Sensor Web Architecture: Core Services

#Xingchen Chu[1], Tom Kobialka[2], Rajkumar Buyya[1]
GRIDS Lab[1] and NICTA Victoria Lab[2]
Department of Computer Science and Software Engineering
Faculty of Engineering
The University of Melbourne, Australia
{xchu,tkob,raj}@csse.unimelb.edu.au

Abstract

As sensor network deployments begin to grow there emerges an increasing need to overcome the obstacles of connecting and sharing heterogeneous sensor resources. Common data operations and transformations exist in deployment scenarios which can be encapsulated into a layer of software services that hide the complexity of the underlying infrastructure from the application developer. NOSA is a built upon the Sensor Web Enablement (SWE) standard defined by the Open GIS 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 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.

Keywords
Sensor Web, SensorML, Service-Orientated Architecture, Observation and Measurement, Sensor Collection Service, Sensor Planning Service, Web Notification Service, NOSA.

1. INTRODUCTION

Sensor networks are persistent computing systems composed of large numbers of sensor nodes. Sensor nodes communicate with one another over wireless low-bandwidth links and have limited processing capacity. Sensor nodes work together to collect information about their surrounding environment, this may include things like temperature, light intensity or GPS location. As sensor networks grow and rapidly improve in their ability to measure real-time information in an accurate and reliable fashion, a new research challenge, on how to collect and analyze this generated information presents itself.

Deployment scenarios for sensor networks are countless and diverse, sensors may be used for military applications, weather forecasting, tsunami detection, pollution detection, for power management in schools and office buildings. In many of these cases the software management tools for data aggregation, archiving and decision making are tightly coupled with the application scenario. However, as sensor systems begin to grow and mature a set of common data operations and transformations begin to emerge. For example, all application scenarios will need to send queries to a sensor network and retrieve some resulting data. Some scenarios may require information from historic queries be stored in a repository for further analysis. Others may require regular queries to be scheduled and automatically dispatched without external operator intervention. There is a growing need to share resources among diverse network deployments to aid in tasks like decision making. For example, a tsunami warning system may rely on water level information from two geographically distributed sets of sensors developed by competing hardware vendors. This presents significant challenges in resource interoperability, fault tolerance and software reliability.

In NICTA Open Sensor Web Architecture (NOSA), we aim to implement a set of uniform operations and a standard representation for sensor data which will fulfill the software needs of a sensor network regardless of the deployment scenario. We adopt a Service Orientated Architecture (SOA) approach to describe, discover and invoke services from a heterogeneous platform using XML and SOAP standards. Services are defined for common operations including data aggregation, scheduling, resource allocation and resource discovery. Combing sensors and sensor networks with a SOA is an important step forward in presenting sensors as important resources which can be discovered, accessed and where applicable, controlled via the World Wide Web. We refer to this combination of technologies as the Sensor Web. It opens up the opportunity for linking geographically distributed sensor and computational resources into a sensor-grid.

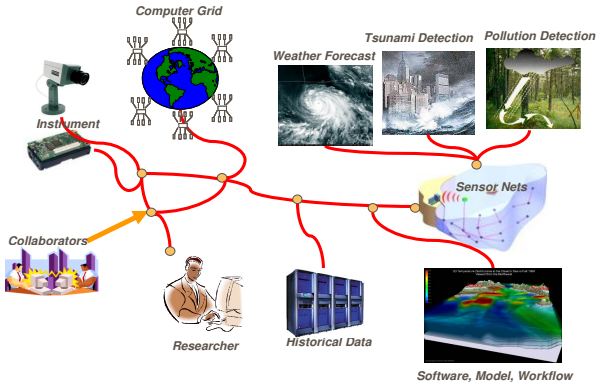

Fig. 1: Vision of the Sensor Web.

Fig. 1 demonstrates an abstract vision of the Sensor Web, various sensors and sensor nodes form a web view and are treated as available services to all the users who access the Web. A researcher may wish to predict if a tsunami is going to occur, they may query the entire Sensor Web and retrieve

the response either from real-time sensors that have been registered on the Web or from historic sensor data available in 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.

In Section 2 we describe the OGC Sensor Web Enablement (SWE) method, Section 3 introduces the Service-Orientated Architecture, and Section 4 details the Design and Implementation behind NOSA, including detailed sections on the core implemented services including the Sensor Collection Service (SCS), Sensor Planning Service (SPS) and Web Notification Service (WNS). Section 5 provides a detailed performance evaluation of NOSA.

