Service Oriented Sensor Web

Xingchen Chu and Rajkumar Buyya

Grid Computing and Distributed Systems Laboratory
Dept. of Computer Science and Software Engineering
The University of Melbourne, Australia
{xchu, raj}@csse.unimelb.edu.au

Abstract

The Sensor Web is an emerging trend which makes various types of web-resident sensors, instruments, image devices, and repositories of sensor data, discoverable, accessible, and controllable via the World Wide Web. A lot of effort has been invested in order to overcome the obstacles associated with connecting and sharing these heterogeneous sensor resources. This chapter emphasizes the Sensor Web Enablement (SWE) standard defined by the OpenGIS Consortium (OGC), which is composed of a set of specifications, including SensorML, Observation & Measurement, Sensor Collection Service, Sensor Planning Service and Web Notification Service. It also presents a reusable, scalable, extensible, and interoperable service oriented sensor Web architecture that (i) conforms to the SWE standard; (ii) integrates Sensor Web with Grid Computing and (iii) provides middleware support for Sensor Webs. In addition, this chapter describes the experiments and an evaluation of the core services within the architecture.

Keywords: Sensor Web, SensorML, Observation & Measurement, Sensor Collection Service, Sensor Planning Service, Web Notification Service.

1. Introduction

Due to the rapid development of sensor technology, current sensor nodes are much more sophisticated in terms of CPU, memory, and wireless transceiver. Sensor networks are long running computing systems that consist of a collection of sensing nodes working together to collect information about, for instance, light, temperature, images and other relevant data according to specific applications. Wireless sensor networks have been attracting a lot of attention from both academic and industrial communities around the world. The ability of the sensor networks to collect information accurately and reliably enables building both real-time detection and early warning systems. In addition, it allows rapid coordinate responses to threats such as bushfires, tsunamis, earthquakes, and other crisis situations.
However, the heterogeneous features of sensors and sensor networks turn the efficient

collection and analysis of the information generated by various sensor nodes into a rather challenging task. The main reasons for that are the lack of both uniform operations and a standard representation for sensor data that can be used by diverse sensor applications. There exists no means to achieving resource reallocation and resource sharing among applications as the deployment and usage of the resources has been tightly coupled with the specific location, sensor application, and devices used.

The Service Oriented Architecture (SOA) provides an approach to describe, discover, and invoke services from heterogeneous platforms using XML and SOAP standards. The term 'service' not only represents a software system but also refers to hardware and any devices that can be used by human beings. A service may be an online ticket booking system, a legacy database application, a laser printer, a single sensor or even an entire network infrastructure. Bringing the idea of SOA to sensors and sensor networks is a very important step forward to presenting the sensors as reusable resources which can be discoverable, accessible and where applicable, controllable via the World Wide Web. Furthermore, it is also possible to link distributed resources located across different organizations, countries, or regions thus creating the illusion of a sensor-grid, which enables the essential strengths, and characteristics of a computational grid.

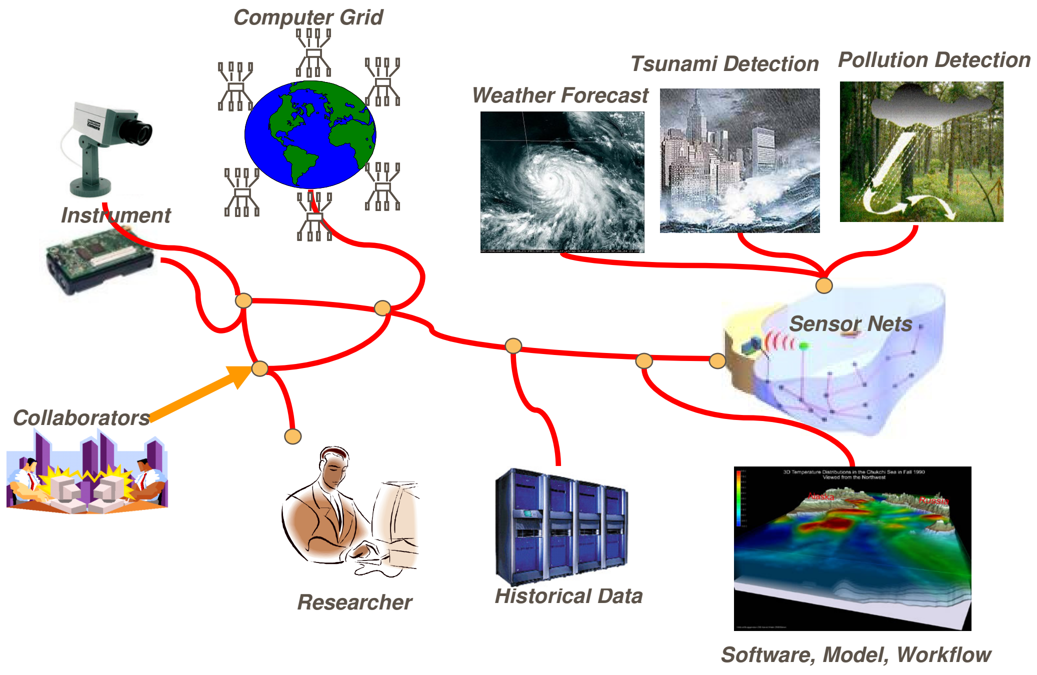


Fig. x.1: Vision of the Sensor Web.

Fig. x.1 demonstrates an abstract vision of the Sensor Web, which is the combination of SOA, grid computing and sensor networks. Various sensors and sensor nodes form a web view and are treated as available services to all the users who access the Web. Sensor Web brings the heterogeneous sensors into an integrated and uniform platform supporting dynamic discovery and access. A sample scenario would be the client (may be the researchers or other software, model and workflow system), who wants to utilize the information collected by the deployed sensors on the target application, such as weather forecast, tsunami or pollution detection. The client may query the entire sensor web and get the response either from real-time sensors that have been registered in the web or existing data from a remote database. The clients are not aware of where the real sensors are and what operations they may have, although they are required to set parameters for their plan and invoke the service (similar to when people perform a search on Google, filling in the search field and clicking the search button). The primary goal of the Sensor Web is to offer reliable and accessible services to the end-users. In other words, it provides the middleware infrastructure and the programming environment for creating, accessing, and utilizing sensor services through the Web.

The remainder of this chapter is organized as follows. Related work on sensor middleware support, sensor-grid, and sensor web is described in Section 2. Section 3 details the emerging standard of the Sensor Web: Sensor Web Enablement. Section 4 describes OSWA, a service oriented sensor web architecture, and the design and implementation of its core services. Evaluation of applying the middleware to a simple temperature monitoring sensor application is discussed in Section 5. This chapter concludes with the summary and the future work.

2. Related Work

A lot of effort has been invested in building middleware support for making the development of sensor applications simpler and faster. Impala (Liu and Martonosi, 2003) designed for the ZebraNet project, considers the application itself while adopting mobile code techniques to upgrade functions on remote sensors. The key to the energy efficiency provided by Impala is making sensor node applications as modular as possible, thus imposing small updates that require little transmission energy. MiLAN (Heinzelman et al., 2004) is an architecture that extends the network protocol stack and allows network specific plug-ins to convert MiLAN commands into protocol-specific commands. Bonnet et al., 2000 implemented Cougar, a query-based database interface that uses a SQL-like language to gather information from wireless sensor networks. However, most of these efforts concentrate on creating protocols and are designed to ensure the efficient use of wireless sensor networks. In contrast to these middleware, Mires (Soutoo et al., 2004) is a message-oriented middleware that is placed on top of the operating system, encapsulates its interfaces and provides higher-level services to the Node Application. The main component of Mires is a publish/subscribe service that intermediates communication between middleware services, which might be used as the foundation of Sensor Web middleware.

Besides middleware support for the sensor applications, integrating sensor networks with grid computing into a sensor grid is also quite important. Tham and Buyya (Tham and Buyya, 2005) outlined a vision of sensor-grid computing and described some early work in sensor grid computing by giving examples of a possible implementation of distributed information fusion and distributed autonomous decision-making algorithms. Discussion about the research challenges needed to be overcome before the vision becomes a reality have also been presented. A data-collection-network approach to address many of the technical problems of integrating resource-constrained wireless sensors into traditional grid applications have been suggested by Gaynor et al., 2004. This approach is in the form of a network infrastructure, called Hourglass that can provide a grid API to a heterogeneous group of sensors. Those, in turn, provide fine-grained access to sensor data with OSGA standards. Another sensor grid integration methodology introduced by Ghanem et al., 2004 utilized the grid services to encompass high throughput sensors, and in effect make each sensor a grid service. The service can be published in a registry by using standard methods and then made available to other users.

