Sensor Web: Integration of Sensor Networks with Web and Cyber Infrastructure

Tomasz Kobialka, Rajkumar Buyya, Peng Deng, Lars Kulik, Marimuthu Palaniswami
University of Melbourne, Australia

ABSTRACT

As sensor network deployments grow and mature there emerge a common set of operations and transformations. These can be grouped into a conceptual framework called Sensor Web. Sensor Web combines cyber infrastructure with a Service Oriented Architecture (SOA) and sensor networks to provide access to heterogeneous sensor resources in a deployment independent manner. In this chapter we present the Open Sensor Web Architecture (OSWA), a platform independent middleware for developing sensor applications. OSWA is built upon a uniform set of operations and standard data representations as defined in the Sensor Web Enablement Method (SWE) by the Open Geospatial Consortium (OGC). OSWA uses open source and grid technologies to meet the challenging needs of collecting and analyzing observational data and making it accessible for aggregation, archiving and decision making.

INTRODUCTION

Sensor networks are persistent computing systems composed of large numbers of sensor nodes. These nodes communicate with one another over wireless low-bandwidth links and have limited processing capacity. They work together to collect information about their surrounding environment, which may include temperature, light or GPS information. As sensor networks grow and their ability to measure real-time information in an accurate and reliable fashion improves, a new research challenge, how to collect and analyze recorded 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 and for power management in schools and office buildings. In many of these cases the software management tools for data aggregation and decision making are tightly coupled with each application scenario. However, as these systems grow and mature, a set of common data operations and transformations begin to emerge. All application scenarios will need to query a sensor network and retrieve some observational data. Some scenarios may require information from historic queries be stored in a repository for further analysis. They may require regular queries to be scheduled and automatically dispatched without external operator intervention. Scenarios may need to share information among themselves to aid in decision making tasks. For example, a tsunami warning system may rely on water level information from two geographically distributed sets of sensors developed by competing hardware vendors. These requirements present significant challenges in resource interoperability, fault tolerance and software reliability. A

solution to these emerging challenges is 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 application or deployment scenario.

A Service Oriented Architecture (SOA) allows us to describe, discover and invoke services from heterogeneous platforms using XML and SOAP standards. Services can be defined for common operations including data aggregation, scheduling, resource allocation and resource discovery. We can exploit these properties by combing sensors and sensor networks with a SOA to present 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. Taking this concept a step further, when interlinked, geographically distributed services form what is called a Sensor Grid which is a key step in the integration of sensor networks and the distributed computing platforms of SOA and Grid Computing. The integration of Sensors Networks with the cyber infrastructure of Grid Computing brings several benefits to the community. The heavy load of information processing can be moved from sensor networks to the backend distributed systems. This separation is beneficial because it reduces the energy and power needed by the sensors, allowing them to concentrate on sensing and sending information. Cross-organizational collaboration is streamlined, because geographically distributed resources can be accessed over common Grid protocols. Data produced by heterogeneous resources can be combined with the aid of common XML formats, eliminating data incompatibility issues.

Figure 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 wishing to predict whether a tsunami is going to occur may query the entire Sensor Web and retrieve the response either from real-time sensors that have been registered on the web or from historical data in database. Data from all sources can be aggregated and used by modeling or visualization tools to aid in tsunami prediction. This can be shared among collaborative parties which may run algorithms or transformations over the raw data with the aid of grid resources. In this way, individual resources can be coupled together to perform complex tasks which where not previously achievable.