2. OGC SENSOR WEB ENABLEMENT

Sensor network applications have been successfully developed and deployed around the world. Concrete examples include deployments on Great Duck Island [3], Cane-toad monitoring [4] and for Soil Moisture Monitoring [5]. However, lack of software interoperability prevents users from accessing resources generated by these applications without specialized tools. Moreover, lack of semantics to describe the sensors makes it impossible to build a uniform registry to discover and access these sensors. In addition, internal information is often tightly coupled with the specific deployment application rather than making use of standard data representations, this restricts the ability for mining and analyzing the data for decision making.

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 receive a higher quality of service. These needs drive the Open Geospatial Consortium (OGC) [1] to develop the geospatial standards that will make the "open sensor web" vision a reality [2].

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) [7] – Information model and XML encodings that describe either a single sensor or sensor platform in regard to discovery, query and control of sensors.
2. Observation and Measurement (O&M) [14] – Information model and XML encodings for observations and measurement.
3. Sensor Collection Service (SCS) [17] – Service to fetch observations, which conform to the O&M 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) [18] – Service to help users build a feasible sensor collection plan and to schedule requests for sensors and sensor platforms.
5. Web Notification Service (WNS) [19] – Service to manage client sessions and notify the client about the outcome of the requested service using various communication protocols.

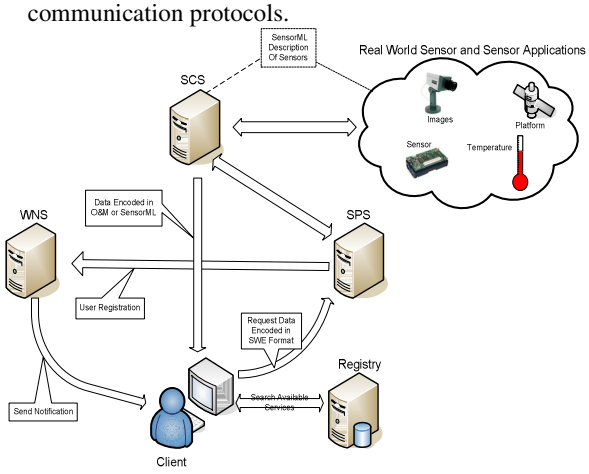

Fig. 2: A typical collaboration within Sensor Web Enablement Framework.

As [6] 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. 2 demonstrates a typical collaboration between services and data encodings of SWE.

3. SERVICE-ORIENTED SENSOR WEB

NICTA Open Sensor Web Architecture (NOSA) is an OGC SWE standard compliant software infrastructure for providing service based access to and management of sensors. NOSA is a platform for integration of sensor networks and emerging distributed computing platforms such as SOA and Grid Computing. The integration brings 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. This separation is beneficial because it reduces the energy and power needed by the sensors, allowing them to concentrate on sensing and sending information. The information processing and fusing is performed on a separate distributed system. Moreover, individual sensor networks can be linked together as services, which can be registered, discovered and accessed by different

clients using a uniform protocol. As [8] stated, Grid-based sensor applications are capable of providing advanced services for smart-sensing by developing scenario-specific operators at runtime.

The various components defined for NOSA are showed in Fig. 3. Four layers have been defined, namely Fabric, Services, Development and Application. 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. NOSA provides the following sensor services:

1. Sensor notification, collection and observation;
2. Data collection, aggregation and archival;
3. Sensor co-ordination and data processing;
4. Faulty sensor data correction and management, and;
5. Sensor configuration and directory service

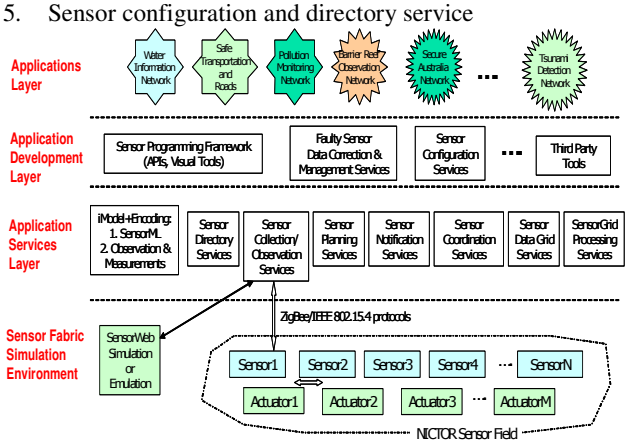

Fig. 3: High-level view of NICTA Open Sensor Web Architecture.