Nickerson et al., 2005 described a Sensor Web Language (SWL) for mesh architecture, which provides a more robust environment to deploy, maintain and operate sensor networks. As they stated, greater flexibility, more reliable operation and machinery to better support self-diagnosis and inference with sensor data has been achieved with the mesh architecture support in SWL. At the GeoICT Lab of York University, an open geospatial information infrastructure for Sensor Web, named GeoSWIFT, has been presented, which is built on the OpenGIS standards. According to Tao et al., 2004, XML messaging technology has been developed, serving as a gateway that integrates and fuses observations from heterogeneous spatial enabled sensors. The IrisNet architecture at Intel Research, introduced by Gibbons et al., 2003, is a software infrastructure that supports the central tasks common to collect, filter and combine sensor feeds and perform distributed queries. There are two-tiers 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. Finally, the most important effort that has been made to Sensor Web is the Sensor Web Enablement (SWE) introduced by Reichardt, 2005. 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. The following section of this chapter discusses SWE standards in more in detail.

3. Standard: OCG Sensor Web Enablement

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

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

- Cane-toad Monitoring Application: two prototypes of wireless sensor networks have been set up, which can recognize vocalizations of at maximum 9 frog species in Northern Australia. Besides monitoring the frogs, the researchers also plan to monitor breeding populations of endangered birds, according to Hu et al., 2003.
- Soil Moisture Monitoring Application: Cardell-Oliver et al., 2004 presents a prototype sensor network for soil moisture monitoring that has been deployed in Pinjar, located in north of Perth, WA. The data is gathered by their reactive sensor network in Pinjar, and sent back to a database in real time using a SOAP Web Services.

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

Imagine hundreds of in-site or remote weather sensors providing real-time measurements of current wind and temperature conditions for multiple metropolitan regions. A weather forecast application may request and present the information directly to end-users or other data acquisition components. A collection of Web-based services may be involved in order to maintain a registry of available sensors and their features. Also consider that the same Web technology standard for describing the sensors, outputs, platforms, locations and control parameters is in use beyond the boundaries of regions or countries. This enables the interoperability necessary for cross-organization activities, and it provides a big opportunity in the market for customers to get a better, faster and cheaper service. This drives the Open Geospatial Consortium (OGC) to develop the geospatial standards that will make the "open sensor web" vision a reality.[1]

As the Sensor Web Enablement (SWE) becomes the de facto standard regarding Sensor Web development, understanding SWE is crucial for both researchers and developers. In general, SWE is the standard developed by OGC that encompasses specifications for interfaces, protocols and encodings that enable discovering, accessing, obtaining sensor data as well as sensor-processing services. The following are the five primary specifications for SWE:

1. Sensor Model Language (SensorML) – Information model and XML encodings that describe either single sensor or sensor platform in regard to discover, query and control sensors.
2. Observation and Measurement (O&M) – Information model and XML encodings for observations and measurement.

[1] http://www.geoplace.com/uploads/FeatureArticle/0412ee.asp

3. Sensor Collection Service (SCS) – Service to fetch observations, which conforms to the Observations and Measurement information model, from a single sensor or a collection of sensors. It is also used to describe the sensors and sensor platforms by utilizing SensorML.
4. Sensor Planning Service (SPS) – Service to help users build feasible sensor collection plan and to schedule requests for sensors and sensor platforms.
5. Web Notification Service (WNS) – Service to manage client session and notify the client about the outcome of her requested service using various communication protocols.

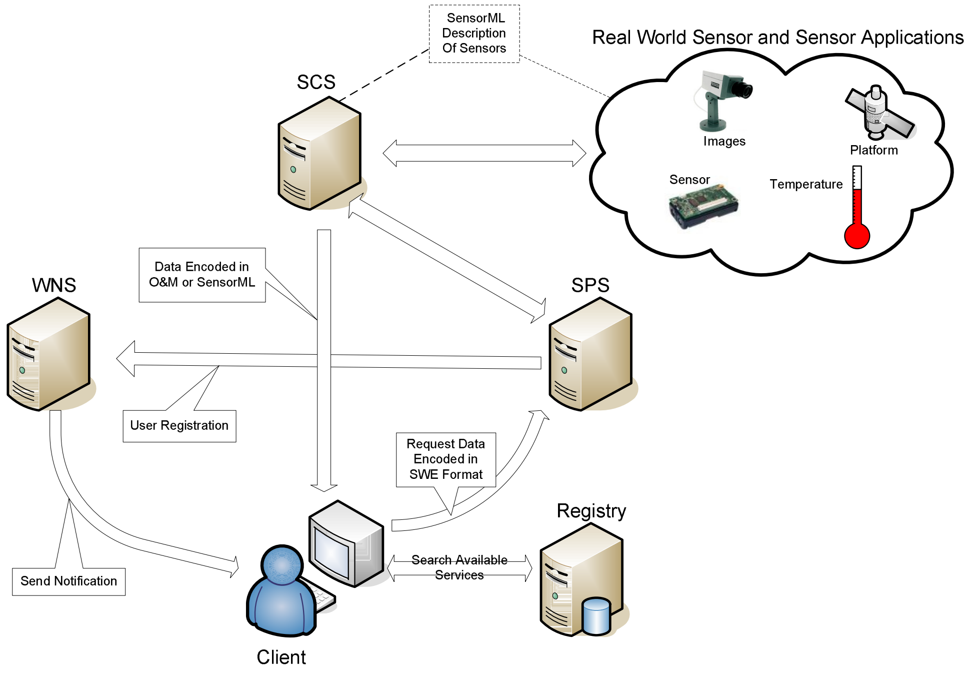

SensorML Description Of Sensors
Real World Sensor and Sensor Applications
SCS
Images
Platform
Sensor
Temperature
Data Encoded in O&M or SensorML
WNS
SPS
User Registration
Request Data Encoded in SWE Format
Registry
Search Available Services
Send Notification
Client

Fig. x. 2: A typical collaboration within Sensor Web Enablement Framework.

As Reichardt, 2005 stated, 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, controllable via the World Wide Web. In other words, the goal is to enable the creation of Web-based sensor networks. Fig. x. 2 demonstrates a typical collaboration between services and data encodings of SWE.

3.1 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 processes this information. Moreover, a uniform encoding benefits the integration of heterogeneous sensors and sensor platforms as it provides an integrated and standard view to the client. The Sensor Model Language (SensorML) is a new XML encoding scheme that may make it possible for clients to remotely discover, access, and use real-time data obtained directly from various Web-resident sensors. SensorML describes the geometric, dynamic, and observational features of sensors of all kinds.

SensorML is a key component for enabling autonomous and intelligent sensor webs, and provides the information needed for discovery of sensors, including sensor's capabilities, geo-location and taskability rather than a detailed description of the sensor hardware. Moreover, both in-site and remote sensors, on either static or dynamic platforms are supported by SensorML. Fig. x.3 depicts the basic structure of SensorML. The information provided in SensorML includes the sensor name, type, and identification (identifiedAs); time, validity, or classification constraints of the description (documentConstrainedBy); a reference to the platform (attachedTo); the coordinate reference system definition (hasCRS); the location of the sensor (locatedUsing); the response semantics for geolocating samples (measures); the sensor operator and tasking services (operatedBy); metadata and history of the sensor (describedBy); and metadata and history of the document itself (documentedBy).

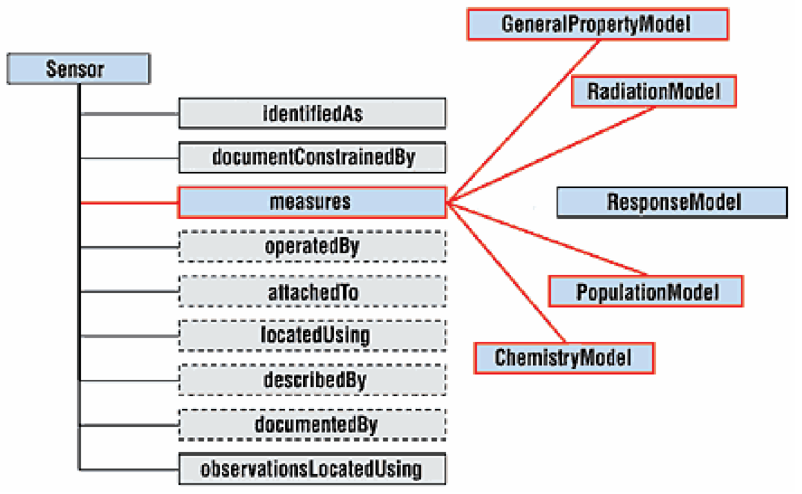


Fig. x. 3: High-level structure of SensorML (Reichardt, 2005).

Besides the importance of SensorML in SWE framework, SensorML itself is an independent standard rather than part of the SWE framework, which means that it can be used outside the scope of SWE. Other benefits of adopting SensorML include (i) enabling long-term archive of sensor data to be reprocessed and refined in the future and (ii) allowing the software system to process, analyze and perform a visual fusion of multiple sensors[2].

3.2 Observation and Measurement

Besides collaborating SensorML which contains information about sensors and sensor platforms, SWE utilizes Observation and Measurement (O&M)[3]. O&M is another standard information model and XML encoding that is important for Sensor Web to find universal applicability with web-based sensor networks. The O&M model is required specifically for the Sensor Collection Service and related components of OGC Sensor Web Enablement, which aims at defining terms used for measurements,

[2] http://member.opengis.org/tc/archive/arch03/03-0005r2.pdf, Sensor Model Language IPR, OGC 03-005.