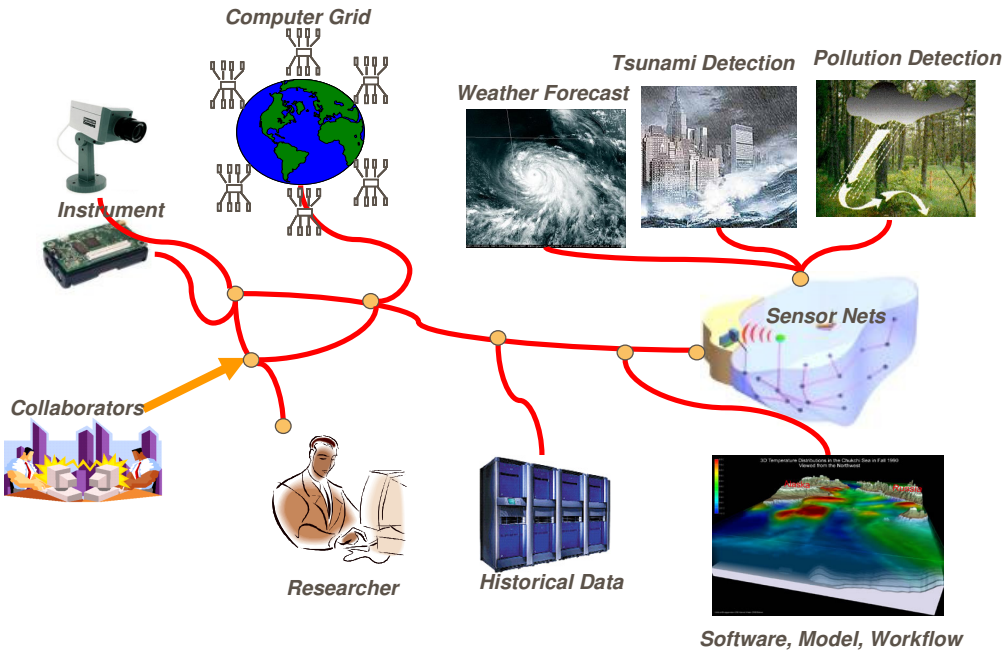

Figure 1. Abstract vision of the Sensor Web

Driven by the growing demand for data sharing among geographically distributed heterogeneous sensor networks the Open Geospatial Consortium (OGC) (Open Geospatial Consortium, 2008), a geospatial standards authority, has defined the Sensor Web Enablement (SWE) method. SWE includes specifications of interfaces and encodings that enable discovering, accessing, and obtaining sensor data as well as sensor-processing services. These specifications form the blueprint upon which the Sensor Web architecture can be developed. In this chapter we present an implementation of the SWE method which we refer to as Open Sensor Web Architecture (OSWA). We explore the technologies and challenges that have arisen from our experiences with implementing the OSWA. A key aim of which is to provide a software infrastructure that simplifies the task of application development for heterogeneous wireless sensor networks. We present a critical analysis of the proposed standards developed for Sensor Web by the OGC including the challenges and benefits of working with standards bodies. We introduce the descriptions of core services and encodings which form the SWE, including Sensor Model Language, Observations and Measurements, Transducer Model Language, Sensor Observation Service, Sensor Planning Service, Web Notification Service and Sensor Alert Service. We describe the design and architecture for each of the core services including the challenges and solutions in developing services for heterogeneous sensor hardware and operating system resources. We provide an analysis of a SunSPOT sensor network deployment using a gesture recognition application deployed onto OSWA which includes design and implementation details and results. Finally we conclude by proposing our vision for the future growth of Sensor Web and our OSWA.

RELATED WORK

The integration of sensor networks and grid computing into a sensor grid was initially outlined by Tham and Buyya (2005). Tham and Buyya introduced some early work in the field by proposing the possible implementation of distributed information fusion and distributed autonomous decision-making algorithms. Gaynor et al. (2004) presents a data-collection-network approach to over come the technical problems of integrating resource constrained wireless sensors into grid applications. This takes the form of network infrastructure with a grid API to access heterogeneous sensor resources, referred to as Hourglass. Reichardt (2005) introduced the Sensor Web Enablement (SWE) method which is an important step in connecting sensor networks with web and cyber infrastructure. The method consists of a set of standard services and encoding which can be used to build a framework for discovering and interacting with web-connected sensors and for accessing sensor networks over the web.

52North (Simonis, 2004) is an initiative supported by the Institute for Geoinformatics at the University of Munster, Germany. 52North has developed an open source software set based on the SWE method. They have developed a set of Java web services based on the specifications and data encodings as well as several SWE clients capable of communicating with services and visualizing observational data. Services developed by 52North are deployed as standard Web Services, and the focus of this project is on geospatial data. Sensor observations are retrieved from a geographic information systems (GIS) database called PostGIS and encoded in SWE descriptions. PostGIS acts as an interface between the service and the sensor systems. Interfaces are defined so that new sensor databases or sources can be easily integrated into the architecture.

The GeoICT group at York University (Tao et al. 2004) has built an OGC SWE compliant Sensor Web infrastructure. They have developed a Sensor Web client capable of visualizing geospatial data, and a set of stateless Web Services called GeoSWIFT. The GeoSWIFT Sensing Server implements all the interfaces of a typical observation service and is capable of communicating with Webcams. They have also created an initial Registry Service.