Besides the core services derived from the SWE, such as the SCS, SPS and WNS, there are several other important services in the service layer. The Sensor Directory Service provides the capability of searching for and registering remote 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 publishes and maintains replicas of sensor data collected from sensor deployments. The Sensor Grid Processing Service collects the sensor data and processes it utilizing grid services. The development layer focuses on providing useful tools in order to ease and accelerate the development of sensor applications.

NOSA 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. 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. 4 depicts a prototype instance of NOSA, 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 NOSA. In addition, the design and implementation of the core services are presented in this section.

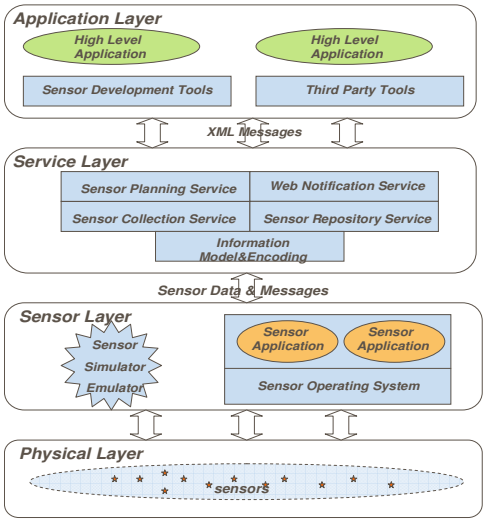

Fig. 4: A prototype instance of NOSA

4. DESIGN AND IMPLEMENTATION

Currently, the primary design and implementation of NOSA focuses on the core services including SCS, WNS, and SPS (which extend from the SWE) as well as the Sensor Repository Service (SRS) that provides a persistent data storage mechanism for the sensor and the observation data.

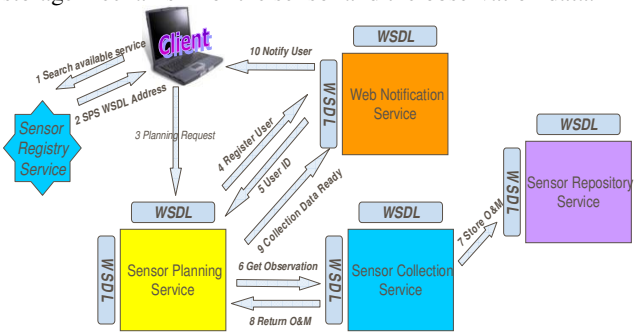

Fig. 5: A typical invocation for Sensor Web client.

Fig. 5 illustrates an example of a client collection request and the invocations between relating services. As soon as the end user forwards an observation plan to the SPS, the service checks the feasibility of the plan and submits it if feasible. The user will be registered in the WNS during this process and the user id will return to the SPS. The SPS is responsible for creating the observation request according to user's plan and retrieving the O&M encoded data from the SCS. 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.

The following sections describe in detail the core set of implemented services implemented in NOSA, namely the SCS, SPS and WNS.

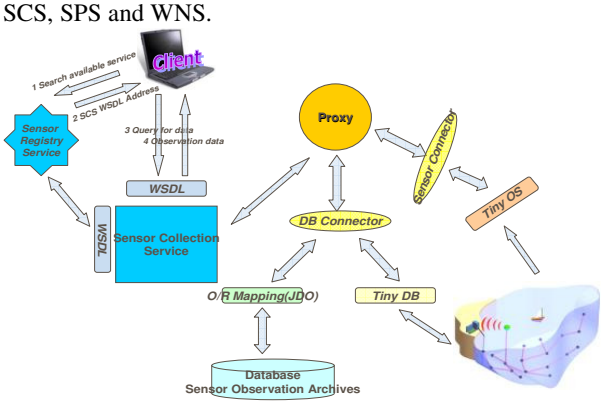

Fig. 6: Sensor Collection Service Architecture.