[3] http://portal.opengeospatial.org/files/index.php?artifact_id=1324, Observation & Measurement, OGC 03-022r3.

and relationships between them.

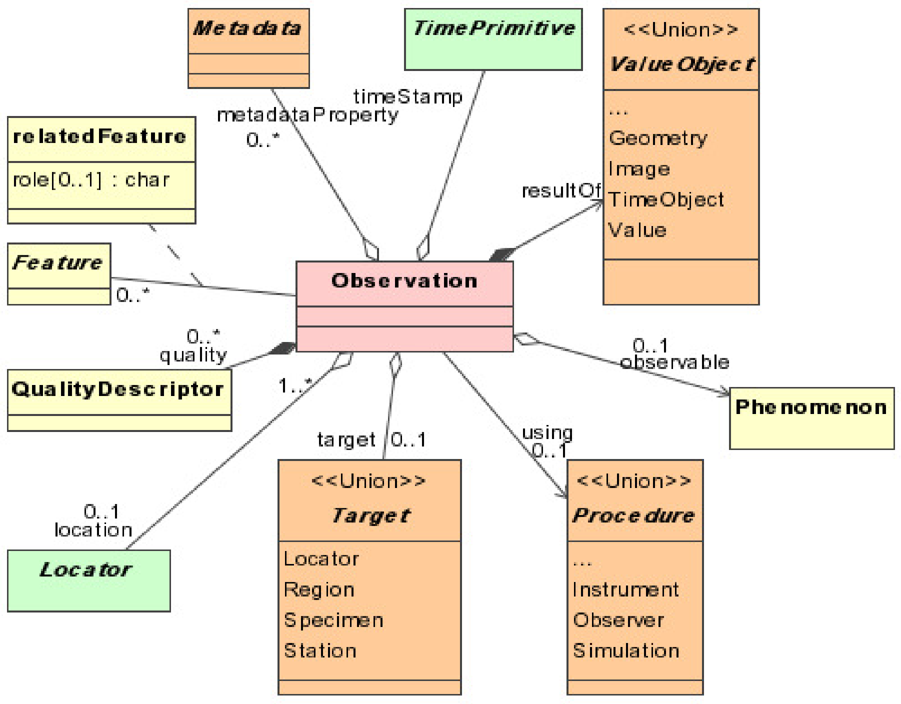


Fig. x. 4: Basic Observation structure[3].

As Fig. x. 4 indicates, the basic information provided by Observation includes the time of the event (timeStamp); the value of a procedure such as instrument or simulation (using); the identification of phenomenon being sampled (observable); the association of other features that are related to the Observation (relatedFeature); the common related features that have fixed role such as Station or Specimen (target); the quality indicators associated with the Observation (quality); the result of the Observation (resultOf); location information (location) and the metadata description (metadataProperty). Moreover, the observation data can be either single or compound values that may contain a collection or an array of observations.

3.3 SWE Services

SWE not only utilizes the information model and encoding like SensorML and Observation and Measurements (O&M), but also defines several standard services that can be used to collaborate with sensor networks in order to obtain data. Currently, the SWE contains three service specifications including Sensor Collection Service (SCS), Sensor Planning Service (SPS) and Web Notification Service (WNS). As the SWE is still evolving, new services will be developed to satisfy emerging requirements of Sensor Web development. A new service called Sensor Alert Service has recently been introduced, which specifies how alert or "alarm" conditions are defined, detected and made available to interested users. Also, a new TransducerML[4] has also been defined, which is an XML based specification that describes how to

[4] http://www.iriscorp.org/tml.html, Transducer Markup Language

capture and “time tag” sensor data. However, as these two specifications are still quite new, this chapter will only discuss the three well-known specifications in details.

One of the most important services is the Sensor Collection Service (SCS) which is used to fetch observations from a sensor or a constellation of sensors. It plays a role of intermediary agent between a client and an observation repository or near real-time sensor channel. The getObservation method of SCS accepts queries from the client within certain range of time as input parameters and responses with XML data conformed to Observation & Measurement information model. The describeSensor and describePlatform methods are used to fetch the sensor’s information based on SensorML. Each client that intends to invoke the Sensor Collection Service must strictly follow the SCS specification.

The Sensor Planning Service (SPS) is intended to provide a standard interface to handle asset management (AM) that identifies, uses and manages available information sources (sensors, sensor platforms) in order to meet information collection (client’s collection request) goals. SPS plays a crucial role as a coordinator which is responsible for evaluating the feasibility of the collection request from the client and, if valid, submitting the request by querying the SCS about the Observation data. The DescribeCollectionRequest operation of SPS presents the information needed for the client’s collection request. The GetFeasibility method of SPS accepts requests from the clients and makes a ‘yes’ or ‘no’ decision according to specified rules regarding to the feasibility of the collection. Clients can invoke the SubmitRequest method to actually schedule their requests and submit to the SCS once the GetFeasibility method responses with ‘yes’. SPS also defines UpdateRequest, CancelRequest and GetStatus methods to manage and monitor the collection request made by users.

In general, the synchronous messaging mechanism is powerful enough to handle collections of in-situ sensors. However, observations that require evaluating collection feasibility or intermediate user notifications are not suitable for synchronous operations. The Web Notification Service (WNS) is an asynchronous messaging service, which is designed to fulfill the needs of supporting these complicated scenarios. Sending and receiving notifications are the major responsibilities of the WNS, which can utilize various communication protocols including HTTP POST, email, SMS, instant message, phone call, letter or fax. Besides, WNS also takes charge of user the management functionality that is used to register user and trace the user session over the entire process. Operations including doNotification, doCommunication and doReply are defined to conduct both one-way and two-way communication between users and services whereas registerUser handles user management, which allows registering users to receive further notifications.

4. Service-Oriented Sensor Web

Open Sensor Web Architecture (OSWA) is an OGC Sensor Web Enablement standard compliant software infrastructure for providing service oriented based access to and management of sensors created by 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. The integration has brought several benefits to the community. First, 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 registered, discovered and accessed by different clients using a uniform protocol. Moreover, as Tham and Buyya, 2005 stated, Grid-based sensor applications are capable of providing advanced services for smart-sensing by developing scenario-specific operators at runtime.

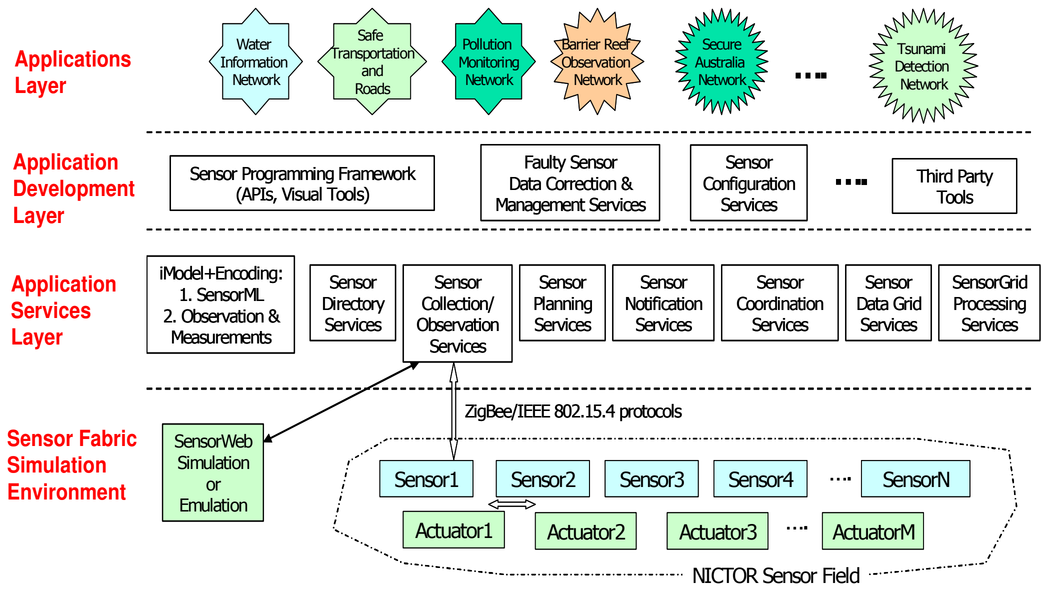

Applications Layer
Water Information Network
Safe Transportation and Roads
Pollution Monitoring Network
Barrier Reef Observation Network
Secure Australia Network
Tsunami Detection Network
Application Development Layer
Sensor Programming Framework (APIs, Visual Tools)
Faulty Sensor Data Correction & Management Services
Sensor Configuration Services
Third Party Tools
Application Services Layer
iModel+Encoding: 1. SensorML 2. Observation & Measurements
Sensor Directory Services
Sensor Collection/ Observation Services
Sensor Planning Services
Sensor Notification Services
Sensor Coordination Services
Sensor Data Grid Services
SensorGrid Processing Services
ZigBee/IEEE 802.15.4 protocols
Sensor Fabric Simulation Environment
SensorWeb Simulation or Emulation
Sensor1
Sensor2
Sensor3
Sensor4
SensorN
Actuator1
Actuator2
Actuator3
ActuatorM
NICTOR Sensor Field

Fig. x. 5: High-level view of Open Sensor Web Architecture.

