

Open Sensor Web Architecture: Core Services

By
Xingchen Chu

Under the Supervision of
Dr. Rajkumar Buyya

A minor project thesis submitted in partial fulfillment
of the requirement for the degree of
Master of Information Technology

Grid Computing and Distributed Systems Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia

Dec 2005

Table of Content

CHAPTER 1 INTRODUCTION	1
CHAPTER 2 RELATED WORKS.....	4
2.1 <i>Sensor Applications</i>	4
2.2 <i>Sensor Middleware</i>	6
2.3 <i>Sensor Grid and Sensor Web</i>	7
CHAPTER 3 OPEN SENSOR WEB ARCHITECTURE.....	8
3.1 <i>Overview</i>	8
3.2 <i>Sensor Web Enablement</i>	9
3.2.1 <i>SensorML</i>	10
3.2.2 <i>Observation and Measurement</i>	11
3.2.3 <i>SWE Services</i>	11
CHAPTER 4 DESIGN AND IMPLEMENTATION	13
4.1 <i>Overview</i>	13
4.1.1 <i>Service Layer and Service Oriented Architecture</i>	14
4.1.1.1 <i>Jini</i>	14
4.1.1.2 <i>Web Services</i>	16
4.1.2 <i>Information Model and XML Data Binding</i>	17
4.1.3 <i>Sensor Operating System</i>	18
4.1.4 <i>Sensor Data Persistence</i>	20
4.2 <i>Services Architecture</i>	21
4.2.1 <i>Sensor Collection Service</i>	21
4.2.2 <i>Planning and Notification Service</i>	22
4.3 <i>Implementation</i>	23
4.3.1 <i>Sensor Collection Service</i>	23
4.3.1.1 <i>Core Sensor Collection Service Components</i>	24
4.3.1.2 <i>Connector Component</i>	26
4.3.1.3 <i>Data Handler Component</i>	27
4.3.1.4 <i>Data Format Component</i>	28
4.3.1.5 <i>Configuration</i>	29
4.3.2 <i>Sensor Planning Service</i>	30
4.3.3 <i>Web Notification Service</i>	31
4.3.4 <i>Sensor Repository Service</i>	32
CHAPTER 5 EVALUATION	34
5.1 <i>Test Platform</i>	34
5.2 <i>Temperature Monitoring Application</i>	35
5.3 <i>Empirical Result</i>	37
5.3.1 <i>Client side</i>	37
5.3.2 <i>Performance</i>	38
CHAPTER 6 CONCLUSION AND FUTURE WORKS	41
REFERENCE	43

Abstract

A new trend, called Sensor Web, makes various types of web-resident sensors, instruments, and image devices as well as repositories of sensor data discoverable, accessible and where applicable, controllable via the World Wide. A lot of efforts have been made to overcome the obstacles connecting and sharing the heterogeneous sensor resources. Open GIC Consortium (OGC) has introduced Sensor Web Enablement (SWE) concept that is actually a set of specifications including SensorML, Observation & Measurement, Sensor Collection Service, Sensor Planning Service and Web Notification Service to implement the Sensor Web. The Open Sensor Web Architecture (OSWA) proposed by NICTA, at University of Melbourne extends the SWE and further integrates the Sensor Web and Grid Computing as well as providing middleware support of Sensor Web. This thesis describes the design and implementation of the subset of OSWA core middleware including planning, notification, collection and repository services. These services have been deployed as Web Services to support both auto-sending and query based sensor applications running on top of TinyOS [28]. A simple temperature monitoring sensor application has also developed and deployed onto the Crossbow's motes. Moreover, integration test has been conducted under both real sensor hardware provided by Crossbow's Development Kit [37] as well as a simulation environment called TOSSIM [36]. Performance evaluation has also been executed in our experiment to demonstrate the capability and scalability of the collection service.

Acknowledgement

I would like to express my gratitude to my supervisor Dr. Rajkumar Buyya. I thank him for his continues encouragement, support and for sharing his knowledge and experience. Furthermore, I wish to thank Bohdan Durnota, who is the Visiting Researcher, Grid Computing Laboratory of University of Melbourne. He takes a lot of his time with me to discuss the issues of OGC SWE and the solutions as well. He also shares his great experience in the sensor application area with me. Besides that, I would like to thank my teammate Jiye Lin, for the pleasure time we discuss the problems and play basketball. Finally, I wish to express my great appreciation to my wife Siyin Sun, for her support and understanding over time.

Chapter 1 Introduction

As the rapid development of sensor technology, sensor nodes that integrate several kinds of sensors, CPU, memory and a wireless transceiver are currently much more powerful and smarter. The Sensor networks are long running computing systems that consist of a collection of sensing nodes working together to collect information about light or temperature as well as images and other relevant data according to their specific applications. Wireless sensor networks have been attracting a huge number of attentions from both academy as well as industry all around the world. As [1] states, the great ability of the sensor networks to collect information of accuracy and reliability enables building early warnings and rapid coordinated responses to threats such as large forest fire, terrible Tsunamis, earthquakes and so forth.

Because of the heterogeneity feature of sensor networks, how to process and make use of the information gathered by the sensor nodes becomes a rather challenging task of the research area. The Service Oriented Architecture (SOA) makes it possible to describe, discovery, and invoke services from heterogeneous platform using SOAP and XML standard. It is a very important step forward to present the sensors as important resources which can be discoverable, accessible and where applicable, controllable via the World Wide Web. Furthermore, it also possible to integrate these resources with grid computing technology to create an illusion of sensor-grid that enables the essential strengths and characteristics of sensor networks and grid computing. In the sensor-grid integration strategy described by [9], various sensor networks are treated as resources, users can query data through grid resource broker and the broker will generate the response according to the resources it requires.

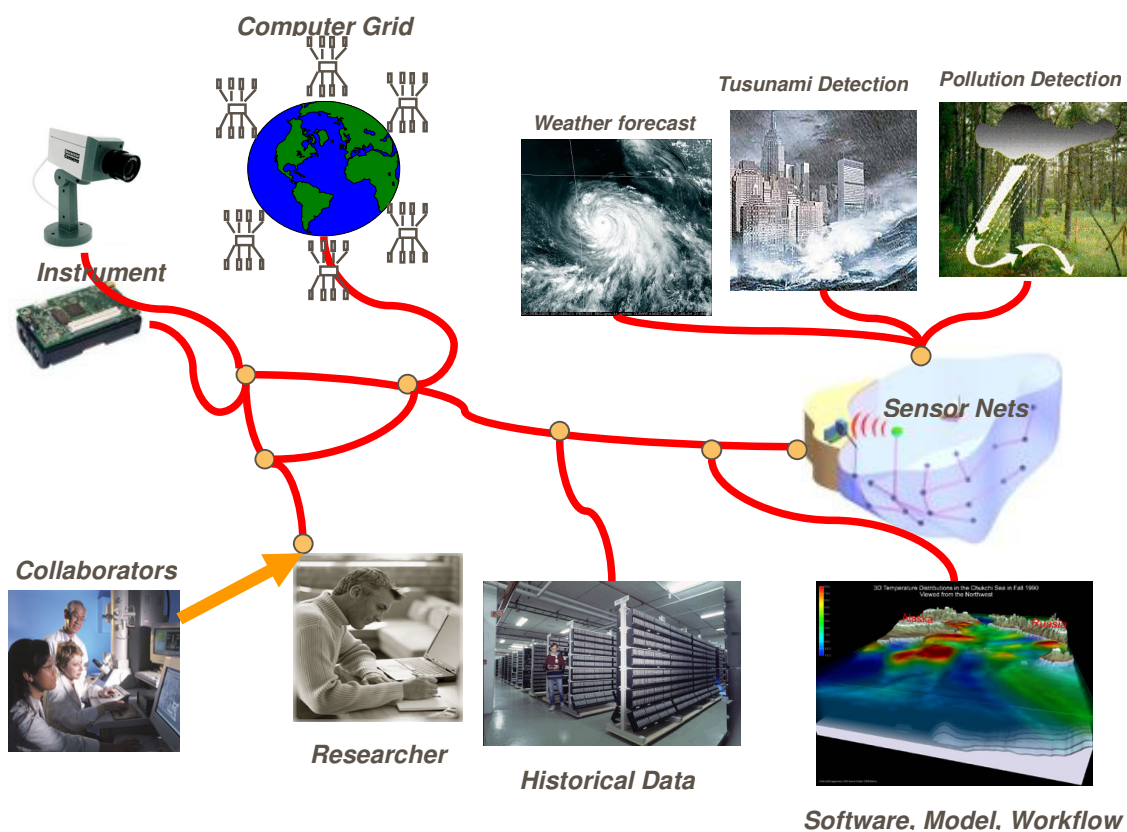


Figure 1 Vision of Sensor Web.

The combination of SOA, grid computing and sensor networks make it possible to form a web view of different sensor and sensor nodes and treat them as available services to all of the users who access the web. It brings the heterogeneous sensors into an integrated and uniform platform with the ability of discovering and accessing. At NICTA (National ICT Australia Ltd) University of Melbourne, there is an effort called the OSWA (Open Sensor Web Architecture) to utilize the combination of these three important technologies. It aims at providing the middleware support and programming environment for creating, accessing and utilizing sensor services through the web.

The remainder of this thesis is organized as follows. Related works about the sensor applications, sensor middleware supports, sensor-grid and sensor webs are described in chapter 2. The details of OSWA and the Sensor Web Enablement are introduced in chapter 3. Chapter 4 concentrates on the design and implementation our prototype middleware. Evaluation has been made in chapter 5 applying the middleware with a

simple temperature monitoring sensor application. Finally, the thesis concludes with an outline of future works on the sensor web and the OSWA.

Chapter 2 Related Works

In this chapter, we briefly discuss the related works in the areas of sensor applications, sensor middleware, sensor grid and sensor web.

2.1 Sensor Applications

Project	Research Group	Devices Used
Great Duck Island Application	Researchers from UCB/Intel Research Laboratory	UC Berkeley Mica Mote: Atmel Atmega 103 microcontroller and Mica Weather Board includes temperature, photoresistor, barometer, humidity and thermopile sensors [2]
Cane-toad Monitoring	Researchers from the School of Computer Science and Engineering, University of New South Wales	Crossbow's Mica mote family: Mica2 and X-Scale Single Board Computer [5]
Soil Moisture Monitoring	Researchers from School of Computer Science and Engineering, University of Western	Crossbow's Mica2 433 MHz motes and MDA300 sensor boards [6]
Integrated Vehicle Health Monitoring (IVHM)	Researchers from CSIRO Australia, part of NASA Robust Aerospace Vehicle Program (RAV)	Texas Instruments TMS320F2810 processor for data acquisition and an array of acoustic emission sensors for the proto-type Concept Demonstrator [7]

Researchers of the Great Duck Island application deployed a mote-based tiered sensor network on Great Duck Island, Maine, to monitor the behavior of storm petrel [2]. 32 motes were placed at the area of interests, and they are grouped into sensor patches to transmit sensor reading to a gateway, which is responsible for forwarding the data from the sensor patch to a remote basestation, as [2] state. The basestation then provides data logging and replicates the data every 15 minutes to a Postgress database

in Berkeley over satellite link.

Similarly, the Cane-Toad Monitoring application is designed to monitor *cane toads* in Kakadu National Park and the Roper valley of Northern Territory, Australia [3,5]. Two prototypes of wireless sensor networks have been set up, which can recognize vocalizations of at maximum 9 frog species in northern Australia. Besides monitoring the frogs, the researchers are also planning to monitor breeding populations of endangered birds in the future.

The purpose of the Soil Moisture Monitoring application is to better manager surface water in soil, researchers are trying to design, implement and field-test a prototype wireless sensor network for soil moisture monitoring. According to [4], a prototype sensor network for soil moisture monitoring has been deployed at Pinjar, a place located north of Perth WS on 25 June 2004. All the data is gathered by their reactive sensor network at Pinjar, and sent back to a database in real time using a SOAP web service. Besides, [4] also presents that 15 soil moisture probes have deployed to measure surface soil moisture when flood or drought occurs in early 2005.

A practical IVHM system, according to [7], will be a network containing thousands or millions of heterogeneous sensors and is used to give a proper response in real-time as soon as the damage has been detected. They have developed the Concept Demonstrator (CD) as a prototype and test-bed for detecting and measuring impacts on the aluminum skin of a space vehicle, where the sensing is performed using an array of acoustic emission sensors mounted on the inside of the skin. However, as [8] states, the IVHM system is only designed to be a powerful and flexible test-bed and an experimental platform, not intending to be a realistic practical prototype.