A. *Sensor Collection Service*

Within the core services of NOSA, the SCS is one of the most important components residing in the service layer. The SCS is the fundamental and unique component that communicates directly with sensor networks, it is responsible for collecting real time sensing data and then translating the resulting raw information into a XML based O&M encoding for other services to utilize and process. The SCS is the gateway for entering into the sensor networks from outside clients. The design of the SCS provides an interface to both streaming data and query based sensor applications that are built on top of TinyOS [9] and TinyDB [10] respectively. Fig. 6 illustrates the architecture of the SCS. The service conforms to the interface definition that is described by the OGC SCS Specification and has been designed as a Web Service which works by connecting via a proxy to either real sensors or a remote repository database. Clients need to query the Sensor Registry Service to retrieve an available SCS WSDL address a data query request is then sent via SOAP to the SCS in order to obtain the resulting encoded 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 which 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 simple to extend for further development if alternative sensor operating systems are adopted by the sensor networks, these may include MANTIS [11] or Contiki [12]. Besides a sensor operating system specific connector 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.

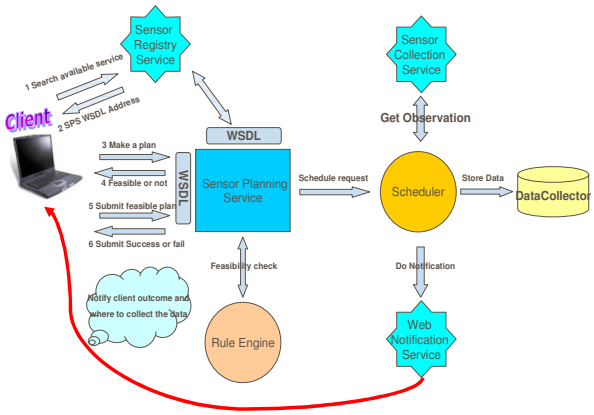

Fig. 7: Sensor Planning Service Architecture.

B. *Sensor Planning Service*

The design of the SPS considers both the short-term and long-term requirements of the 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. 7, the 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 is as following; (i) the scheduler composes a collection request according to user's plan and then invokes the getObservation call on the SCS, (ii) it asks the DataCollector to store the resulting observation data for users to collect afterward, and (iii) sends notification to the WNS indicating the outcome of the collection request. The time of the execution in the scheduler varies based on the requirements of the user's plan. The client receives a response indicating that ther plan will be processed right after the plan is submitted to the SPS. The scheduler deals with the remaining time consuming activities. The client may get the notification from the WNS as soon as the WNS receives a message from the scheduler, the client can then collect results from the DataCollector.

C. *Web Notification Service*

The current design of WNS is displayed in Fig. 8, it 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.

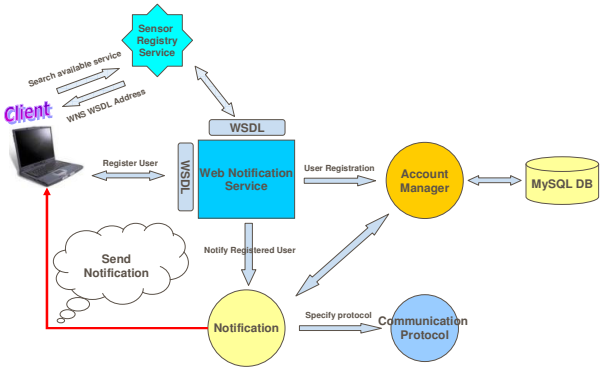

Fig. 8: Web Notification Service Architecture.

5. PERFORMANCE EVALUATION

The experiment platform for the services was built on TOSSIM (described by [15] 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 [16] which consists of 3 Mica2 Radio boards, 2 MTS300 Sensor Boards, a MIB510 programming and serial interface board. The experiment concentrated 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 for the entire system.

