocols, SNMP. However, it is not possible, because the unique challenges posed by WSNs for network management make traditional network management techniques impractical. For example, the following characteristics of wireless sensor networks, which really matter to the design of network management system, make SNMP not applicable to WSNs:

First, there is no address for each sensor, and specifying sensors is difficult. The only way to transmit messages among WSNs is to broadcast the message to all sensor nodes no matter the message is planned to be sent to all sensor nodes or only one specific node. The communication overhead becomes too high when applying SNMP directly to WSNs. Second, for some self-configured WSNs, the management server does not have all information of sensor nodes. In order to apply SNMP directly, it requires each sensor node to maintain a MIB, and the big size of MIB makes it impractical for the storage-constrained sensor nodes. Third, due to the high density of the deployment of sensor nodes, sensor-specific failures become very common. This is a unique

characteristic of WSNs, and is not handled by SNMP.

Ad Hoc Network Management Protocol ANMP and Guerilla are two protocols designed for managing mobile wireless ad-hoc networks, but they can only be used with certain types of WSNs [22] [31]. ANMP uses hierarchical clustering of nodes to reduce the number of messages exchanged between the manager and the agents. It is an extended SNMP with the differences including MIB extensions, dynamic configuration of agents, dynamic extension of the agents, and an applicationspecific security module. The main contribution of ANMP is to make SNMP work for wireless networks. Guerilla is another adaptive management architecture for ad hoc networks, which provides management flexibility and continuity by making its nomadic managers adapt to dynamic network conditions. It employs a two-tier infrastructure: the higher tier consists of groups of peer-to-peer nomadic managers that process management intelligence, adapt to network dynamics, collaborate among one another; the lower tier consists of active probes that may be dispatched to remote nodes to perform localized management operations. The nomadic managers and active probes facilitate disconnected management operations and reduce consumption of wireless bandwidth.

Management Architecture for Wireless Sensor Networks MANNA [29], is a management solution specific for WSNs, but it adopts ad hoc network management techniques. It provides a general framework for policy-based management of sensor networks. It collects dynamic management information, maps this into WSN models, and executes management functions and services based on WSN models. MANNAs management policy specifies management functions that should be executed if certain network conditions are met. WSN models maintain the information about the state of the network. MANNA defines the relationship among WSN models in a Management Information Base (MIB). MANNA adapts to dynamic WSN behaviors by analyzing and updating the MIB. MIB update is a centralized operation and expensive in terms of energy consumption. Moreover, WSN uncertainties and delay may affect the accuracy of collected management information. To keep the MIB up-to-date, it is critical to determine the right time to query for management

information and the right frequency for obtaining management information.

Another system based on traditional network management systems is BOSS. It proposes a service discovery management architecture for WSNs. The architecture is based on UPnP, the standard service discovery protocol for network management. To make UPnP run on resource-constrained sensor nodes Song et al. implements an UPnP agent in the base station, called Bridge Of the SensorS (BOSS), which provides a bridge between a managed sensor network and a UPnP network. The proposed system consists of three main components: UPnP control point, BOSS, and non-UPnP sensor nodes. The control point is a powerful logical device with sufficient resources to run the UPnP protocol and manage a sensor network using the services provided by BOSS, e.g. PCs, PDAs, and notebooks. BOSS is a base node that acts as the mediator between non-UPnP sensor nodes and UPnP control point and is implemented in the base station. Each node in a sensor network is a non-UPnP device with limited resources and sensing capability. The base node carries the network management computation burden, rather than the resource-constrained sensor nodes. The control point can specify which events of non-UPnP sensors it is interested in.

A management framework called Sensor Network Management Protocol, sNMP [12], is proposed by Deb et al. The sNMP framework has two functions: First, it defines sensor models that represent the current state of the network and defines various network management functions. Second, it provides algorithms and tools for retrieving network state through the execution of the network management functions. Models for sensors include network topology (node connectivity), energy map (node battery power), and usage patterns. Deb et al. suggest that sensor models could be used for different network management functions. The human manager could use the current knowledge of network topology for future node deployment. By measuring network states periodically, the human manager can monitor and maintain the network by identifying which parts of the network have a low performance, and taking corrective actions as necessary. From periodic monitoring of network states, the human manager could also analyze network dynamics to predict network failures and then take preventive actions.