2.2 Sensor Middleware

Project	Description	Remarks
Impala [10]	It designed for use in ZebraNet project as a middleware architecture that enables modularity, adaptivity and reparability in wireless sensor network	It adopts mobile code techniques to upgrade functions of the remote sensors The key to energy efficient for Impala is for the sensor node applications to be as modular as possible, enabling small updates that require little transmission energy
MiLAN	MiLAN allows applications to specify a policy for managing the network and the sensors [11]	An architecture extends into network protocol stack and allows network specific plug-ins to convert MiLAN commands to protocol-specific commands
Cougar [12]	Being described as a device database system, that can execute queries	It implements a query-based database interface and uses SQL-like language for gaining information of wireless sensor networks
Mires [13]	A message-oriented middleware and placed on top of Operating System	It provides high-level services to the Node application and implements a publish/subscribe service intermediating the communication between middleware services.

2.3 Sensor Grid and Sensor Web

Project	Description	Remarks
Hourglass [14]	A data-collection-network approach to address many of the technical problems of integrating resource-constrained wireless sensors into traditional grid applications has been suggested	It provides a grid API to a heterogeneous group of sensors. Those, in turn, provide fine-grained access to sensor data with OSGA standards
Sensor Grids for Air Pollution Monitoring	[15] introduced a sensor grid integration methodology by using of grid services to encompass high throughput sensors	Making each sensor becoming a grid service. The service can be published in a registry using standard methods and make available to other users.
Sensor Web Enablement (SWE) [16]	SWE consists of a set of standard services to build a unique and revolutionary framework for discovering and interacting with web-connected sensors and for assembling and utilizing sensor networks on the web	SWE is still in the draft process and will be evolving in the near future, a lot of new specifications will be developed and introduced to the actual development of Sensor Web
GeoSWIFT [17]	An open geospatial information infrastructure for Sensor Web built at GeoICT Lab of York University	GeoSWIFT builds on the OpenGIS standards, XML messaging technology has been developed, serving as a gateway that integrates and fuses observations from heterogeneous spatial enabled sensors
IrisNet [18]	An software infrastructure that supports the central tasks command to collect, filter and combines sensor feeds and perform distributed queries at Intel Research Center	There are two-tier of IrisNet architecture including sensing agents that provide a generic data acquisition interface to access sensors, and organizing agents that are the nodes implementing the distributed database.

Chapter 3 Open Sensor Web Architecture

3.1 Overview

Open Sensor Web Architecture is an OGC's Sensor Web Enablement standard compliant software infrastructure for providing SOA based access to and management of sensors proposed at NICTA/Melbourne University. OSWA is a complete standards compliant platform for integration of sensor networks and emerging distributed computing platform such as SOA and grid computing. There are several benefits that the integration has brought to the community. First of all, the heavy load of information processing can be moved from sensor networks to the backend-distributed systems such as Grids. The separation is either saving a lot of energy and power of sensor networks just concentrating on sensing and sending information or accelerating the process for processing and fusing the huge amount of information by utilizing distributed systems. Moreover, individual sensor networks can be linked together as services, which can be register, discover and access by different clients using a uniform protocol. What is more, according to [9], Grid-based sensor applications can provide advanced services for smart-sensing by developing scenario-specific operators at runtime.

The various components defined in OSWA are showed in Figure 2. Four layers have been defined which are SensorWeb Fabric, SensorWeb Core Middleware, SensorWeb User-Level Middleware and SensorWeb Applications respectively. Fundamental services are provided by low-level components whereas components at higher-level provides tools for creating applications and management of lifecycle of data captured through sensor networks. The OSWA based platform provides a number of sensor services such as:

- Sensor notification, collection and observation
- Data collection, aggregation and archive
- Sensor coordination and data processing
- Faulty sensor data correction and management, and
- Sensor configuration and directory service

The project mainly focuses on providing an interactive development environment, an open and standards-compliant SensorWeb services middleware and a coordination language to support the development of various sensor applications.

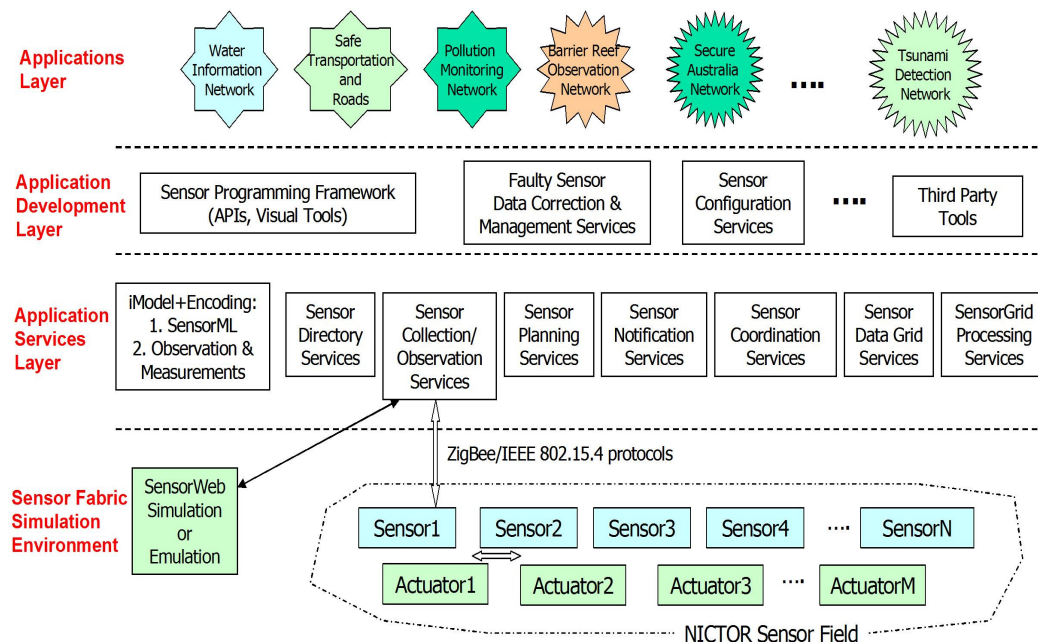


Figure 2 Open Sensor Web Architecture Overview [9].

3.2 Sensor Web Enablement

The proposed OSWA core middleware is the SWE standard compliant system, as the SWE becomes the de-facto standard description of SensorWeb development area. It is very important to understand the details of SWE. Sensor Web Enablement is the standard specification developed by OGC, which consists of five sub specifications including SensorML, Observation and Measurement, Sensor Collection Service, Sensor Planning Service and Web Notification Service. As [16] states, the purpose of SWE is to make all types of web-resident sensors, instruments and imaging devices,

as well as repositories of sensor data, discoverable, accessible and, where applicable and controllable via the World Wide Web. In other words, the goal is to enable the creation of web-based sensor networks. Figure 3 demonstrates the architecture and collaboration between components of SWE.

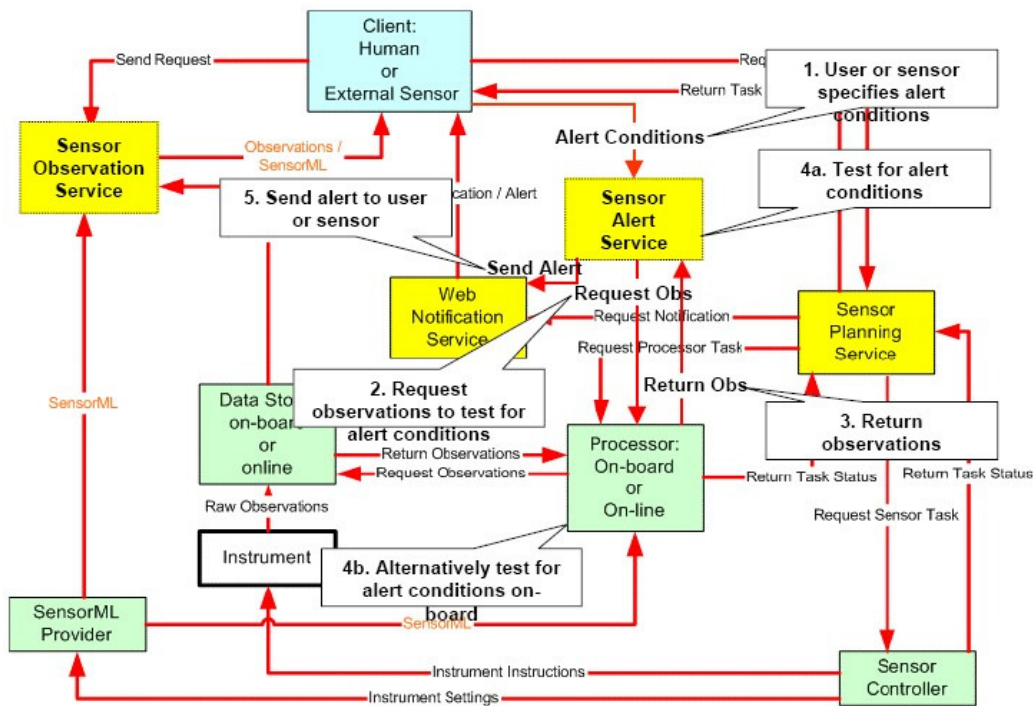


Figure 3 Sensor Web Enablement Architecture [16].

3.2.1 SensorML

Web-enabled sensors provide the technology to achieve rapid access to various kinds of information from the environment. Presenting sensor information in standard formats enables integration, analysis and creation of various data “views” that are more meaningful to the end user and to the computing system which process these information. Moreover, a uniform encoding benefits the integration of heterogeneous sensors and sensor platforms to form a uniform, integrated and standard view to the client. The details of encoding of SensorML are described in [19], which is basically defined over several XML schemas and extends and reuses a lot of elements from another OpenGIS standard GML [20]. SensorML is a key component for enabling autonomous and intelligent sensor webs. It provides the information needed for

discovery of sensors, including sensor's capabilities, geo-location and taskability. Both single sensors and platforms holding numbers of sensors can be described by SensorML.

3.2.2 Observation and Measurement

Besides providing SensorML that contains information about sensors and sensor platforms, SWE defines another standard information model and XML encoding for observations and measurements that are important for Sensor Web to find universal applicability with web-based sensor networks, according to [16]. The detailed conceptual model and encoding for observation and measurements can be found at [21], where the observation has been defined to be an event with a result which is a value describing some phenomenon. An observation extends the GML feature type model, which means it contains the essential properties required by GML feature type such as id, metadataProperty, description, name, location and boundedBy. Observation and measurement model is required specifically for Sensor Collection Service and related components of an OGC Sensor Web Enablement capability.

3.2.3 SWE Services

SWE not only provides the information model and encoding like SensorML and Observation & Measurements, but also defines several standard services that can be used to collaborate with to obtain data from sensor networks. One of the most important one is the Sensor Collection Service that is useful to fetch observations from a sensor or constellation of sensors. As [16] states, it plays a role of intermediary agent between a client and an observation repository or near real-time sensor channel. [22] defines the essential operations needed for the service as well as the required parameters for each operations. Each client who intends to invoke the Sensor Collection Service must strictly follow the standard specified by [22].

There are another two useful services provided by SWE including Sensor Planning Service and Web Notification Service. The Sensor Planning Service focuses on dealing with planning for the collection actions, which determines the feasibility of the collecting request and calls the Sensor Collection Service to collect the observation data if possible. Web Notification Service maintains an asynchronous dialogues between client and one or more other services for long duration processes, which may be quite useful when many collaborating services are required to satisfy a client request.

As the SWE is still evolving, new services will be come out to satisfy other requirements of Sensor Web development. Until recently, a new service called Sensor Alert Service has been introduced, which specifies how alert or “alarm” conditions are defined, detected and made available to interested users. Besides, a new TransducerML[23] has also been defined, which is an XML based specification that describes how to capture and “time tag” sensor data.

