Web Services as application enabler for Sinkless Wireless Sensor Networks

Nuru Yakub Othman

A Thesis

in

the Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science at
Concordia University
Montréal, Québec, Canada

February 2007

Your file Votre référence
ISBN: 978-0-494-28923-5
Our file Notre référence
ISBN: 978-0-494-28923-5

Abstract

Web Services as application enabler for Wireless Sensor Network

Nuru Yakub Othman

It is foreseen that in the near future, several Wireless Sensor Networks (WSN) will be deployed to provide data and services to end user applications. WSN will play an important role in context aware applications and services, which adapt to situations of the surrounding environment and that of their host devices. However, conventional WSNs are built using proprietary protocols and use proprietary data access frameworks. They necessitate the presence of gateways or protocol converters between the WSN and application devices, and require special expertise from application developers. Further, dealing with heterogeneous sensor networks is difficult, as the protocols and data access mechanisms are different. There is a need to have a framework that is based on open standards to simplify application development and deal seamlessly with heterogeneous WSNs. This thesis proposes a Web Services based sinkless architecture for WSN as a potential solution. A prototype of an embedded web services platform is also built. It allows application devices to interact directly with the sensor nodes without going through a gateway. The thesis includes a survey and evaluation of several frameworks to highlight the potentials of web services. The use of Web Services will also attract a larger pool of application developers, leading to more innovative applications. Further, as the framework is built over IP, WSNs will no longer be treated as foreign islands. They will be part and parcel of ICT infrastructure in enterprises, as well as fit in easily in Value Added Services infrastructures for Next Generation Telecommunication Networks.

Acknowledgements

This work would not have been possible without the directions and continuous support from my supervisors; Dr. Roch H. Glitho and Dr. Ferhat Khendek. Their sound advice was vital in putting this work in focus and within the right scope. Being a member of Telecommunication Services Engineering Research Laboratory (TSELab) has been a valuable experience. I wish to thank all the group members for their comments, ideas and constructive criticisms, and sharing with me their experiences and perspectives. The biweekly meetings and presentations have been very useful indeed.

I also wish to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) and Ericsson Research for their financial and other material support.

My personal thanks also go to my family and friends for their support, encouragement and understanding. It all counts.

Nuru Yakub Othman, February 2007

Table of Contents

Table of Contents v
List of Figures vii
List of Tables viii
List of Acronyms and Abbreviations ix
Chapter 1 Introduction 1
1.1 Research Domain 1
1.2 Problem Statement and Contribution of the Thesis 2
1.3 Organization of the Thesis 3
Chapter 2 Background Information on Sensors, WSN and Web Services 5
2.1 Sensors 5
2.1.1 Definition 5
2.1.2 Hardware Components 6
2.1.3 Software Components 8
2.2 Wireless Sensor Networks 9
2.2.1 Description 9
2.2.2 Entities and their roles 9
2.2.3 Characteristics distinguishing WSN from other Networks 10
2.2.4 WSN Applications 11
2.3 Web Services Framework 12
2.3.1 Definition and Basic Principles 12
2.3.2 The 3 roles and 3 operations 13
2.3.3 Associated Technologies 15
2.4 Chapter Summary 17
Chapter 3 WSN Application Enablers: State of the Art and evaluation 18
3.1 WSN Application enablers 18
3.2 Driving forces behind the evolution of the frameworks 18
3.3 Categorization and Evaluation of the frameworks 20
3.3.1 Criteria for evaluation 20
3.3.2 Evaluation of Frameworks based on common technologies 24
3.3.3 Selected Proprietary frameworks 30
3.3.4 Evaluation Summary 33
3.4 Chapter Summary 34
Chapter 4 Proposed Web Services framework 35
4.1 Web Services for WSN 35
4.1.1 Web Services at the gateway 35
4.1.2 Web Services on the aggregator 38
4.1.3 Web Services on the sensor nodes 40
4.2 The Proposed Architecture 42
4.2.1 Motivations for Sinkless Wireless Sensor Network 43
4.2.2 The proposed model, its architectural entities and interactions 45
4.2.3 Protocol Stack and Addressing 47
4.2.4 Performance Pre-Assessment 49
4.3 Chapter Summary 51

Chapter 5 TinyWS prototype – Design and Implementation 52
5.1 TinyWS 52
5.2 Objectives 53
5.3 The Setup 54
5.4 Hardware and Software Tools 54
5.5 The design 56
5.5.1 The Services provided 57
5.5.2 The SOAP Messages 58
5.5.3 The Software Architecture 62
5.6 Challenges faced, lessons learned and related work 67
Chapter 6 Conclusion 72
6.1 Thesis Contributions 72
6.2 Future Work 74
References 76
Appendix A: Sample WSDL file for the TinyWS prototype 88

List of Figures

Figure 2.1 Main Hardware components of a sensor node 7
Figure 2.2: Some of the Common Sensor Platforms today 7
Figure 2.3: Software Components of a sensor node.....8
Figure 2.4: Wireless Sensor Network 9
Figure 2.5 Web Services roles and operations.....14
Figure 3.1 Agilla Mobile Agent framework for WSN.....28
Figure 4.1 Web Services on Sensor Gateway.....36
Figure 4.2 Web Services on Sensor Gateway, with UDDI in place.....37
Figure 4.3 WS on the gateway, entity interactions.....37
Figure 4.4 WSN for remote Monitoring 38
Figure 4.5 Web Services on the Aggregator.....39
Figure 4.6 Web Services at Aggregator, entity interactions.....40
Figure 4.7 Web Services on the Aggregator, indoor scenario 40
Figure 4.8 Web Services on sensor nodes.....41
Figure 4.9 WS on the sensor nodes, entity interactions.....42
Figure 4.10 WS on the sensor node, Indoor example 42
Figure 4.11 Sinkless WSN with routing done by the application devices 44
Figure 4.12 Web Services based sinkless architecture.....45
Figure 4.13 Entity Interactions for WS based sinkless WSN 46
Figure 4.14 Spatial IP address assignment.....48
Figure 4.15 Simulation results – Energy dissipation.....49
Figure 4.16 Simulation results – System lifetime.....50
Figure 4.17 Simulation results – Network load.....50
Figure 5.1 TinyWS: Setup of the prototype.....54
Figure 5.2 TelosB Sensor Node and its block diagram.....55
Figure 5.3 The SOAP Request for service SubscribeData.....59
Figure 5.4 The SOAP Response for service SubscribeData.....59
Figure 5.5 The SOAP Request for service ToggleLED.....60
Figure 5.6 The SOAP Response for service ToggleLED.....60
Figure 5.7 The SOAP Request for service GetLEDStatus.....60
Figure 5.8 The SOAP Response for the service GetLEDStatus.....61
Figure 5.9 The SOAP Request embedded in HTTP message 61
Figure 5.10 The SOAP Response embedded in HTTP message 62
Figure 5.11 Tiny WS Software Architecture.....63
Figure 5.12: Software Architecture for the client library 65

List of Tables

Table 1: An example modified SQL query for WSN 26
Table 2: Summary of the evaluation of the frameworks..... 33

List of Acronyms and Abbreviations

API:	Application Programming Interface
IEEE:	Institute of Electrical and Electronics Engineers
IETF:	Internet Engineering Task Force
OGC:	Open Geospatial Consortium
SOA:	Service Oriented Architecture
SOAP:	Simple Object Access Protocol
UDDI:	Universal Description, Discovery and Integration standard
WS:	Web Services
WSDL:	Web Service Description Language
WSN:	Wireless Sensor Network
W3C:	World Wide Web Consortium
XML:	eXtended Markup Language

Chapter 1 Introduction

1.1 Research Domain

The field of Wireless Sensor Networks (WSN) has become one of the very active research areas, with a lot of potentials to improve the quality of life through its various applications. It has been identified as one of the most important technologies of the 21st century [1]. The MIT enterprise technology review has placed WSN at the top of the list of ten emerging technologies that will change the world [2]. The technology was originally motivated by military applications, but as the sensor nodes become smaller, more powerful and cheaper, the applications have greatly diversified towards civilian. Notable application areas include environmental monitoring [17], Habitat Monitoring [18], Smart buildings [78], Structural monitoring [19], medical/health monitoring [80] and a wide range of industrial and commercial applications [54].

Most of the WSN research activities have focused on system architecture for miniaturization of the nodes and lowering their cost, low power radio links and energy aware routing algorithms. At a higher level, very active research activities are on the middleware frameworks for accessing data, reprogramming the sensor network and application integration. In conventional Wireless Sensor Networks, sinks collect data from sensors and/or aggregators and interact with applications via centralized gateways. However, there are situations in which this sink based modus operandi may not be suitable. In such cases, it is preferred that the applications interact directly with the sensors. This thesis focuses on data access framework in this operational setting.

1.2 Problem Statement and Contribution of the Thesis

There are several development frameworks for enabling end user applications to access sensor data. However, most of these frameworks used today are proprietary and highly specialised, meant for developers who are already sensor experts or are expected to undergo a learning curve in order to use them. This is particularly the case in the sinkless architecture where the applications are expected to interact directly with the sensor nodes. To allow general developers from diverse backgrounds, and to allow access to different kinds of sensors (heterogeneity), there is a need for a framework that is common to general application developers. This thesis proposes the use of Web Services as a potential framework to fulfill that need. It looks at the potential benefits and the various possibilities of using Web Services in WSN, proposes a comprehensive architecture and builds a prototype as a proof of concept.

So, the problem addressed by this thesis is the difficulty of accessing sensor data and other interactions between application devices and WSN due to the use of proprietary protocols and frameworks in WSN. The thesis advocates the use of open and standard protocols and providing data access via common frameworks. In particular, Web Services based framework has been chosen as a potential solution. The framework is expected to simplify and motivate development of new end user applications that make use of sensor data, as well as simplify the integration of sensor data to existing applications.

The contributions of this thesis include:

(i) A systematic survey and evaluation of several frameworks for accessing sensor data, from application developers point of view

(ii) Highlighting the potentials of web services as a common framework for heterogeneous WSN

(iii) Proposing a comprehensive web services based architecture for Wireless Sensor Network

(iv) Development of the first embedded Web Services platform residing on the actual sensor nodes.

(v) Enhancement of the idea of using conventional protocols such as TCP/IP in WSN instead of the proprietary ones.

1.3 Organization of the Thesis

Chapter 2 provides more detailed background information on the technologies and concepts involved. It gives a formal definition of sensors and explains its hardware and software components. Wireless Sensor Networks, their various entities, characteristics and applications are explained. Lastly, an overview of Web Services framework, its associated technologies and benefits are explained.

Chapter 3 surveys and evaluates the existing frameworks (State of the Art) for data collection and application development in Wireless Sensor Networks. It proposes a set of evaluation criteria and uses them to evaluate the existing frameworks. In doing so, it highlights the potentials of web services as a very promising framework.

In chapter 4, the various possibilities of using web services in Wireless Sensor Networks are presented, with relevant use cases. A comprehensive web services based

sinkless architecture is then proposed, with motivating application scenarios and use cases. A discussion of the protocol and addressing issues and how they can be overcome today is also presented.

As a proof of concept, Chapter 5 describes the design and implementation the prototype, TinyWS, an embedded web services platform on the sensor nodes. The challenges faced, the lessons learned as well as related work are also presented.

Chapter 6 concludes and highlights potential future work.

Chapter 2 Background Information on Sensors, WSN and Web Services

This chapter provides some basic background information on sensors, wireless sensor networks and Web Services.

2.1 Sensors

2.1.1 Definition

Sensors are electronic devices that can detect physical phenomena or stimuli from the environment and produce an electrical signal. They convert (changes in) the measured physical dimension into an electrical dimension (or changes thereof) that can be processed or electronically transmitted. They are limited in energy, communication and processing capabilities. They collect data and perform necessary actions or, transmit the data to other nodes so that certain actions can be performed. The collected data could be temperature, pressure, sound, existence of chemical or biological substances or certain changes in them, location or movement (motion detection), or other observable phenomena of interest to applications. A single sensor device can be made to observe a single phenomenon or several of them. A simple example of sensor is a device that notifies a home security system of potential trouble such as a break-in, a fire or other undesired events.

Sensors can work individually or in collaboration with other nodes to form a network, in which case they can aggregate data and provide a more complete view of the environment with respect to the phenomena under observation.

2.1.2 Hardware Components

The main hardware components of a sensor node are:

- **Sensing unit (s)**: The main functionality is to do the actual sensing or detection, by measuring the physical data from the target area. Upon detection of the phenomenon, an electrical signal or analog voltage is generated by this unit as a continual waveform, which gets digitized by an analog-to-digital converter (ADC) before being delivered to the processing unit for further analysis [3] [92].
- **Processing unit**: usually associated with a small storage unit, it manages the collaboration among the units within the sensor node as well as collaboration with other nodes to carry out the assigned sensing tasks. Families of this unit include microcontrollers, microprocessors and field-programmable gate arrays (FPGA). The storage unit, usually in the form of flash memory, helps to minimize the size of transmitted messages, to hold some data during local computation and when the node is responsible for data aggregation [93].
- **Communication unit (transceiver)**: connects to the network, does the transmission and reception of data. Radio Frequency (RF) is the dominant scheme in these units, but infrared and laser have also been used.
- **Power unit**: provides power to the sensor node, usually supported by batteries or other power scavenging units such as solar cells.

There may also be some additional application specific components such as location finding system. The components are combined to make a sensor as small in size

as possible. These hardware components and their technology trends are well explained in [3], [5] and [89]. A general architecture is shown in Figure 2.1

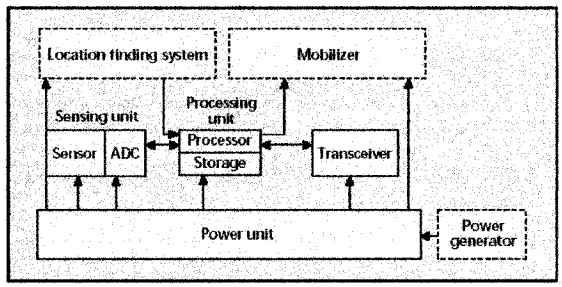

Figure 2.1 Main Hardware components of a sensor node

Among the common hardware platforms as shown in Figure 2.2 are the Motes (Mica, Micaz, TelosB) from the University of Carlifornia at Berkleley and Crossbow[6], BTNodes Platform [7] which are Bluetooth based, the ScatterWeb platforms [8] by Freie University in Berlin, Teco nodes by Telecooperation Office (TECO)/University of Karlsruhe [9] and TMoteSky and other nodes by Moteiv[10].

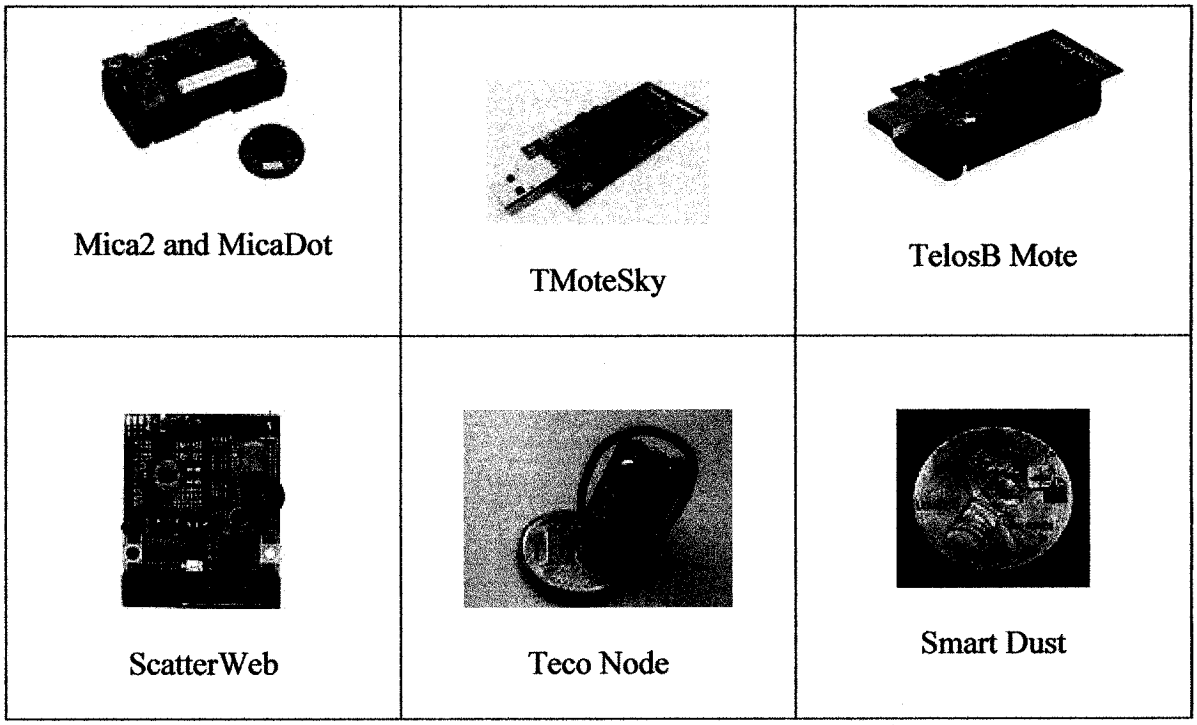

Figure 2.2: Some of the Common Sensor Platforms today

The current research aims to build the smallest possible sensor with least energy consumption. The Smart Dust project [79] intends to build a very small sensor, a few millimetres in volume, which can remain suspended in the air.