Louis Lee et al. propose an adaptive policy-based management system for WSNs, called Wireless Sensor Network Management System (WinMS). The end user predefines management parameter thresholds on sensor nodes that are used as event triggers, and specifies management tasks to be executed when the events occur. A local network management scheme provides autonomy to individual sensor nodes to perform management functions according to their neighborhood network state, such as topology changes and event detections. The central network management scheme uses the central manager with a global knowledge of the network to execute corrective and preventive management maintenance. The central manager maintains an MIB that stores WSN models that represent network states. The central manager analyzes the correlation among WSN models to detect interesting events such as areas of weak network health, possible network partition, noisy areas, and areas of rapid data changes. An advantage of WinMS is that its lightweight TDMA (Time Division Multiple Access) protocol provides energy-efficient management, data transport and local repair. Its systematic resource transfer function allows non-uniform and reactive sensing in different parts of a network, and it provides automatic self-configuration and self-stabilization both locally and globally by allowing the network to adapt to current network conditions without human intervention. A disadvantage of WinMS is that the initial setup cost for building a data gathering tree and node schedule is proportional to network density. Tolle and Culler propose Sensor Network Management System SNMS. It is an interactive system for monitoring the health of sensor networks. SNMS provides two main management functions: query-based network health data collection and event logging. The query system allows the user to collect and monitor physical parameters of the node environment. The event-driven logging system allows the user to set event parameters and nodes in the network will report their data if they meet the specified event thresholds. The main advantage of SNMS is that it introduces overhead only for human queries and so has minimal impact on memory and network traffic. SNMS further minimizes energy consumption by bundling the results of multiple queries into a single message instead of returning results individually. The main drawbacks of SNMS are that the network management function is limited

to passive monitoring only, requiring human managers to submit queries and perform post-mortem analysis of management data. Furthermore, SNMPs centralized-processing approach requires continuous polling of network health data from managed nodes to the base station, and this can burden sensor nodes that should minimize transmissions in order to extend network lifetime.

8.5 Remote Procedure Call In Sensor Web Service Framework

The core RPC service is a lightweight communication layer built on top of the standard TinyOS communication stack. The service is realized as a single nesC module. Every proxy and skeleton generated by the RPC compiler uses this module to send and receive marshalled invocation requests. In addition to providing transport services, the RPC core supports discoverybased binding, and provides buffering and arbitration support. Hence, if a remote invocation is placed while the radio is busy, the marshalled invocation will be buffered for later transmission. Buffered requests are sent in the order that the invocations were placed. A similar buffering strategy is useful when messages are received. Receiver-side buffering prevents messages from being dropped when the destination skeleton is busy servicing a request.

The Sensor Web Service Framework uses the Sensor Interpretation Standard to identify which RPC commands are supported by the sensor network, and how the clients can send out the RPC commands to the sensor network. For example, when a client wants to send out a RPC command to the sensor network, it will search in the `<Write>` component to see which RPC commands are provided by the sensor network. Suppose the client wants to send out the `<setReportLevel>` command, which is provided by the sensor network (Figure 8.1), it will also look into the `<MessageHierarchy>` component to understand how to construct the RPC message (Figure 8.2). After the RPC message is constructed, the client will use the `<DataFormat>` component to fill the empty RPC message with some values. As explained in the chapter 4, the value could be a fixed value or could be user-input value (when the input keyword is used) (Figure 8.3)

```
<Write> <data dataName="setReportLevel" excuteType="write" />
</Write>
```

Figure 8.1: Setreport Level RPC Command

```
<struct structName="setReportLevel">
<field message="TOSMessage" messageField="addr" encode="dec"
value="65535" />
<field message="TOSMessage" messageField="group" encode="dec"
value="125" />
<field message="RpcCommandMessage" messageField="unix_time"
encode="hex" value="4a122482" />
<field message="RpcCommandMessage" messageField="user_hash"
encode="hex" value="4c6e9f74" />
<field message="RpcCommandMessage" messageField="returnAddress"
encode="dec" value="0" />
<field message="RpcCommandMessage" messageField="responseDesired"
encode="dec" value="1" />
<field message="RpcCommandMessage" messageField="commandID"
encode="dec" value="1" />
<field message="RpcCommandMessage" messageField="dataLength"
encode="dec" value="2" />
<field message="RpcCommandMessage" messageField="address"
encode="dec" value="65535" />
<field message="SetReportLevelMsg" messageField="type" encode="dec"
value="input" />
<field message="SetReportLevelMsg" messageField="level"
encode="dec" value="input" />
</struct>
```