Chapter 4 Design and Implementation

4.1 Overview

OSWA defines the big picture of how to build the Sensor Web and how each components should collaborate with each other, it is still a lot of design issues to consider not only including the common problems faced with other distributed system such as security, multithreading, transactions, maintainability, performance, scalability and reusability, but also need some critical decisions to be made about which alternative technologies are best suitable for the project. Figure 4 depicts the

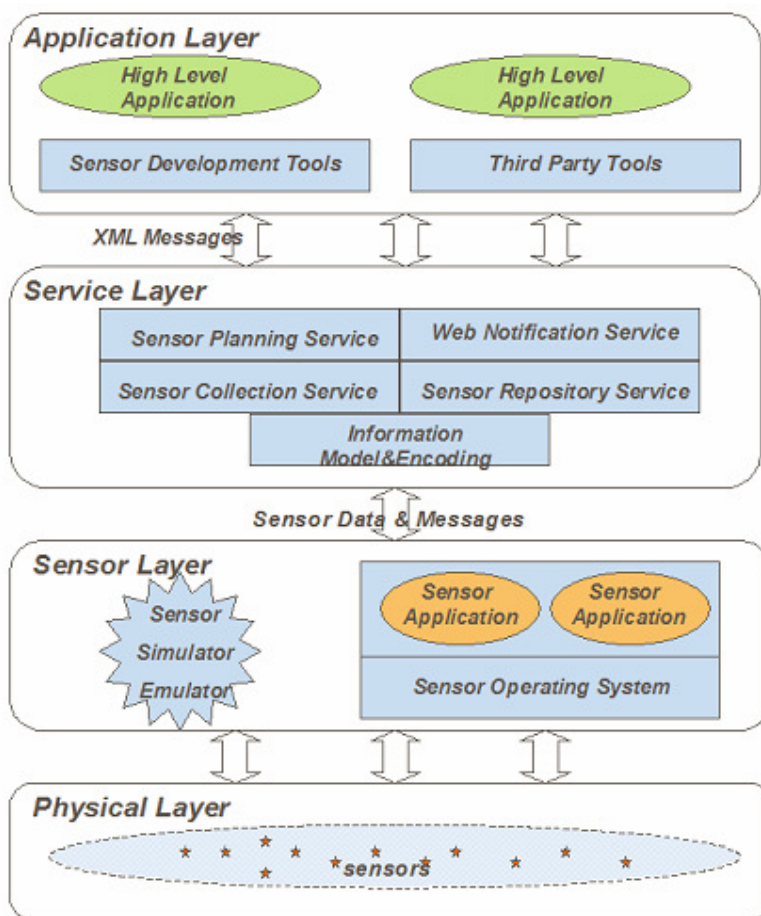


Figure 4 Overview Layered Architecture of OSWA.

layered architecture of the OSWA, this thesis concentrates on the Service Layer and Sensor Layer especially on the design and implementation of Sensor Collection Service, the XML encoding and its communication with the sensors and sensor networks. The following section will describe the key technologies that relevant to different layers of the OSWA.

4.1.1 Service Layer and Service Oriented Architecture

The SOA is the essential infrastructure that supports the Service Layer and plays a very important role to present the core middleware components as services for client to access. The reason why OSWA heavily relies on SOA is quite obvious since SOA is a loose coupling distributed architecture and can make interoperability of the heterogeneous systems possible. Moreover, SOA allows the services to be published, discovered and invoked by each other on the network dynamically. All the services communicate with each other through predefined protocols via messaging mechanism, which supports both synchronous and asynchronous communication model. As the sensor networks basically differ from each other, trying to put different sensors on the web through uniform operations to discover and access them requires the adoption of SOA. There are two famous and mature implementations that confirm to SOA including Jini developed by Sun Microsystems and Web Services promoted by majority of industry vendors. Both of them have their own advantages and disadvantages, which discusses in the following sections.

4.1.1.1 Jini

Jini is a lightweight environment for dynamically discovering and using services on a network. According to [24], resources in Jini system can be implemented as either hardware devices, software program, or a combination of the two, which can also be added and deleted as services. Consequently, Jini enables users to share services and resources over a network and to access them anywhere on the network. As [24] states, services in a Jini system communicate with each other by using a service protocol,

which is a set of predefined interfaces written in Java including a universal lookup service and Java RMI. Jini also support advanced mechanism like leasing, distributed transaction, security and events. Figure 5 illustrates the basic Jini programming model, every client who wants to use a service or resource need to contact lookup service to find the service by its java interface type and possible type attributes, once the service or resource from service provider has joined to Jini previously, lookup service moves a copy of service object to the client. The three components in Jini programming model exactly reflect the roles they play in SOA.

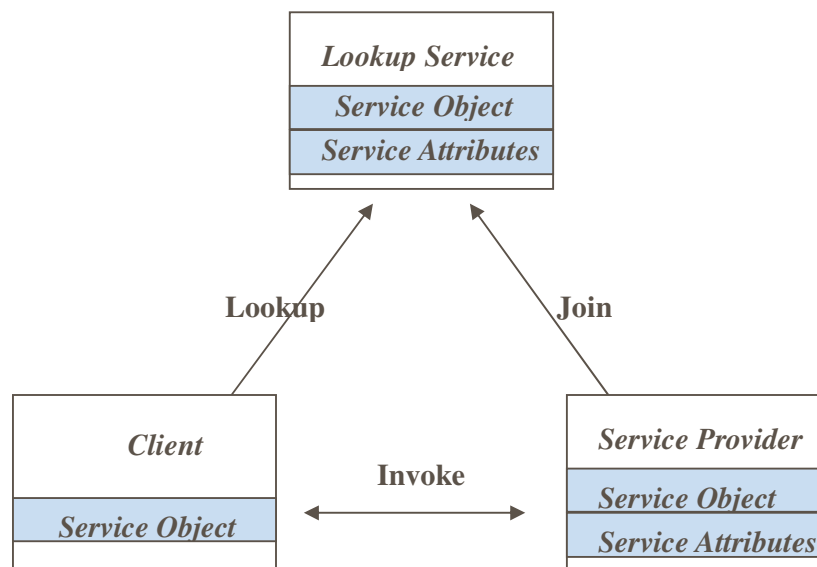


Figure 5 Jini Basic Programming Model.

The biggest advantage of Jini is that it allows the hardware devices plug into the network as software services easily, which means clients can access and control if authorized hardware devices utilizing simple interfaces without being aware of their complexity in Jini environment. Consequently, Jini is best suitable for the applications like Smart Home that require communicating with diversity of appliances such as printer, fax machine, light, alarm, TV, CD, or even refrigerator and microwave. Moreover, Jini introduces the concept of leasing that means every service has to be granted access over a period of time by leasing first, and whenever the leasing is expired, the leased resources are forced to be released. Leasing enhance the capability of self-healing to Jini environment once failures happen on the network.

However, Jini relies heavily on Java Virtual Machine that means every service intending to join the network needs to support Java VM. The restriction largely limits the usage of Jini worldwide, especially for those who are not willing to use Java platforms. Furthermore, Jini uses Java RMI as its standard communication model that has potentially low interoperability and does not inherently support asynchronous communication model that is vital important for most SOA based applications.

4.1.1.2 Web Services

Web Services, technologically, depend on a group of standard specifications including HTTP, XML, Simple Object Application Protocol (SOAP), Universal Description Discovery and Integration (UDDI), Web Services Description Language (WSDL). XML is the key of Web Services technology, which is the standard format of the messages exchanges between services, and moreover all most every specifications used in Web Services are themselves XML data such as SOAP and WSDL. SOAP provides a unique framework that for packaging and transmitting XML messages over variety of network protocols, such as HTTP, FTP, SMTP, RMI/IIOP or proprietary messaging protocol [26]. WSDL describes the operations supported by web services

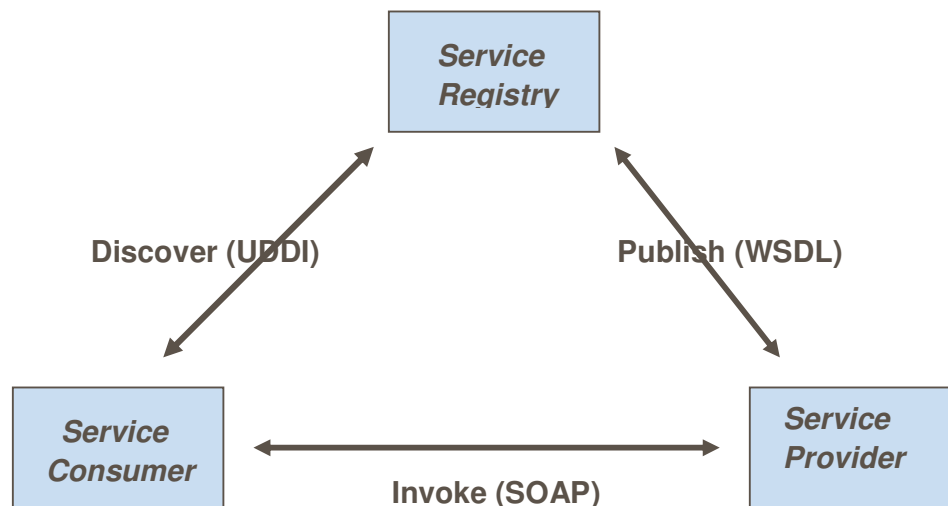


Figure 6 Typical architecture of Web Service.

and the structure of input and output messages required by these operations as well as the important information about web services including definition, support protocol, processing model and address. The architecture for Web Services is showed in Figure

6, it is quite similar with Jini except that the lookup service in Web Services is UDDI, a more complex but powerful model for registry service.

There are several aspects that Web Services is exceeding Jini. First of all, unlike Jini, Web Services is a language and platform neutral technology that can be implemented using any programming language in any platform. For example, a service written in C# can be accessed by a client written in Java. Furthermore, the use of XML and XML-based protocols such as SOAP and WSDL largely improves the interoperability of the applications, as both parties share one services need to conform to the operations and the passing message format defined by WSDL. What is more, the message-oriented approach of Web Services is inherent support both synchronous and asynchronous communication model. In addition, the adoption of SOAP as its standard transport protocol makes it extremely flexible with diversity of network protocols. As OSWA is primarily based on SensorML and other related XML data models, Web Services provide a much better solution compared with Jini in terms of interoperability and flexibility.

4.1.2 Information Model and XML Data Binding

The information model of OSWA is based on Observation&Measurement and SensorML, both of them are built upon XML standard and defined by XML Schema. Transforming data representation of the programming language to XML that conform to XML Schema refers to XML data binding and is a very important and error-prone issue that may affect the performance and reliability of the system. In general, there are two common approaches to solve this problem. The first and obvious way is to build the encoder/decoder directly by hand using the low-level SAX parser or DOM parse-tree API, however doing so is likely to be tedious and error-prone and generating many redundant codes that are quite hard to maintain. A better approach to deal with the transformation is to use XML data binding mechanism that automatically generates the required codes according to DTD or XML Schema. As

[27] states, Data binding approach provides a simple and direct way to use XML in the applications without being aware of the details structure of XML documents, instead working directly with the data content of those documents. Moreover, [27] also points out that Data binding approach makes accessing to data faster since it requires less memory than a document model approach like DOM or JDOM for working with documents in memory.

There are quite a few Java Data binding tools available such as JAXB, Castor, JBind, Quick and Zeus according to [27], and another greater tool Apache XMLBeans [34]. Although JAXB is the reference implementation by Sun, it is not suitable for OSWA as the data model definition of SensorML [19] uses a lot of advanced features of XML Schema that JAXB does not support currently. However, it will be a good choice in the future and definitely play an important role in working with XML and Java technologies, as Sun promises that JAXB 2.0 [33] will 100% support XML Schema and become standard part of J2EE platform. Among those open source tools, XMLBeans seem to be the best choice not only because it provides fully support for XML Schema, but also does it provide extra valuable features like XPath and XQuery supports, which directly enables performing queries in the XML documents.

4.1.3 Sensor Operating System

As OSWA has the ability of dealing with heterogeneous sensor networks that may adopt quite different communication protocols including radio, blue tooth, and ZigBee/IEEE 802.11.4 protocols, it is quite desirable that the operating system level support for sensor networks can largely eliminate the work of developing device drivers and analyzing various protocol stacks directly in order to concentrate on higher-level issues related to the middleware development.

TinyOS [28] is the de-facto standard and very mature Operating System for wireless sensor networks, which consists of a rich set of software components developed by

NesC language, ranging from application level routing logic to low-level network stack. TinyOS provides a set of Java tools in order to communicate with sensor networks via a program called SerialForwarder, which runs as a server on the host machine and forward all the package receiving from sensor networks to the local network, depicted by Figure 7. Once the SerialForwarder program is running, the software located on the host machine can parse the raw package and process the desired information. TinyDB [29] is another useful component built on top of TinyOS, which constructs an illusion of distributed database running on each node of the sensor networks. SQL-like queries including simple and even grouped aggregating queries can be executed over the network to acquire data of sensors on each node.

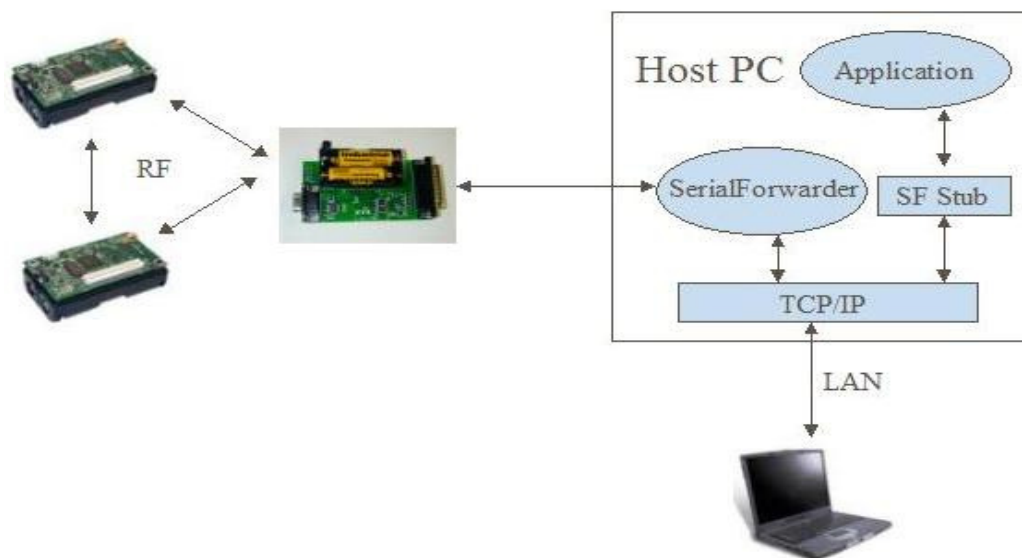


Figure 7 TinyOS SerialForwarder Architecture.