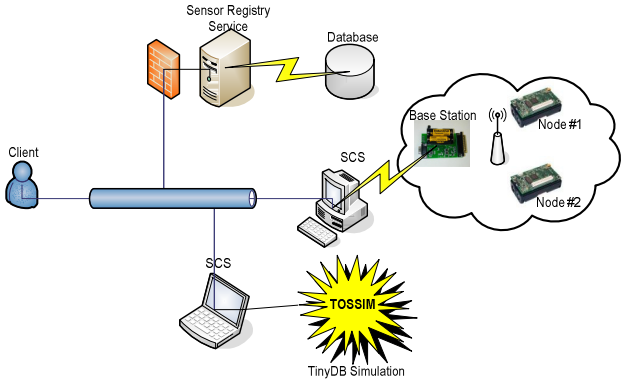

Fig. 9: Deployment of Experiment

Fig. 9 illustrates the SCS as deployed on Apache Tomcat 5.0 server running on two different machines, one of which is hosting the TinyDB application under TOSSIM and the other the Temperature Monitoring application under Crossbow's motes. A Sensor Registry Service is also configured on a separate machine that provides the functionality to access the sensor registry and data repository.

A simple temperature monitoring application has been developed. The application is programmed using nesC [13] and uses simple logic, which 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 or energy saving mechanisms. Before installing the application on the Crossbow motes, the functionality is verified under the TOSSIM simulator. Once the application has been successfully installed onto each mote via the programming board, a wireless sensor network is setup using the 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. 10 displays the results retrieved by a client from the SCS interfaced with the temperature monitoring application. The light intensity level is illustrated by the graph plot; two individual sensors each take recordings at 100ms intervals. Recordings from sensor one are illustrated in the top half of the window and sensor two on the bottom half. A change in the graph plot indicates a variance in the incoming light intensity for each sensor. The left-hand-side column contains SensorML descriptions of the sensors retrieved by the client from the SCS.

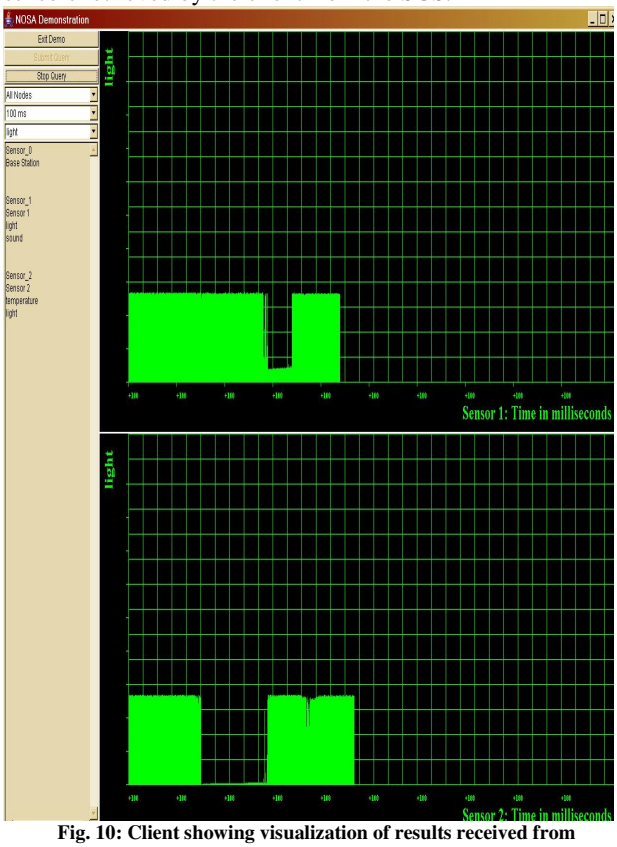

Fig. 10: Client showing visualization of results received from temperature monitoring application called from SCS

Regarding scalability, a simulation program that can stimulate a varying number of clients running simultaneously 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 displayed in the following figures. Fig. 11 displays the result of the auto-sending mode application; the response time is moderate when the number of clients who request the observation simultaneously 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 show in Fig. 12 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 increase. The reason why the query-based approach has very poor performance is due to the execution mechanism of TinyDB. A lot of time is 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 completed or is terminated. A solution to this problem may require the TinyDB application to 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. There are several possible ways to enhance the performance. A caching mechanism may be one of the possible approaches, recently collected observation data can be cached in a proxy for a limited time period, such that clients requesting the same set of observation data can simply read it from the cache.

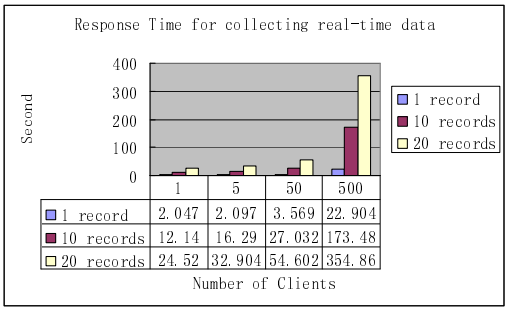

Fig. 11: Performance for collecting auto-sending data.