Figure 8.2: setReportLevel RPC Command Detail

```
<MessageHierarchy>
<TOSMessage type="133">
<NetworkMessage type="0">
<ApplicationMessage type="0">
<RpcCommandMessage> <SetReportLevel /> </RpcCommandMessage>
</ApplicationMessage>
</NetworkMessage>
</TOSMessage>
</MessageHierarchy>
```

Figure 8.3: Hierarchy of The SetReportLevel Message

CHAPTER NINE

IMPLEMENTATION AND TESTING

9.1 Development Environment

9.1.1 Software Development Environment

In order to test the Sensor Web Service Framework performance, two parts need to be developed separately: the Sensor Web Service Framework and the data sources.

The Sensor Web Service Framework is developed in Java version 6. Java version 6 is the newest version of Java, which supports multiple network communication services including Java RMI. Furthermore, Sensor Web Service Framework uses Java RMI as its standard communication model that has potentially low interoperability and does not inherently support asynchronous communication that is of vital importance for most SOA based applications. However, Java relies heavily on the Java Virtual Machine (JVM) that means every service intending to join the network needs to support JVM. The restriction largely limits the usage of Sensor Web Service Framework worldwide, especially for those who are not willing to use Java platforms. In order to overcome this disadvantage, we have created installation packages for each component of our Sensor Web Service Framework. The installation package have embedded the JVM within our Sensor Web Service Framework's component so that our software can work on the computers, which do not have their own JVM.

The data sources can be a MS Access database, which is developed using SQL or it could also be a wireless sensor network, which is developed in the Nesc language. NesC (network embedded systems C) is a dialect of the C programming language optimized for the memory limitations of sensor networks. It is a component-based, event-driven programming language used to build applications for the TinyOS [20]. TinyOS is an operating environment designed to run on embedded devices used in distributed Wireless Sensor Networks. In TinyOS, the programmer

can develop several components separately such as sensing component, network component, and Remote Procedure Call (RPC) component. Components then provide certain interfaces to their users and in turn use other interfaces from underlying components. For example, the network component can provide the interface for the sensing component and the RPC component to send out their messages.

In addition, TinyOS provides a set of Java tools in order to communicate with sensor networks via a program called SerialForwarder, which runs as a server on the host machine and forwards all the packages received from sensor networks to the local network.

9.1.2 Hardware Development Environment

Currently, three types of sensor node are widely used in sensor network community include the MICAz mote which has 4K RAM space and 8K ROM space [1], the Tmote Sky sensor node [26] with 8K RAM and 40K ROM, and the new generation platform, Intel iMote2 sensor node. All of these nodes adopt the Chipcon CC2420 [3] radio chip as the radio component, which is a 2.4 GHz IEEE 802.15.4 and Zigbee compliant RF transceiver. In order to minimize code size as required by the memory constraints inherent in sensor networks and guarantee the real-time characteristic, several event driven and lightweight operating systems are developed for WSNs.

For those high-fidelity sensing application such as volcano monitoring, due to limited RAM, these two families of motes need to buffer data to EEPROM or Flash storage component. However, the time delay and energy consumption on the external storage medium is a constraint for real-time applications. In order to meet the requirements of timeliness and high data rate network, we need to use the iMote2 which has more computation capacity. In this project, we also used the iMote2 to setup the wireless sensor network for our test environment.

The iMote2 sensor node [17] is an advanced wireless sensor node platform. The platform is built around a low power XScale processor, PXA271. It integrates an 802.15.4 radio (ChipCon 2420) and a built in 2.4 GHz antenna. The iMote2 platform is a modular stackable platform and

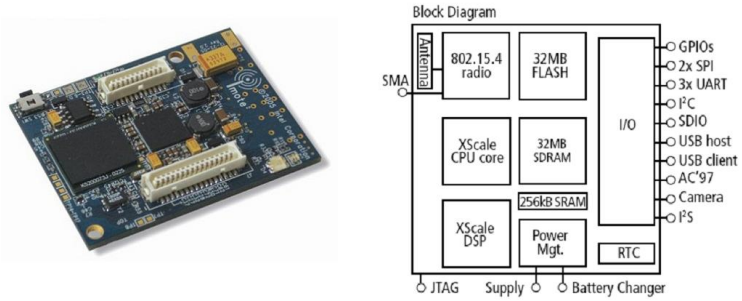

Figure 9.1: iMote2 Sensor Node

can be stacked with sensor boards to customize the system to a specific application, along with a "power board" to supply power to the system.