Besides TinyOS, there are other Operating Systems existing as well. MANTIS [30] is a lightweight multithreaded sensor operating system, which supports C API enabling the cross-platform supports and reuse of C library. Moreover, it supports advanced sensor features including multi-model prototyping environment, dynamic reprogramming and remote shell. Contiki [31], which is designed for memory constrained systems, is another event-driven sensor operating system like TinyOS with highly dynamic nature that can be used to multiplex the hardware of a sensor network across multiple applications or even multiple users.

4.1.4 Sensor Data Persistence

Persistence is one of the most important aspects for the purpose of manipulating the huge amount of data that relevant to both sensor observation as well as sensor information. As the standard data format exchanging between services are XML data that conform to O&M and SensorML schema, transformations need to be done between different views of data including XML, Java object and relational database. In order to ease the operation of the transformation, the O/R mapping solution has been adopted to support the data persistence.

Java Data Objects (JDO) is one of the most popular O/R mapping solutions, which defines a high-level API that allows applications to store Java objects in a transactional data store by following defined standards. It supports transactions, queries, and the management of an object cache. JDO provides for transparent persistence for Java objects with an API that is independent of the underlying data store. JDO allows you to very easily store regular Java classes. JDOQL is used to query the database, which uses a Java-like syntax that can quickly be adopted by those familiar with Java. Together these features improve developer productivity and no transformation codes need to be developed manually as JDO has done that complicated part underneath. To make use of JDO, the JDO Metadata is needed to describe persistence-capable classes. The information included is used at both enhancement time and runtime. The metadata associated with each persistence-capable class must be contained within an XML file. In order to allow the JDO runtime to access it, the JDO metadata files must be located at paths that can be accessed by Java classloader.

4.2 Services Architecture

4.2.1 Sensor Collection Service

Sensor Collection Service is one of the most important component resides in the OSWA core middleware layer. As can be seen in Figure 1, Sensor Collection Service is the fundamental and unique component that needs to communicate directly with sensor networks, which is responsible for collecting real time sensing data and then translating the raw information into the XML encoding for other services to utilize and process. In other words, Sensor Collection Service is the key entering into the sensor networks from outside clients and its design and implementation will affect the entire OSWA. The design of Sensor Collection Service provides interface to both streaming data and query based sensor applications that are built on top of TinyOS and TinyDB respectively.

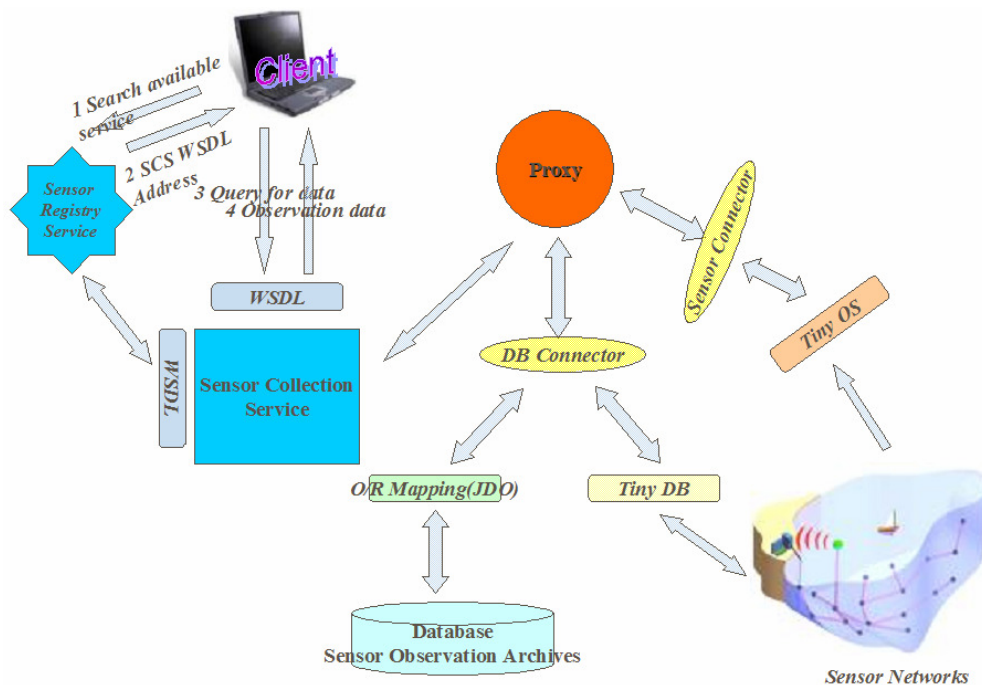


Figure 8 Sensor Collection Service Architecture.

Figure 8 illustrates the architecture of the Sensor Collection Service, it is conformed to the interface definition that is described by [22] and has been designed as a web service that work with a proxy connecting to either real sensor or remote database.

Clients need to enquire the Sensor Registry Service about available SCS WSDL address according to their requirements and send data query via SOAP messages to the SCS in order to obtain the observation data conformed to the XML schema defined by Observation & Measurement [21]. The proxy acts as an agent working with various connectors that connect to the resources holding the information, and encoding the raw observation to O&M compatible data. Different type of connectors have been designed to fit into different kind of resources including sensor networks running on top of TinyOS or TinyDB and remote observation data archives. The proxy needs to process the incoming messages from client to determine what kind of connectors either real-time based or archive based to use.

The design of the SCS is quite flexible and makes it quite easy to extend for further development if different sensor OS are adopted by the sensor networks such as MANTIS or Contiki. The only work is to implement a specific connector in order to connect to those resources and no modifications need to be made in the current system. The design of the proxy also encourages the implementation of cache mechanism to improve the scalability and performance of the SCS. Load balancing mechanism can be added to the system easily as well by simply deploying the web service to different servers.

4.2.2 Planning and Notification Service

Sensor Planning Service and Web Notification Service are another two very important services that are usually working together. Planning Service is the one that actually communicate with the end user and invoke the notification service as well as the collection service. Web Notification service provide an asynchronous way to communication with the end users.

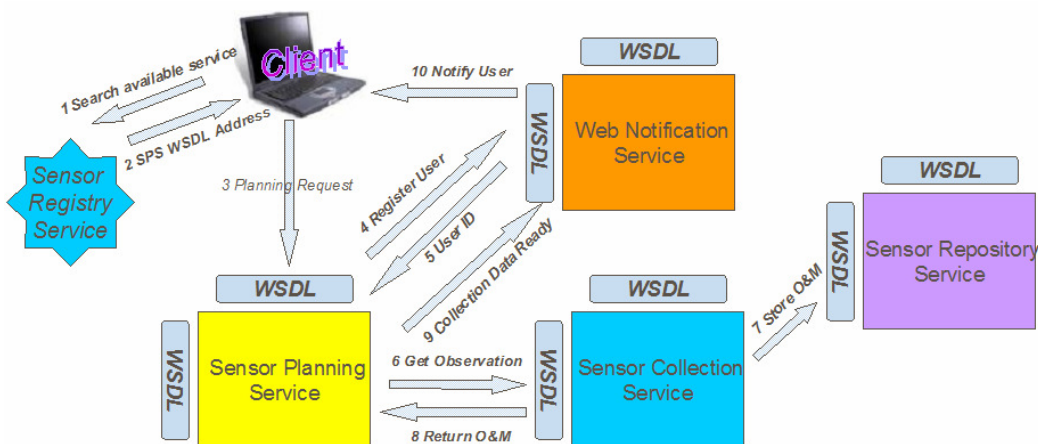


Figure 9 SPS, WNS and SCS Services Architecture.

Figure 9 illustrates the architecture of relating services. Once the end user make an observation plan to the Planning Service, it checks the feasibility of the plan and submit the plan if it is feasible. The user will be registered in the Web Notification Service during this process and the user id will return to SPS. SPS is responsible to create the observation request according to user's plan and get the O&M data from the Sensor Collection Service. As soon as the O&M data has been ready, SPS will send an operation complete message to the WNS along with the user id and task id, WNS will notify the end user to collect the data via email or other protocols it supports.

4.3 Implementation

4.3.1 Sensor Collection Service

Sensor Collection Service is the most important as well as complicated service to implement. It is not only responsible for connecting to various resources to collect their observation data, but also does it parse and format those data into standard O&M format. Our goal is to make the SCS quite flexible and easy to extend for different sensor applications and heterogeneous resources they may choose to use in their applications. There are four important components: Proxy, Connector and Data Handler and Data Formatter in the SCS implementation. Developers can extend the Connector, DataHandler and DataFormatter interfaces to fulfill their specific

application requirements. The following section will discuss these components in details.

4.3.1.1 Core Sensor Collection Service Components

As demonstrated by Figure 10, the SensorCollectionService interface defines the basic operation `getObservation` that is the most important one that actually fetch the observation data to the O&M format. The parameter passing into `getObservation` needs to conform to the XML Schema definition in [22].

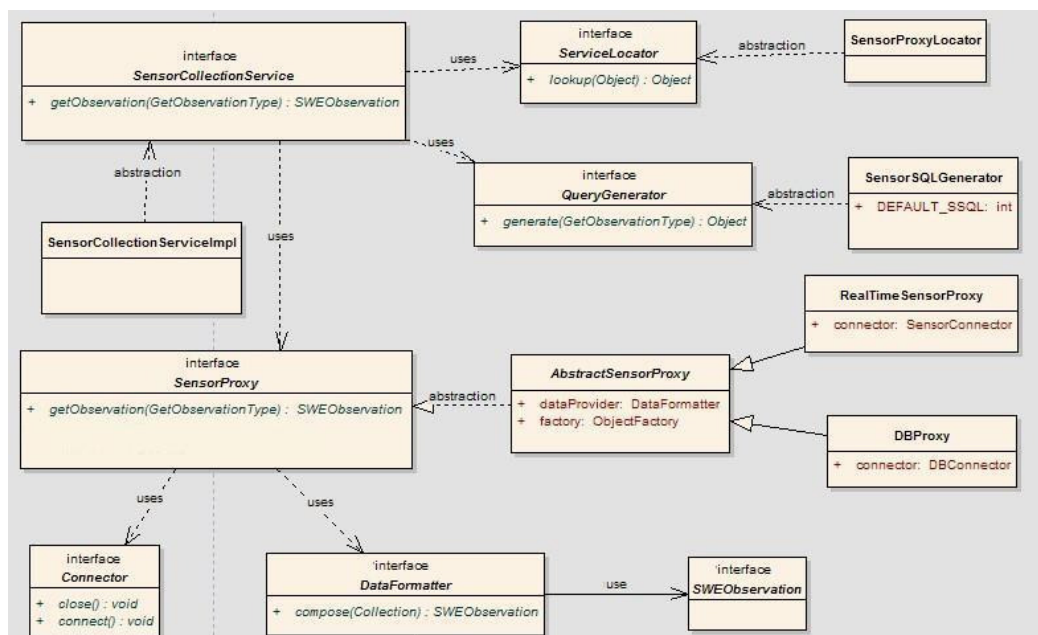


Figure 10 Sensor Collection Service Main Class Diagrams.

Below is an example of this standard parameter to query the sensor whose sensing temperature is larger than 500.

```
<GetObservation xmlns="http://www.opengis.net/scs"
  version="0.1" service="OGC Sensor Collection Service"
  outputFormat="SWEObservation">
  <BoundingBox />
  <Query xmlns="http://www.opengis.net/wrs"
    typeName="TinyDB Observation Example">
    <PropertyName xmlns="http://www.opengis.net/ogc">
      SWEObservation
    </PropertyName>
    <QueryConstraint>
      <Filter xmlns="http://www.opengis.net/ogc">
        <PropertyIsGreaterThan>
```

```

    <PropertyName>temp</PropertyName>
    <Literal>500</Literal>
  </PropertyIsGreaterThan>
</Filter>
</QueryConstraint>
</Query>
</GetObservation>

```

Currently, the implementation supports four kinds of condition rules including less than, larger than, equals and not equals. As getObservation is the key operation of SCS, let us take a close look at the sequential diagram of getObservation showing in Figure 11 and Figure 12 to better understand how it works in both Client and Server side.