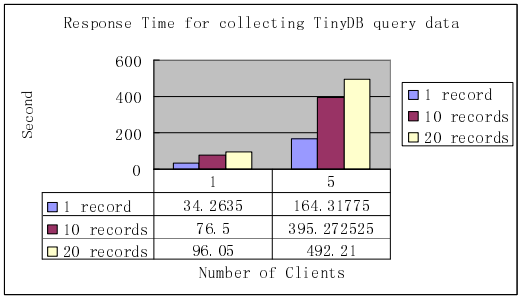

Fig. 12: Performance for collecting TinyDB query data.

However, as the data should be kept as close to real time as possible, it is rather difficult to accurately determine the period of time for the cache to be valid. An approximate 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 time period for setting the cache for each sensor application can vary based on the information the sensor is targeting. Enhancement of query performance may be achieved by utilizing the XQuery of the XML data directly as opposed to querying the real sensor itself and executing the query in a similar fashion to TinyDB.

6. CONCLUSION

NOSA is an implementation of OGC SWE standard which standardizes the vision of Sensor Web. SensorML, O&M, SCS, SPS and WNS are coupled together, to create an integrated platform for registering, discovering and accessing heterogeneous distributed sensors using Web Services. We have introduced the design and implementation of the core services in NOSA. In future work we aim to extend NOSA beyond the SWE and provide additional services for processing information collected from sensor resources accompanied by computational grids. We have detailed the scalability and performance of the prototype SCS which forms the backbone of the core services. The development of NOSA is still in its early stages. Future works include implementing all methods described in the specifications of SWE services but which are not available currently, a caching mechanism for the SCS and extension of notification protocols for the WNS.

ACKNOWLEDGEMENT

We would like to thank Bohdan Durnota for his technical guidance while formulating and designing initial prototype. 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. Special thanks to Ingebjorg Theiss for her contribution of the client GUI snapshot (GUI code presented in Fig. 10).

REFERENCES

[1] http://www.opengeospatial.org/

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

[3] Mainwaring A, Polastre J, Szewczyk R, Culler D, Anderson J (2002) Wireless sensor networks for habitat monitoring, 1st ACM International Workshop on Wireless Sensor Networks and Applications, Sept. 28, Atlanta, GA, USA.

[4] 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. Information Processing in Sensor Networks, April 25-27, Los Angeles, CA, USA.

[5] 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.

[6] Reichardt M (2005) Sensor Web Enablement: An OGC White Paper. Open Geospatial Consortium (OCG), Inc.

[7] http://vast.nsstc.uah.edu/SensorML/

[8] Tham CK, Buyya R (2005) SensorGrid: Integrating Sensor Networks and Grid Computing. CSI Communications 29:24-29.

[9] http://www.tinyos.net/

[10] http://telegraph.cs.berkeley.edu/tinydb/

[11] http://mantis.cs.colorado.edu/index.php/tiki-index.php

[12] http://nescc.sourceforge.net/

[13] http://www.sics.se/~adam/contiki/index.html

[14] Cox S, (2006) Observations and Measurements OGC 05-087r3, Open Geospatial Consortium Inc, http://portal.opengeospatial.org/modules/admin/license_agreement.php?suppressHeaders=0&access_license_id=3&target=http://portal.opengeospatial.org/files/index.php?artifact_id=14034

[15] Levis P, Lee N, Welsh M, Culler D (2003) TOSSIM: Accurate and. Scalable simulation of entire TinyOS applications. 1st Intl. Conf. on Embedded Networked Sensor Systems, November 4-7, Los Angeles, CA, USA.

[16] http://www.xbow.com/Products/productsdetails.aspx

[17] McCarty T, (2003) Sensor Collection Service OGC 03-023r1, Open GIS Consortium Inc.

[18] Simonis I, (2005) Sensor Planning Service OGC 05-089r1, Open GIS Consortium Inc.

[19] Simonis I, Wytzisk A, (2003) Web Notification Service OGC 03-008r2, Open GIS Consortium Inc.