2.1.3 Software Components

The software components differ from one design to another; in general, the two main components are a lightweight operating system and a framework for data collection and programming (application enabler). TinyOS [11] and Contiki [12] are examples of the former whereas TinyDB [13] and Agilla [14] are examples of the latter. The separation between the two software components is not very sharp; in some designs they appear to overlap. Sometimes the separation is removed and a distributed operating system combines the OS and middleware or data retrieval functionality [15]. The general structure is shown in Figure 2.3 below. A survey of specific frameworks (i.e. the programming and data access environments) is provided in Chapter 3.

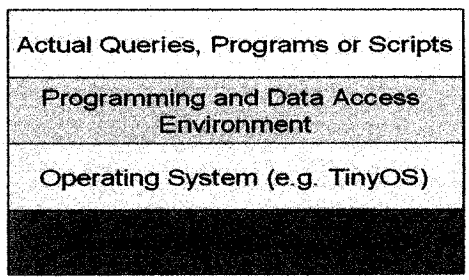

Figure 2.3: Software Components of a sensor node

2.2 Wireless Sensor Networks

2.2.1 Description

Wireless sensor networks (WSN) are distributed sets of collections of small sensors equipped with processors, memory and short range wireless communications. They communicate with each other and with other nodes called sinks and gateways that connect to the outside infrastructure and to the applications. The sensor nodes are usually randomly scattered in sensor field and use multi hop routing to relay information to the sink, possibly after data aggregation (also called fusion) along the way. A typical WSN is shown in Figure 2.4 below.

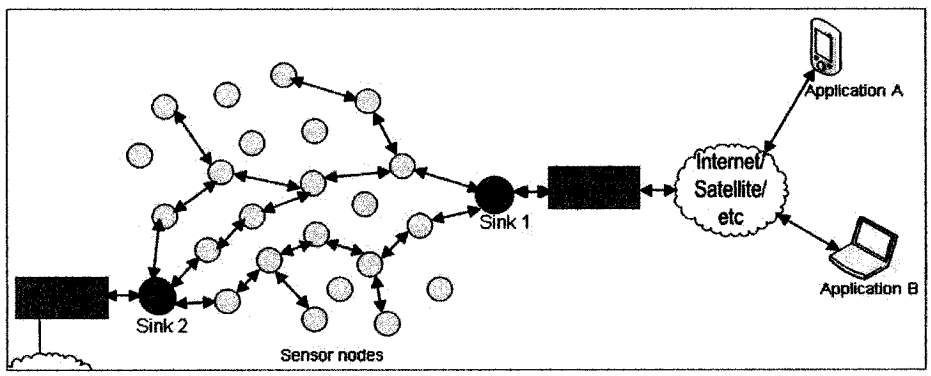

Figure 2.4: Wireless Sensor Network

2.2.2 Entities and their roles

The entities of a conventional WSN are sensors, aggregators, sink and gateway. The role of sensors is to do the actual sensing, perform limited computation to decide where and how to send the data. They also act as routers when multi hop routing is used. The aggregators (not shown in the Figure 2.4) are logical representatives of regions of interest. They summarize (aggregate) data for the region from the nearby sensors. They could be specialized nodes or dynamically selected from among the sensors (e.g. as cluster heads in dynamic cluster of sensors).

The sink node is a collector of data from all the sensors and/or aggregators. It receives requests from applications (via the gateway) and responds with appropriate data. The sinks are equipped with the same 'radio' links and communication protocols as the sensors. The gateway is the one that bridges the WSN to the 'outside world'. It is equipped with dual network interface (dual homed) to be able to communicate with the sink (and sensors) as well as with end user application devices or the external infrastructure. It is responsible for providing the necessary mappings and protocol conversions.

It is also common to find the sink and gateway co-located in the same node. They are more powerful in terms of available energy and processing capabilities. They take the burden of data processing and long range transmission off the sensor nodes. Apart from providing communication interfaces, in many cases, it is the sink and/or gateway nodes that play the role of a sensor network representative. They have full knowledge of the sensor network in terms of the types of sensors available, the services they offer, their logical topology and how to communicate with them. They are also responsible to present these capabilities to external parties.

While the above model is the most common; certain applications [38] [56] and operational settings may not require the sink, gateway or multi-hop routing. We call a WSN without the sink and/or gateway a sinkless WSN and this work has focused on that. A more detailed discussion of sinkless WSN is given in Chapter 3.

2.2.3 Characteristics distinguishing WSN from other Networks

Wireless Sensor Networks have some characteristics that distinguish them from other wireless networks. In particular, they often contain a very large number of nodes

which are very densely deployed (up to 20 per sq. m). The nodes are highly constrained in power, computational capacity and memory. Therefore, energy saving is very crucial. They often require a form of self organizing capability as they are usually deployed in a random manner and in remote or in-accessible areas, operating unattended and, their topology may change very frequently. Further, they often require co-operative effort in collecting, processing and transmitting data in order to save energy. Fault tolerance is also very crucial as the nodes easily die when energy is depleted.

Another characteristic is related to addressing; in conventional WSN, global addressing or identification of specific sensor node is not mandatory, as applications are more interested in the collected data than the actual node. Thus addressing is usually *data-centric,* which could be based on geographic location or location coordinates in space (spatial addressing) in which the address of the node provides hints of its location.

These characteristics play a major role in designing of the sensor nodes, the sensor network and their programming frameworks. They are well explained in [3] and [89].

2.2.4 WSN Applications

The main categories of WSN applications include environmental, military, building automation and security, health, industrial and commercial applications.

Environmental applications are mainly on observation and forecasting systems. These include monitoring of air quality for pollution detection, water pollution detection, forest fire detection, flood detection, monitoring disaster areas as well as intelligent agriculture [86]. Military applications appear to focus on exploration and surveillance, ranging from battlefield surveillance and damage assessment to detection of nuclear, biological and chemical attacks [5].

WSNs are also used in home and office buildings for automation and security. These include automatic door opening and closing, switching the lights ON and OFF depending on room occupancy, intrusion detection, fire detection as well as indoor air quality control.

There has also been a lot of work in applying WSN in the medical and health arena. Notable and experimented area is tele-monitoring of human physiological data such as heart rate and blood oxygen levels to alert the detected unusual levels to the emergency personnel and paramedics [16][87][88].

Other commercial applications include inventory control, asset tracking, intelligent traffic systems and habitat monitoring. References [17] [18] and [19] highlight several applications of WSN.

2.3 Web Services Framework

2.3.1 Definition and Basic Principles

Web Services (WS) [20] are modular programs that can be discovered and invoked across the network. According to Adam Bosworth [21], the term web services refer to an architecture that allows applications to talk to each other. It is a collection of protocols and standards used for exchanging data between applications or systems. The architecture is platform independent, making it an excellent candidate when interoperability between different systems is required. This interoperability is due to the use of XML and other open standards.

The three basic principles of Web Services are coarse grained approach, loose coupling and support for both synchronous and asynchronous communications. Coarse grained approach refers to the high level of abstraction that the web services provide. They can hide the internal complexities of the application and provide only the necessary high level interfaces that the other applications need to invoke and use the service. Loose coupling means no inter-dependency; the two applications that communicate via web services do not interfere nor depend on each others' internal workings; application A which talks to application B should not necessarily be re-written if application B is modified. The service user (i.e. application developer) does not have to know the details of the services implementation, including the programming language used. If the internal implementation changes, without changing the service interfaces and description, the consuming application will continue to work seamlessly. As for support for asynchronous communication, it means one application should not be necessarily blocked while waiting for response from the other application. It should be able to continue with its other tasks and handle the response when it comes.

Web Services is an example of a bigger paradigm known as Service Oriented Architecture (SOA), which defines the use of services to make resources available to network nodes through a well-defined and standard interface.

2.3.2 The 3 roles and 3 operations

The Web Services model defines three main roles and three main operations. The roles are service provider, service requestor and service registry. The three operations are publish, find and bind. Their relationship is as illustrated in Figure 2.5:

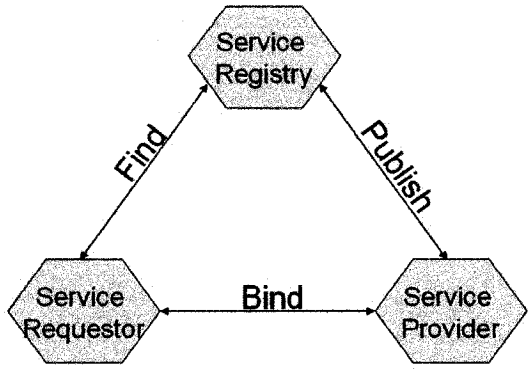

Figure 2.5 Web Services roles and operations

Service provider is the owner and host of the service; it is responsible for creating a service description and publishing it to one or more registries, and receiving invocation messages from one or more service requestors. The registry is a repository of web services descriptions; each description contains all the necessary information to use the described service. The requestor is a client that is able to discover a web service from the registry and invoke it from its provider.

The publish operation is what the service provider uses to make the service descriptions available in the registry. The find operation allows the service requestor to state its search criteria to the registry in determining the service of interest and how to invoke it. The service registry matches the find criteria against its collection of published descriptions. The bind operation is the actual invocation embodying a client-server relationship between the service requestor and the provider.

The most commonly used registry model in web services is based on Universal Description, Discovery and Integration (UDDI) specifications. Publication is done using the XML based Web Services Description Language (WSDL). The communications (operations) among the three web services entities are usually based on XML and use

Simple Object Access Protocol (SOAP). These terms are explained in the next section, and more detailed explanation can be found in [20] and [21].

2.3.3 Associated Technologies

2.3.3.1 XML

XML [73] stands for eXtended Markup Language. It is a markup language for documents containing structured information. It makes use of tags, similar to HTML. While the tags in HTML are fixed and predefined with unique meanings, XML allows one to devise virtually any tag to describe anything, leading to better support for content creation and management. XML carries data in plain text format, and separate the actual data from its representation. Thus the data can be shared in a software and hardware independent way, even between otherwise incompatible systems. It is neutral to programming languages. The data represented in XML format are well structured, usually given descriptive names and obey certain rules. Usually, there are XML processors that are used to easily read and navigate through XML documents to create, modify or delete its elements.

2.3.3.2 SOAP

SOAP [74] refers to Simple Object Access Protocol, which is a simple XML based communication protocol that is independent of any platform or programming language. Its purpose is to transfer XML data from one point to another over the network. The main parts of a SOAP message are the envelope, the headers and the body.

While a number of protocols can be used to carry SOAP, it is a de facto standard to carry it over HTTP. An application receiving a SOAP message will usually use a SOAP processor to identify all parts of the message, verify that the application supports them and thus act on them accordingly, or if the application is not the ultimate destination of the message, forward the message accordingly. Examples of SOAP messages corresponding to certain services will be shown in Chapter 5.

2.3.3.3 WSDL

Web Services Description Language (WSDL) [75] is an XML based language for describing the web services and how to access them. It is an XML grammar that describes the input data to the web service, the operations to be performed on the data and the binding protocol method (e.g. HTTP post) to be used to send them. Appendix A has an example of a WSDL file.

2.3.3.4 UDDI

Universal Description, Discovery and Integration (UDDI) [76] refer to the registry with a form of database that holds information about web services, their providers and how to access the services. It hosts the WSDL files. The specifications allow web service providers to publish their services and web service requestors to enquire and discover services. The publishing and inquiry is done via SOAP. UDDI is considered as 'yellow pages' of web services.

2.4 Chapter Summary

This chapter has provided background information on the main concepts and technologies used in this thesis. Sensors have been defined and their main hardware and software components have been illustrated. A brief description of Wireless Sensor Networks, their main entities and topologies, field of applications and the characteristics that distinguish them from other networks have been highlighted. Lastly a sufficient introduction of web services framework and its associated technologies has been given.

Chapter 3 WSN Application Enablers: State of the Art and evaluation

This chapter defines application enablers, highlights the driving forces behind their evolution, sets some evaluation criteria and then surveys and evaluates the state of the art.

3.1 WSN Application enablers

Application enablers refer to middleware, frameworks or programming mechanisms that allow developers to develop applications quickly and easily. WSN Application enablers are the WSN middleware, frameworks or programming mechanisms used by application developers to develop end user applications that interact with wireless sensor networks. Their main role is to allow retrieval of data from the WSN. They may also allow applications to send information to the sensors in order to change their operating parameters. Further, they may also allow applications to discover the existence of WSN and the services they offer. The term framework will be used as synonym of application enabler.

3.2 Driving forces behind the evolution of the frameworks

The frameworks enabling data retrieval from WSN to applications and other forms of interactions between applications and WSN keep on evolving. The main driving forces behind this evolution include:

- To *address the WSN constraints more efficiently*. Some frameworks' focus is on efficient algorithms that lead to higher savings in energy, processing and communication.
- Improving *deployment and management* of the WSN. This refers to improved self organization, selection of aggregators, finding of best routes to the aggregators and sink and, in case of mobile sensors, reorganization when the sensors have moved.
- *Simplifying application development* by providing appropriate levels of abstraction. Some frameworks are built with the main objective of allowing application developers to easily interact with WSN (easy 'sensor enabling' of applications).
- To provide dynamic *reprogramming of sensors*, adaptation and software *updates on the fly*. This is to avoid redeployment of WSN simply because of the changes in environmental conditions or application requirements.
- *Support for heterogeneity*. This is to allow different kinds of sensors to co-exist and cooperate within the same WSN. Also, to allow different applications to simultaneously 'query' or interact with the same WSN.
- Allowing *discoverability* of the WSN or the individual sensors. The applications have to be aware of the presence of WSN or the actual sensors and the services they offer before they can make use of them. Newer application enablers are taking this into consideration.
- *Trying paradigms* that have been successful in other domains. Some paradigms have worked so well in different areas and have been well received by application

developers or in some cases even attracted them. The move to introduce them into WSN is expected to attract more application developers and lead to more innovative applications.

- *Introducing new business models* for WSN. As the use of WSN evolves, different business roles arise (owners, service providers, end users etc). Application enablers will need to accommodate these roles accordingly.

3.3 Categorization and Evaluation of the frameworks

Different criteria can be used to categorize the WSN frameworks [96] [97]. Since the focus of this work is on the application developers, the categorization is based on the programming paradigms and concepts that the developers should be familiar with in order to use a particular framework. However, this categorization is not very strict, as some frameworks do fall into more than one category, in which case they are placed under the more dominant one. This applies to the frameworks based on common technologies, Section 3.3.2. Some frameworks are very proprietary and can not be conveniently placed in any of the general categories, and thus they are dealt with individually in Section 3.3.3.

3.3.1 Criteria for evaluation

In order to facilitate the use WSN data in more and more applications, there is a need to open up WSN application development by attracting application developers from various fields. A number of criteria need to be satisfied by the WSN application enabler. The ones we have identified are described below.

The first criterion is that the framework *should provide a high level of abstraction*. Developers should not have to learn much, if at all, about sensors and their technologies. Also, since applications are not really interested in readings of individual sensor nodes. Ideally, it is not even necessary for the developers to know or care that the data is coming from sensors. It would be sufficient for them to know that they can obtain information about hotness (temperature), darkness (light), humidity, air quality etc, regardless as to whether the information is coming from sensors or from other sources.

Another criterion is that the framework *should be based on a widely deployed technology* so that it is *easy to integrate with existing applications*. The developers should not be expected to learn new programming paradigms, special commands or languages solely for the purpose of accessing and integrating sensor data with their applications. The framework should be one of, or very close to the widely used frameworks for data access and should be built on top of common protocols supported by many application devices. They should provide interfaces for easy integration to existing applications.

The next criterion is related to performance. The framework should *introduce very little overhead or no overhead* at all in terms of network load, processing and response time. Substantial overhead will affect the network's performance and slow down applications. In cases such as disaster recovery and health monitoring, the sensor data need to be fresh and accessible in real time. Substantial processing delay and poor response time will compromise the usefulness of the collected data. It may also hinder the adoption of the framework by developers.