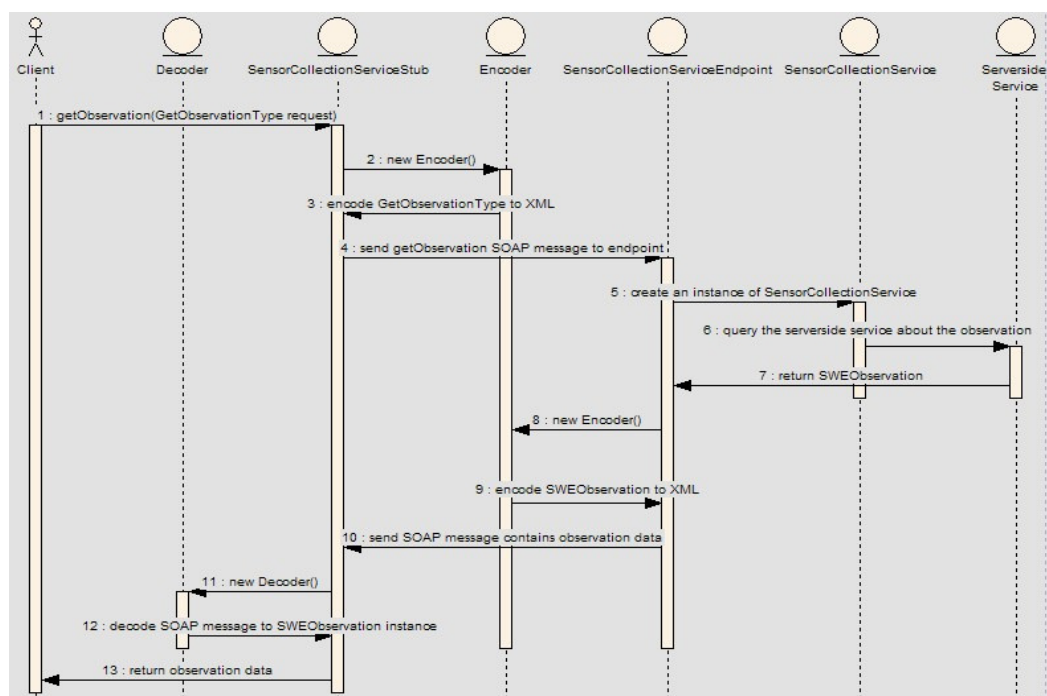


Figure 11 getObservation operation sequential diagram: Client Side.

The client, who intends to get the observation for some purpose, contacts the SensorCollectionServiceStub, which is a stub object running on client side to send and receive SOAP messages to a SensorCollectionServiceEndpoint object which simply delegate the call to SensorCollectionService on the server side via HTTP. The response of the client's request will be a SWEObservation instance that is a Java Object that stands for the O&M XML data from server.

What happens inside server is far more complicated compared with the call on client side. Once receiving a request, SensorCollectionService translates the request data into a query that is generated by QueryGenerator. For example, the previous request XML data will be translated to a query string like “SELECT * FROM Sensors WHERE temp>500”. Then SensorCollectionService looks up the available proxy according to timestamp client required by asking RepositoryService that maintains a table of available proxy instances, either RealTimeProxy or DBProxy.is returned to the SensorCollectionService. As soon as the proxy is available, queries will be sent to the Connector that has registered message listeners and data handlers to process the incoming messages from the real source either sensors or database. All the raw observation data need to be formatted into the required O&M format, which is done by DataFormatter. Before sending the O&M data back to the client, it will be pushed into the repository for further queries.

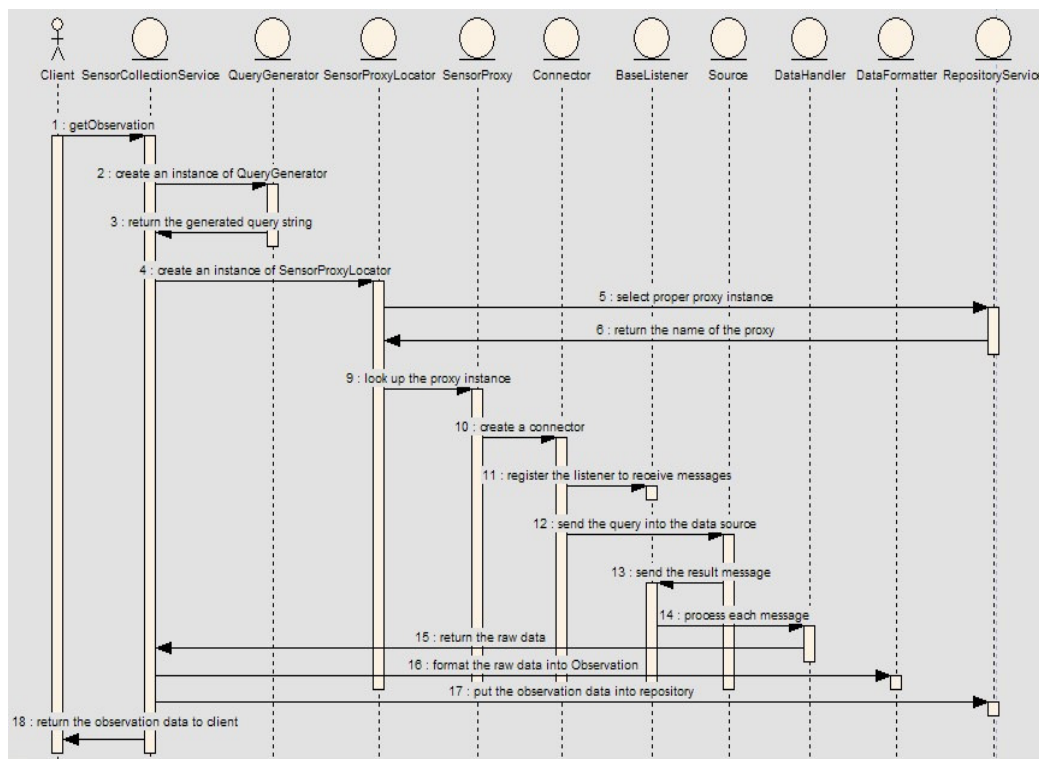


Figure 12 getObservation operation sequential diagram: Server Side.

4.3.1.2 Connector Component

The most important component inside SCS is the Connector, which provides the

gateway to the heterogeneous resources. There are various implementations of connector that each is responsible for connecting to a specific type of resource. TinyDBConnector and SerialBasedSensorConnector are two implementations of the connector showing in Figure 13. Both of them utilize the Java interface called MoteIF provided by TinyOS to listen to the serial server on “[sf@<host-ip>:<port-no>](#)” created by SerialForwarder [35].

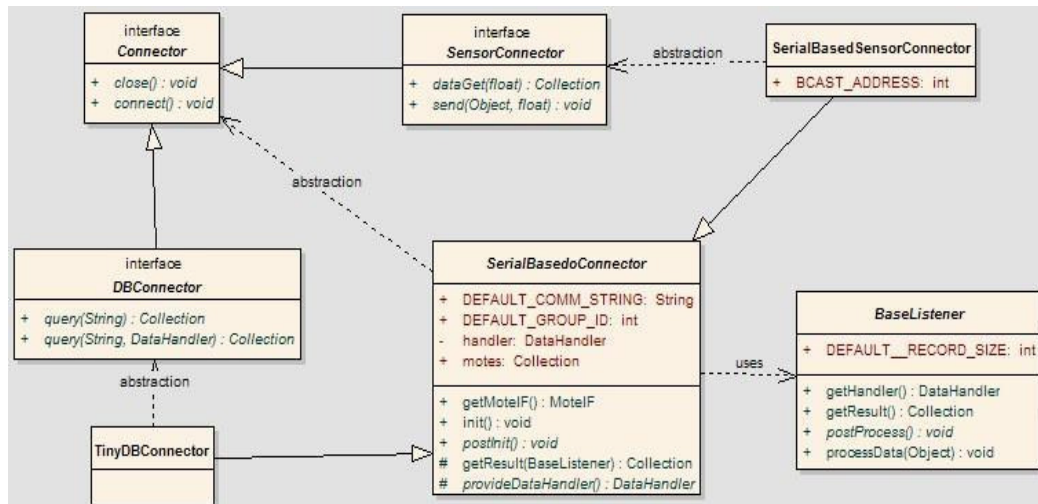


Figure 13 Connector Component Class Diagrams.

As soon as the messages coming from resources arrives in asynchronous way to the listener, a specific data handler for different kinds of application processes each message according to its requirement, and when the number of processed messages reaches the predefined number about how many messages need to handle, the listener will notify the connector to collect the result by setting a dataReady tag. The listener forwards the data to DataHandler to process rather than dealing with the data itself.

4.3.1.3 Data Handler Component

DataHandler is the component that actually processes the data received from the resources. Every connector needs to register a listener and a data handler in order to receive the data from resources and process them, as illustrated in Figure 14. As every application has its unique features of the data it wants to provide, the developer needs to implement their own concrete DataHandler class in order to process their specific requirements of the messages. Two concrete implementations of DataHandler have

been provided which process the message collecting from TinyDB and the Surge application [35] respectively.

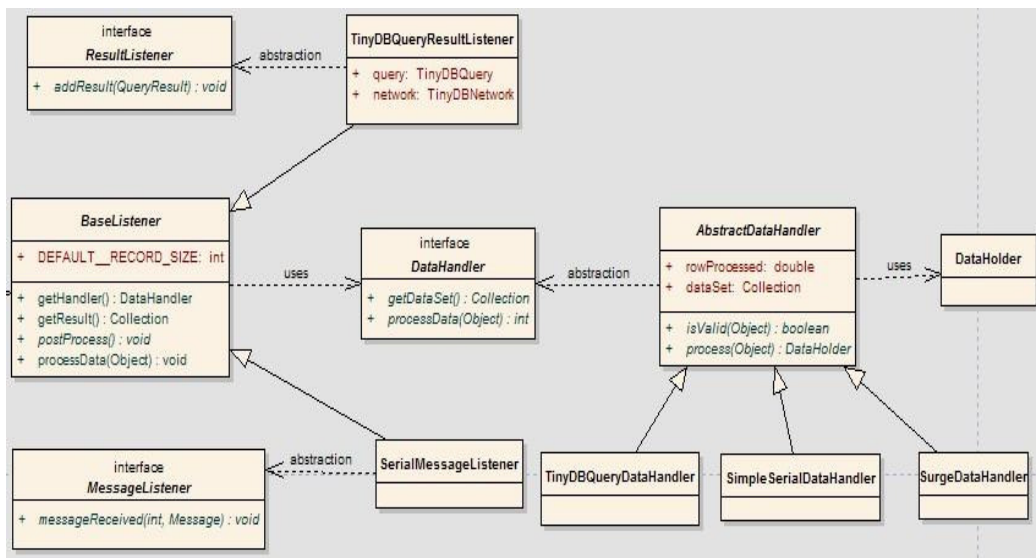


Figure 14 Data Handler Components Class Diagrams.

4.3.1.4 Data Format Component

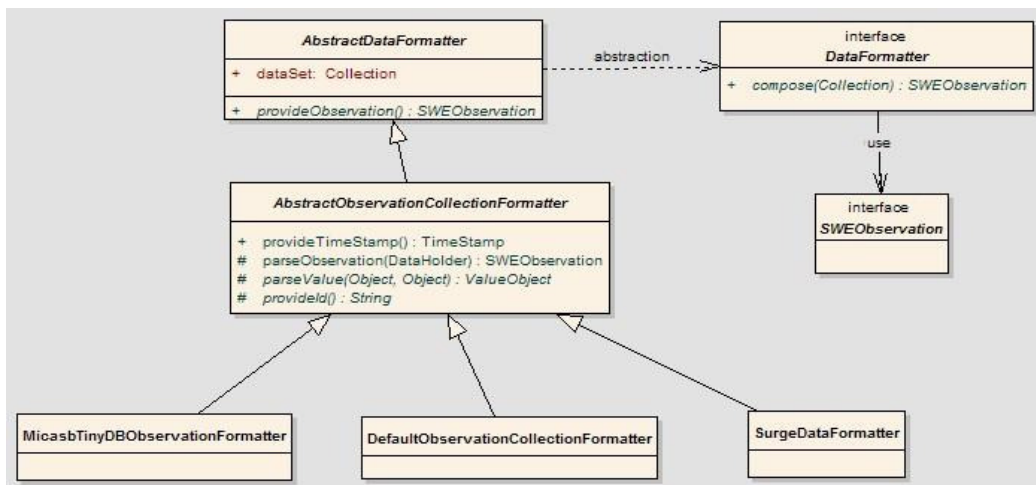


Figure 15 Data Formatter Component Class Diagrams.

As the SCS client needs the standard O&M data format not just the raw observation data, it is essential to convert the raw observation data into the O&M data. The DataFormatter component demonstrated in Figure 15 is the key to this issue. It is quite understandable that each sensor application concentrates on different kinds of information; therefore, there are various implementations of DataFormatter to match

each specific application and define its own dictionary and measurement for the targeting data. MicasbTinyDBObservationFormatter and SurgeDataFormatter are two of those DataFormatters, which deal with the TinyDB application and Surge application [35] respectively.