The various components defined in OSWA are showed in Fig. x. 5. Four layers have been defined, namely Fabric, Services, Development and Application layers. Fundamental services are provided by low-level components whereas higher-level components provide tools for creating applications and management of the 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

Besides the core services derived from SWE, such as SCS, SPS and WNS, there are several other important services in the service layer. Sensor Directory Service provides the capability of storing and searching services and resources. The Sensor Coordination Service enables the interaction between groups of sensors, which monitor different kinds of events. The Sensor Data Grid Service provides and maintains the replications of large amount of sensor data collected from diverse sensor applications. The SensorGrid Processing Service collects the sensor data and processes them utilizing grid facilities. The development layer focuses on providing useful tools in order to ease and accelerate the development of sensor applications. The OSWA mainly focuses on providing an interactive development environment, an open and standards-compliant Sensor Web services middleware and a coordination language to support the development of various sensor applications.

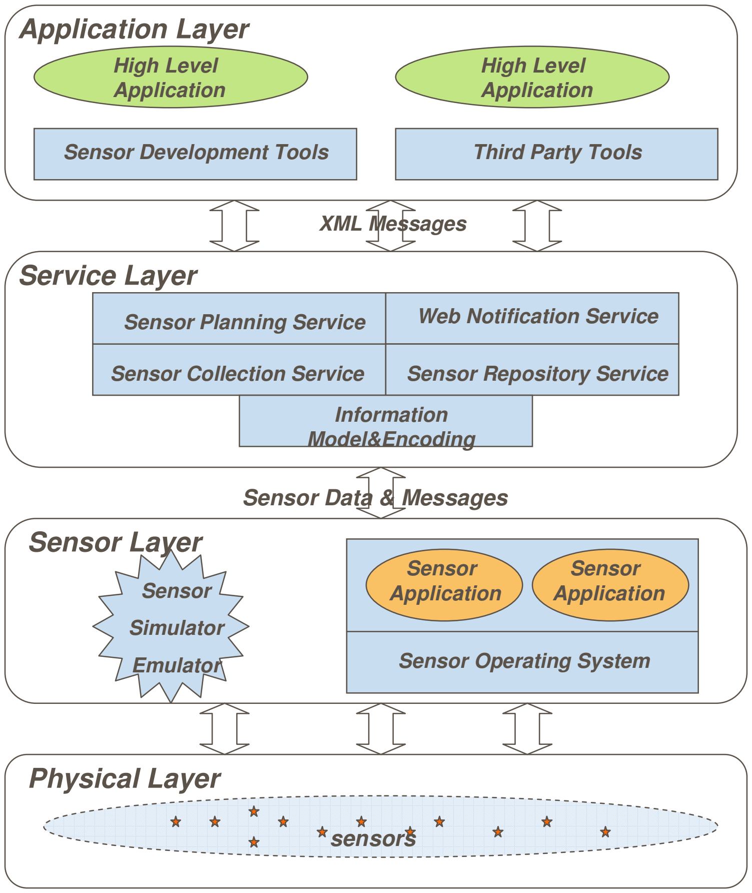


Fig. x. 6: A prototype instance of OSWA.

SWE only provides the principle standard of how the Sensor Web looks, and does not have any reference implementation or working system available to the community; therefore, there are many design issues to consider, including all of the common issues faced by other distributed systems such as security, multithreading, transactions, maintainability, performance, scalability and reusability, and the technical decisions that need to be made about which alternative technologies are best suitable to the system. Fig. x. 6 depicts a prototype instance of the OSWA, the implementation concentrates on the Service Layer and Sensor Layer as well as the XML encoding and the communication between the sensors and sensor networks. The following section will describe the key technologies that are relevant to different layers of the OSWA. In addition, the design and implementation of the core services are presented in this section.

4.1 Technology Issues

In order to better understand the whole Open Sensor Web Architecture including its design and implementation, several critical technologies are discussed briefly, which form the fundamental infrastructure across several layers of OSWA.

4.1.1 Service Layer and SOA

The SOA is the essential infrastructure that supports the Service Layer and plays a very important role in presenting the core middleware components as services for client access. The main reason for Sensor Web relying heavily on SOA is because it simplifies integration of distributed heterogeneous systems which are loosely coupled. 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 a messaging mechanism which supports both synchronous and asynchronous communication models. Since each sensor network differs from each other, trying to put different sensors on the web, and providing discovery and accessibility requires the adoption of SOA.

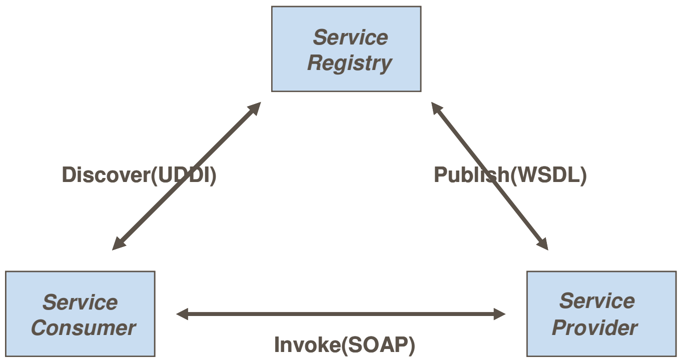


Fig. x. 7: Typical architecture of Web Service.

Web Services is one of the most popular implementations of SOA and 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. Web Services, technologically, depends 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 to Web Services technology, which is the standard format of the messages exchanged between services, and moreover almost every specifications used in Web Services are themselves XML data such as SOAP and WSDL. SOAP provides a unique framework that is used for packaging and transmitting XML messages over variety of network protocols, such as HTTP, FTP, SMTP, RMI/IIOP or proprietary messaging protocol[5]. WSDL describes the operations supported by web services and the structure of input and output messages required by these operations. It also describes important information about the web services definition, support protocol, processing model and address. The typical architecture for Web Services is showed in Fig. x. 7. Service consumers may search the global registry (i.e. UDDI registry) about the WSDL address of a service that has been published by the service provider, and the consumers can invoke the relevant calls to the service once they obtain the WSDL for the service from the registry. As OSWA is primarily based on XML data model, Web Services provide a much better solution in terms of interoperability and flexibility.

4.1.2 Information Model and XML Data Binding

The information model of OSWA is based on Observation and Measurement and SensorML, both of them are built upon XML standards and are defined by an XML Schema. Transforming the data representation of the programming language to XML that conforms to an 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 most 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 require generating a lot of redundant codes that are hard to maintain.

A better approach to deal with the transformation is to use an XML data binding mechanism that automatically generates the required code according to a DTD or an XML Schema. The data binding approach provides a simple and direct way to use XML in applications without being aware of the detailed structure of an XML document, and instead working directly with the data content of those documents. Moreover, the data binging approach makes access to data faster since it requires less memory than a document model approach like DOM or JDOM for working with documents in memory[6]. There are quite a few Java Data binding tools available such as JAXB, Castor, JBind, Quick, Zeus and Apache XMLBeans. Among those open source tools, XMLBeans seem to be the best choice not only because it provides full support for XML Schema, but also does it provide extra valuable features like XPath and XQuery supports, which directly enables performing queries on the XML documents.

4.1.3 Sensor Operating System

OSWA has the ability of dealing with heterogeneous sensor networks that may adopt quite different communication protocols including radio, blue tooth, and ZigBee/IEEE

[5] http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/wsa.pdf, Web Services Architecture, W3C, Feb 2004

[6] Sosnoski D (2003) XML and Java Technologies: Data binding, Part 1: Code generation approaches – JAXB and more. http://www-128.ibm.com/developerworks/xml/library/x-databdopt/

802.11.4 protocols. As a result, 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 (Hill et al., 2000) 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 (Gay et. al., 2003) 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. SerialForwarder runs as a server on the host machine and forwards the packets received from sensor networks to the local network, depicted by Fig. x. 8. Once the SerialForwarder program is running, the software located on the host machine can parse the raw packet and process the desired information. TinyDB (Maden, 2003) 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.