Microsoft has released the MSR SenseWeb Project (Suman, Jie & Feng 2006) which allows users to publish their sensor data on a portal web site. Microsoft has implemented its own XML ontology along with a set of querying and tasking mechanisms. The ontology is influenced by encodings introduced in the SWE method. Support is provided for sensors running TinyOS and devices such as webcams. Microsoft is not affiliated with the OGC Consortium and there is no support for Linux based operating systems. The current application of SenseWeb is limited to publishing data, with little support for post processing, although it is likely that this will change as the project matures.

OSWA is an implementation of the SWE method that extends the typical Web Service interface definitions by implementing them as Stateful Web Services called WSRF. To our knowledge there are no other published SWE implementations which use WSRF. WSRF opens the door for OSWA services to communicate with data and computational grid resources. OSWA supports heterogeneous sensor resources on TinyDB, SunSPOT, TinyOS and Linux. Implementations such as GeoSWIFT and 52North typically support one or two sensor operating systems, although they include constructs to extend these. Microsoft's SenseWeb Project includes support of TinyOS, but not Linux. No other SWE method implementations support the same diversity of operating systems as OSWA. In OSWA we have introduced a caching method into the service responsible for communicating with the sensor networks, this is a novel feature which improves performance and has not been implemented by any other research groups. There are many research groups working on sensor node middleware solutions. This is middleware which resides on top of the sensor operating system. Some notable projects include MiLAN (Heinzelman et al., 2004), Agilla (Fok, Roman & Lu, 2005), DSWare (Li, Son & Stankovic, 2003) and MagnetOS (Barr et al., 2002). It is our intention to expand OSWA into the sensor operating system level and provide a lower level middleware solution. Future research opportunities include developing a specific service for this purpose.

SENSOR WEB ENABLEMENT

As sensor network deployments grow obstacles begin to arise as an outcome of connecting heterogeneous sensor resources and sharing observational data. A research challenge presents itself in how to collect and analyze observational data from heterogeneous sensor networks and make it accessible for aggregation, archiving and decision making.

The Sensor Web Enablement (SWE) method is defined by the Open Geospatial Consortium (OGC); it includes specifications for interfaces, protocols and encodings which enable implementation of interoperable and scalable service-oriented networks of heterogeneous sensor systems and client applications (Botts, Percivall, Reed, & Davidson, 2007). OSWA is an implementation of the SWE method which consists of the following XML encodings and interfaces:

1. Sensor Model Language (SensorML) – A set of standard models and XML schema for describing sensor systems and processes.

2. Observations and Measurements Schema (O&M) – A set of standard models and XML schema for describing physical phenomena observed by sensor systems.

3. Transducer markup Language (TML) – A XML schema and encoding for describing real-time streaming data recorded by transducers.

4. Sensor Observations Service (SOS) – A web service interface definition for requesting observations from sensor networks and observation repositories.

5. Sensor Planning Service (SPS) – A web service interface definition for scheduling and planning observational requests to sensor networks.

6. Web Notification Service (WNS) – A web service interface definition for the transmission of messages between SWE services.

7. Sensor Alert Service (SAS) – A web service interface definition for publishing and subscribing to alerts from sensors.

Services and encodings presented in the SWE method are decoupled from any particular deployment scenario. Interfaces are defined in such a manner that services responsible for performing independent tasks can co-ordinate with each other to complete a common goal. When coupled together services form a middleware layer which is capable of meeting the complex demands of a heterogeneous multi-user system.

The implementation of service interfaces, based on a common set of standards, has many advantages. Research groups or commercial companies are free to design their own service implementations with the confidence that services will be capable of interacting with one another. A group at the University of Melbourne can build a SPS while another group in Europe builds a SOS. Service descriptions can be published on the World Wide Web so that both groups can use each other's resources, mutually benefiting both teams. Standard models for data encoding such as SensorML, O&M and TML allow data to be shared among implementations and encourage collaboration among SWE implementations. The use of XML as a basis for the schemas allows for platform independence, software can be developed to run on Linux or Microsoft platforms with the confidence that there will be no data incompatibilities. The rich semantic capability of XML is well suited for data exchange, and capable of meeting growing needs in data encapsulation.