4.3.1.5 Configuration

The Sensor Collection Service is designed to be highly adaptable and pluggable, and different service providers of the SCS may require having their own configurations. A configuration file named application.properties is available to accomplish this task along with a class called ObjectFactory who can create object and get attributes from the configuration file. Instances of Proxy, Connector, DataHandler and DataFormatter are created via calling ObjectFactory's createObject method, which looks up the concrete class name defined in the configuration file. Application specific settings can also be provided in the configuration file. Below is an example configuration file residing in the host machine running TinyDB application in TOSSIM [36] simulation environment. All supported configuration keys are defined in GlobalConstant.java.

```
#new setting for query generator
org.sensorweb.core.generator.QueryGenerator=
    org.sensorweb.core.scs.SensorSQLGenerator
#class for sensor proxy for tinydb applications
org.sensorweb.core.sensor.Proxy=
    org.sensorweb.core.scs.DBProxy
#class for db connector
org.sensorweb.core.db.Connector=
    org.sensorweb.core.scs.tinyos.tinydb.TinyDBConnector
#class for data formatter of tinydb application
org.sensorweb.core.DataFormatter=
    org.sensorweb.core.scs.tinyos.tinydb.MicasbTinyDBObservationFormatter

#note settings for simulation communication connection
serial.comm-string=tossim-serial
#default group-id for tinydb application
serial.group-id=125
#tinydb setting
tinydb.catalog=D:/jakarta-tomcat-5.0/webapps/SensorWeb/WEB-INF/catalog.xml
#application settings
record.number=5
```

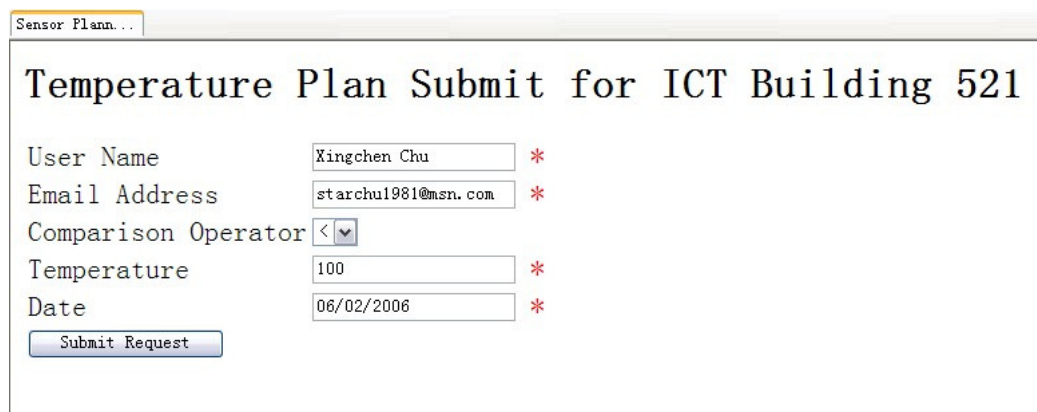
4.3.2 Sensor Planning Service

The Sensor Planning Service implementation currently only supports two standard methods of the OGC SPS specification, they are *getFeasibility* and *submitRequest*. Both of them accept a request collection that is defined to take a simple planning for monitoring different range of temperature. The SPS will be the service that communicates directly with the SensorWeb Clients who wants to collect some data.

The GetFeasibility method will check the client's request according to the rule defined to support the valid range of temperature to monitor. If the request is feasible, the GetFeasibility method will then send request to the WNS to register the user in order to receive any notification. The response of the GetFeasibility method will be a Boolean value to indicate whether the request is feasible and a user id. Once the request is feasible, the client can then submit their request form to SPS by invoking the submitRequest method defined in SPS. This method deals with the request and generates the query, which is conformed to the SCS getObservation input parameter schema, according to the request, and then asks the SCS to get the o&m information. If the getObservation succeeds, SPS will notify WNS OPERATION_COMPLETE, otherwise an OPERATION_FAILED notification will be sent. The response of the submitRequest will be a task id for clients to get the observation data. Asynchronous communication has been used in SPS, once the user submit the request, it will immediately get the response task id and clients will be notified by the WNS if the data is ready to collect.

As Figure 16 shows, the SPS will ask the users to fill a request form with required information related to user as well as their plan. We implement a very simple plan scenario that allows the users to adjust the temperature they want to monitor., the users need to specify the operator as well as the base temperature in the plan along with their names and contact details in order to receive the notification. Once users click the submit button, the request will be processed at the backend and users will the

notification as soon as the request has been done.



Sensor Plann...

Temperature Plan Submit for ICT Building 521

User Name	<input type="text" value="Xingchen Chu"/>	*
Email Address	<input type="text" value="starchu1981@msn.com"/>	*
Comparison Operator	<input type="text" value="<"/>	
Temperature	<input type="text" value="100"/>	*
Date	<input type="text" value="06/02/2006"/>	*

Figure 16 Simple Temperature Planning Request Form.

4.3.3 Web Notification Service

The WNS implementation supports two basic methods that defined in OGC WNS specification, which are *registerUser* and *doNotification*. Also in order to manager the user account, a JDBC-based *UserAccountManager* has been implemented to maintain the user account. Besides, a *CommunicationProtocol* interface has also been provided to support various protocol defined in WNS. Currently, only email protocol has been implemented that can be used to notify clients by emails.

The *registerUser* method accepts the name and contact of the client, and save them in the database through the *JDBCUserAccountManager.saveUser* method. It also request the *IDGenerator* to generate a *BigInteger* value as the user id. And this id will be returned to client. The *doNotification* method process the incoming message that contains the user id and it find the contact details of the user according to the id and then set the proper communication protocol for the client. And it will send the message to the user through the specific protocol. In our case, the protocol is to send message by email. And the email will tell the client the URL to collect the data according to a key composed by user id and task id. The WNS is mainly called by the SPS and won't accept request directly from the end user. It just sends notifications to the client if the SPS asking it to do it by invoking *doNotification* method, such as the

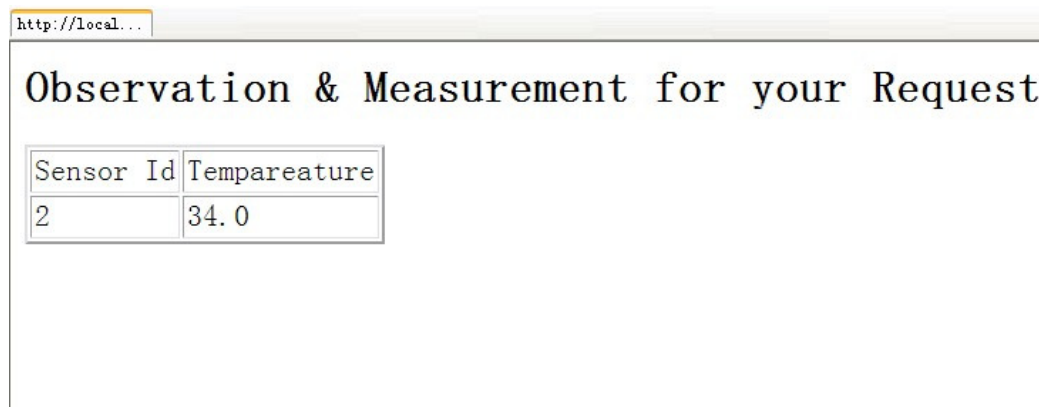
way the submitRequest method does. Figure 17 demonstrates the notification email received by the users to indicate the information they need to get the observation data.

```
发件人 : <notification-admin@unimelb.edu.au>
发送 : 2006年1月6日 13:58:08
收件人 : starchu1981@msn.com
主题 : SensorWeb Client Notification

-----Notification Received-----
Status : Operation completed
Information
  Data is Ready
  You can collect the data at
  http://localhost:8180/SensorWeb/DataCollector?key=1002-1003
```

Figure 17 Notification Email to tell user to collect data.

Once the user clicks the URL, the observation data will be parsed and showed by a DataCollector servlet, as Figure 18 shows. Also extension can be easily made by implementing the DataCollector interface to allow getting the raw XML data for further processing.



The screenshot shows a web browser window with the address bar containing 'http://local...'. The main content area displays the heading 'Observation & Measurement for your Request' in a serif font. Below the heading is a table with two columns: 'Sensor Id' and 'Temperature'. The table contains one row of data: '2' under 'Sensor Id' and '34.0' under 'Temperature'.

Sensor Id	Temperature
2	34.0

Figure 18 Observation Data demonstrated by the DataCollector servlet.

4.3.4 Sensor Repository Service

Repository of sensor observation data as well as sensor information is quite important for other services like collection service that may manipulate data over the repository as well as querying it. Just like other services, Sensor Repository Service is deployed as a web service that can be accessed via SOAP messages. The primary idea to implement the repository as web services is trying to make the sensor repository

globally accessible that better supports other services to access it.

There are two important sets of operations that are allowed in the Sensor Repository Service including data manipulation and data query operations. Currently, the service only support save and query operations, which can write the O&M data into the repository and get the O&M data according to the query passing to the operation. JDO and JPOX 1.0 as its reference implementation have been adopted to implement the operation, which has created 72 tables required for storing all O&M data. To make those objects accessible to JDO runtime, as showed by Figure 19, a metadata file is required for each object to describe the mapping between the object fields and database fields, and each class needs to be enhanced by the JDO runtime.

```
<?xml version="1.0"?>
<!DOCTYPE jdo PUBLIC
  "-//Sun Microsystems, Inc.//DTD Java Data Objects Metadata 2.0//EN"
  "http://java.sun.com/dtd/jdo_2_0.dtd">
<jdo>
  <package name="org.sensorweb.model.gml.base">
    <class name="AbstractGMLType" identity-type="application" requires-extent="tru
      <inheritance strategy="new-table"/>
      <field name="id" primary-key="true" persistence-modifier="persistent"/>
      <field name="description" persistence-modifier="persistent"/>
      <field name="names" persistence-modifier="persistent">
        <collection element-type="java.util.Collection">
          </collection>
        </field>
      <field name="metadataProperties" persistence-modifier="persistent">
        <collection element-type="java.util.Collection">
          </collection>
        </field>
    </class>
  </package>
</jdo>
```

Figure 19 An example of metadata file.

As the SOAP messages are O&M XML data, they cannot be directly stored into the repository, it is necessary to decode the O&M data into enhanced java objects that are able to make persistence by JDO and vice verse. Furthermore, the queries passed to the query operation are XML data that follows the same schema as the parameter passed to the getObservation operation of Collection Service. A JDOQueryGenerator class has been provided to support the translation between the query parameter to the JDOQL, which can be used to query objects by JDO. Although, O&M data can be stored and queried via the Sensor Repository Service, it currently does not support the manipulation of SensorML data that means that the discovery and registry of sensors are not available.

Chapter 5 Evaluation

5.1 Test Platform

The test platform for the prototype system is built upon TOSSIM [36] and Crossbow's MOTE-KIT4x0 MICA2 Basic Kit [37] that consists of 3 Mica2 Radio board, 2 MTS300 Sensor Boards, a MIB510 programming and serial interface board. Figure 20 demonstrates the deployment diagram of the entire system. The Sensor Collection Service has been deployed on Apache Tomcat 5.0 on two different machines that run TinyDB application under TOSSIM and Temperature Monitoring Application under Crossbow's motes respectively. Meanwhile, a Sensor Registry Service is also configured on a separate machine that provides the functionality to access sensor repository.

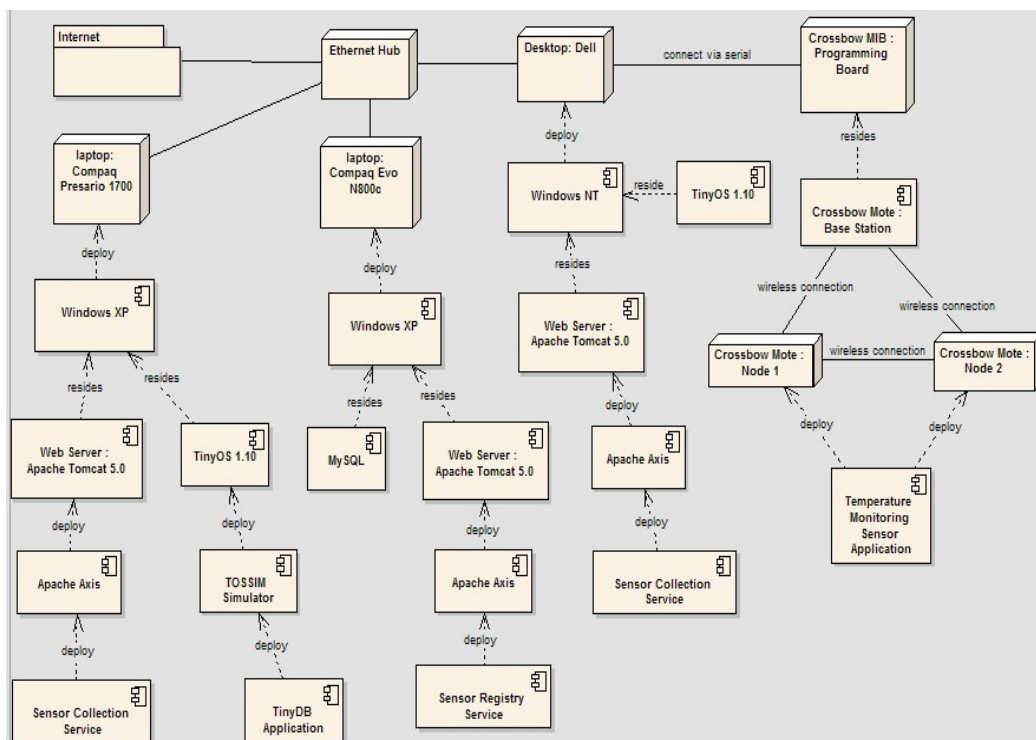


Figure 20 Deployment Diagram of Prototype system.