The PXA271 processor on iMote2 can operate in a low voltage (0.85 V) and a low frequency (13 MHz) mode, hence enabling low power operation. The frequency can be scaled to 104 MHz at the lowest voltage level, and can be increased up to 416 MHz with Dynamic Voltage Scaling. It also integrates 256 KB of SRAM divided into 4 equal banks of 64 KB. The PXA271 is a multi-chip module that includes three chips in a single package, the processor, 32 MB SDRAM and 32 MB of flash. The processor integrates many I/O options making it extremely flexible in supporting different sensors, A/Ds, radio options, etc. These I/O options include I2C, 3 Synchronous Serial Ports one of which dedicated to the radio, 3 high speed UARTs, GPIOs, SDIO, USB client and host, AC97 and I2S audio codec interfaces, fast infrared port, PWM, and Camera Interface. The processor also adds many timers and a real time clock. The PXA271 also includes a wireless MMX coprocessor to accelerate multimedia operations. It adds 30 new media processor instructions, support for alignment and video operations and compatibility with Intel MMX and SSE integer instructions.

9.1.3 Test Environment Setup

Based on the work in previous chapters, the Sensor Web Service Framework was designed and implemented, which supports a data collection service, event alerting service, data publishing and

remote control service. To test the validity of these core set of management services in our design, I setup a small wireless sensor network consisting of eight sensor nodes. This sensor network was running the Oasis application, which is used in the Oasis project to monitor the volcano. I will call this WSN as the data source 1. In addition, I also created a small bookstore database to demonstrate the functionality of the DXM Database. I will call this database as the data source 2. The data source 1 and data source 2 connected to different DXM and running on separate computers. Then I have another two separate machines to run the RTSC peer, and one extra machine to run the RC server. Finally, I have setup ten clients machine to request data from these two data sources simultaneously.

9.1.4 DXM Setup

In the Sensor Web Service Framework, each data provider must define their dataset in the XML file. The DXM component will use the XML to interpret input from the datasource. The most important attributes of the datasource are:

- Datasource name is used as the end-user search keyword.
- The database and datastream datasource must use different parsers. Therefore, the datasource type is important for the DXM to know which parser can be used.
- Data name is one of the search criteria. For example, the user can choose to search for Computer Book.
- Data value is one of the search criteria. For example, the user can choose to search for a price between $10 and $20 (Table 1).

When the inputstream meets the search criteria, the DXM will send the data, as well as this XML, to the client machine so that they know how to visualize this dataset.

Table 9.1: Example of a simple DXM's XML file

DataSource	DataSourceType	DataName	DataValue
Book	Database	BookName	Price
Volcano	DataStream	Seismic	Amplitude
Volcano	DataStream	Infrasonic	Amplitude

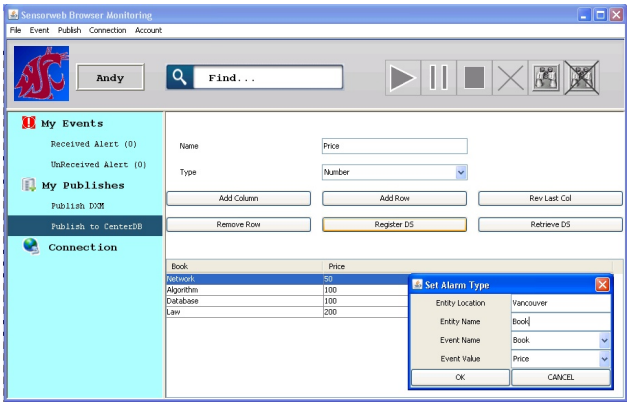

Figure 9.2: Publish data to central database

9.1.5 Sensorweb Browser Demonstration

To lay the groundwork for our exploration of the Sensor Web Service Framework design, I will give a brief description of the Sensorweb Browser component of the Sensor Web Service Framework. The Sensorweb Browser is the end-user's application that allows the naive user to publish or query real-time information through the Sensor Web Service. In Figure 9.2, we see a screen shot of the Sensorweb Browser. In this panel, the Sensorweb Browser provides the user with the ability to create their own dataset. This dataset consists of different columns, which are either text or numeric. After creating the dataset, the user can publish his dataset to the central server so that others can find it.