The feedback of user experience and contribution of ideas to standards bodies is an important step in developing a community and promoting broader adoption of standards among researchers and businesses. Standards can only mature if they are underpinned by practical experience. However, in an emerging technology such as Sensor Web where feedback from deployment experience may be quite high, this presents an interesting challenge. Development efforts of pioneering adopters who have invested in early standards may seem in vain, as their systems can quickly cease to conform to the most recent standards release. The growth of XML tools is one technology which can ease this burden. A variety of tools exist which facilitate the generation of code from XML schemas. This it aids the developer by reducing some of the tedious works. Ultimately, however, it is important that researchers, developers and standards bodies work together to foster a strong community which can meet these challenges.

Moodley & Simonis (2006) raise some drawbacks on the SWE approach. SWE does not have a formal conceptual model that links all the services and encodings together. This complicates the task of combing data with different granularities of time, space and measured phenomena. The encodings lack explicit semantics, so it is difficult to discover if two or more sets of observed phenomena are related. SensorML is an attempt at partially meeting this requirement. However, although it can be used to describe the sensors themselves, it does not provide a semantic description of the sensor and the phenomenon that it measures.

A vision for Sensor Web is to have service components working together to execute a user request and achieve a common goal as illustrated in Figure 2. A SWE enabled client is interested

in retrieving observational data from a set of physical sensor nodes. The client knows the physical location of the sensors it is interested in, the duration for which it is interested in reading the observational data for (1 hour, 1 day, 1 week, etc.) and the observational data (light, acceleration, temperature, etc.) it requires. The client constructs a request containing this information and sends it to the SPS. The SPS then determines the Universal Resource Indicator (URI) of the appropriate SOS instance by querying a registry of available services. When a SOS instance comes online it automatically registers its capabilities with the Registry. If the client requires alerts, the SPS subscribes to a SAS, if events described in the alert request occur the SAS will automatically inform the WNS, which will perform an action or communicate this information back to the client. Once the subscriptions have been dealt with the SPS will query the appropriate SOS instance which will send the request to a sensor network and retrieve the observational result. The SPS will notify the WNS that the request has been completed; the WNS will forward the location of the observation data and the outcome of the plan to the SWE client. The client can then collect the observational data.

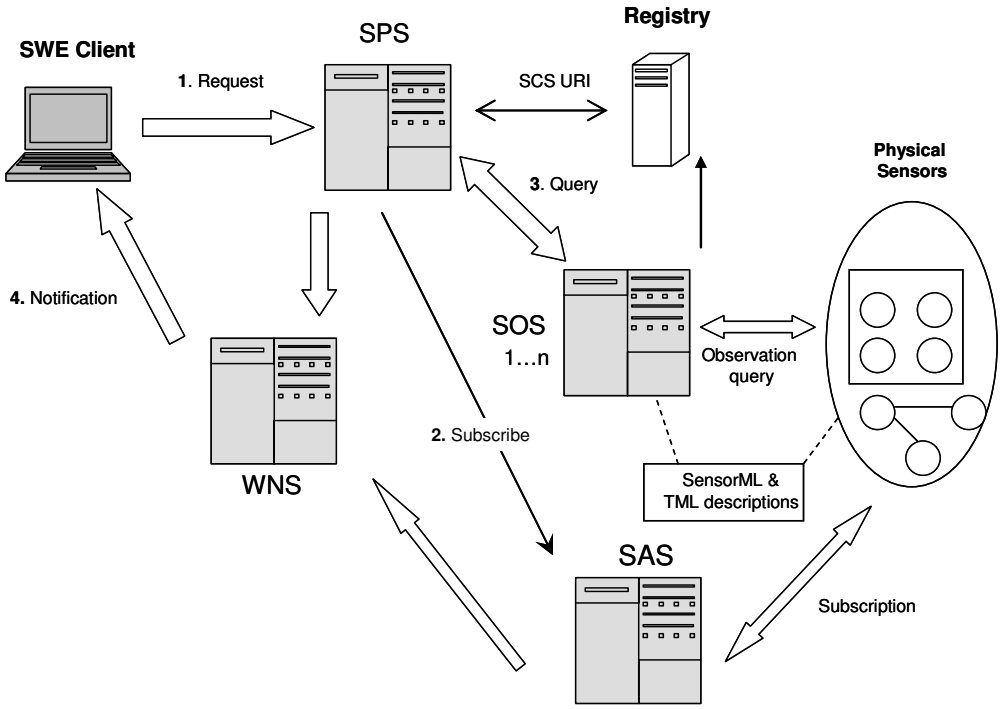

Figure 2. Sensor Web Enablement Service interaction

The SWE presents a framework of service descriptions and XML schemas for communication protocols. A research challenge lies in the design and architecture of the services and specifications in a robust, efficient, platform independent and secure manner. In OSWA we attempt to tackle this challenge, using the SWE method as a basis upon which to build robust platform independent middleware.