As one of the criteria, we propose that the framework *should support both synchronous and asynchronous modes* of data access. Upon requesting for data, the

application should not be forced to wait for response (blocked). The framework should allow applications to express interest in certain data and, from then on, be provided with the data and updates thereof whenever available until the interest expires or the application expresses that it is no longer interested in receiving the data. In other words, applications should not have to unnecessarily keep on re-querying (polling) for the same kind of information in a given time span.

It is also necessary that the framework *should allow simultaneous access* by different users or applications. Different applications may query the same network for the same or different kinds of data at the same time. For example, 3 different applications may submit the following requests:

Application A: "I need a list of rooms whose current temperature is above 20 degrees"

Application B: "For the next 24 hours, send me a notification whenever lights go OFF or ON in any room"

Application C: "Provide me with a list of rooms which currently have lights ON and there is some noise"

The framework should allow for independent simultaneous and sharable access to the data by different applications. This means, the framework should not focus on application specific sensors; it should cater for heterogeneity.

Another criterion is the ability to *support different programming languages*. The framework should not be tied to a specific programming language. It should allow access by applications written in different programming languages. This will attract a larger number of application developers, as there will be no need to learn a new language. Further, existing applications written in different languages will not have to be

substantially changed or re-developed in a different language just for the sake of incorporating sensor data.

The last criterion is that the framework *should support different business models.* As the WSNs continue to emerge, different business entities will be involved. The owners may decide to charge directly or via third parties. These raise the need for the frameworks to support different business models. This requirement can be broadly divided into two sub requirements; *providing Security mechanisms* and *providing publication and discovery mechanisms.*

To support different business models, the framework should provide authentication and authorization mechanisms. Only eligible parties should have access to the data; and they should only access the data that they are authorized to. Further, as charging may be involved; there is definitely a need for a non-repudiation mechanism. In principle, the mechanism should allow a third party to verify a claim from either party (provider or consumer) regarding service usage, to prevent the consumer from denying usage or the provider from charging for unused service. Security mechanism should also provide data integrity, which includes correctness, reliability and freshness. By freshness we mean real time results within reasonable time thresholds, not a replay of previously collected data which may possibly be out of date.

As the users, (or rather their devices on which applications will be running) are not expected to know of the existence of the WSNs on a certain area or, more importantly the services they offer and how they can be accessed, it is important that the framework provide publication and discovery mechanisms. Coupled with the abstraction requirement, the framework should allow the WSNs to publish their capabilities, the data

they offer and how they can be accessed. On the other hand, the framework should allow potential applications to discover the available information and how to access them when the user is interested.

3.3.2 Evaluation of Frameworks based on common technologies

These include Application Programming Interfaces (APIs) and specialised languages, Database approach, Mobile Code approach, Virtual Machine approach and Web Services Approach. We survey and evaluate each of them here.

3.3.2.1 APIs and specialized languages

As common to embedded systems, manufacturers of sensors also provide Application Programming Interfaces (APIs) or specialized languages to expose capabilities of the sensors for the developers to access and manipulate them. They normally require sensor programming boards attached to serial or USB ports. This method is very powerful as the developer has full access to all the sensor capabilities but it does demand the developer to be well versed in sensor technology. It is used to build most other frameworks. The most popular example of specialised language is the nesC [22]. There are also APIs in high level languages such as C/C++ and Java; examples are CricketLib [23] and EmberNet [24].

In general, this framework does not offer a high level of abstraction nor does it come with security and discovery mechanisms. It is tied to the specific programming language, so it does not offer language neutrality. Also, as the framework is mostly meant for low level development, it becomes necessary for developers to go through a learning curve on sensors and WSN, thus it is not easy to integrate with applications. It is possible for this approach to offer both synchronous and asynchronous access, as well as

simultaneous access by different applications. No significant overhead is expected from this approach.

It should be noted that this category refers to 'plane' APIs that are not exposing another mechanism like database or mobile code. The other categories discussed here may also use APIs, but since they are just wrapping some other approaches, they are placed in the relevant category.

3.3.2.2 Database Approach

This approach uses the concept of distributed and relational databases with some modifications. The WSN is either modeled as a single database table, with columns representing various characteristics of the sensor node, their location and the phenomena to be sensed and rows representing the individual sensors, or a distributed database with each node acting as a virtual database site. The analogy between data generation (detection) and routing in WSN as compared to data storage and query processing in databases led to this model. Applications issue SQL like queries, and receive responses via the gateway/sink as if the WSN is a database system. The queries are high level; the application developer doesn't have to know the details as to which sensor node sent the response or what in-network processing and aggregation was done. This is also referred to as Data Centric approach.

The queries can be "one shot" relational queries with a fixed answer set, or ongoing continuous queries that produce a bounded or unbounded stream of results. An example query is as shown in Table 1 below:

SELECT AVG(volume), room FROM sensors WHERE
floor = 6 GROUP BY room HAVING AVG(volume) > 10
SAMPLE INTERVAL 3s FOR 30s

Table 1: An example modified SQL query for WSN

In this example, the WSN is modeled as a single table named *sensors* and has room, floor and volume among its columns. The query is meant to retrieve a list of rooms in the sixth floor of a building and their corresponding volume (sound) level, provided that the average level collected by all sensors in that room exceeds ten. The data should be furnished every three seconds for the next thirty seconds.

Abstracting the WSN as a database enables simpler and faster application development. The most popular example in this category is TinyDB [13] which comes with its SQL variation called TinySQL. Other examples include "TASK: Sensor Network in a box" [25] , IrisNet [26] which uses XML instead of SQL, Cougar [27], SINA [28] which comes with a query language called Sensor Query and Tasking Language (SQTL) containing flavours of SQL and XML.

This framework can provide a high level of abstraction. Database is also a widely used technology and easy to integrate with applications. It is not tied to a specific programming language. Simultaneous access by different applications is generally supported. Apart from non-repudiation, the approach could provide the required security features, but most implementations have chosen not to. Using triggers, it is possible to provide both synchronous and asynchronous modes, but again, most implementations have chosen not to use them, presumably to avoid processing overhead on the sensor nodes. Rather, modifications have been done to SQL queries to add duration and sample

interval to provide for continuous stream of data instead of the usual fixed set results in conventional databases. No significant overhead has been reported for SQL based queries; it may be significant for XML based access. The framework does not provide publication and discovery mechanisms. Almost any programming language can be used with this framework.

3.3.2.3 Mobile Code Approach (Mobile Agent and Active Networks)

Mobile agents are software modules that can execute in more than one node in a special sequence, carrying intermediate results from one node to another. They may eventually return final results to the original node. Applied to WSN, the sensors are equipped with a lightweight mobile agent platform and the applications inject scripts or programs (i.e. mobile agents) into the WSN via the gateway; the agents collect sensor data and statefully migrate or copy themselves to other nodes and communicate with such remote copies. The collected data is sent back to the sink/gateway ready to be delivered to the requesting applications. The agents can also reconfigure or upgrade the nodes they visit, making this approach primarily used for upgrading the sensor nodes 'on the fly'.

As for Active Network approach, the nodes are made capable of performing customized operations on the data flowing through them or, the flowing data can trigger execution of certain programs on the nodes. The sensor nodes built on Active Network concept are sometimes called Active Sensors.

Agilla [14] is one such example of a lightweight mobile agent platform for sensor nodes running TinyOS. Figure 3.1 shows Agilla in action. The agents' routes are pre-planned by relatively more powerful backend system but new routes can be adopted in case of network changes or failures.

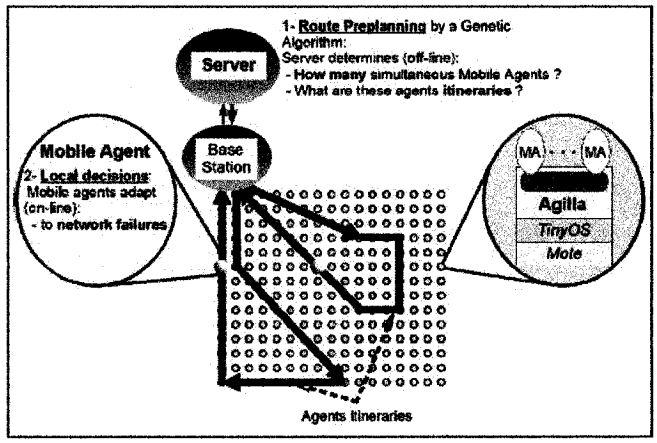

Figure 3.1 Agilla Mobile Agent framework for WSN

Other example frameworks using the mobile code approach include SensorWare [29] which takes most of its concepts from active networks, Janus [30] and Smart Messages [31]. Impala [32], while primarily event based, also employs mobile code.

This framework can offer a high level of abstraction but some implementations do not. Mobile code is not a widely deployed programming paradigm. While we would theoretically expect high overhead due to migration and replication of scripts and agents in the network, tests have proven otherwise [29]. It is possible to have both synchronous and asynchronous access, as well as simultaneous access by different applications. Security, publication and discovery are not provided. A specific programming language has to be used with this framework, depending on the platform.

3.3.2.4 Virtual Machine Approach

As a concept, this is very similar to Java Virtual Machine except that it is the developer who is expected to build a customized virtual machine instead of just building

applications that run on it. Therefore, the level of abstraction is not fixed, but rather decided by the developer. The optimization focuses on a specific application.

The motivation behind this approach is the fact that an application specific framework can be more efficient than a general framework that should cater for a wide range of applications. The most popular example in this category is Mate' [33].It is architecture for constructing application specific virtual machines on top of TinyOS. It consists of scripting language and an interpreter to execute the scripts. It is the scripts that are used by the developers for their applications to interact with the WSN.

The power of Virtual machines is that they can be used to derive and build other frameworks faster and with less lines of code. For example, one can build a virtual machine that can translate SQL queries and transmit sensed data, thereby coming up with a database approach. In fact, Mate' has been used to redevelop TinyDB faster and with higher energy savings (11 – 30%). SensorWare, a mobile code approach, has been built using Mate'.

This approach obviously requires extensive knowledge of sensors; it does not provide high level of abstraction. It is not a widely deployed technology. Tests have shown very good performance. It can support both synchronous and asynchronous data dissemination, and can allow for simultaneous access. It is not independent of programming language, does not address security or provide publication and discovery mechanisms.

3.3.2.5 Web Services Approach

Web Services are modular, independent and self describing programs that can be discovered and invoked across the network. The framework can and has been applied to

sensor networks in different forms, as will be detailed in Chapter 4. A common form today is by implementing web service based sensor gateway. The gateway hides the complexity of the sensor network and exposes the sensor data as web services. The implementation of the web services involve mapping to the APIs or other underlying frameworks.

In principal, web services are meant to offer high level of abstraction, therefore it is very convenient to 'general' developers. It is now a widely used technology among application developers; it is easy to integrate with existing applications. However there are well known issues with web services performance, their use introduces significant overhead. The framework does allow both synchronous and asynchronous accesses. There is no problem in allowing independent, simultaneous accesses to sensor data by different applications. The framework supports several programming languages. It does not come with security as yet, but there are established specifications to provide security mechanisms for web services as provided in [34] without non repudiation, which is covered in [35]. The framework provides its own publication and discovery mechanisms.

3.3.3 Selected Proprietary frameworks

Apart from the frameworks that are based on common technology approaches, there are numerous proprietary frameworks with different design goals. The significant ones for application developers include Sharman [36], Mires [37] and TinyLIME [38].

3.3.3.1 Sharman

This framework is a java based service gateway that integrates sensors into heterogeneous ad hoc networks. The gateway uses service wrappers to support different

standards such as Jini [39], UpnP [40] and possibly others, simultaneously. Applications can choose any of these standards to access the WSNs. A proxy on the gateway translates the Jini, UpnP and possibly other commands into Sensor specific commands. This shields the complexity of sensors from the developers.

The framework does offer a high level of abstraction but is not yet among the widely deployed technologies, thus, in most cases it does involve a learning curve to integrate with applications. Performance has not been reported, but it appears to have significant overhead. Synchronous and asynchronous modes are both supported, and simultaneous access from different applications is allowed. The frameworks architecture is meant to allow access by applications independent of their programming language. Security has not been addressed. The use of Jini and UpnP does offer support for publication and discovery.

3.3.3.2 Mires

Mires is a message oriented middleware that allows applications to communicate with sensors in a publish/subscribe way. It is developed directly on top of TinyOS. The nodes publish the list of available services towards the sink node without transmitting the actual data. The end user application has to subscribe by selecting one or more services. Upon this subscription, the nodes start and continue to transmit the corresponding data to the end user application. The application communicates with this middleware by sending special commands or messages.

Mires does provide a high level of abstraction, as the end user application is only supposed to identify the service of interest to start receiving the corresponding data. It is

not a widely used approach; the special commands/messages are proprietary. Reducing overhead is one of this framework's goals, although no performance tests have been reported. It can allow multiple accesses by different applications, in both synchronous and asynchronous modes. This middleware is tied to a mini programming language of its own for the command set and messages. It is built with publication and discovery in mind. Security has not been addressed.

3.3.3.3 TinyLIME

This framework is based on LIME [41] (Linda in Mobile Environments) which is very common in Mobile ad-hoc Networks (MANETs). TinyLIME has been built with the goal of removing centralization of WSN on the gateway. It therefore allows applications to discover and directly 'talk' to individual sensor nodes in their neighbourhood. It comes with proprietary APIs that combine concepts of mobile code and tuple spaces to provide a shared memory between applications and the sensor nodes, thereby allowing the discovery of the sensors and access to their data without needing a gateway or sink.

TinyLIME can provide a high level of abstraction, as it has taken care of the lower level issues. It is based on a paradigm mainly used in MANET, thus not yet widely adopted by general developers. There is some overhead involved when this approach is used. Synchronous and asynchronous modes are supported, and so is simultaneous access to data. The framework is mainly Java based, so it is not independent of programming language. Discovery mechanism is among its main focus, but it has not addressed security.

3.3.4 Evaluation Summary

As we have seen, in most frameworks, the criteria we have set are met to some degree; certain aspects are supported whereas others are not. A tabulated summary is shown in Table 2; the evaluation metric will be showing **F** for Fully met, **P** for Partially met and **N** for Not met at all.

	High Abstraction	Widely deployed Techn.	Low overhead	Synchronous & asynch	Simultaneous access	Language neutrality	Security mechanisms	Discoverability
Commands and APIs	N	P	F	F	F	N	N	N
Database Approach	F	F	P	P	F	F	P	N
Mobile Code Approach	P	N	P	F	F	N	N	N
Virtual Machine Approach	N	N	F	F	F	N	N	N
Web Services Approach	F	F	N	F	F	F	P	F
Sharman	F	N	P	F	F	F	N	F
Mires	F	N	P	F	F	N	N	F
TinyLIME	F	N	P	F	F	N	N	F

Table 2: Summary of the evaluation of the frameworks

It can be seen that Web Services is one of the very promising frameworks; the main drawback being performance. Performance is a well known issue with web services,

a number of research activities are under way to address it, and there are some suggested techniques to reduce its impact.

Another survey and evaluation of the frameworks is provided in [90] and [91], using different set of criteria which are not selected from application developer point of view. However, they do not include web services among the surveyed and evaluated frameworks.

3.4 Chapter Summary

This chapter has looked into frameworks for enabling applications to make use of sensor data. The driving forces behind their evolution have been highlighted. A number of necessary requirements for promoting the use of WSN data have been set, followed by a survey and evaluation of the prominent frameworks. The evaluation reveals that Web Services are among the very promising application enablers, though not necessarily the best.

Chapter 4 Proposed Web Services framework

This chapter has two main sections. The first section outlines the various possibilities of applying Web Services to WSN, with use cases and possible application scenarios. The second section explains the Web Service based sinkless architecture for WSN as proposed and envisioned by this thesis.

4.1 Web Services for WSN

By looking at the main entities of WSN and their roles, the Web Services model and some application scenarios; Web Services can be applied to WSN in three main forms:

(i) having the web services platform on the gateway and the applications bind to the gateway

(ii) having the web services on the aggregators and the applications bind to the aggregator, and

(iii) having the web services platform on the sensor nodes themselves and the applications bind directly to the nodes

The next subsections details these possibilities.