TOSSIM [36] is a discrete event simulator that can simulate thousands of motes running complete sensor applications and allow a wide range of experimentation. Moreover, a GUI called TinyViz providing detailed visualization and actuation of a running simulation has been developed in order to allow developers to easily transition between running an application on motes and in simulation. To make use of TOSSIM, the *serial.comm-string* property in the configuration file needs to be set as *tossim-serial* in order to tell the connector to creating the MoteIF listening the server on *sf@tossim-serial*.

5.2 Temperature Monitoring Application

A simple temperature monitoring application is also developed in order to test the SCS. The application is programmed using NesC [35] and the logic is really simple, which just simply broadcasts the sensing temperature, light and node address to the sensor network over a fixed period of time. The simple application does not consider any multi-hop routing and energy saving mechanism, more complicated application has also planned to use such as Surge application. Before installing the application to the Crossbow's mote, the functionality can be verified via the TOSSIM simulator. Figure 21 simply illustrates the application running in TOSSIM.

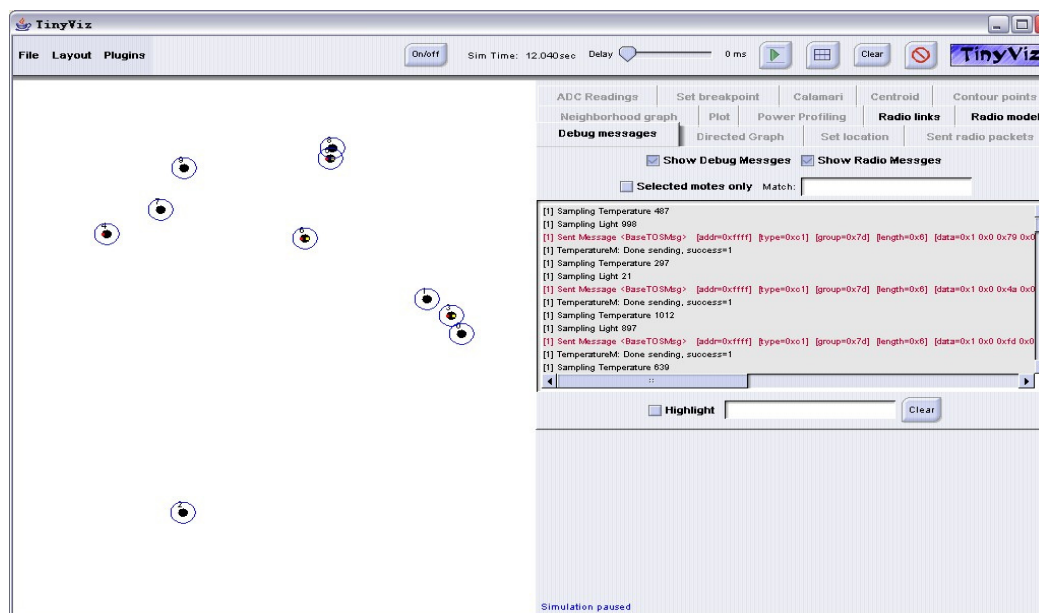


Figure 21 Simulation of Temperature Monitoring Application.

The code snippet below illustrates the core part of the application.

```

event result_t Timer.fired()
{
    call Temperature.getData();
    return SUCCESS;
}
task void sendTask() {
    atomic {
        pack.address = TOS_LOCAL_ADDRESS;
        *((struct TempMsg*)msg.data) = pack;
    }
    if (call SendMsg.send(TOS_BCAST_ADDR, sizeof(struct TempMsg),&msg))
    {
        call Leds.yellowToggle();
    }
}
async event result_t Temperature.dataReady(uint16_t thisData) {
    atomic {
        pack.temperature = thisData;
    }
    call Light.getData();
    return SUCCESS;
}
async event result_t Light.dataReady(uint16_t thisData) {
    atomic {
        pack.light = thisData;
    }
    post sendTask();
    return SUCCESS;
}

```

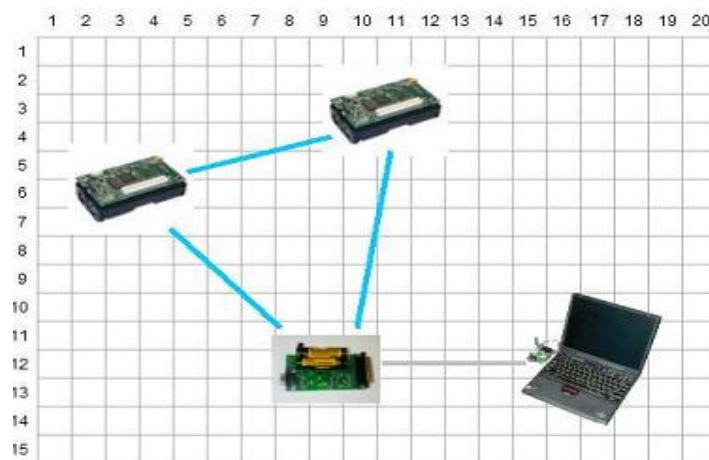


Figure 22 – Network Topology of the Temperature Application

Once the application has been successfully installed into each mote via the programming board, a wireless sensor network has been built with two nodes, and one base station connecting to the host machine via the serial cable. Figure 22 demonstrates the topology of the temperature application. Besides installing the application itself, the SerialForwarder program also needs to run on the host machine in order to forward the data from the sensor network to the server. The *serial.comm-string* in the configuration file must be set to the same value as the server name of SerialForwarder as depicted by Figure 23.

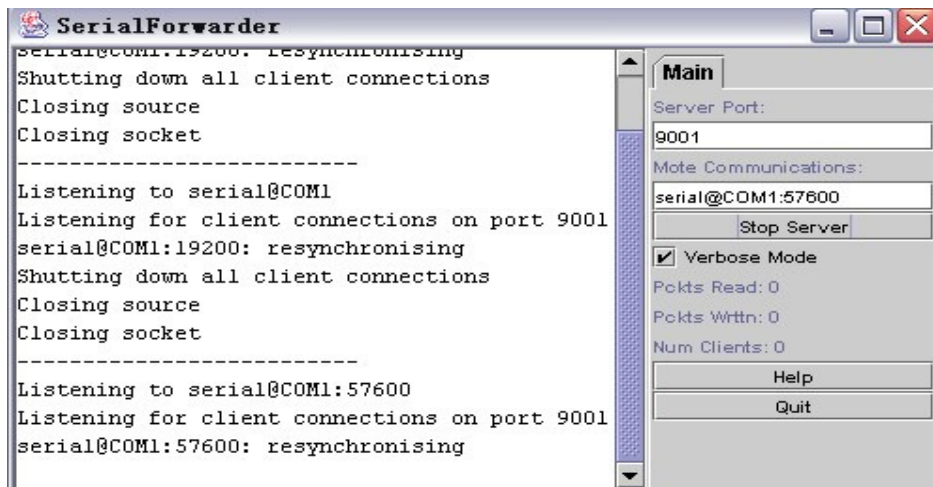


Figure 23 SerialForwarder Program GUI.

5.3 Empirical Result

5.3.1 Client side

Both the TinyDB under TOSSIM and Temperature Monitoring Application have been tested using the same test client. The result for TinyDB application is demonstrated in Figure 24 and Figure 25.

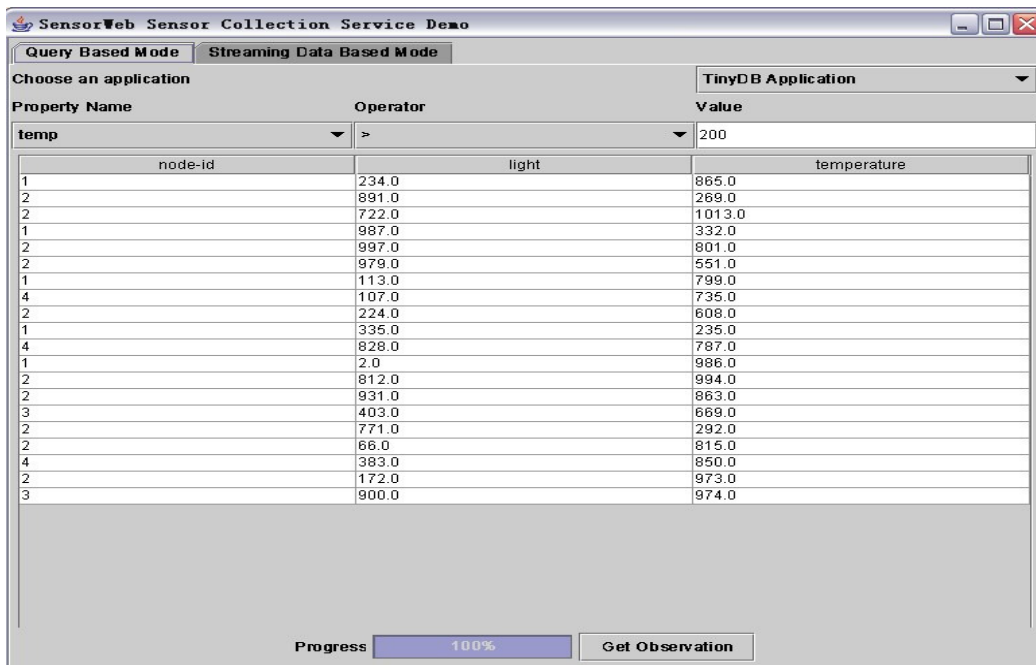


Figure 24 Test Result for TinyDB application under TOSSIM.

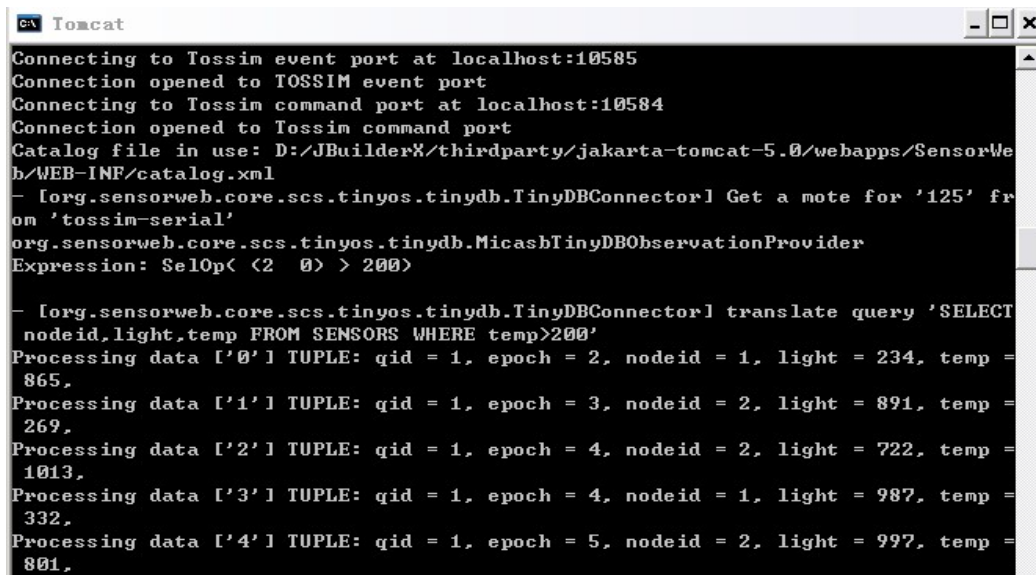


Figure 25 Output Information under Tomcat 5.0 Server.

5.3.2 Performance

Performance test has been concentrated on the SCS, as it plays the critical role and most heavily load service in the entire system. The performance measured by second for both auto-sending and query-based application running on top of TinyOS are

demonstrated by Figure 26 and 27.