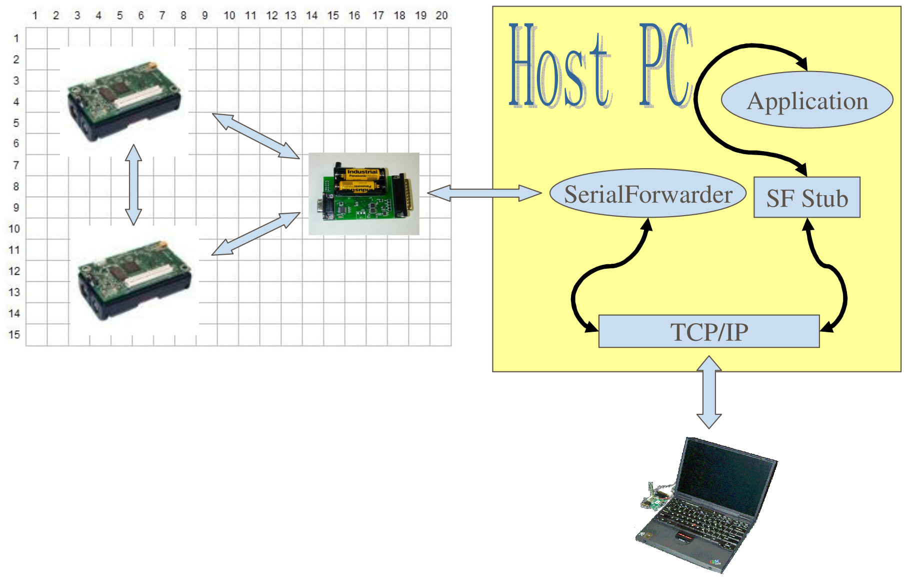


Fig. x. 8: TinyOS SerialForwarder Architecture.

Besides TinyOS, there are other Operating Systems existing as well. MANTIS (Abrach et. al., 2003) 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 (Dunkels et. al., 2004), which is designed for memory constrained systems, is another event-driven sensor operating system like TinyOS with a 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 relevant to both sensor observation and sensor information. As the standard format for exchanging data between services is XML data which conforms 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 assocoiated 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 the Java classloader.

4.2 Design and Implementation

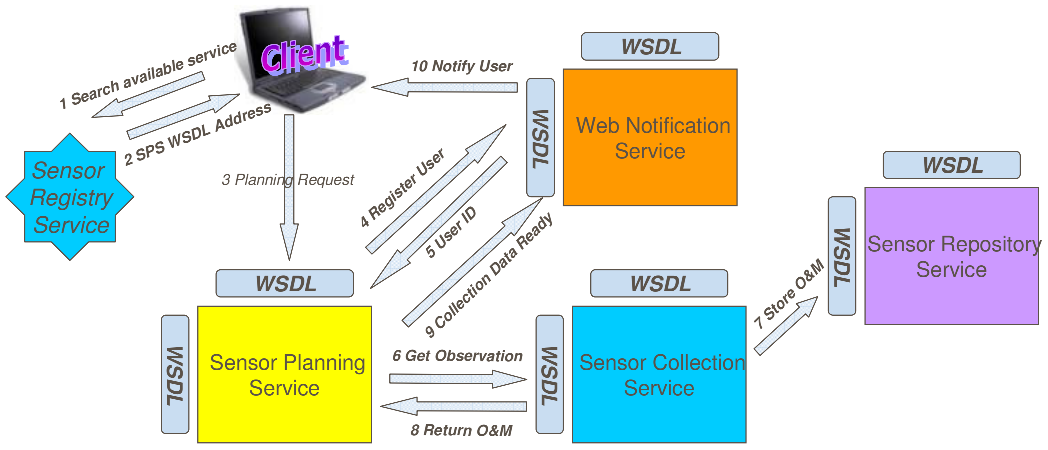


Fig. x. 9: A typical invocation for Sensor Web client.

Currently, the primary design and implementation of OSWA focuses on its core services including SCS, WNS, and SPS (those extends from SWE) as well as the Sensor Repository Service that provides the persistent machanism for the sensor and the observation data. Fig. x. 9 illustrates an example of the client collection request and the invocations between relating services. As soon as the end user forwards an observation plan to the Planning Service, it checks the feasibility of the plan and submits it if 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 for creating the observation request according to user's plan and retrieving the O&M data from the Sensor Collection Service. Once the O&M data is ready, the SPS will send an operation complete message to the WNS along with the user id and task id. The WNS will then notify the end user to collect the data via email or other protocols it supports.

4.2.1 Sensor Collection Service

Within those core services of OSWA, Sensor Collection Service (SCS) is one of the most important components residing in the service layer. The 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, SCS is the gateway for entering into the sensor networks from outside clients and its design and implementation will affect the entire OSWA. The design of SCS provides an interface to both streaming data and query based sensor applications that are built on top of TinyOS and TinyDB respectively.

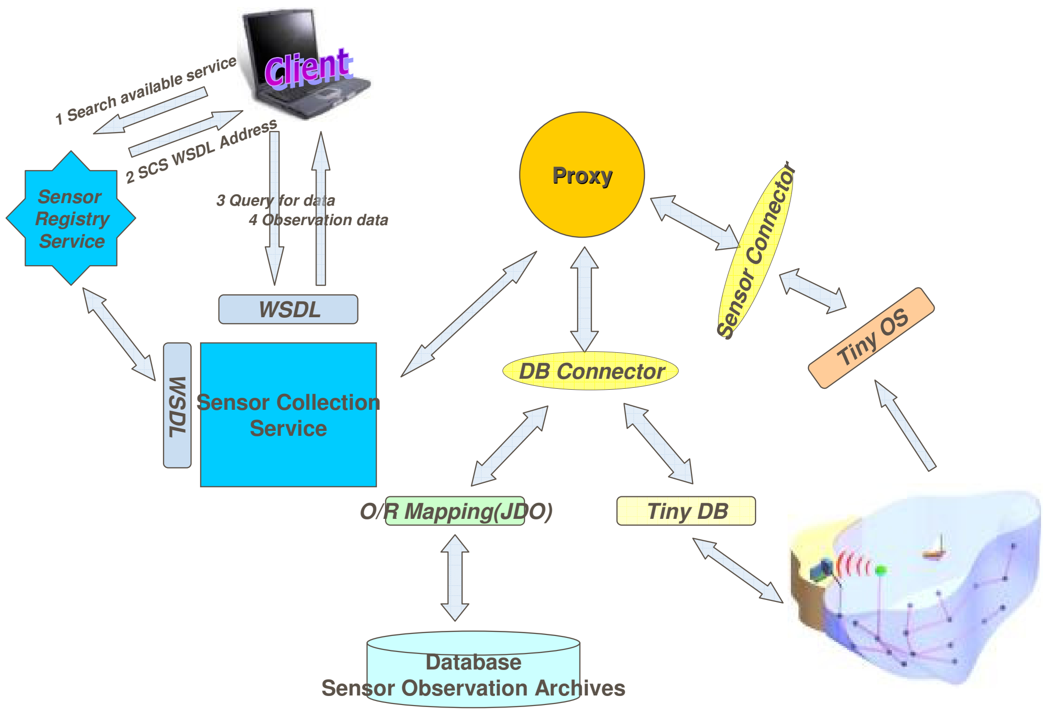


Fig. x. 10: Sensor Collection Service Architecture.

Fig. x. 10 illustrates the architecture of the Sensor Collection Service. It conforms to the interface definition that is described by the SCS specification and has been designed as web services that work with a proxy connecting to either real sensors or a remote database. Clients need to query the Sensor Registry Service about available SCS WSDL addresses according to their requirements and send a data query via SOAP to the SCS in order to obtain the observation data conforming to the O&M specification. The proxy acts as an agent working with various connectors that connect to the resources holding the information, and encode the raw observation into O&M compatible data. Different types of connectors have been designed to fit into different types 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 the client in order to determine what kind of connectors, either real-time based or archive based, to use. The design of the SCS is flexible and makes it easy to extend for further development if different sensor operating systems 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 a cache mechanism to improve the scalability and performance of the SCS. Load balancing mechanisms can be added to the system easily as well, by simply deploying the web service to different servers.

4.2.2 Sensor Planning Service

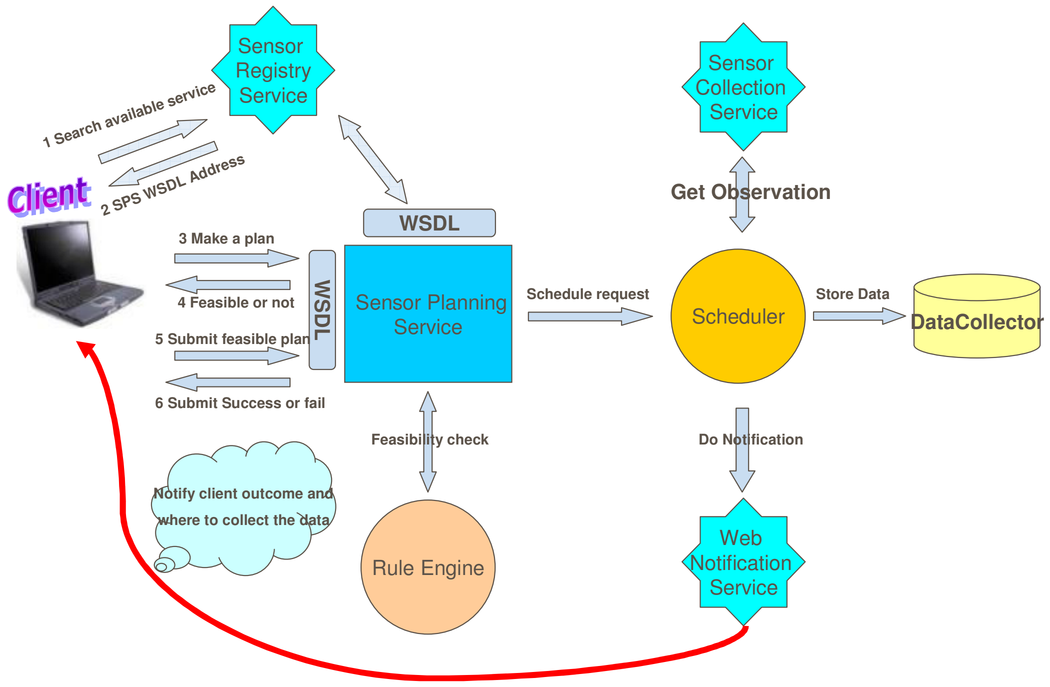


Fig. x. 11: Sensor Planning Service Architecture.