4.1.1 Web Services at the gateway

This is the situation today, as far as the use of web services in WSN is concerned. Web Services platform and the hosted services are located at the WSN gateway. Figure 4.1 illustrates this situation:

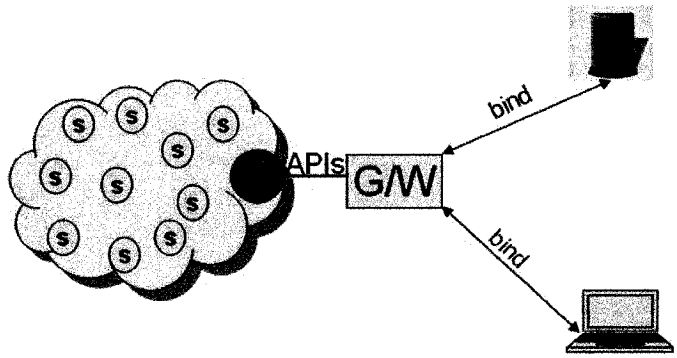

Figure 4.1 Web Services on Sensor Gateway

In this model, the sensors, aggregators and sinks communicate via their own proprietary protocols and frameworks. At the gateway, there is a mapping from the proprietary framework(s) to web services. Thus the services and data offered by WSN are exposed to application developers via web services. The end user applications communicate with the gateway via SOAP by invoking a particular web service (i.e. they bind to the gateway). In its simplest form, the gateway does the mapping of the SOAP requests to the format and protocol of the framework used by the rest of the WSN entities behind it. It also collects the responses from the network, builds the corresponding SOAP response and forwards them to the requesting applications. A more practical web service based gateway does more than just mapping the requests and responses. The services are expected to do some processing of the collected raw sensor data and provide more meaningful information. For example, instead of merely providing actual temperature values as furnished by temperature sensor, it could allow applications to be alerted when the temperature is above certain thresholds. Or, instead of supplying coordinates of a particular object as detected by location sensors, it could allow applications to request notifications when the object of interest is within certain proximity. More complex services can be provided.

The services offered by the Web Services based WSN gateways can be published in the UDDI registry for the end user applications to find. Figure 4.2 illustrates this.

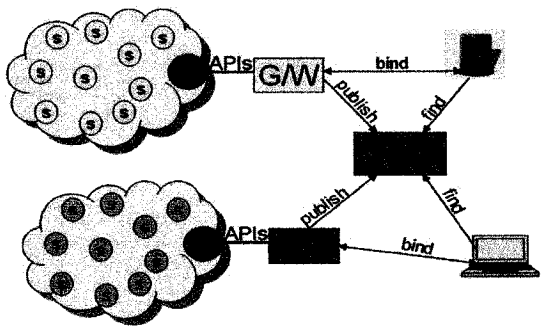

Figure 4.2 Web Services on Sensor gateway, with UDDI in place

The actual interactions among the WSN entities, the web services gateway, UDDI registry and user application are as shown in Figure 4.3 below. In the figure, native F/W means the proprietary framework and associated protocols within the WSN; s1, s2 and s3 are the sensor nodes.

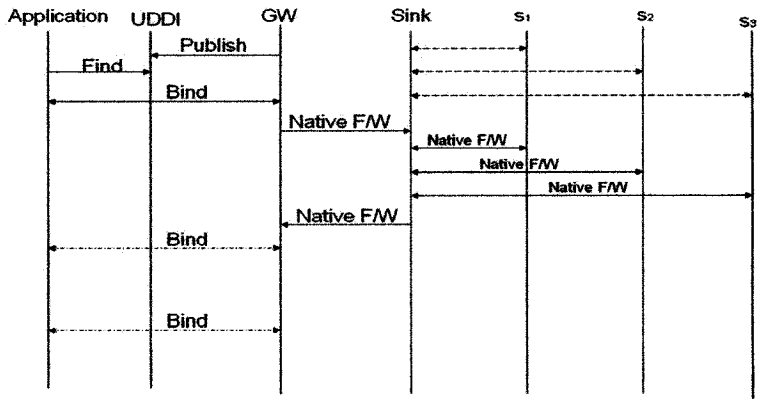

Figure 4.3 WS on the gateway, entity interactions

This model is the best, actually the only realistic option, when the WSN is meant for remote monitoring. In remote monitoring, as in Figure 4.4 it would not be practical for the requesting applications to bind directly to sensor nodes or the aggregators. This is because the sensors and aggregators are normally not equipped with long range

communication mechanism, they are far from the requesting application devices and, normally the requestors in this case are only interested in the abstract and general view of the network, not specific sensors. The roles of the sink and gateway are particularly important in this case.

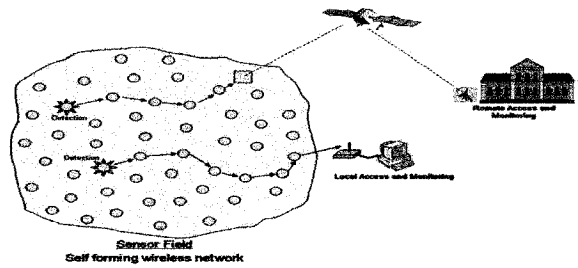

Figure 4.4 WSN for remote monitoring

The model is also used in non-remote monitoring as it fits well in today's conventional WSN topology, which almost always has the gateway in place.

4.1.2 Web Services on the aggregator

Another model, which we have yet to see implemented, is to have the web services and their platform located on the aggregators. This model is useful when:

- the aggregator covers a meaningful area of interest to applications, for example a room.
- the requesting application devices are often closer to the aggregators (than going through a gateway). This happens when the user is within or around the WSN.
- the requesting applications require extended and continuous access to data from a particular aggregator.

There are two possibilities for the UDDI, we can have it as a stand alone entity or distributed over several aggregator nodes as an overlaid entity. Figure 4.5 illustrates this model, with a UDDI as a stand alone entity.

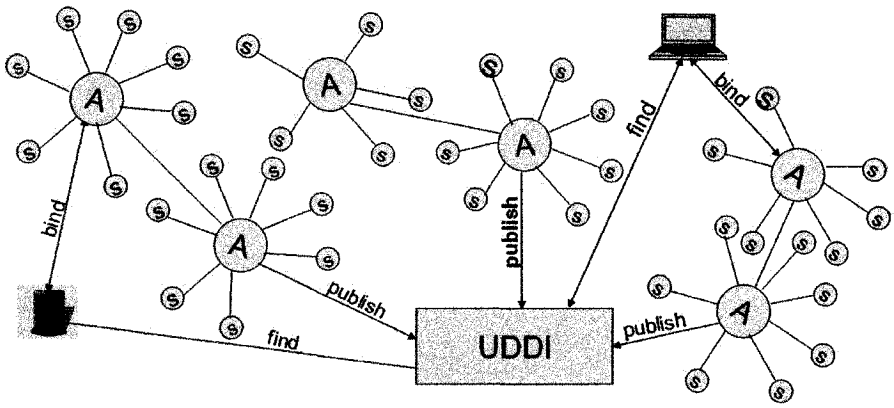

Figure 4.5 Web Services on the Aggregator

The role of sink and/or gateway is not relevant in this case. It is considered as a sinkless architecture. The application devices interact directly with the aggregator nodes. The communication between the aggregators and the individual sensor nodes could either be via proprietary mechanisms or via web services. The latter would need the sensor nodes to host web services too.

Figure 4.6 shows the message flows in this case, assuming that the UDDI is a standalone entity and the communication between the sensor nodes and the aggregator is via proprietary mechanisms whereas between aggregator and application devices, as well as the UDDI entity is via web services.

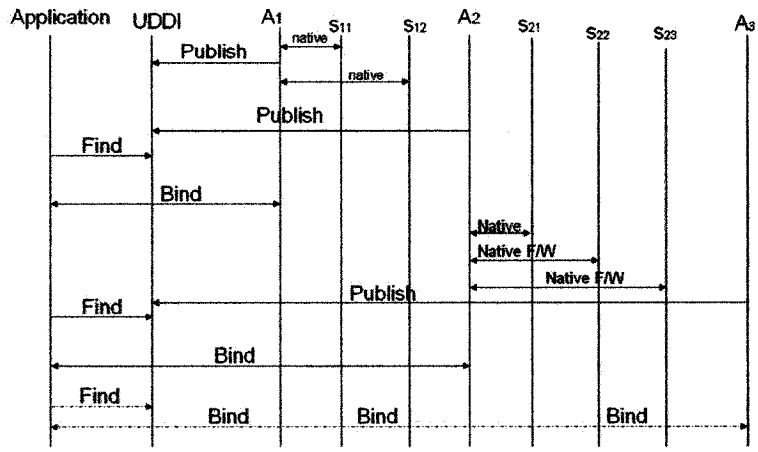

Figure 4.6 Web Services at Aggregator, entity interactions

Figure 4.7 shows an indoor scenario where the applications bind to the aggregators.

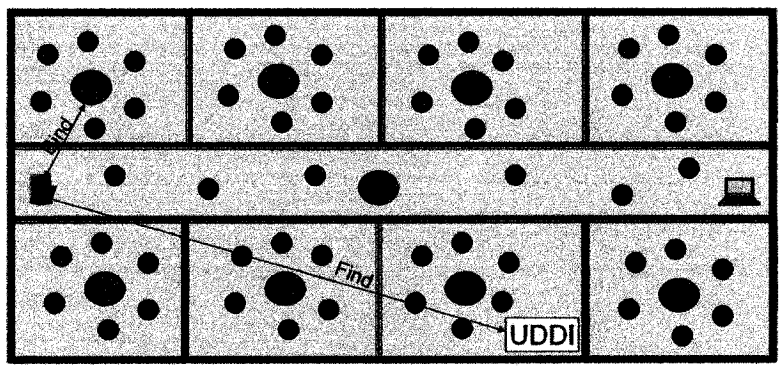

Figure 4.7 Web Services on the Aggregator, indoor scenario

4.1.3 Web Services on the sensor nodes

Another possible model, which is also yet to be implemented and is the work of this thesis, is to have the web services and their platform on the sensor nodes themselves. The model is useful when the individual sensor nodes, rather than the aggregators, are representatives of areas of interest, and the applications are often closer to them or require extended data access from them.

Section 4.2 further builds a case for this model. It is also a sinkless model. One possible setup of this model is shown in Figure 4.8

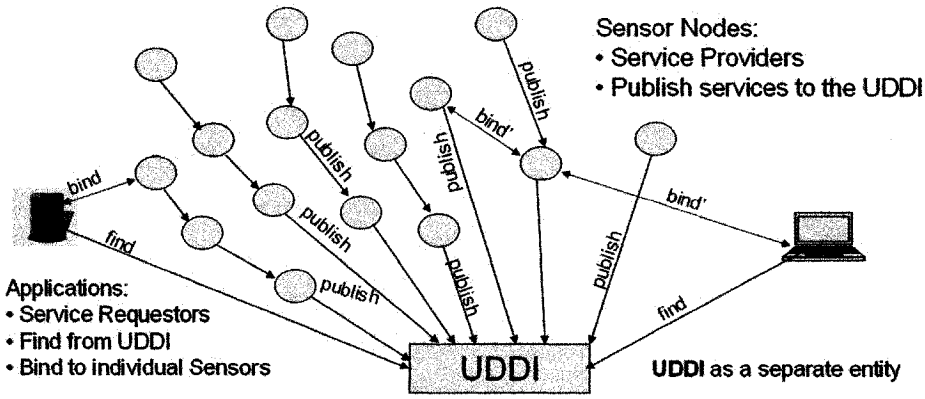

Figure 4.8 Web Services on sensor nodes

Again, the roles of sink and gateway are not required in this case, as the applications bind directly to the sensor nodes. This is referred to as a sinkless architecture. The sensors themselves have a small built in web services platform and host the necessary services. They act as service providers and publish their services to a designated UDDI registry. Here too, the UDDI could be a centralized special node or distributed over several nodes as an overlaid entity.

The interactions and message flows are as shown in Figure 4.9. The messages can be sent via single hop or multi hop. This is a web services only network without any proprietary frameworks. It should be noted that this architecture necessitates that the application devices share the same network interfaces and protocols as the sensor nodes.

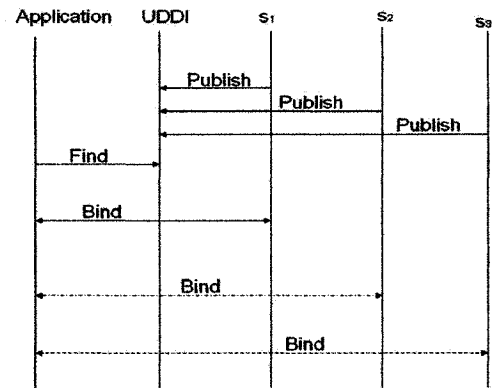

Figure 4.9 WS on the sensor nodes, entity interactions

This model is applicable mostly to indoor applications and few outdoor scenarios. An example application scenario is shown in Figure 4.10, a modified version of Figure 4.7. A building with each of its room having a sensor node that sufficiently covers intended data collection for the room.

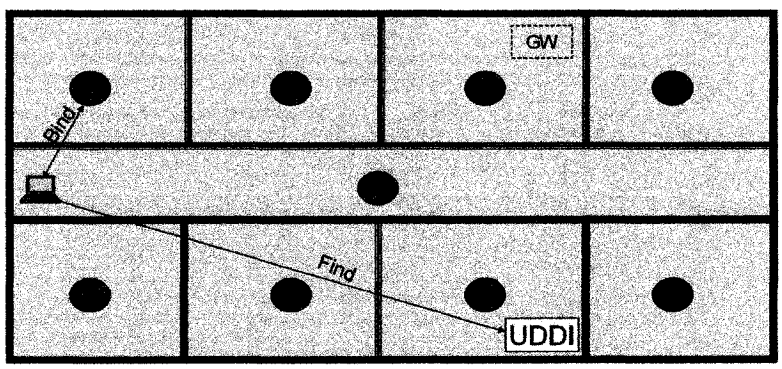

Figure 4.10 WS on the sensor node, Indoor example

4.2 The Proposed Architecture

The architecture proposed by this thesis is a web services based sinkless Wireless Sensor Network. As mentioned earlier, a sinkless WSN is the one that has neither a sink nor a gateway. The end-user applications interact directly with the individual sensors or aggregators, as shown in previous figures and message interactions. This second part of

the chapter highlights some motivating scenarios for a sinkless architecture, presents the proposed web services model, its architectural entities and the protocol issues.

4.2.1 Motivations for Sinkless Wireless Sensor Network

There are several scenarios in which it is more beneficial to have direct interaction between applications and the sensors. The cases include onsite battlefield assessment, specialized indoor monitoring and certain rescue operations.

We take onsite battlefield assessment as an outdoor example. In this case, several sensors are scattered all over the field to detect landmines and other dangerous substances. The soldiers are moving within the sensor field carrying application devices. They need to be notified of the danger around them. It would be more beneficial from the performance standpoint if the application devices can interact directly with the sensors in their neighborhood than having the sensors send the data all the way to the sink and gateway, which will then forward it to the device. In cases where it is necessary to have the data sent to a centralized location or to other users, it will be sent by the application devices communicating with each other and with the centralized infrastructure. This approach will save energy of the sensors as they do not need to implement long range communication or run complex multi-hop routing algorithms. The application devices are relatively more powerful and thus better equipped to perform the routing or the long range communication, if necessary. Figure 4.11 illustrates this scenario. In this case, the application devices can route data to other devices and, eventually, to the centralized location.

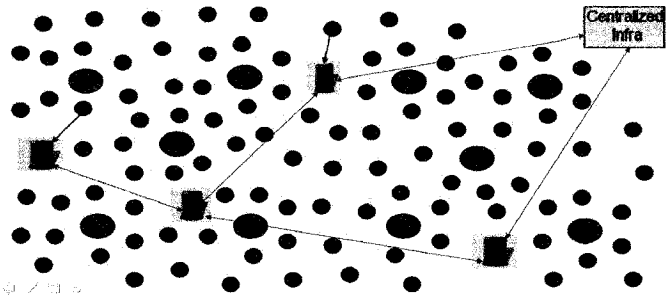

Figure 4.11 Sinkless WSN with routing done by the application devices

Another scenario, which is an indoor case, is a specialized monitoring of humidity, temperature, or air quality in a building. Consider a building with several rooms, each of which is equipped with a sensor, assumed to be sufficient for providing representative data for the room. If the end user interest is in a specific room and/or the application device is often very close to the room of interest, or is moving within and around the building, then direct interaction with the individual sensor would be more suitable than communicating through the gateway.

Sinkless WSN could also be used in rescue operations in large buildings, where the rescuers (humans or robots) are moving inside the building and are interested in receiving sensed events of interest close to them so that they can respond immediately. Direct interaction with neighboring sensors will be more efficient. The received data could also be used by applications to add more features such as navigation directions.

Several other scenarios can be thought of where a sinkless architecture is preferred, when one or more of the following situations exist:

- Individual sensors are representative of an area of interest
- Application device is within or moving around the WSN
- Application needs data from sensors in close proximity
- Applications need extended access to data from a specific sensor

It should also be noted that, if the rooms or areas of interests are too large to be covered by an individual sensor, the aggregators could be used and have the application devices interact with them. The same motivating use cases will apply to these aggregators.

There are other inherent advantages of having a sinkless architecture. The decentralization removes the traffic burden from the sensors and aggregators that would have been close to the sink, as they would have to handle a lot more communications in routing data to and from the sink. It is also not necessary for the sensors to run complex multi hop routing algorithms. Communication can be as simple as single hop from the sensor to the application device, or, when aggregators are used, single hop from the sensor to the aggregator and single hop from the aggregator to the application device. These advantages have been demonstrated mathematically in [84] and [85].