In Figure 9.3, our system gives the users the ability to publish their own database or their WSN datastream. This ability illustrates our idea of providing a realtime and situation-aware information sharing service over the Internet in the P2P manner. When data providers update their local database or WSN datastream, the clients machines will immediately receive the alert.

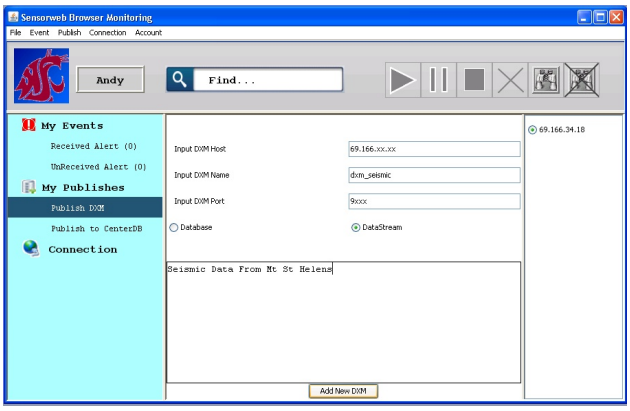

Figure 9.3: Publish DXM connection. DXM can serve the Serial Forwarder or the Microsoft Access Database

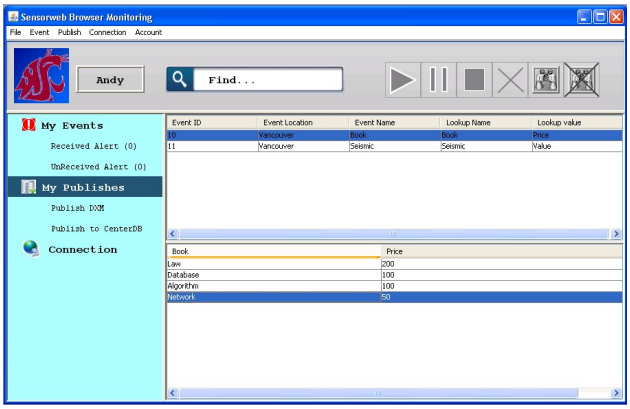

Figure 9.4: Published Event of the current users

In the central database approach, the data providers must also update their data into the central database. Currently, our system can support the Microsoft Access database and WSN datastreams coming from a Serial Forwarder. The Serial Forwarder is a Java application developed by the Tinyos community. It can receive the inputstream from the sink node and send the outputstream to a TCP port. In addition, minor changes can be made so that the system can support different databases such as MySQL or Microsoft SQL Server. In the Sensor Web Service Framework, the Data Exchange Middleware (DXM) is used to connect the Serial Forwarder, or a database, to the other components of the Sensor Web Service Framework.

When the user logs into the system, he can see his current published events as in figure 9.4.

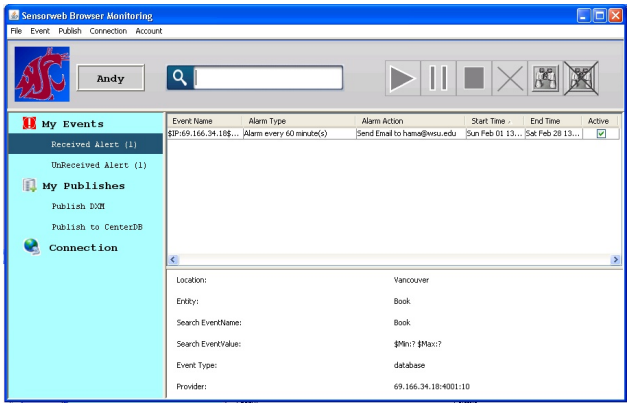

Figure 9.5: Event Management Functions allow users to set alarm time, alarm type and email address to send alert mes sage

When the event is registered into the Sensor Web Service, it will be given a timestamp. If the event is inactive for a long period of time, it will be deleted from the server, and the publisher will receive a message asking whether he wants to republish it or not. The most important function of the Sensor Web Service is the event querying and alerting. In figure 9.5, the user first searches for the book event, which already exists in the system. Therefore, this event is shown in the list of ”Received Events”. Then, the user can further set the alerting properties; such as, how often the event should be alerted, what email address will receive the alert, and when the event should be monitored. For example, if he chooses to monitor events between 9:00 AM and 9:00 PM, he will not receive an email alert when the event is received outside of this time window. When the user receives the alert, he can view the event details as in figure 9.6. In this version, we support the table view for database event and the oscilloscope view for datastream event such as seismic or infrasonic stream.