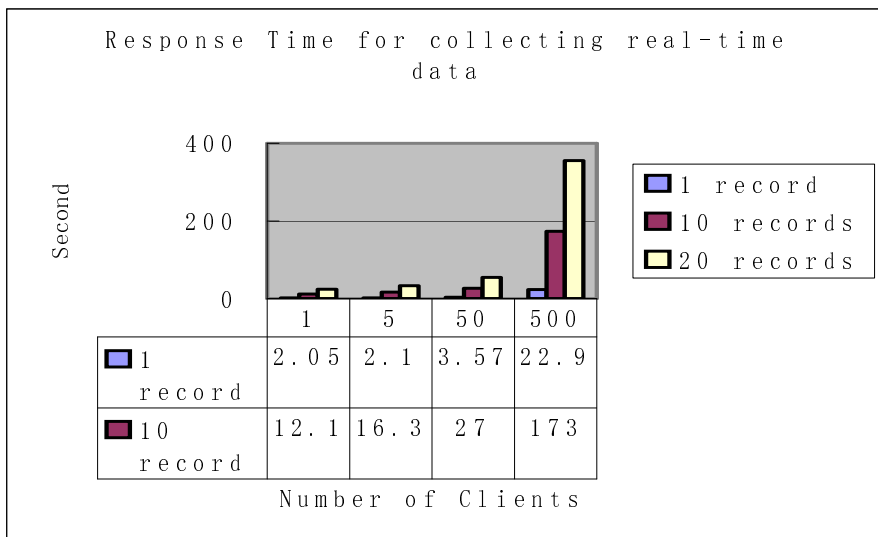


Figure 26 Performance for collecting auto-sending data.

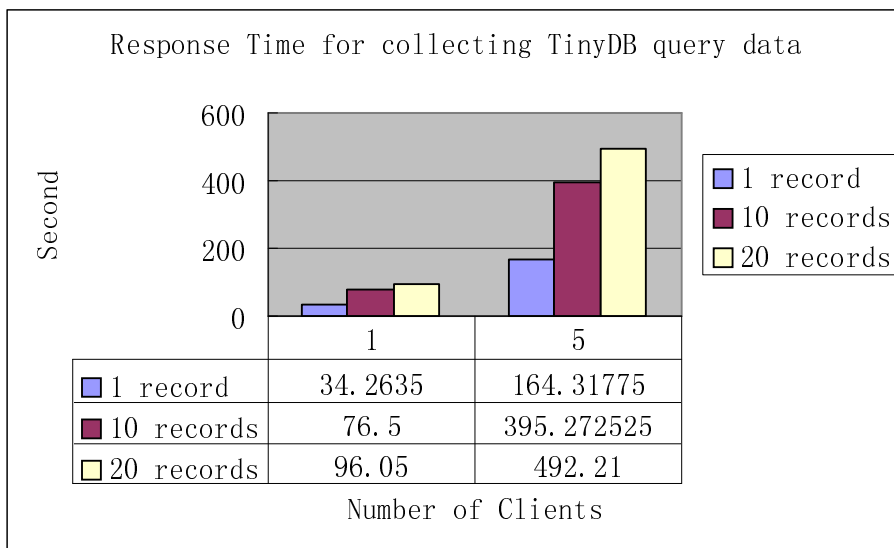


Figure 27 Performance for collecting TinyDB query data.

As can be seen from Figure 26, the result for the auto-sending mode application is quite moderate when the number of clients who request the observation simultaneity is quite small. Even the number of clients reaches 500, the response time for small number of records is also acceptable. In contrast, the result showed in Figure 27 is fairly unacceptable even just one client requesting one observation takes 34 second. The response time increases near linearly when the number of clients and the number of records go up. The reason why the query-based approach has very pool

performance is due to the execution mechanism of TinyDB. A lot of time has been spent on initializing each motes and the application can only execute one query at one time, which means another query needs to wait until current query has been stopped or terminated. A solution to this problem may require the TinyDB application run a generic query for all clients, and the more specific query can be executed in-memory according to the observation data collecting from the generic query.

There are several possible ways to enhance the performance of the Sensor Collection Service. Cache mechanism may be one of the possible approach, the recent collected observation data can be cached in the proxy for a given period of time and the clients who request the same set of observation data can be read the observation data from the cache. However, as the data should be remained as real time as possible, it is quite hard to determine the period of time for the cache to be valid. It may be decided according to the dynamic feature of the information the application is targeting. For example, the temperature for a specific area may not change dynamically by minutes or by hours. Consequently, the period of time setting for the cache can differ from sensor application itself based on the information the sensor targeting. Another enhancement of query performance may be achieved by utilizing the query mechanism such as XQuery of the XML data directly other than asking the real sensor itself executing the query like TinyDB application.

Chapter 6 Conclusion and Future Work

In this thesis, we have surveyed the related works about the sensor applications, sensor middleware, sensor grid integration as well as sensor web to show the importance of those work and the challenges it has to face with. Then, we have discussed the proposed Open Sensor Web Architecture (OSWA) that aims at providing an integration platform to discover, access heterogeneous sensor networks and then process the information collected from those resources accompanying with Grid Technology. Moreover, a service-oriented prototype of Sensor Collection Service, Sensor Planning Service, Web Notification Service and Sensor Repository Service, have been designed and implemented targeting the sensor applications running on top of TinyOS.

However, we are still in the earlier stage of having the entire OSWA implemented and even those services that we have implemented are not a fully functioned. Sensor Collection Service is the key component of the entire OSWA, which affects the whole performance and reliability of the system. A lot of issues remain to future investigation focusing on aspects of reliability, performance optimization and scalability. There are a couple of efforts that are needed to make to enhance the SCS and other services in the following stage.

- The query mechanism for the Sensor Collection Service will be enhanced to support in-memory XML document query. XPath and XQuery technologies are planning to be adopted, as they are the standard way to query XML document. The outcome of this enhancement is expected to improve the performance by moving the heavy workload of queries from sensor network itself to the host machine.
- Caching mechanism may be implemented and placed into the Proxy to further

enhance the performance and scalability.

- Publish/Subscribe messaging mechanism needs to be included as the alternative communication model coexistence with the synchronous approach in the SCS.
- Other methods that described in the specifications of SWE services but are not available currently need to be implemented.
- Other notification protocols needs to be built for the WNS in the future.
- Sensor Registry via SensorML needs to be developed in order to support the worldwide sensor registry and discovery.

Reference

1. Sensor Networks Homepage, <http://www.sensornetworks.com.au/index.html>.
2. Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. Wireless sensor networks for habitat monitoring, In *ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, Atlanta, GA, September 2002.
3. A HYBRID SENSOR NETWORK FOR CANE-TOAD MONITORING, <http://www.cse.unsw.edu.au/~sensar/research/projects/cane-toads/>.
4. Soil Moisture Monitoring with Wireless Sensor Networks project Homepage, <http://www.csse.uwa.edu.au/adhocnets/WSNgroup/soil-water-proj/>.
5. W. Hu, V.N. Tran, N. Bulusu, C.T. Chou, S. Jha and A. Taylor. "The Design and Evaluation of a Hybrid Sensor Network For Cane-toad Monitoring", In *Proceedings of Information Processing in Sensor Networks (IPSN2005/SPOTS 2005)*, Los Angeles, CA, April 2005.
6. R. Cardell-Oliver, K. Smettern, M. Kranz and K. Mayer. "Field Testing a Wireless Sensor Network for Reactive Environmental Monitoring", *International Conference on Intelligent Sensors, Sensor Networks and Information Processing ISSNIP-04*, Melbourne, December 2004
7. M. Hedley, M. Johnson, C. Lewis, D. Carpenter, H. Lovatt and D. Price. "Smart Sensor Network for Space Vehicle Monitoring", In *Proceedings of the International Signal Processing Conference*, Dallas, Texas, March 2003.
8. D. C. Price , D. A. Scott, G. C. Edwards, A. Batten, A. J. Farmer, M. Hedley, M. E. Johnson, C. J. Lewis, G. T. Poulton, M. Prokopenko, P. Valencia and P. Wang. "An Integrated Health Monitoring System for an Ageless Aerospace Vehicle", *Structural Health Monitoring* 2003
9. Chen-Khong Tham and Rajkumar Buyya, "SensorGrid: Integrating Sensor Networks and Grid Computing", Technical Report, GRIDS-TR-2005-10, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, June 24, 2005
10. T. Liu and M. Martonosi, "Impala: a Middleware System for Managing Autonomic, Parallel Sensor Systems", In *Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming*, San Diego, CA, USA, June 2003.
11. W. Heinzelman, A. Murphy, H. Carvalho and M. Perillo, "Middleware to Support Sensor Network Applications", *IEEE Network Magazine Special Issue*, pp. 6-14, January 2004.
12. P. Bonnet, J.E. Gehrke, and P. Seshadri, "Querying the Physical World", *IEEE personal Communications*, Vol. 7, No. 5, pp. 10-15, October 2000.
13. E. Soutoo, G. Guimaraes, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, L. Freire, "A Message-Oriented Middleware for Sensor Networks", 2nd International Workshop on Middleware for Pervasive and Ad-Hoc Computing, October 2004, Toronto, Ontario, Canada.

14. Mark Gaynor, Steven L. Moulton, Matt Welsh, Ed LaCombe, Austin Rowan, John Wynne. "Integrating Wireless Sensor Networks with the Grid", IEEE Internet Computing, vol. 08, no. 4, pp. 32-39, July/August, 2004.
15. M.Ghanem, Y.Guo, J.Hassard, M.Osmond, and M.Richards. "Sensor Grids for Air Pollution Monitoring", UK e-Science All Hands Meeting, Nottingham UK, September 2004.
16. M. Reichardt, "Sensor Web Enablement: An OGC White Paper", Open Geospatial Consortium (OGC), Inc, 2005.
17. S.H.L.Liang, V.Tao and A.Croituru. "Sensor Web And GeoSWIFT- An Open GeoSpatial Sensing Service", ISPRS XXth Congress, Istanbul, Turkey, 2004.
18. PB.Gibbons, B. Karp, Y.Ke, S.Nath, S.Seshan, "IrisNet: An Architecture for a Worldwide Sensor Web" IEEE Pervasive Computing, vol. 2, no 4, 2003, pp. 22-33.
19. SensorML, <http://member.opengis.org/tc/archive/arch03/03-0005r2.pdf>.
20. Geography Markup Language, version 3, OGC Document Number: 02-023r4, URL: <http://www.opengis.org/>.
21. Observation & Measurements, version 0.9.2, OGC Document Number: 03-022r3, http://portal.opengeospatial.org/files/index.php?artifact_id=1324.
22. Sensor Collection Service IPR, OGC 03-023, <http://member.opengis.org/tc/archive/arch03/03-023r1.pdf>.
23. Transducer Markup Language, <http://www.iriscorp.org/tml.html>.
24. Jini™ Specification version 2.0, Sun Microsystems, http://www.sun.com/software/jini/specs/jini2_0.pdf
25. J. K. Waters, "Web services: The next big thing?", March 2002, <http://www.adtmag.com/article.asp?id=6124>,
26. W3C, "Web Services Architecture", February 2004, <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/wsa.pdf>
27. D. Sosnoski, "XML and Java Technologies: Data binding, Part 1: Code generation approaches – JAXB and more" Jan 2003, <http://www-128.ibm.com/developerworks/xml/library/x-databdopt/>
28. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. "System architecture directions for networked sensors". In *Proc. ASPLOS-IX*, November 2000.
29. S. R. Madden. "The Design and Evaluation of a Query Processing Architecture for Sensor Networks", PhD thesis, UC Berkeley, Decmeber 2003, <http://www.cs.berkeley.edu/~madden/thesis.pdf>.
30. H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, J. Deng, and R. Han. "MANTIS: system support for Multimodal Networks of In-Situ sensors", In *Proc. WSNA'03*, 2003.
31. A. Dunkels, B. Gronvall, and T. Voigt. "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors", In First IEEE Workshop on Embedded Networked Sensors, 2004.
32. TinyOS Tutorial 6, <http://www.tinyos.net/tinyos-1.x/doc/tutorial/lesson6.html>.
33. JAXB specification version 2.0, proposed final draft, download page,

- <http://jcp.org/aboutJava/communityprocess/pfd/jsr222/index.html>.
34. Apache XMLBeans, <http://xmlbeans.apache.org/>.
 35. D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. “*The nesC language: A holistic approach to networked embedded systems*”, In ACM SIGPLAN Conference on Programming Language Design and Implementation, 2003.
 36. P. Levis, N. Lee, M. Welsh, and D. Culler, “*TOSSIM: Accurate and Scalable simulation of entire TinyOS applications*”, Proceedings of. SenSys’03, First ACM Conference on Embedded Networked Sensor. Systems, 2003.
 37. Crossbow Technology Inc, <http://www.xbow.com/>.