4.2.2 The proposed model, its architectural entities and interactions

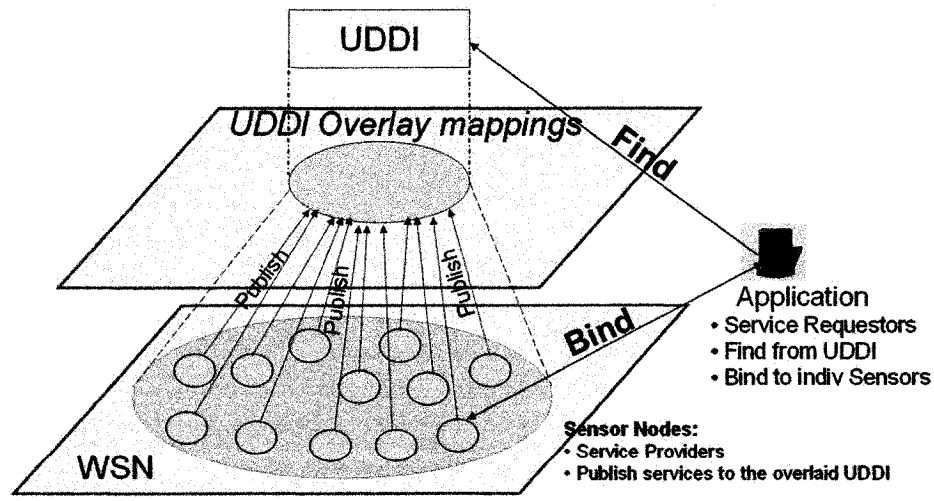

Figure 4.12 Web Services based sinkless architecture

Figure 4.12 shows the overall proposed architecture. There is no conventional WSN gateway or sink; the web services reside on the actual sensor nodes. These nodes are therefore service providers, providing relevant sensor services (data related to sensed phenomena) whereas applications are requestors or clients of such services. While the registry could be a specialized physical entity, the proposed model is a completely decentralized approach having a distributed UDDI overlaid on the sensor network. The sensors publish descriptions of their services to the overlaid UDDI. These descriptions will basically include the services offered and how to access them. The descriptions will be hosted in one or more sensor nodes; an appropriate overlay middleware will provide the required mappings to their actual hosts. As long as the description can be found, the actual location of the UDDI record is not important to the requesting applications.

The messages exchanged among the three entities, and their order, are as shown in Figure 4.13. s1, s2 and s3 are the sensor nodes acting as service providers.

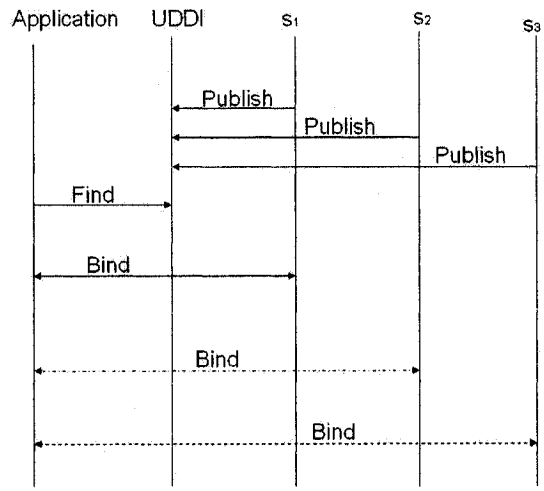

Figure 4.13 Entity Interactions for WS based sinkless WSN

Again it should be noted that a similar model as above can be used for aggregators; the aggregators may host the web services and the overlaid UDDI.

Applications would interact with the aggregators via web services; the interaction between the aggregators and sensor nodes can be via web services or other mechanisms.

4.2.3 Protocol Stack and Addressing

In conventional WSN, the sensor nodes communicate via proprietary protocols, from radio links to the application level. The WSN is effectively alien to applications. A gateway is therefore necessary to link the two. For application devices to interact directly with the sensors (or aggregators), the two must be compatible in their entire protocol stack. The gateway, which usually performs protocol conversion, is no longer in place.

A promising solution for this protocol compatibility issue is the adoption of the IEEE 802.15.4 standard [42], loosely referred to as Zigbee [43]. A number of Zigbee compliant sensors are already available and Zigbee adapters for the application devices have started to emerge. TelosB[44] and tmotesky[10] are examples of Zigbee based sensor nodes. Zigbee adapters for application devices are also available; For PCs and Laptops, these include UZBee[45], a USB Zigbee adapter, Zigbee Access Point (ZAP) used in [46] and iB-Bean GW-5324-CF [47]. The trend for adoption on mobile devices is also promising; Telecom Italia has released ZSIM [48], a SIM card with integrated Zigbee adaptor.

It is also worth mentioning that, while Zigbee is more promising, Bluetooth could have been a viable alternative. Already there are Bluetooth based sensor nodes, such as BTNode[7], as well as widespread Bluetooth support in application devices.

Another important protocol issue is the fact that most application devices communicate via IP (internet protocol) whereas conventional WSN have not been supporting it. Realising the importance of natural integration of the WSN to the rest of

the world, there is a trend towards IP enabled sensors. There have been a number of attempts to implement TCP/IP on sensor nodes [49] [50]. μIP [51] is one that is relatively more matured; it has been used in a number of projects [52], and ported into several platforms, including the Mote platforms and their TinyOS operating system. Applications can interact with the sensor nodes via TCP/IP. A number of IP based protocols and applications have been experimented. These include HTTP, SIP, DHCP and SNMP [53], [46]. [52] presents a complete IP based sensor network in which application devices can interact directly with the nodes via IP.

Along with the protocol stack is the issue of addressing. In conventional WSN, addressing is data centric, i.e. applications do not have to know or care about the identification of a specific sensor node. With this in mind, μIP for example uses spatial IP address assignment when assigning IP addresses dynamically, which means the address of the sensor is related to its location. Figure 4.14, taken from [55] shows an example where the x and y coordinates of the regions are the least significant digits of the IP address of the corresponding node. Sensor nodes on the same location will have the same IP address without conflict. In sinkless WSN, the application can get the address or identification of a representative sensor (or aggregator) and interact with it.

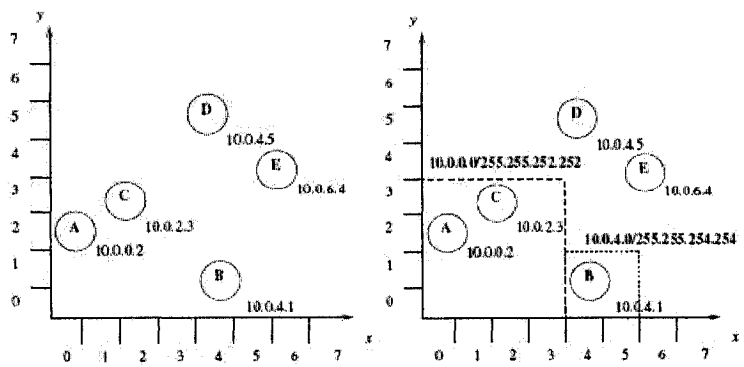

Figure 4.14 Spatial IP address assignment

4.2.4 Performance Pre-Assessment

In [56], we did a performance comparison between conventional sink based architecture and a web service based sinkless architecture, for a sample scenario that suits a sinkless architecture. 50 simulations were done using Matlab, with 400 nodes scattered in an area of 800 by 800 square meters, and an application device moving around it. The metrics considered are the average energy dissipation, the network load and the system lifetime. The comparison was between direct interactions with a specific sensor node versus accessing data of the same sensor node through a gateway. The results showed performance improvement of 22% for average energy dissipation, 30% for system lifetime, but relatively little improvement for network load. The simulation results are shown in Figure 4.15, 4.6 and 4.17 below.

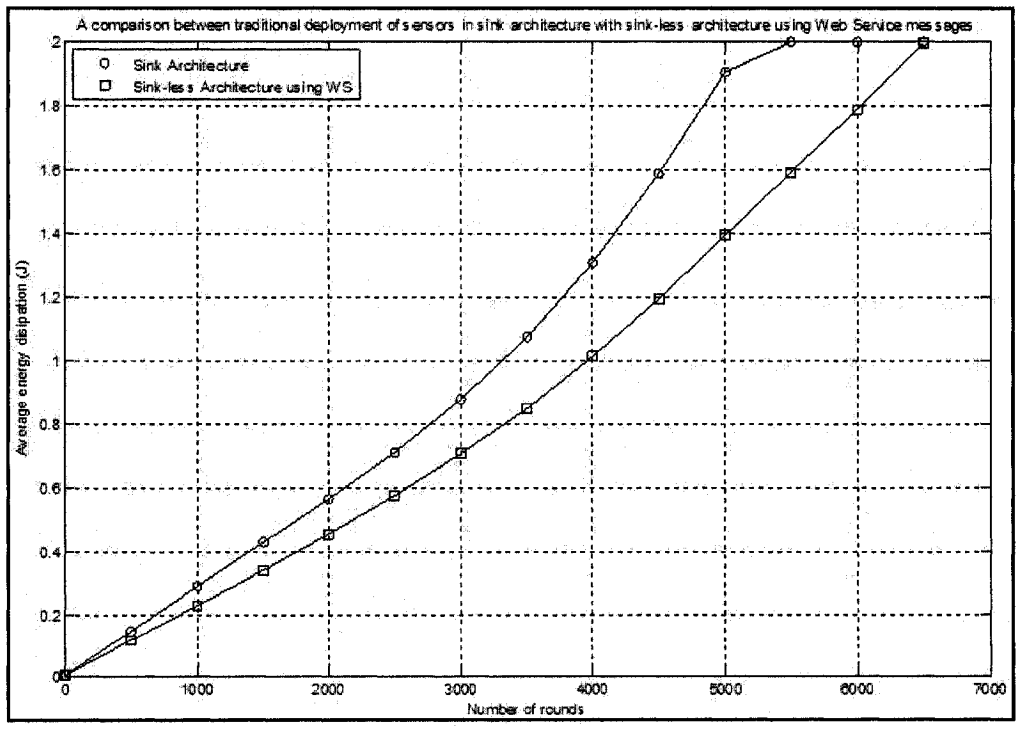

Figure 4.15 Simulation results - Average Energy Dissipation

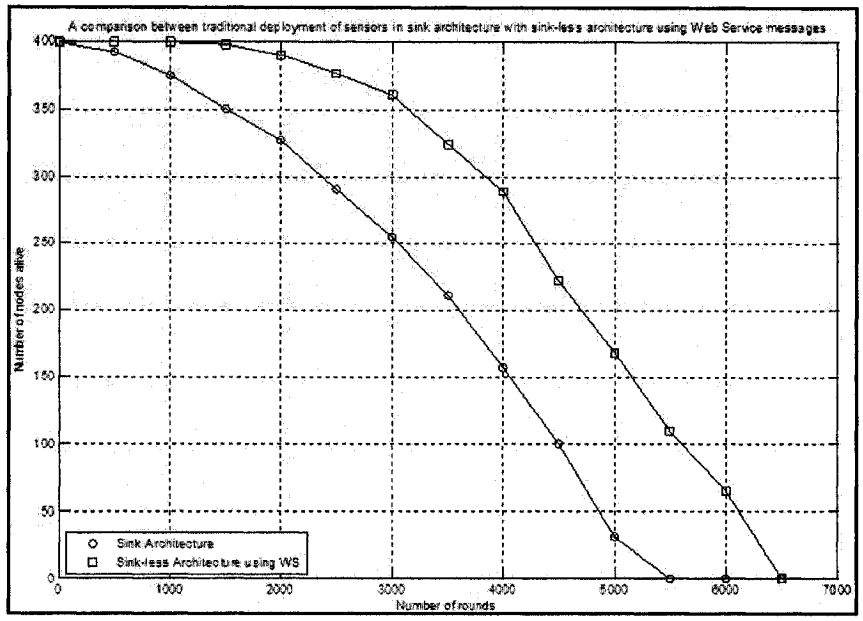

Figure 4.16 Simulation results - System Lifetime

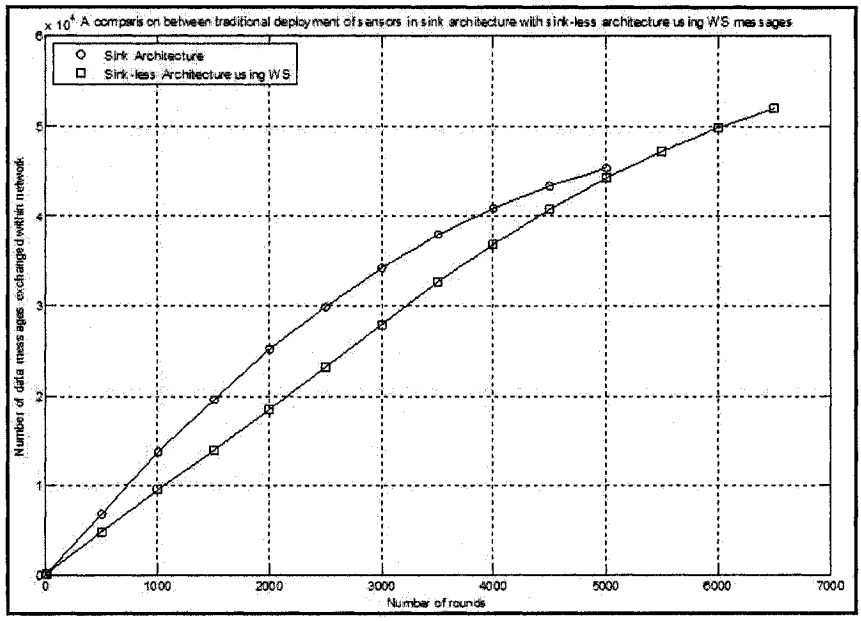

Figure 4.17 Simulation results – Network load (number of exchanged messages)

The simulations did not take into account the web service discovery phase, which may have an impact. Nevertheless, the rationale for using web services is not solely based

on performance, but rather on the other advantages it brings to applications and developers.

Apart from these simulations, [84] and [85] have also argued mathematically and concluded that, energy wise, multi-hop routing and centralized architecture is less efficient in many cases, and is not as beneficial as generally claimed in the rest of the cases. The analysis was done from communication engineering perspective, taking into account several factors such as interference, channel capacity and coding, link model, end to end reliability and energy consumption.

4.3 Chapter Summary

This chapter has explained the various possibilities of using web services in WSN. Some of them have already been used and some are newly proposed. The chapter has also proposed a web services based sinkless architecture, by presenting motivating examples, outlining the protocol and addressing issues and how they can be dealt with. Results of preliminary performance analysis of the proposed architecture have also been briefly presented. The next chapter presents the design and implementation of the prototype.

Chapter 5 TinyWS prototype – Design and Implementation

This chapter presents our design and implementation of a proof of concept for an embedded web services platform on the sensor nodes, in particular the motes [6] [44]. The challenges faced, lessons learned and some related work are also presented.

5.1 TinyWS

TinyWS is the chosen name for our small web services platform that will reside on the sensor nodes. It hosts the services, receives web service requests via SOAP and sends relevant responses via SOAP. The choice of the name TinyWS follows a common convention for numerous other embedded software modules and frameworks for sensors such as TinyOS [11] for the operating system, TinyDB [13] for a database based framework and TinyDiffussion [57] for its routing protocol.

In its simplest form, a web services platform is a SOAP processing engine. A SOAP engine is a software system that is capable of:

- receiving SOAP messages
- extract the actual request from it
- invoke another software module that can service or act upon the request
- receive response from the servicing module and package it as a SOAP message
- send a SOAP response back to the original requesting application

While a SOAP message could be carried over any protocol, the de facto standard today is to carry it over HTTP. SOAP is considered to be HTTP + XML; HTTP message carrying XML content in the body. Further, HTTP is an application layer protocol carried over TCP/IP. The implication here is that a web services platform should be built over TCP/IP and should handle HTTP as a transport protocol for SOAP. Thus, at a minimum, our TinyWS prototype should do just that.

A conventional web services platform is expected to provide a mechanism for deploying new services on the fly, the same way a web server allows deployment of new pages. The TinyWS prototype does not provide that mechanism yet.

5.2 Objectives

The objective of building this prototype is to have a proof of concept that the sensor node is capable of hosting a small Web Services platform and communicate via SOAP over HTTP. It is an initial attempt and practical assessment of the feasibility of the proposed web services based framework. The prototype is also meant to dispel the notion that such a framework is absolutely unthinkable. Another objective is to experience the challenges involved in resource optimization to fit an otherwise resource demanding framework into small and resource limited devices. The lessons learned can be taken into account when a full fleshed framework is to be built.

5.3 The Setup

Figure 5.1 shows the setup of our prototype.