OPEN SENSORWEB ARCHITECTURE

The OSWA is an implementation of the SWE method. The various components defined for OSWA are outlined in Figure 3. We can identify four core layers 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.

We use the SWE specifications as a blueprint upon which to base our Service layer. It is important to feed back real-world deployment experience into the design and architecture of services. Ultimately deployment experience should drive standards improvement, although, given the nature of standards this often a lengthy process. It is through the early embrace of emerging technologies that we can demonstrate their advantages that can then lead to improvements in standards. With this in mind, we have implemented the Service layer as a set of WSRF services. The move to WSRF grew out of a need to support on going queries which persist over time to services. These queries require state information which is difficult to implement in traditional Web Services. WSRF changes the dynamics of the SWE framework because services are no longer passive sources for data to be pulled from. They are active data sources which push data out to clients following a publish-subscribe paradigm. In future work we plan to explore the introduction of additional services, such as an Operator Deployment Service which can facilitate the deployment of application specific operators onto the physical sensor networks. Operators could communicate with one another to form an overlay network which would be hardware transparent and capable of enforcing efficient network communication, fault tolerance, resource management and discovery, code management and energy saving schemes.

A key aim of the OSWA is to provide a software infrastructure that simplifies the task of application development for heterogeneous wireless sensor networks. Once services have been deployed we can further abstract the details of the services into interfaces which we couple together to form an API, this forms the basis of the Application Development layer. Developers can then use the API to build and deploy sensor applications, define relationships between services and build job scheduling schemes through an interactive GUI. A challenge in the development of services is to decouple as much application specific logic from the service code base as possible. It can be difficult to develop services which are neutral to the deployment scenario but still fulfill the idiosyncrasies of a particular application. For example, a set of services which comprise a tsunami monitoring application may also be used for pollution detection. Both of these applications may have the commonality of measuring water temperature or displacement from the same set of sensors but have quite different post processing, scheduling and result outcomes. A common approach is to express these idiosyncrasies using a XML model. Although this often introduces additional computational processing time which may not be acceptable in real time applications, it is important that XML models can provide the semantic descriptions necessary to encapsulate this information.

The Sensor Fabric layer includes the Operating System and application code deployed onto physical sensors which allows them to record observations and network among themselves. Currently it is up to developers to program and deploy applications at the Fabric layer. This is not an ideal situation as it requires the programmer to directly interface with the sensors and manage the storage, processing, recording and transport of observation data. Furthermore physical access to individual sensors is required, making it difficult to update code on large numbers of remote sensing nodes. A solution to this problem is to deploy a sensor node middleware onto the sensors themselves. The middleware acts as an interface to the underlying operating system and provides code management, allocation and migration facilities. It is our intention to expand OSWA into the fabric layer and provide a multi tier middleware solution. The Operator Deployment Service is a step in this direction.

Technologies such as Java, Tomcat, XML, SOAP and Web Services provide great opportunities for developing platform independent applications but come with a cost. OSWA is written in the Java programming language. Java was chosen because it is a platform independent object oriented programming language, software libraries released by sensor hardware vendors such as Crossbow and Sun are available as Java Archived Repositories (JAR) files which are simple to use. A disadvantage of using Java is that it is not as fast in its execution time when

compared to a lower-level language like C. Java comes with a memory and resource footprint which may affect performance when large numbers of simultaneous requests are to be processed by services. However, constant improvements in JVM technologies mean this situation can only improve with time. The platform heterogeneity and ease of programming outweigh any disadvantages associated with Java.

A challenge of OSWA is how to support ongoing sensor queries which persist over time to heterogeneous sensor networks. Traditional Web Services are stateless, making it difficult to create and maintain persistent relationships between services. Stateful Web Services provide access to data values that persist over time and evolve as a result of Web Service interactions. The Web Service Resource Framework (WSRF) defines conventions for managing state so that applications discover, inspect and interact with stateful resources in standard and interoperable ways, as defined by the OASIS standards body. Java WS Core is a component of the Globus Toolkit, a set of software components for building distributed systems and it is a popular Grid middleware platform. WSRF is underlined by a notification-subscription interaction pattern. A client subscribes to a service resource and if any changes of state occur on that resource, the service will automatically notify the client of the changes. This eliminates the need for a client to poll the service for changes, as is typical from Web Services, thus reducing the network traffic among services and improving performance. The introduction of WSRF is a key step forward in evolving sensor web technologies into a sensor grid. Services are deployed on an Apache Tomcat container. Tomcat is a servlet container which provides an environment for java code to run on. It is written in Java and is a stable and free technology maintained by volunteers.