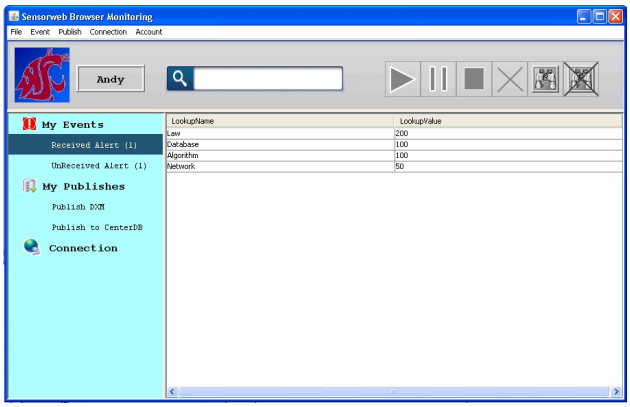

Figure 9.6: View Event Panel shows the event details such as BookName and Price

CHAPTER TEN

CONCLUSIONS

Due to the continuing advances in network and application design in WSNs, the development of a sensor network management system is becoming necessary and possible. Because the significant differences between traditional networks and WSNs, a different management solution for WSNs is required.

10.1 Main Contributions

In this thesis, a SensorWeb Service Framework has been proposed. The SensorWeb Service Framework demonstrates some of the benefit of the realtime and situation-aware information sharing services over the Internet.

- It provides a lightweight standard to interpret the sensor data from various sensor networks.
- It provides the data collection service, the event alerting service, the data publishing service so that the user can publish, query and set alerts on various data sources, which can be a database and a sensor network. It also bridges the gap between the end-users and the data providers so that any changes in the data providers datasource is immediately alerted in the client machine.
- It provides a lightweight sensor network management based on RPC.
- It provide a content-based routing overlay network to publish or query the data efficiently in a P2P manner. Finally, the Sensor Web Service Framework combined the TCP Hole Punching and Relay Method to travel NAT reliably. It is our hope that the application will help achieve the full potential of the WSNs to enable realtime information sharing.

10.2 Future Work

One limitation to the current designed Sensorweb Service Framework is the dependency on the communication module of the RTSC Cloud. As mentioned earlier, we use the TCP Hole Punching Method to setup the direct connection between the Sensorweb Browser and the DXM. However, the successful ratio of this method is low. The quality of the Collection and Dissemination communication patterns will directly effect the correctness and efficiency of management functions. Therefore, more research is needed to improve the communication module between the Sensorweb Browser and the RTSC Cloud.

Another special issue related to our Sensorweb Service Framework design is the security. By providing the RPC function to the end-users, the clients can remotely access a nodes functions or variables over a wireless network. The consistency of application information between client and server side is needed to avoid faulty actions. Therefore, we also need to consider to provide authentication boundary to protect our data sources.

BIBLIOGRAPHY

[1] Micaz datasheet http://www.xbow.com/products/product_pdf_files/wireless_pdf/micaz_datasheet.pdf.

[2] Ogc http://www.opengeospatial.org/.

[3] Chipcon CC2420 Datasheet: http://focus.ti.com/lit/ds/symlink/cc2420.pdf, Texas Instruments, 2007.

[4] B. N. Bershad, T. E. Anderson, E. D. Lazowska, and H. M. Levy. Lightweight remote procedure call. *ACM Transactions on Computer Systems*, 8(1):37–55, February 1990.

[5] Andrew Biggadike, Daniel Ferullo, Geoffrey Wilson, and Adrian Perrig. Natblaster: Establishing tcp connections between hosts behind nats. In *SIGCOMM Asia Workshop*, 2005.

[6] Mike Botts, George Percivall, Carl Reed, and John Davidson. Sensor web enablement: Overview and high-level architecture. Technical report, December 2007.

[7] De Bruijn. A combinatorial problem. 49:758–764, 1946.

[8] Hybrid Sensor Network For cane Toad Monitoring. http://www.cse.unsw.edu.au/~sensar/research/projects/cane-toads/.

[9] R. Cardell-Oliver, K. Smettern, M. Kranz, and K. Mayer. Field testing a wireless sensor network for reactive environmental monitoring. December 2004.

[10] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the world with wireless sensor networks. In *IEEE ICASSP Conference*, May 2001.