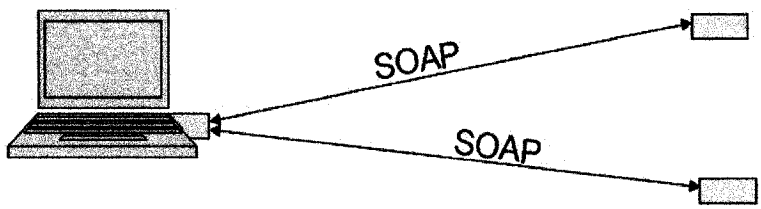

Figure 5.1 TinyWS: Setup of the prototype

The prototype has two parts; the TinyWS which is on the sensor nodes and a client application running on the laptop. A sensor node is also attached on the laptop to act as a Zigbee [43] based wireless network interface card. In terms of the Web Services roles, the remote sensor nodes act as service providers. The laptop, representing client devices, host end user application which act as web service consumer. This prototype does not include a UDDI registry and its respective operations.

It should be noted that the sensor node attached to the laptop is not considered to be a sensor in this case. It is a network card in the absence of a real Zigbee network card.

5.4 Hardware and Software Tools

The main hardware involved are the sensor nodes and the laptop.

The sensor nodes used are the TelosB motes from Crossbow Technologies, part number TPR2420. The actual node and its block diagram are as shown in Figure 5.2.

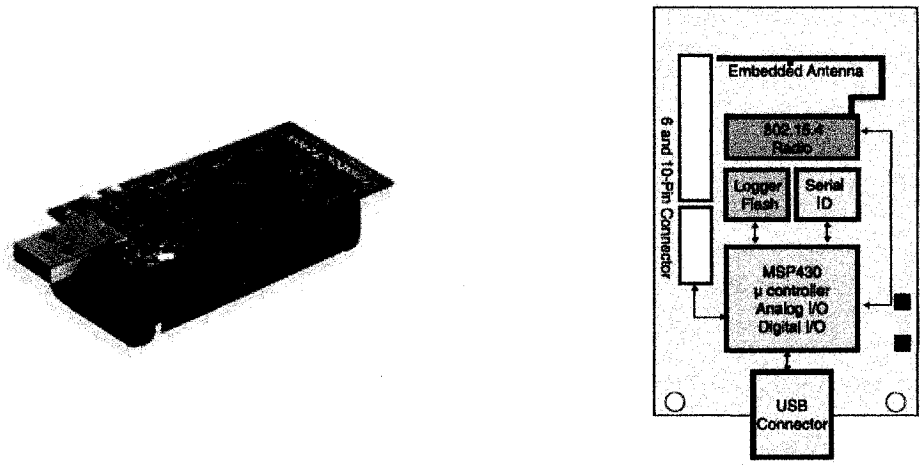

Figure 5.2 TelosB Sensor Node and its block diagram

The TelosB motes have IEEE 802.15.4 or Zigbee compliant RF Transceiver, operate at 2.4 – 2.4835GHz frequency band, have integrated onboard antenna, offer 250Kbps data rate. They are based on 8 Mhz Texas Instrument MSP430 microcontrollers, with 10KB RAM and 1MB of external flash.

The Software tools and libraries used in developing the TinyWS platform are TinyOS operating system, the nesC language, the TOSSIM Simulator, TinyOS port of the uIP [51] TCP/IP stack and the TinyXML parser.

TinyOS [11] is a lightweight operating system specifically designed for network embedded systems. It has component based architecture, whose components are modular and reusable by applications. nesC [22] is an extension of the C language primarily intended for embedded systems. It is considered a *static language* in the sense that it does not offer dynamic memory allocation, everything must be pre-decided and is checked at compile time. TOSSIM [58] is a simulator for TinyOS sensor networks that allows developers to test, debug and analyze applications without compiling them into the actual sensor nodes.

uIP [51] is an implementation of the TCP/IP protocol stack, in C language, intended for small 8-bit and 16-bit microcontrollers. It provides the necessary protocols for Internet communication, with a very small code footprint and RAM requirements. The stack has also been ported into TinyOS by re-implementing it using the nesC language and programming model. TinyXML [59] is a general purpose XML parser for TinyOS. It is based on another small and efficient parser called Parsifal [60]. However, considering resource limitations on sensor nodes, a number of features had to be removed. Thus the TinyXML is non-validating and has no support for Namespaces and DTD (Document Type Definition). Lack of these features would not affect our prototype. The parser is contributed by researchers from University of Yeditepe [61]

The Software tools used in developing the client stubs or libraries for the client application are Xerces-J XML Parser [62] (a popular XML parser for Java Langunge), jakarta HTTP client library [63] (a full compliant HTTP 1.1 library), the Java programming language and JDK 1.5 compiler.

5.5 The design

As a proof of concept, the prototype has been designed to host very simple services. The focus was to make sure that the nodes can receive and process SOAP messages, invoke the right service and send back corresponding SOAP response. The simple services, their corresponding SOAP requests and responses as well as the software architecture are presented in the following subsections.

5.5.1 The Services provided

Three simple services have been designed to cover common requests from sensors, which include data collection, changing operational settings of the nodes and checking certain node status.

The three services are:

(i) **SubscribeData**(char phenomena, int samples, int interval)

The functionality of this service is to collect sensor readings. It triggers the sensor to sense of the specified phenomena and provide the sensed data. The service takes three parameters:

- phenomena: this is a string value specifying the phenomenon to be sensed. In this case it is either temperature or light.
- samples: an integer value specifying the number of sample readings to be collected
- interval: an integer value specifying the interval between the sensor readings, in milliseconds.

The service is an example of a common query for sensor data.

(ii) **ToggleLED**(char theLED, int N, int interval)

The functionality of this service is to change operational settings of the sensor; in particular switching the sensor LED lights to ON or OFF depending on current status (toggling). The service takes 3 parameters:

- theLED: this is a string value specifying the LED to be toggled. The value can be RED, GREEN or YELLOW*.
- N: an integer value specifying the number of times that the toggling should take place
- interval: an integer value specifying the interval between the toggling, in milliseconds.

The service is an example of a command (instead of a query). The real life use would be a command to change various operational mode of the sensor, such as sending the node to sleep mode or wake up mode. A similar service could also be designed to change the status of the sensor from a normal sensor node to an aggregator, among other uses.

(iii) **GetLEDStatus()**

The functionality of this service is to get the Status of the three LED lights. This status is returned as a hexadecimal value; 000 returned as 0, meaning all the lights are OFF, 111 returned as 7, meaning all lights are ON, and the other combinations/value in between.

The service is another example of a query that takes no parameters.

5.5.2 The SOAP Messages

Inline with SOAP 1.1 specification, the SOAP requests and responses are as shown below. They all include the SOAP envelope and SOAP body. The name of the actual web service is preceded by the letter m and a colon in the corresponding XML tag of the same name as the service. This 'service tag' nests the tags that represent the

* This LED is actually blue on the sensor node, but it is specified as YELLOW for backward compatibility

parameters and their values. For the sake of simplification and taking into account the limited buffer resources on the sensor nodes, these SOAP messages neither include the lengthy XML namespaces nor the SOAP headers. The SOAP specification asserts that the two are not mandatory.

For SubscribeData service, the SOAP request and the SOAP response are as shown in Figure 5.3 and Figure 5.4

```
<SOAP-ENV:Envelope>
  <SOAP-ENV:Body>
        <m:SubscribeData>
              <phenomena> temp | light </phenomena>
              <samples> N </samples>
              <interval> T </interval>
        </m:SubscribeData>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 5.3 The SOAP Request for service SubscribeData

```
<SOAP-ENV:Envelope>
  <SOAP-ENV:Body>
        <m:SubscribeDataResponse>
              <phenomena> temp | light </phenomena>
              <value> nn.nn </value>
        </m:SubscribeDataResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 5.4 The SOAP Response for service SubscribeData

The SOAP request and SOAP response for the ToggleLED service are as shown in Figure 5.5 and 5.6

```
<SOAP-ENV:Envelope>
  <SOAP-ENV:Body>
        <m:ToggleLED>
            <ledType> RED|GREEN|YELLOW </ledType>
            <samples> N </samples>
            <interval> T </interval>
        </m:ToggleLED>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 5.5 The SOAP Request for service ToggleLED

```
<SOAP-ENV:Envelope>
  <SOAP-ENV:Body>
        <m:ToggleLEDResponse>
          <status>red_on | red_off | green_on |
                  green_off | yellow_on | yellow_off
          </status>
        </m:ToggleLED>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 5.6 The SOAP Response for service ToggleLED

The SOAP Request and Response for the GetLEDStatus Service are as shown in Figure 5.7 and 5.8

```
<SOAP-ENV:Envelope>
  <SOAP-ENV:Body>
        <m:GetLEDStatus>
        </m:GetLEDStatus>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
</SOAP-ENV:Envelope>
```

Figure 5.7 The SOAP Request for service GetLEDStatus

```
<SOAP-ENV:Envelope>
 <SOAP-ENV:Body>
        <m:GetLEDStatusResponse>
                <LEDStatus> N* </LEDStatus>
        </m:GetLEDStatusResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 5.8 The SOAP Response for the service GetLEDStatus

As mentioned earlier, SOAP messages are usually transported over HTTP. The SOAP message is carried in the body of the HTTP message. Figure 5.9 and 5.10 shows examples of the resulting HTTP messages with the SOAP messages embedded in their bodies. The examples are for the request and response of the SubscribeData service.

```
POST /TinyWS HTTP/1.1
Host: 10.0.0.11
Content-Type: text/xml; charset="utf-8"
Content-Length: nnn
SOAPAction: ""

<SOAP-ENV:Envelope>
 <SOAP-ENV:Body>
        <m:SubscribeData>
                <phenomena> temp | light </phenomena>
                <samples> N </samples>
                <interval> T </interval>
        </m:SubscribeData>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 5.9 The SOAP Request embedded in HTTP message

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnn

<SOAP-ENV:Envelope>
 <SOAP-ENV:Body>
        <m:ToggleLEDResponse>
                <status>red_on | red_off |green_on |
                        green_off | yellow_on |yellow_off
                </status>
        </m:ToggleLED>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 5.10 The SOAP Response embedded in HTTP message

These are standard HTTP messages. The requests can be produced by any HTTP client and the response can be understood by them. Most Web Services platforms have a way of automatically generating a WSDL file for the services offered. TinyWS platform doesn't do this. However, a WSDL file to describe these sample services is included as Appendix A.

5.5.3 The Software Architecture

The prototype has two parts, the TinyWS platform on the sensor node and the client library and application on the client's device, in our case a laptop.

5.5.3.1 TinyWS Architecture

The software architecture for the TinyWS is as shown in Figure 5.11

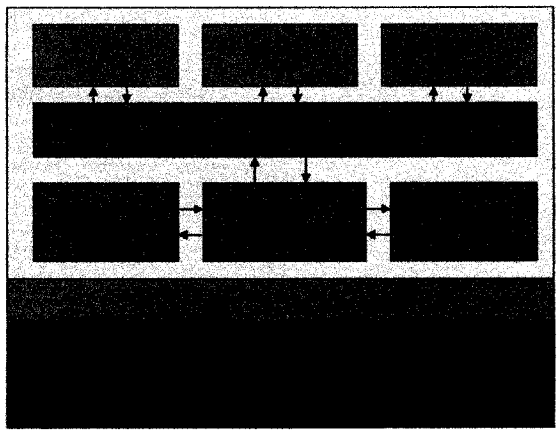

Figure 5.11 TinyWS Software Architecture

The Sensor Hardware is the TelosB Mote; it appears in the figure just for convenience as it is not a software module. TinyOS is the open source operating system for small devices, mostly used in sensors. It has been developed using the nesC language and provides a lot of interfaces that can be reused by the software modules that are built on top of it. The µIP TCP/IP stack is a light weight stack ported on nesC to run on the sensor nodes and enable them to communicate with applications running on IP based devices.

The modules developed from scratch or modified from their original form are the µHTTP, the SOAPProcessor (also called TinySOAP), the TinyXML parser, TinyWSHost and the services to be offered. The functionalities of each of these modules are as explained below:

- **µHTTP:** This module provides the necessary functions to handle HTTP messages. It uses the µIP functions to provide the HTTP protocol. The functionalities of this module are:
    - receiving HTTP messages from client applications
    - extracting SOAP/XML messages from the body of the received HTTP

messages and pass them to the TinySOAP processor

- receive SOAP/XML response messages and embed them into HTTP messages
- Sending HTTP responses (carrying SOAP) back to the requesting application

- **SOAPProcessaor/TinySOAP:** This is the module that processes the SOAP messages. The role of this module is to:
  - receive the SOAP message as extracted and handed over by the μHTTP module
  - use the TinyXML parser to extract the actual service requests and the associated parameters from the received SOAP message.
  - pass the requests and parameters to the TinyWSHost component for service invocation
  - receive responses from the TinyWSHost
  - use the TinyXML parser to create the SOAP response
  - pass the SOAP response to the μHTTP module
- **TinyXML:** This is a stripped down XML parser that assists the TinySOAP processor by:
  - Parsing the SOAP/XML to retrieve the tags and values
  - Creating the SOAP/XML responses from the parameters passed on by TinySOAP
- **TinyWSHost:** This is the module that hosts the actual web services. So, its functionalities are:

- Receiving the requests from TinySOAP Processor and invoking the right service
- Receiving responses from the actual service logic and pass them to the TinySOAP Processor

- **WS_1, WS_2, WS_3:** These are the actual services provided, containing the actual service logic.

5.5.3.2 The Client Library and application

The role of this client library is similar to the client stubs provided by Tomcat [64] web services platform or the client .jar files provided by BEA Web Logic [65] web services platform. They hold the necessary functions and SOAP proxy that will produce the SOAP/XML message embedded in HTTP. These are usually built automatically by the platform. In this small prototype, the library had to be developed.

The Software Architecture for the client library and sample application is as shown in Figure 5.12 below:

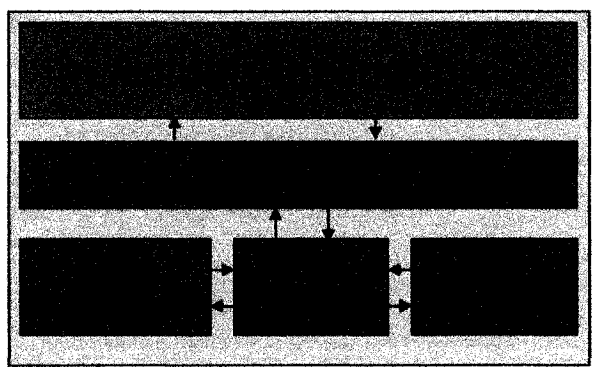

Figure 5.12: Software Architecture for the client library

The modules developed from scratch are the SOAP Processor, the TinyWSClientAPI and the sample application. The XML Parser and the HTTPLib are the

open source libraries (the Xerces-J XML Parser and the Jakarta HTTP library respectively).

The functionalities of the modules in Figure 5.12 are as explained below:

- **HTTPLib:** This is a library that receives and sends HTTP messages. It sends the HTTP messages carrying SOAP/XML to the TinyWS and, it receives the respective HTTP responses, carrying SOAP/XML from TinyWS.
- **TinyWSClientAPI:** Provides the interfaces for invoking the web services. In our prototype, the interfaces are:
    - SubscribeData (.., .., ..)
    - ToggleLED (…, …., …)
    - GetLEDStatus()
- **SOAP Processor:** This module is responsible for
    - Converting the requests and parameters from the TinyWSClientAPI into SOAP/XML messages
    - Embedding the SOAP/XML messages into the body of a HTTP message and thus creating the actual HTTP message to be sent out by HTTPLib.
    - Extracting the SOAP/XML message from the body of HTTP message received by HTTPLib
    - Extract the actual response data from the SOAP/XML messages and pass them to the TinyWSClientAPI to update the application.
- **Client Application:** This represents the actual end user application logic.

The client library is developed in Java and currently runs in the Linux environment. However, any language that supports web services could have been used, on any operating system. Also, different XML parsers and HTTP libraries could be used. What is important is that the exchanged SOAP messages are carried in standard HTTP messages between the client application and the TinyWS platform.

5.6 Challenges faced, lessons learned and related work

The main challenges in this implementation were the limited resources on the target sensor nodes on one hand and the complexity of HTPP and SOAP/XML processing on the other. Both HTTP and XML are verbose text based protocols, which tend to demand higher memory usage, processing and bandwidth, all of which are quite limited in the sensor nodes. The *static* nature of the nesC language added more challenges; a common way of optimizing memory usage is dynamic memory allocation, but the language does not support it. A work around on this was to create a special memory pool when necessary.