The design of the Sensor Planning Service (SPS) should consider the both short-term and long-term user's plan, which means that the SPS must provide response to the user immediately, rather than blocking to wait for the collection results. Shown in the Fig. x. 11, SPS utilizes a rule engine which reads a specific set of predefined rules in order to clarify the feasibility of the plan made by the user. The implementation of the rule engine can be quite complicated, expecting the system to accept rules within a configuration file as plain text, xml-based or other types of rule-based languages. . Currently, the rule engine is implemented as an abstract class that can be extended by the application developers to specify a set of boundary conditions that define the feasibility of the applications. For example, in a simple temperature application, a boundary condition for the temperature may be a range from 0 to 100.

The most important component that makes the SPS suitable for short or long term plan execution is the Scheduler which is implemented as a separate thread running in the background. The execution sequence of the Scheduler (i) composes the collection request according to user's plan and then invokes the getObservation of the SCS, (ii) asks the DataCollector to store the observation data in order for users to collect afterward, and (iii) sends notification to the WNS indicating the outcome of the collection request. Notice that the time of the execution happened in the scheduler varies baesd on the requirements of the user's plan. The clients will get a response indicating that their plans will be processed right after they submit their plan to the SPS. The scheduler deals with the remaining time consuming activities. The clients may get the notification from the WNS as soon as the WNS receives the message from the scheduler, and collect the results from the DataCollector.

4.2.4 Web Notification Service

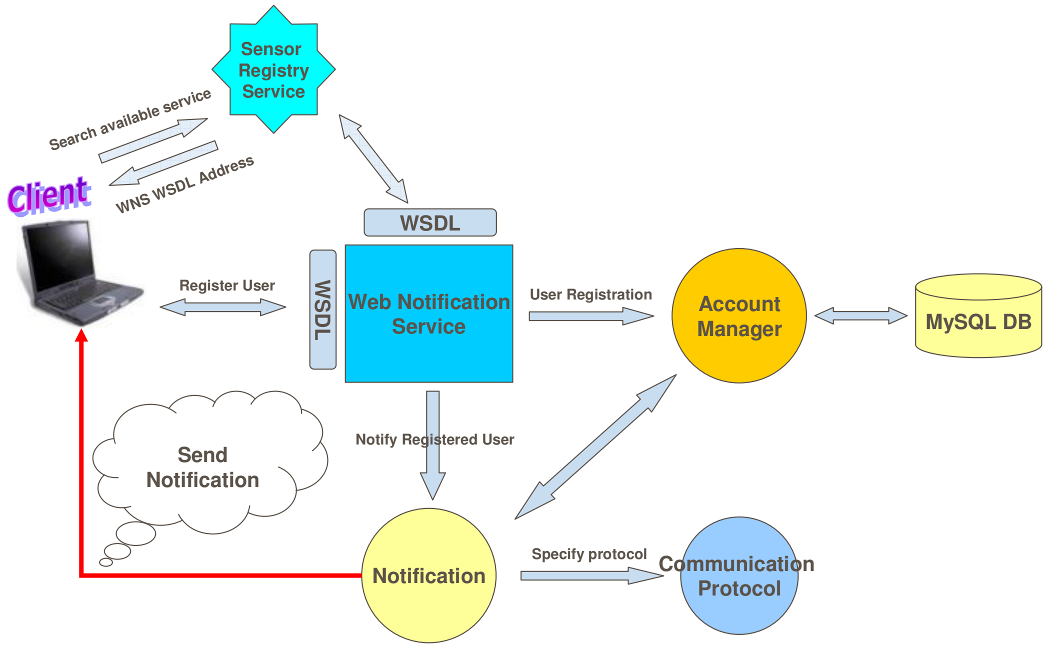


Fig. x. 12: Web Notification Service Architecture.

The current design of Web Notification Service is showed in Fig. x. 12, which contains two basic components: AccountManager and Notification. The SPS may request to register users via WNS, which asks the AccountManager to manage the user account in the DBMS in order to retrieve user information in the subsequent operations. The Notification is used to create a specific communication protocol and send the messages via the protocol to the user that has been registered in the DBMS. Currently, an EmailCommunicationProtocol has been implemented to send messages via email. Further implementations can be easily plugged into the existing architecture by implementing the CommunicationProtocol interface with a send method.

5. Experimentation and Evaluation

As the OWSA aims at providing a platform to serve numerous users globally through the internet, it is quite important to test the services, and ensure that they are scalable and performing reasonably. The experiment platform for the services is built on TOSSIM (described by Levis et al., 2003 as a discrete event simulator that can simulate thousands of motes running complete sensor applications and allow a wide range of experimentation) and Crossbow's MOTE-KIT4x0 MICA2 Basic Kit[7] that consists of 3 Mica2 Radio board, 2 MTS300 Sensor Boards, a MIB510 programming and serial interface board. The experiment concentrates on the SCS, due to the fact that it is the gateway for other services to sensors, which would be the most heavily loaded service and possible bottleneck of the entire system. As can be seen in Fig. x. 13, SCS 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 registry and data repository.

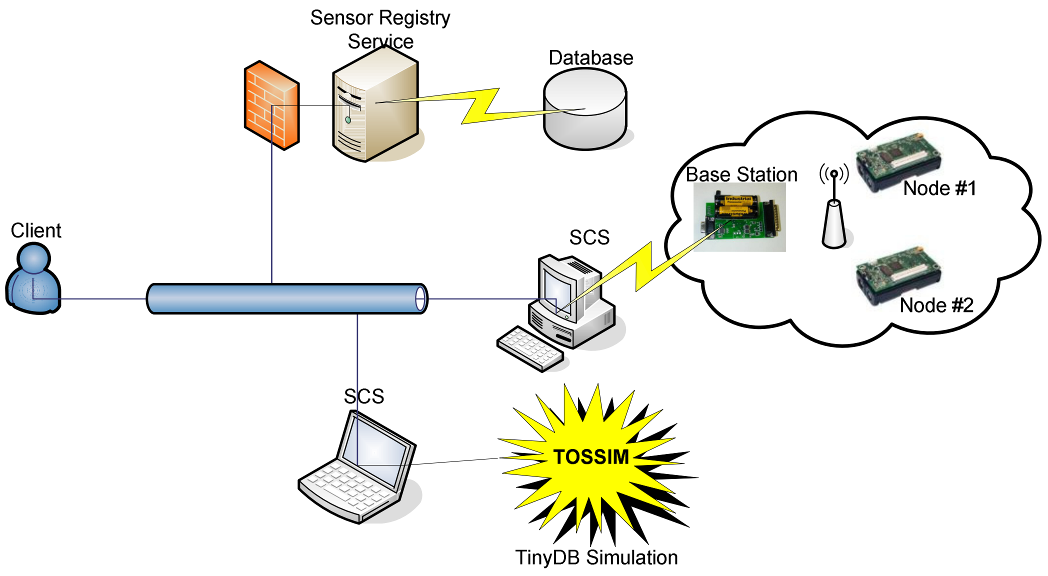
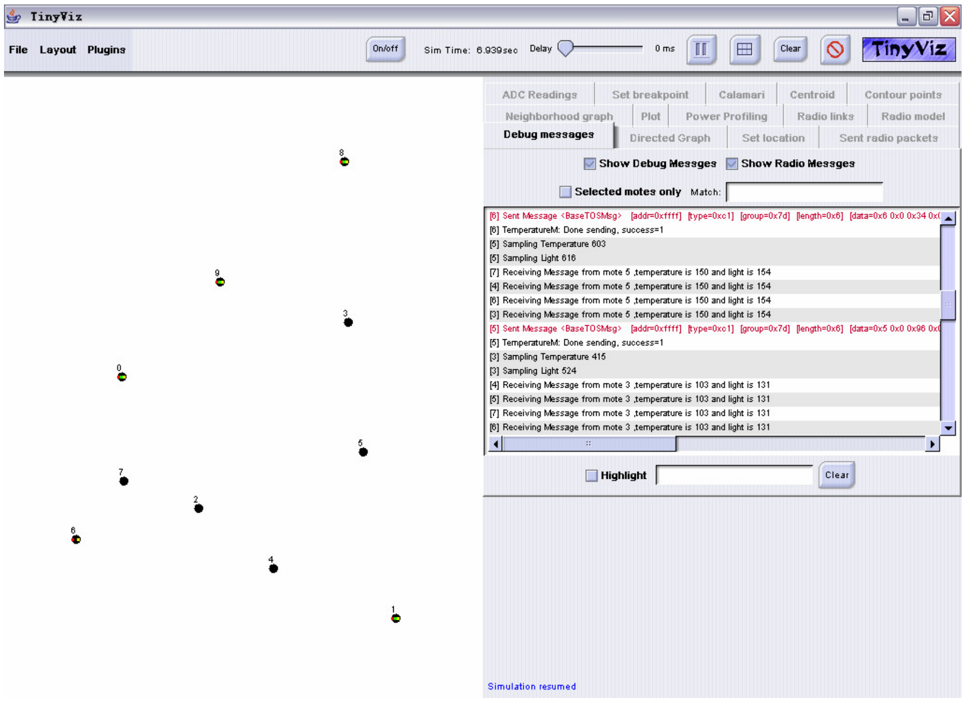


Fig. x. 13: Deployment of Experiment.