[11] Bryan Ford, Pyda Srisuresh, and Dan Kegel. Peer-to-peer communication across network address translators. In *USENIX*, 2005.

[12] Saikat Guha, Yutaka Takeda, and Paul Francis. Nutss: A sipbased approach to udp and tcp network connectivity. In *SIGCOMM Workshops*, August 2004.

[13] Carl Hartung, Richard Han, Carl Seielstad, and Saxon Holbrook. Firewxnet: A multitiered portable wireless system for monitoring weather conditions in wildland fire environments. In *the 4th international conference on Mobile systems, applications and services*, June 2006.

[14] Douglas Herbert, Vinaitheerthan Sundaram, Yung-Hsiang Lu, Saurabh Bagchi, and Zhiyuan Li. Adaptive correctness monitoring for wireless sensor networks using hierarchical distributed run-time invariant checking. *ACM Trans. Auton. Adapt. Syst.*, 2(3), September 2007.

[15] Soil Moisture Monitoring With Wireless Sensor Networks Project Homepage. http://www.csse.uwa.edu.au/adhocnets/wsngroup/soil-water-proj/.

[16] W. Hu, V. N. Tran, N. Bulusu, C. T. Chou, S. Jha, and A. Taylor. The design and evaluation of a hybrid sensor network for cane-toad monitoring. In *In Proceedings of Information Processing in Sensor Networks*, April 2005.

[17] Intel. iMote2 Datasheet: http://www.xbow.com/Products/Product_pdf_files /Wireless_pdf/Imote2_Datasheet.pdf.

[18] Holger Karl and Andreas Willig. *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons, June 2005.

[19] Sukun Kim, Shamim Pakzad, David Culler, James Demmel, Gregory Fenves, Steve Glaser, and Martin Turon. Wireless sensor networks for structural health monitoring. In *Proc. 4th ACM conference on Embedded networked sensor systems (SenSys 2006)*, November 2006.

[20] Philip Levis, Sam Madden, David Gay, Joseph Polastre, Robert Szewczyk, Alec Woo, Eric Brewer, and David Culler. The emergence of networking abstractions and techniques in

tinyos. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, page 1, Berkeley, CA, USA, 2004. USENIX Association.

[21] F. L. Lewis. Wireless sensor networks. 2004.

[22] Zhigang Li, Xingshe Zhou, Shining Li, Gang Liu, and Kejun Du. *Issues of Wireless Sensor Network Management*, pages 355–361. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2005.

[23] B. Lo and G. Z. Yang. Key technical challenges and current implementations of body sensor networks. In *Proc. 2nd International Workshop on Body Sensor Networks (BSN 2005)*, April 2005.

[24] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. Wireless sensor networks for habitat monitoring. In *ACM International Workshop on Wireless Sensor Networks and Applications*, September 2002.

[25] Minenet. https://sensorweb.vancouver.wsu.edu/wiki/index.php/cs580.

[26] Moteiv. *Tmote Sky Datasheet http://www.sentilla.com/pdf/eol/tmote-sky-datasheet.pdf*, 2006.

[27] Suman Nath, Jie Liu, and Feng Zhao. *SensorMap for Wide-Area Sensor Webs*. Microsoft Research, July 2007.

[28] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. Stun simple traversal of user datagram protocol(udp) through network address translators (nats),. Technical report, March 2003.

[29] Linnyer B. Ruiz, Jose M. Nogueira, and Antonio A. F. Loureiro. Manna: A management architecture for wireless sensor networks. *IEEE Communications Magazine*, 41(2):116–125, February 2003.

[30] Siva and B. S. Manoj. *Ad Hoc Wireless Networks Architectures and Protocols*. Communications Engineering and Emerging Technologies. Prentice Hall, 2004.

[31] Hyungjoo Song, Daeyoung Kim, Kangwoo Lee, and Jongwoo Sung. Upnp-based sensor network management architecture. In *The Second International Conference on Mobile Computing and Ubiquitous Networking*, April 2005.

[32] Wenzhan Song, Renjie Huang, Mingsen Xu, Andy Ma, Behrooz Shirazi, and Richard Lahusen. Air-dropped sensor network for real-time high-fidelity volcano monitoring. In *Mobisys*, June 2009.

[33] Wenzhan Song and Xiangyang Li. Cbrbrain: Provide content based routing service over internet backbone. In *IEEE ICCCN 2004*.