Other challenges arose from the fact that most of the crucial elements of this implementation are very new, undocumented and in some cases incomplete. This was particularly the case in the setup of TelosB programming environment in Linux, using the uIP port of TCP/IP stack in TinyOS and the TinyXML parser.

Sensor network programming is hard, adapting to the TinyOS programming paradigm is by itself a significant challenge. A substantial paradigm shift into component based, event driven concurrency programming is required. Contrary to conventional programming, development of TinyOS applications frequently involves calling the actual

operating system modules. Compiling the application includes recompilation of the modules from the operating system.

There is no clear separation between the OS, the uIP protocol and the applications to be built. The same buffers had to be shared between the IP protocol and the application (i.e. the TinyWS modules), which further adds to the challenge of handling shared variables, which is not so straight forward in NesC language.

The TinyOS version of uIP also posed its challenges. While the original stack is reported to be fully compliant with relevant RFCs and is well documented, its TinyOS port is yet to be fully complete and has no documentation beyond a few start-up instructions to confirm its installation.

What has been learnt is that apart from having very limited resources, the sensor nodes are capable of hosting a functional web services platform, serving several services. Although our prototype has limited features, the resources used leave a lot of room for additional features. While the node has 1MB ROM and 10Kb RAM, this simplified prototype appears to have used only 51.6 Kb ROM and 4.137 Kb of RAM.

The figures above suggest that the resource concern is more on the RAM, which seems to have taken about 41%. While this figure is relatively small, it should not suggest that having a web services platform on the sensor node is trivial. Apart from programming techniques used for memory optimization, a number of non compulsory features had to be dropped. Some features that are usually included in conventional TCP/IP stack, XML Parsers and SOAP were dropped, leaving only the minimum allowed by the relevant specifications and necessary for a simple prototype. In particular, the TCP/IP stack was made to allow only a single connection on a single port (listening for

incoming SOAP messages). More connections on the same port as well as opening additional ports for standard tools such as telnet and ping (ICMP) and others would consume more resources. The XML parser was simplified to only parse non complex XML messages, thus it does not handle Data Type Definition (DTDs) for validation. This also lowered the required computational resources without affecting core functionality of the prototype. As for SOAP, the SOAP envelopes usually contain lengthy strings representing XML name space and encoding style which would take a lot of buffer/memory. As SOAP can do without them, these were also dropped from the design, leading to further savings on resources.

In practical terms, the platform will need to scale and stabilize; this necessitates allowing multiple connections, which raise the need for state management for the connections and their corresponding web service requests. Nevertheless, the resources used by the simplified prototype suggest that having a web services platform on sensor node is not unrealistic. The conclusion would have been different if the prototype would take, say 95% of the resources, after all the simplifications.

While the resources do allow adding more features to the prototype platform, the full compatibility with other IP based applications and other web services platforms will need significant work on the XML parser and the uIP protocol modules.

As for related work, we have looked at four aspects; survey and evaluation of the state of the art in WSN middleware or application enablers, the use of web services in WSN, the sinkless architecture and development of embedded web services platforms. We explain how they relate to this work and highlight the points in which they differ.

While there are number of publications that have presented a survey of other aspects of WSN, the only one found to have surveyed the application enablers or middleware for WSN is the one reported in [90] and [91]. While it has dealt with most of the frameworks mentioned in this thesis, it has not included web services and it has used different set of criteria, not selected from application developers' point of view.

In general, the use of WS in WSN has been at the gateway level or further away from the sensor nodes; the implementation has mainly involved mapping WS to proprietary APIs or other underlying frameworks. This includes one of our publication [81] in which we have built a prototype of a WS based WSN gateway for location and environmental Sensors. SensorWeb project by OGC [68] is another example.

To the best of our knowledge, there is no work in the public domain on an actual implementation of a web services platform on individual sensor nodes, especially on TinyOS [10]. However, there are some proposals; apart from our publication [56], another proposal presented in [98] has modeled the WSN in terms of three roles and three operations of Web Services, and has proposed that the sensor nodes should host a web services platform. However, this proposal still assumes the presence of a sink node; the requesting applications bind to the sink node, not to individual sensors, and there is no TCP/IP stack on the sensor nodes. The proposal has not been implemented in actual sensor nodes.

There is another form of Web Services called REST (Representational State Transfer) [94], which is light weight and does not use SOAP. It has also been attempted in WSN in the form of TinyREST [95] built on top of TinyOS. However, the model still

relies on the presence of a sink/gateway, and, REST web services have yet to be widely adopted by developers, and not yet standardized.

As this thesis includes development of an embedded web services platform, it is also worth mentioning that there is a lot of interest in embedded web services, [99] and [100] are among those initiatives. gSOAP, explained in [101] is a SOAP toolkit meant for embedded devices, but has yet to be used on sensors.

As for the sinkless model, it has also been proposed in [38], which has an architecture that allows applications to discover and interact directly with relevant sensor nodes within single hop distance from the application device. The interaction is via TinyLIME APIs, which require understanding of the combined concepts from mobile code, tuple spaces and events; as such they are not as easy to use nor are they widely adopted as web services.

Chapter 6 Conclusion

The Wireless Sensor Networks today use proprietary mechanisms in providing data access. They use non standard protocols, which necessitate the need for gateways as bridges between application devices and the sensors. To facilitate a wider use of sensor data and motivate more innovative applications, WSN should use common and standard protocols as most application devices and provide data access via frameworks that can be used by general application developers. Web Services framework has been seen as a very promising framework but often considered unsuitable for WSN. The limited resources in sensor nodes were perceived to inhibit even the adoption of standard networking protocols such as TCP/IP.

With the latest developments in electronics, smaller but more powerful sensors continue to emerge. With necessary optimization techniques, it is now possible to have small footprints of the standard networking protocols such as TCP/IP. This work has leveraged on these developments by proposing the use of Web Services as a framework for interaction between the WSN and the end user applications; thus creating a room for more innovative applications. A prototype has been built which confirms that the idea of having web services on the sensor node is feasible. Web Services performance is still an issue that needs to and is being worked on.

6.1 Thesis Contributions

While most of the research on WSN focuses on low level routing mechanisms for large sensor networks, energy savings and hardware optimization, often by introducing new protocols and techniques, this thesis advocates the use of open standards and

common tools so as to attract a larger pool of application developers as well as allow access to sensors from more application devices. Among the contributions, a systematic survey and evaluation of existing data access mechanism, with criteria set from application developer's point of view is provided. In doing so, this thesis has also highlighted the potentials of Web Services as compared to the other frameworks. It has also presented a number of architectures for applying web services to WSN, by looking at the roles of the various WSN and mapping them to the main roles and operations of Web Services framework. A comprehensive Web Services based architecture for WSN has then been proposed, which introduce the idea of having web services at the sensor level rather than the usual convention of having it at the gateway only. As a proof of concept and assessing the feasibility of the proposed architecture, a prototype of an embedded Web Services platform residing on the sensor nodes, named TinyWS, has been built; thus dispelling the notion that web services on the sensor node is absolutely impossible.

Having a framework that is built on open networking protocols and standards has further advantages. IP based WSN provide users with full access to sensor services through common IT methodologies. The seamless integration of the sensor networks to existing IT infrastructure brings about the ability to use conventional network utilities in monitoring the network down to individual sensor level. A sensor node could be assigned a domain name, appear on SNMP management tool or its route traced using existing common tools.

Web Services are known as popular solutions in offering interoperability among dissimilar systems. Thus if the framework is supported on sensor nodes from different vendors, it is a very promising solution for interoperability in heterogeneous WSN.

Among the exciting value added services in next generation networks (3G, 4G and beyond) are context aware applications. WSN data will need to be accommodated in various applications offered through IP Multimedia System (IMS)[66]. As these network and systems are IP based, the benefit of having IP based WSN are clear. As more and more mobile devices are equipped with XML and Web Services enabled development tools such as J2ME [47], access to Web Services enabled WSN could be used to extract data for use in a local application running on the mobile device or; the sensed information could be relayed over the IP network backbone to centralized infrastructure where more applications and services can be offered.

Parts of this work have been presented and published in IEEE Symposium on Computer and Communications 2006 (ISCC'06) [81] and IEEE Consumer Communications and Networking 2007 (CCNC'07) [56]. A third publication is under review for IEEE Symposium on Computer and Communications 2007 (ISCC'07) [82].

6.2 Future Work

As this is among the initial attempts of using Web Services on WSN, especially on the individual sensor nodes, there is a lot of room for future work.

As this work has focused more on web services interactions between applications and sensors, it can be extended to accommodate relatively larger WSN, where the nodes themselves need to exchange data via web services. As such the necessary services for network formation and cooperation will need to be designed to allow SOAP based communication among the sensor nodes, regardless of the sensor type.

The work has proposed the use of an overlaid UDDI as a means to avoid centralized entities. The details and realization of this aspect is another area that needs to be addressed and developed further.

Another area to be looked at is dynamic and remote deployment of new services on the sensor nodes. One possibility is to have remote reprogramming via Agentified Web Services, in which some services can handle mobile code.

On the implementation side, apart from additional work required on the TCP/IP stack and the XML processing modules, another area to be worked on is to employ Web Services acceleration techniques, such as the ones suggested in [69] [70] [71]. Energy saving techniques should also be considered in improving the performance of TinyWS. The newly released TinyOS 2.0[72] has a number of new features that improve dynamic memory allocation, among other things, which could greatly improve the handling of the message buffers in TinyWS implementation.

References

[1] C.Y. Chong and S.P. Kumar, "Sensor networks: Evolution, opportunities, and Challenges", Proceedings of the IEEE, vol. 91, n.8, 2003, pp. 1247-1256.

[2] "10 Emerging Technologies That Will Change The World", An MIT Enterprise Technology Review, vol. 106, no. 1, pp. 33 – 49, Feb 2003. Available at: http://www.technologyreview.com/InfoTech/13060

[3] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: A survey," Computer Networks, vol. 38, pp. 393-422, 2002.

[4] David Culler, Deborah Estrin, and Mani Srivastava, Guest Editors' Introduction: "Overview of Sensor Networks", IEEE Computer, Vol. 37, No. 8, August 2004.

[5] I. Khemapech, I. Duncan, and A. Miller, ``A survey of wireless sensor networks technology," in PGNET, Proceedings of the 6th Annual PostGraduate Symposium on the Convergence of Telecommunications, Networking & Broadcasting (M. Merabti and R. Pereira, eds.), (Liverpool, UK), pp. xx-xx, EPSRC, June 2005.

[6] Crossbow Technology, Mica2 Multi-Sensor Module, available at: http://www.xbow.com/Products/productsdetails.aspx?sid=75

[7] J. Beutel, O. Kasten, M. Ringwald, F. Siegemund, and L. Thiele, "Poster abstract: Btnodes - a distributed platform for sensor nodes," in Proceedings of the First International Conference on Embedded Networked Sensor Systems (SenSys-03) Los Angeles, CA, Nov. 2003

[8] ScatterWeb Website: http://www.scatterweb.com/

[9] TECO Particle Sensor Website:

http://particle.teco.edu/

[10] TMote Sky Sensor Node Website:

http://www.moteiv.com/products-tmotesky.php

[11] TinyOS WebSite:

http://www.tinyos.net

[12] Contiki - A Dynamic Operating System for Memory-Constrained Networked Embedded Systems. Website:

http://www.sics.se/contiki/

[13] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. "Tinydb: An acquisitional query processing system for sensor networks. "Transactions on Database Systems (TODS), 2005.

[14] Chien-Liang Fok, Gruia-Catalin Roman, Chenyang Lu. "Mobile Agent Middleware for Sensor Networks: An Application Case Study" In Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks (IPSN'05), Los Angeles, California, April 25-27, 2005, pp. 382-387

[15] Kay Romer, "Programming Paradigms and Middleware for Sensor Networks". GI/ITG Fachgespräch Sensornetze, Karlsruhe, February 2004

[16] T. Arampatzis, J. Lygeros, and S. Manesis, "A survey of applications of wireless sensors and wireless sensor networks," Proceedings of the IEEE International Symposium on Intelligent Control, 2005.

[17] K. Martinez, J.K. Hart, and R. Ong, "Sensor Network Applications – Environmental Sensor Networks", IEEE Computer Society, August 2004

[18] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless Sensor Network for Habitat Monitoring", ACM WSNA'02, Atlanta, Georgia, USA, September 2002.

[19] Merrett, G. V., Harris, N. R., Al-Hashimi, B. M. and White, N. M., "Energy Controlled Reporting for Industrial Monitoring Wireless Sensor Networks". In Proceedings of IEEE Sensors 2006 (in press), Daegu, Korea.

[20] Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N. and Weerawarana, S.: Unraveling the Web Services Web An Introduction to SOAP, WSDL, and UDDI, IEEE Internet Computing, March 2002

[22] McKusick, K. "A conversation with Adam Bosworth", ACM Queue, vol. 1, no. 1, March 2003, Available at:

http://www.acmqueue.org/modules.php?name=Content&pa=showpage&pid=29

[22] Gay, D., Levis, P., Behren, R., Welsh, M., Brewer, E., Culler D.: The nesC Language: A Holistic Approach to Networked Embedded Systems. Proceedings of the ACM SIGPLAN 2003, San Diego, California, USA, pp. 1–11, 2003

[23] Cricket Version 2 User manual, MIT Computer Science and Artificial Intelligence Lab, available at http://nms.csail.mit.edu/projects/cricket/v2man-html/, Jan 2005

[24] Ember, company website:

http://www.ember.com

[25] Buonadonna, P., Gay, D., Hellerstein, J. M., Hong W. and Madden, S.: TASK: Sensor Network in a Box, Intel Research Berkeley and UC Berkeley, 2005

[26] Nath S., Deshpande A., Ke, Y., Gibbons, P. B., Karp, B., Seshan, S., "IrisNet: An Architecture for Internet-scale Sensing Services". In proceedings of the 29th VLDB

Conference, Berlin, Germany, 2003

[27] Yong Yao and J. E. Gehrke. "The Cougar Approach to In-Network Query Processing in Sensor Networks", Sigmod Record, vol. 31, no. 3, September 2002.

[28] C. Shen, C. Srisathapornphat, and C. Jaikaeo, "Sensor Information Networking Architecture and Applications," IEEE Pers. Commun., Aug. 2001, pp. 52–59.

[29] SensorWare Project

http://sensorware.sourceforge.net/ and

http://nesl.ee.ucla.edu/projects/sensorware/

[30] Adam Dunkels, Richard Gold, Sergio Angel Marti, Arnold Pears, and Mats Uddenfeldt. "Janus: An architecture for flexible access to sensor networks". In First International ACM Workshop on Dynamic Interconnection of Networks (DIN'05), Cologne, Germany, September 2005.

[31] P. Kang, C. Borcea, G. Xu, A. Saxena, U. Kremer and L. Iftode, "Smart Messages: A Distributed Computing Platform for Networks of Embedded Systems", The Computer Journal, Special Focus Mobile and Pervasive Computing, British Computer Society, Oxford University Press, Vol 47/4, pp 475-494, July 2004.

[32] Ting Liu and Margaret Martonosi. "Impala: A Middleware System for Managing Autonomic, Parallel Sensor Systems". ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'03), June 2003. Paper available at:

http://www.cs.princeton.edu/~tliu/p71-tliu.pdf

[33] P.Levis and D. Culler. "Mate': A Tiny Virtual Machine for Sensor Networks." In International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, USA, Oct. 2002.

[34] "Specification: Web Services Security (WS-Security)", available at:

http://www-106.ibm.com/developerworks/webservices/library/ws-secure/

[35] "Web Services Security: Non-Repudiation proposal draft 05", available at:

http://xml.coverpages.org/ReactivityWS-NonRepudiation-05.pdf

[36] Peter Schramm, Edwin Naroska, Peter Resch, Platte Platte, Holger Linde, Guido Stromberg, Thomas Sturm. "A Service Gateway for Networked Sensor Systems," IEEE Pervasive Computing, vol. 03, no. 1, pp. 66-74, January-March 2004.