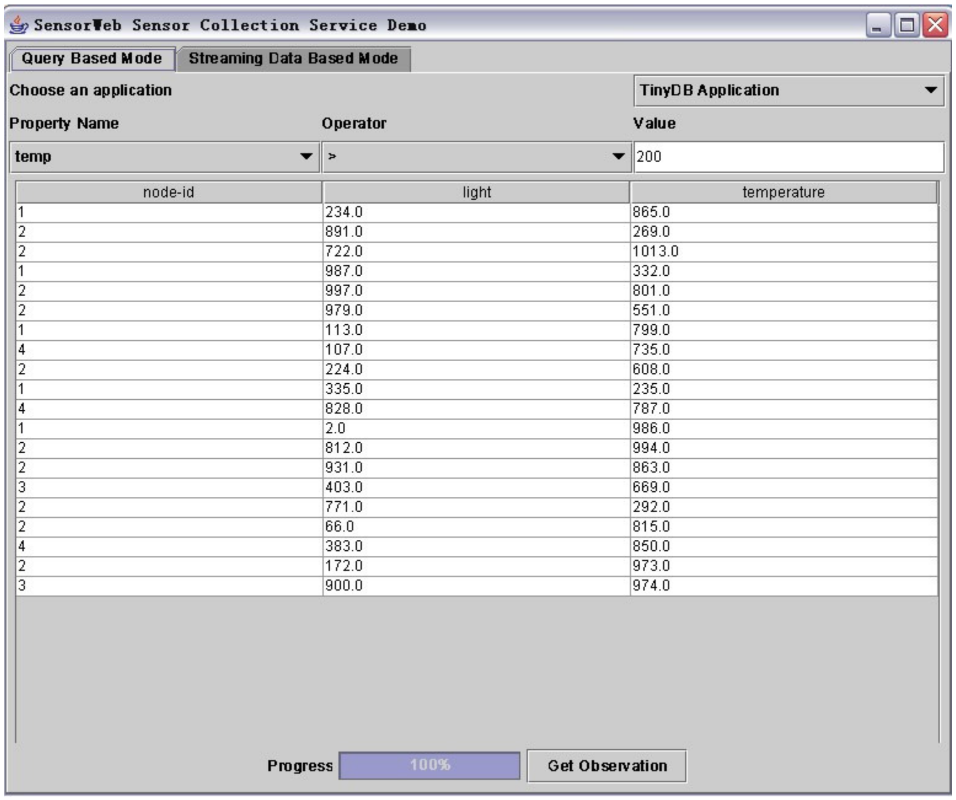
A simple temperature monitoring application has also been developed. The application is programmed using nesC and uses simple logic, which just broadcasts the sensing temperature, light and node address to the sensor network at regular intervals. The simple application does not consider any multi-hop routing and energy saving mechanism. Before installing the application to the Crossbow's mote, the functionality can be verified via the TOSSIM simulator. Fig. x. 14 demonstrates the simulation of the temperature application running under the TOSSIM visual GUI.

[7] http://www.xbow.com/Products/productsdetails.aspx?sid=67. Crossbow Technology Inc



```
[6] Sent Message <BaseTOSMsg>  [addr=0xffff]  [type=0xc1]  [group=0x7d]  [length=0x6]  [data=0x6 0x0 0x34 0x
[6] TemperatureM: Done sending, success=1
[5] Sampling Temperature 603
[5] Sampling Light 616
[7] Receiving Message from mote 5 ,temperature is 150 and light is 154
[4] Receiving Message from mote 5 ,temperature is 150 and light is 154
[6] Receiving Message from mote 5 ,temperature is 150 and light is 154
[3] Receiving Message from mote 5 ,temperature is 150 and light is 154
[5] Sent Message <BaseTOSMsg>  [addr=0xffff]  [type=0xc1]  [group=0x7d]  [length=0x6]  [data=0x5 0x0 0x96 0x
[5] TemperatureM: Done sending, success=1
[3] Sampling Temperature 415
[3] Sampling Light 524
[4] Receiving Message from mote 3 ,temperature is 103 and light is 131
[5] Receiving Message from mote 3 ,temperature is 103 and light is 131
[7] Receiving Message from mote 3 ,temperature is 103 and light is 131
[6] Receiving Message from mote 3 ,temperature is 103 and light is 131
```

Fig. x. 14: Simulation of Temperature Monitoring Application under TOSSIM


SensorWeb Sensor Collection Service Demo
Query Based Mode
Streaming Data Based Mode
Progress
100%
Get Observation

Choose an application: TinyDB Application

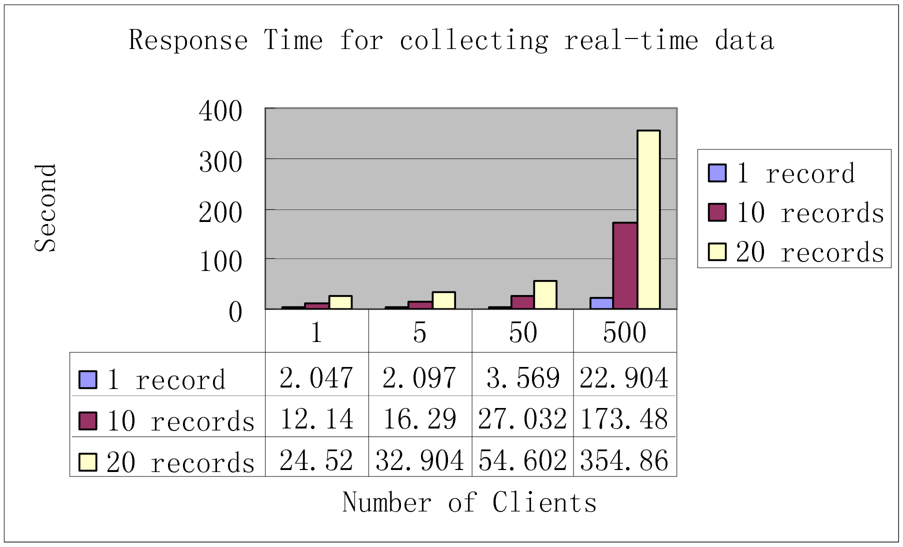
Property Name	Operator	Value
temp	>	200

node-id	light	temperature
1	234.0	865.0
2	891.0	269.0
2	722.0	1013.0
1	987.0	332.0
2	997.0	801.0
2	979.0	551.0
1	113.0	799.0
4	107.0	735.0
2	224.0	608.0
1	335.0	235.0
4	828.0	787.0
1	2.0	986.0
2	812.0	994.0
2	931.0	863.0
3	403.0	669.0
2	771.0	292.0
2	66.0	815.0
4	383.0	850.0
2	172.0	973.0
3	900.0	974.0

Fig. x. 15: Swing client showing query result for TinyDB application under TOSSIM

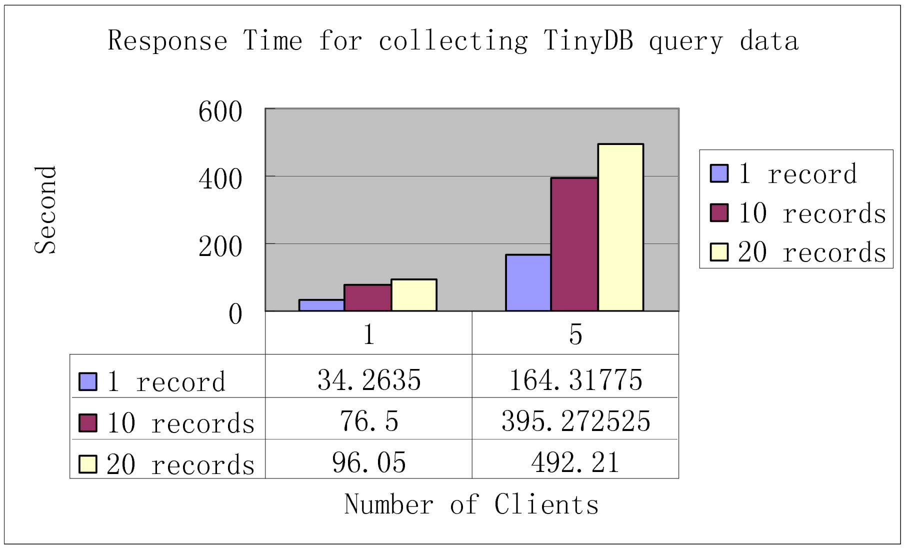
Once the application has been successfully installed onto each mote via the programming board, a wireless sensor network has been built with two nodes, and one base station connecting to the host machine via the serial cable. Besides installing the application itself, the SerialForwarder program also needs to run on the host machine in order to forward the data from the sensor network to the server. Fig. x. 15 demonstrates the list of results for a simple query “temp>200” to the sensors running TinyDB application under TOSSIM.

Regarding scalability, a simulation program that can stimulate different numbers of clients running at the same time has been used exclusively for the SCS. The performance measured by time variable (per second) for both auto-sending and query-based applications running on top of TinyOS is showed in the following figures. As can be seen from Fig. x. 16, the result of the auto-sending mode application is moderate when the number of clients who request the observation simultaneity is small. Even when the number of clients reaches 500; the response time for a small number of records is also acceptable. In contrast, the result showed in Fig. x. 17 is fairly unacceptable as even just one client requesting a single observation takes 34 seconds. 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 poor performance is due to the execution mechanism of TinyDB. A lot of time has been spent on initializing each mote, and the application can only execute one query at one time, which means another query needs to wait until the current query has been stopped or has 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 collected from the generic query.



	1	5	50	500
🟦 1 record	2.047	2.097	3.569	22.904
🟥 10 records	12.14	16.29	27.032	173.48
🟨 20 records	24.52	32.904	54.602	354.86

Fig. x. 16: Performance for collecting auto-sending data.



	1	5
🟦 1 record	34.2635	164.31775
🟥 10 records	76.5	395.272525
🟨 20 records	96.05	492.21

Fig. x. 17: Performance for collecting TinyDB query data.

There are several possible ways to enhance the performance. A caching mechanism may be one of the possible approaches, 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 kept as close to real time as possible, it is quite hard to determine the period of time for the cache to be valid. A decision can be made according to the dynamic features 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 for each sensor application can vary based on the information the sensor is 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 similar to TinyDB.

6 Summary and Future Works

In this chapter, we have introduced a new buzzword: Sensor Web in the research and academic community of sensor and sensor networks. There are a lot of efforts that aim to provide middleware support to the sensor development. Among those, the most important one is OGC's SWE standard that standardizes the vision of Sensor Web. SensorML, O&M, SCS, SPS and WNS together, to create an integration platform to register, discover and access anonymous and heterogeneous sensors distributed all over the world through internet. A service oriented Sensor Web framework named Open Sensor Web Architecture (OSWA) has been discussed along with the design implementation of the core services targeting the sensor applications running on top of TinyOS. OSWA extends SWE and provides additional services to process the information collected from those resources accompanied by computational grids. In addition, the experiment of the scalability and performance of the prototype system is also presented.

Although the services are all working properly with acceptable performance, we are still at an early stage of having the entire OSWA implemented. Even those services that we have implemented are not yet fully functional. The Sensor Collection Service is the key component of the entire OSWA, which affects the performance and reliability of the entire system. A lot of issues are left to future investigation, focusing on aspects of reliability, performance optimization and scalability. There are a couple of efforts that are needed to be made to enhance the SCS and other services in the next stage.

- The query mechanism for the Sensor Collection Service will be enhanced to support in-memory XML document querying. XPath and XQuery technologies are planned to be adopted, as they are the standard way to query XML documents. The outcome of this enhancement is expected to improve the performance by moving the heavy workload of queries from the sensor network itself and onto the host machine instead.
- A caching mechanism may be implemented and placed into the Proxy to further enhance the performance and scalability.
- Other methods that are described in the specifications of the SWE services but are currently not available still need to be implemented.
- Other notification protocols need to be built for the WNS in the future.
- Sensor Registry via SensorML needs to be developed in order to support the worldwide sensor registration and discovery.
- Both XML-based configuration and rule-based configuration language may be developed in order to ease the deployment of the services.

Acknowledgements

We would like to thank Bohdan Durnota for his technical guidance while formulating and designing initial prototype of SensorWeb. We thank Jiye Lin for his contribution towards the development of SensorWeb repository. We thank all members of the NOSA project especially those involved in advancing SensorWeb into the future.

Bibliography

1. Liu T, Martonosi M (2003) Impala: a Middleware System for Managing Autonomic, Parallel Sensor Systems. In Proceedings of the 9th ACM SIGPLAN symposium on Principles and practice of parallel programming, June 11-13, San Diego, CA, USA.
2. Heinzelman W, Murphy A, Carvalho H, Perillo M (2004) Middleware to Support Sensor Network Applications. IEEE Network Magazine 18: 6-14.
3. Bonnet P, Gehrke J, Seshadri P (2000) Querying the Physical World. IEEE personal Communications 7:10-15.
4. Soutoo E, Guimaraes G., Vasconcelos G., Vieira M, Rosa N, Ferraz C, Freire L (2004) A Message-Oriented Middleware for Sensor Networks. Proceedings of 2nd International Workshop on Middleware for Pervasive and Ad-Hoc Computing, October 18-22, Toronto, Ontario, Canada.

5. Tham CK, Buyya R (2005) SensorGrid: Integrating Sensor Networks and Grid Computing. CSI Communications 29:24-29.
6. Gaynor M, Moulton SL, Welsh M, LaCombe E, Rowan A, Wynne J (2004) Integrating Wireless Sensor Networks with the Grid. IEEE Internet Computing 8:32-39.
7. Ghanem M, Guo Y, Hassard J, Osmond M, and Richards M (2004) Sensor Grids for Air Pollution Monitoring. Proceedings of UK e-Science All Hands Meeting, 31st Aug- 3rd September, Nottingham UK.
8. Nickerson BG, Sun Z, Arp JP (2005) A Sensor Web Language for Mesh Architectures. 3rd Annual Communication Networks and Services Research Conference, May 16-18, 2005, Halifax, Canada.
9. Tao V, Liang SHL, Croitoru A, Haider Z, Wang C (2004) GeoSWIFT: Open Geospatial Sensing Services for Sensor Web. In: Stefanidis A, Nittel S (eds), CRC Press, pp.267-274.
10. Gibbons PB, Karp B, Ke Y, Nath S, Seshan S (2003) IrisNet: An Architecture for a Worldwide Sensor Web. IEEE Pervasive Computing 2: 22-33.
11. Reichardt M (2005) Sensor Web Enablement: An OGC White Paper. Open Geospatial Consortium (OCG), Inc.
12. Mainwaring A, Polastre J, Szewczyk R, Culler D, Anderson J (2002) Wireless sensor networks for habitat monitoring. In Proceedings of the first ACM International Workshop on Wireless Sensor Networks and Applications, Sept. 28, Atlanta, GA, USA.
13. Hu W, Tran VN, Bulusu N, Chou CT, Jha S, Taylor A (2005) The Design and Evaluation of a Hybrid Sensor Network For Cane-toad Monitoring. In Proceedings of Information Processing in Sensor Networks, April 25-27, Los Angeles, CA, USA.
14. Cardell-Oliver R, Smettern K, Kranz M, Mayer K (2004) Field Testing a Wireless Sensor Network for Reactive Environmental Monitoring. International Conference on Intelligent Sensors, Sensor Networks and Information Processing, December 14-17, Melbourne, Australia.
15. Hill J, Szewczyk R, Woo A, Hollar S, Culler D, and Pister K (2000) System architecture directions for networked sensors. In Architectural Support for Programming Languages and Operating Systems, November 12-15, Cambridge, MA, USA.
16. Madden SR (2003) The Design and Evaluation of a Query Processing Architecture for Sensor Networks. PhD thesis, UC Berkeley, USA.
17. Abrach H, Bhatti S, Carlson J, Dai H, Rose J, Sheth A, Shucker B, Deng J, Han R (2003) MANTIS: system support for MultimodAl NeTworks of In-Situ sensors. Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications, September 19, San Diego, CA, USA.
18. Dunkels A, Gronvall B, Voigt T (2004) Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks, November 16-18, Tampa, Florida, USA.
19. Gay D, Levis P, von Behren R, Welsh M, Brewer E, Culler D (2003) The nesC language: A holistic approach to networked embedded systems. Proceedings of Programming Language Design and Implementation (PLDI), June 8-11, San Diego, CA, USA.
20. Levis P, Lee N, Welsh M, Culler D (2003) TOSSIM: Accurate and. Scalable simulation of entire TinyOS applications. In Proc. of the 1st Intl. Conf. on Embedded Networked Sensor Systems, November 4-7, Los Angeles, CA, USA.