[37] Eduardo Souto, Germano Guimarães, Glauco Vasconcelos, Mardoqueu Vieira, Nelson Rosa, Carlos Ferraz, "A Message-Oriented Middleware for Sensor Networks", Toronto, Ontario, Canada, 2004, pages 127 – 134

[38] Carlo Curino et al, "TinyLIME: Bridging Mobile and Sensor Networks through Middleware" Proceedings of the 3rd IEEE Int'l Conf. on Pervasive Computing and Communications (PerCom 2005)

[39] Jini Network Technology Website:

http://www.sun.com/software/jini/

[40] uPnP Website:

http://upnp.org/

[41] Murphy, A.L., Picco, G.P., Roman, G.C., "LIME: a middleware for physical and logical mobility", Distributed Computing Systems, 2001. 21st International Conference on, 16-19 April 2001 Page(s):524 - 533

[42] IEEE 802.15.4 Standard Specification, available at:

http://standards.ieee.org/getieee802/download/802.15.4-2003.pdf

[43] Zigbee Alliance, Zigbee specification 1.0, June 2005, available at:

http://www.zigbee.org/en/index.asp

[44] TelosB Mote Research platform, Crossbow product website:

http://www.xbow.com

[45] UZBee, A USB to Zigbee Adapter. Product by Flexipanel. Company Website:

http://www.flexipanel.com/

[46] Christian, Andrew; Hicks, Jamey; Avery, Brian; Kuris, Ben; Denning, Don; Ayer, Steven; Ankcorn, John, "Fingertips of the Network: Featherweight Communicators and Sensors", HP Lab Technical report, available at:

http://www.hpl.hp.com/techreports/2005/HPL-2005-114.pdf

[47] The iB-Bean GW-5324-CF Zigbee Adapter PCMCIA card, product Website:

http://www.wirelessmeasurement.com/millennial_24.html

[48] "TIM unveils Z-SIM SIM card with radio technology", an article by John Tilak on Digital Media news for Europe. Available online at:

http://www.dmeurope.com/default.asp?ArticleID=11958

[49] Xiaohua Luo, Kougen Zheng, Yunhe Pan, Zhaohui Wu, "A TCP/IP implementation for wireless sensor networks", Systems, Man and Cybernetics, 2004 IEEE International Conference on Volume 7, 10-13 Oct. 2004 Page(s):6081 - 6086 vol.7

[50] Adam Dunkels, Thiemo Voigt, and Juan Alonso. "Making TCP/IP Viable for Wireless Sensor Networks." In Proceedings of the First European Workshop on Wireless Sensor Networks (EWSN'04), work-in-progress session, Berlin, Germany, January 2004.

[51] The uIP embedded TCP/IP stack, Project web site:

http://www.sics.se/~adam/uip/

[52] Adam Dunkels, Thiemo Voigt, Niclas Bergman, and Mats Jönsson. "The Design and

Implementation of an IP-based Sensor Network for Intrusion Monitoring", in Swedish National Computer Networking Workshop, Karlstad, Sweden, November 2004.

[53] A. Christian, J. Healey, "Gathering Motion Data Using Featherweight Sensors and TCP/IP over 802.15.4", IEEE International Symposium on Wearable Computers, On-Body Sensing Workshop, October 2005. Available at:
http://www.hpl.hp.com/techreports/2005/HPL-2005-188.pdf

[54] Xingfa Shen , Zhi Wang, and Youxian Sun, "Wireless Sensor Networks for Industrial Applications", Fifth World Congress on Intelligent Control and Automation, WCICA, 2004, Vol 4, pp 3636-3640

[55] Adam Dunkels, Thiemo Voigt, and Juan Alonso, "Making TCP/IP Viable for Wireless Sensor Networks." In Proceedings of the First European Workshop on Wireless Sensor Networks (EWSN'04), work-in-progress session, Berlin, Germany, January 2004.

[56] N. Y. Othman, S. Chebbine, R.Glitho and F. Khendek, "A Web Services Based Architecture for the Interactions between End-users Applications and Sinkless Wireless Sensor Networks", to appear in proceedings of IEEE Consumer Communications and Networking Conference 2007 (CCNC'07), January 11 – January 13, 2007, Las Vegas, Nevada, USA.

[57] General purpose XML Parser for TinyOS (TinyXML), project website:
http://ics.yeditepe.edu.tr/tnl/html/tinyxml.html

[58] TinyDiffusion Application Programmer's Interface, version 0.1
http://www.isi.edu/scadds/papers/tinydiffusion-v0.1.pdf

[59] Philip Levis, Nelson Lee, Matt Welsh, and David Culler, "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications", In Proceedings of the First

ACM Conference on Embedded Networked Sensor Systems (SenSys) 2003, November 2003

[60] Parsifal XML Parser. Project website:

http://www.saunalahti.fi/~samiuus/toni/xmlproc/

[61] University of Yeditepe, Computer Engineering Department:

http://ics.yeditepe.edu.tr/

[62] Xerces-J XML Parser, project web site:

http://xerces.apache.org/xerces-j/

[63] Jakarta Commons HTTP client, project website:

http://jakarta.apache.org/commons/httpclient/

[64] Apache Tomcat Server

http://tomcat.apache.org/

[65] BEA Web Logic Server Website:

http://www.bea.com/

[66] Miguel-Angel, García-Martín, Gonzalo Camarillo, "The 3G IP Multimedia Subsystem (IMS): Merging the Internet and the Cellular Worlds", John Wiley and Sons, 2004

[67] The Java Micro Edition (J2ME) Platform website:

http://java.sun.com/javame/index.jsp

[68] Open Geospatial Consortium (OGC) Inc

http://www.opengeospatial.org/

[69] N. Abu-Ghazaleh, M. Lewis, M. Govindaraju, "Differential Serialization for Optimized SOAP Performance", in: Proc. 13th IEEE International Symposium on

High Performance Distributed Computing, Honolulu, Hawaii, June 2004, pp. 55-64.

[70] K. Chiu, M. Govindaraju, R. Bramley, "Investigating the Limits of SOAP Performance for Scientific Computing", in: Proc. 11th IEEE International Symposium on High Performance Distributed Computing, Edinburgh, Scotland, July 2002, pp. 246-254.

[71] M. Migliardi, R. Podesta, "Performance Improvement in Web Services Invocation Framework", in: Proc. 18th International Parallel and Distributed Processing Symposium, Santa Fe, New Mexico, April 2004, pp. 110-122.

[72] TinyOS 2.0 Documentation. Available at:
http://www.tinyos.net/tinyos-2.x/doc/

[73] XML specification, available at:
http://www.w3.org/XML/

[74] Simple Object Access Protocol (SOAP) Specification 1.1, available at:
http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[75] Web Services Description Language (WSDL) 1.1, available at:
http://www.w3.org/TR/wsdl

[76] Universal Data Discovery and Integration (UDDI) Specifications, available at:
http://www.uddi.org/

[77] SensorML
http://vast.uah.edu/SensorML/

[78] M. Srivastava, R. Muntz, and M. Potkonjak, "Smart kindergarten: Sensor-based wireless networks for smart developmental problem solving environments," Proc. 7th Ann. Int'l Conf Mobile Computing and Networking, Rome, Italy, pp. 132-138, Jul. 2001

[79] J.M. Kahn, R.H. Katz, and K.S.J. Pister, "Next Century Challenges: Mobile Networking for "Smart Dust"", MobiCom'99, Seattle, Washington, 1999.

[80] G. Amato, S. Chessa, F. Conforti, A. Macerata, and C. Marchesi, "Health Care Monitoring of Mobile Patients", News No. 60, January 2005. Available online at: http://www.ercim.org/publication/Ercim_News/enw60/amato.html

[81] Truong Ta, Nuru Yakub Othman, Roch H. Glitho and Ferhat Khendek, "Using Web Services for Bridging End User Applications and Wireless Sensor Networks", in proceedings of IEEE Symposium of Computer and Communications 2006 (ISCC'06), June 26th – June 29th , Cagliari, Italy.

[82] Nuru Yakub Othman, Roch H. Glitho and Ferhat Khendek, "The Design and Implementation of a Web Service Framework for Individual Nodes in Sinkless Wireless Sensor Networks", under Review for IEEE Symposium of Computer and Communications 2007 (ISCC'07), July 1st – 4th, Aveiro, Portugal.

[83] Imad Mahgoub, Mohammad Ilyas, "Smart Dust: Sensor Network Applications, Architecture, and Design", CRC Press, Taylor and Francis Group, 2006.

[84] M. Haenggi, "Twelve reasons not to route over many short hops," In Proc. IEEE Vehicular Technology Conference. (VTC'04), Los Angeles, CA, Sep. 2004 Available at: http://www.nd.edu/~mhaenggi/pubs/vtc04.pdf

[85] M. Haenggi, "Energy-balancing Strategies for Wireless Sensor Networks", In Proceedings of the 2003 International Symposium on Circuits and Systems (ISCAS '03), May 2003. Available at: http://www.nd.edu/~mhaenggi/pubs/iscas03.pdf

[86] J. Burrel, T. Brooke, and R. Beckwith, "Vineyard computing: Sensor Networks in agricultural production", IEEE Pervasive Computing, vol. 3, no. 1, pp. 38-45, 2004

[87] T. Fullford-Jones, D. Malan, M. Welsh, M. Gaynor, and S. Moulton, "CodeBlue: An ad hoc sensor network infrastructure for emergency medical care", in International Workshop on Wearable and Implantable Body Sensor Networks, London, UK, 2004.

[88] D. Myung, B. Duncan, D. Malan, M. Welsh, M. Gaynor, and S. Moulton, "Vital dust: Wireless sensor network for real time patient monitoring", in 8th Annual England Regional Trauma Conference, Burlington, MA, 2002.

[89] Puccinelli D, Haenggi M, "Wireless Sensor Networks: Applications and Challenges of Ubiquitous Sensing", IEEE Circuits and Systems Magazine, vol. 5, issue. 3, 2005 pp. 19 – 31.

[90] Salem Hadim, Nader Mohamed, "Middleware: middleware challenges and approaches for Wireless Sensor Networks", IEEE Distributed Systems Online, vol. 7, issue. 3, March 2006.

[91] Salem Hadim, Nader Mohamed, "Middleware for Wireless Sensor Networks: A Survey", First International Conference on Communication System Software and Middleware, 08-12 Jan, 2006, pp 1- 7

[92] Jason Lester Hill, "System Architecture for Wireless Sensor Network", PhD dissertation, University of California, Berkeley, 2003.

[93] J. Feng, F. Koushnfar, and M. Potkonjak, "System-Architecture for Sensor Networks. Issues, Alternatives and Directions", ICCD'02, 2002

[94] Goth G., "Critics Say Web Services Need a REST", IEEE Distributed Systems

Online, vol. 5, issue 2, Dec 2004, page 1.

[95] Thomas Luckenbach, Peter Gober, Andreas Kotsopoulos, Kyle Kim, Stefan Arbanowski, "TinyREST - A Protocol for Integrating Sensor Networks into the Internet", Proceedings of REALWSN 2005

[96] Kirsten Terfloth and Jochen Shiller, "Driving forces behind middleware concepts for Wireless Sensor Networks", Proceeding of REALWSN Workshop, Stockholm, Sweden, June 2005

[97] Y. Yu, B. Krishnamachari, and V.K. Prasanna, "Issues in Designing Middleware for Wireless Sensor Networks", In IEEE Network Magazine, Jan/Feb 2004, vol. 18, issue. 1, pp. 15 - 21

[98] Flávia Coimbra Delicato, Paulo F. Pires, Luci Pirmez, and Luiz Fernando Rust da Costa Carmo (Federal University of Rio de Janeiro, Brazil) "A Flexible Web Service based Architecture for Wireless Sensor Networks" 23rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03), 2003, Providence, Rhode Island, USA, pp 730

[99] Dimitris Lioupis, Michalis Stefanidakis, "A Web Service for Embedded Distributed Computation", 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP'05), Feb 9 -11, 2005, pp. 20-25.

[100] Robert van Engelen, "Code Generation Techniques for Developing Web Services for Embedded Devices", in the proceedings of the 9th ACM Symposium on Applied Computing SAC, Nicosia, Cyprus, 2004, pp. 854-861.

Appendix A: Sample WSDL file for the TinyWS prototype

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/">
 <types>
  <s:schema elementFormDefault="qualified" targetNamespace="http://www.openuri.org/"
xmlns:s="http://www.w3.org/2001/XMLSchema">
    <s:element name="subscribeData">
     <s:complexType>
      <s:sequence>
       <s:element name="phenomena" type="s:string" minOccurs="0"/>
       <s:element name="samples" type="s:int"/>
       <s:element name="Interval" type="s:int"/>
      </s:sequence>
     </s:complexType>
    </s:element>
    <s:element name="subscribeDataResponse">
     <s:complexType>
      <s:sequence>
       <s:element name="phenomena" type="s:string"/>
           <s:element name="value" type="s:int"/>
          </s:sequence>
     </s:complexType>
    </s:element>
    <s:element name="toggleLED">
     <s:complexType>
      <s:sequence>
       <s:element name="ledType" type="s:string" minOccurs="0"/>
       <s:element name="samples" type="s:int"/>
       <s:element name="Interval" type="s:int"/>
      </s:sequence>
     </s:complexType>
    </s:element>
    <s:element name="toggleLEDResponse">
     <s:complexType>
      <s:sequence>
       <s:element name="status" type="s:string"/>
      </s:sequence>
     </s:complexType>
    </s:element>
    <s:element name="getLEDStatus">
     <s:complexType>
      <s:sequence/>
     </s:complexType>
    </s:element>
    <s:element name="getLEDStatusResponse">
     <s:complexType>
      <s:sequence>
           <s:element name="LEDStatus" type="s:int"/>
      </s:sequence>
     </s:complexType>
    </s:element>
  </s:schema>
 </types>
```

```
<message name="subscribeDataSoapIn">
  <part name="parameters" element="s0:subscribeData"/>
</message>
<message name="subscribeDataSoapOut">
  <part name="parameters" element="s0:subscribeDataResponse"/>
</message>
<message name="toggleLEDSoapIn">
  <part name="parameters" element="s0:toggleLED"/>
</message>
<message name="toggleLEDSoapOut">
  <part name="parameters" element="s0:toggleLEDResponse"/>
</message>
<message name="getLEDStatusSoapIn">
  <part name="parameters" element="s0:getLEDStatus"/>
</message>
<message name="getLEDStatusSoapOut">
  <part name="parameters" element="s0:getLEDStatusResponse"/>
</message>
<message name="subscribeDataHttpPostIn">
  <part name="phenomena" type="s:string"/>
  <part name="samples" type="s:string"/>
  <part name="Interval" type="s:string"/>
</message>
<message name="subscribeDataHttpPostOut"/>
<message name="toggleLEDHttpPostIn">
  <part name="ledType" type="s:string"/>
  <part name="samples" type="s:string"/>
  <part name="Interval" type="s:string"/>
</message>
<message name="toggleLEDHttpPostOut"/>
<message name="getLEDStatusHttpPostIn"/>
<message name="getLEDStatusHttpPostOut"/>
<portType name="TinyWSSoap">
  <operation name="subscribeData">
    <input message="s0:subscribeDataSoapIn"/>
    <output message="s0:subscribeDataSoapOut"/>
  </operation>
  <operation name="toggleLED">
    <input message="s0:toggleLEDSoapIn"/>
    <output message="s0:toggleLEDSoapOut"/>
  </operation>
  <operation name="getLEDStatus">
    <input message="s0:getLEDStatusSoapIn"/>
    <output message="s0:getLEDStatusSoapOut"/>
  </operation>
</portType>
<portType name="TinyWSHttpPost">
  <operation name="subscribeData">
    <input message="s0:subscribeDataHttpPostIn"/>
    <output message="s0:subscribeDataHttpPostOut"/>
  </operation>
  <operation name="toggleLED">
    <input message="s0:toggleLEDHttpPostIn"/>
    <output message="s0:toggleLEDHttpPostOut"/>
  </operation>
  <operation name="getLEDStatus">
```

```
    <input message="s0:getLEDStatusHttpPostIn"/>
    <output message="s0:getLEDStatusHttpPostOut"/>
  </operation>
 </portType>
 <binding name="TinyWSSoap" type="s0:TinyWSSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  <operation name="subscribeData">
    <soap:operation soapAction="http://www.openuri.org/subscribeData" style="document"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
  <operation name="toggleLED">
    <soap:operation soapAction="http://www.openuri.org/toggleLED" style="document"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
  <operation name="getLEDStatus">
    <soap:operation soapAction="http://www.openuri.org/getLEDStatus" style="document"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
 </binding>
 <binding name="TinyWSHttpPost" type="s0:TinyWSHttpPost">
  <http:binding verb="POST"/>
  <operation name="subscribeData">
    <http:operation location="/subscribeData"/>
    <input>
      <mime:content type="application/x-www-form-urlencoded"/>
    </input>
    <output/>
  </operation>
  <operation name="toggleLED">
    <http:operation location="/toggleLED"/>
    <input>
      <mime:content type="application/x-www-form-urlencoded"/>
    </input>
    <output/>
  </operation>
  <operation name="getLEDStatus">
    <http:operation location="/getLEDStatus"/>
    <input>
      <mime:content type="application/x-www-form-urlencoded"/>
    </input>
    <output/>
```

```
    </operation>
  </binding>
  <service name="TinyWS">
    <port name="TinyWSSoap" binding="s0:TinyWSSoap">
      <soap:address location="http://10.0.1.2:81"/>
    </port>
    <port name="TinyWSHttpPost" binding="s0:TinyWSHttpPost">
      <http:address location="http://10.0.1.2:81"/>
    </port>
  </service>
</definitions>
```