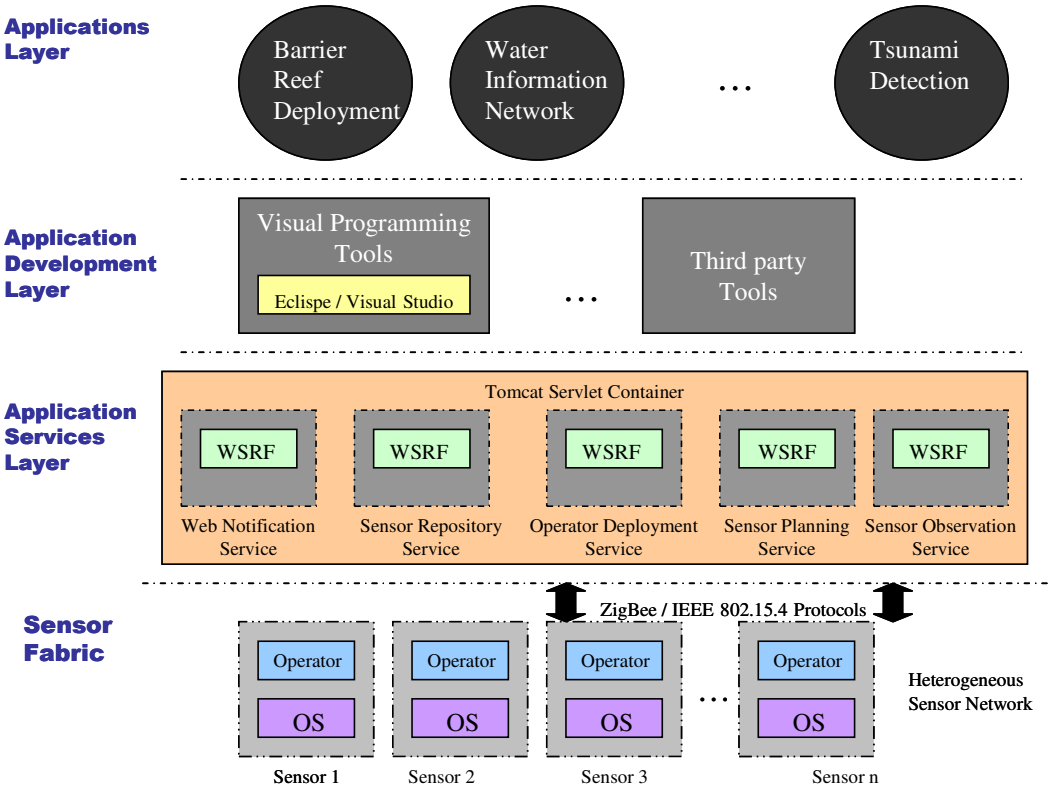

Figure 3. High level view of OSWA

SOAP is used as the communication protocol between clients and services; SOAP relies on XML as its message protocol and HTTP for negotiation and transportation. The use of XML comes with a processing burden. Transformations need to be performed between the data views

of XML and Java object. Managing these relationships manually can be cumbersome and error prone. One solution is to automatically generate Java objects from XML schema using a Java-to-XML binding framework like XMLBeans.

For the remainder of this section we discuss each of the components defined in SWE method, introduce the architecture of these components as implemented by us in the OSWA, and explore the relationships between services and encodings.

Sensor Model Language

SensorML is used to describe the processes and processing components associated with the measurement and post-measurement transformation of observations (Botts, 2007). A process is any entity that takes an input, applies a set of well-defined methods, and results in an output. Processes can be linked together into executable process chains which describe the mapping from input to output between components. This conceptual model for processes in SensorML is illustrated in Figure 4. Process chains are useful in deriving high-level information, which is not otherwise attainable from a single process. For example, a process chain could include the retrieval of raw observational results and the on-demand processing of those results into more meaningful output. SensorML is particularly useful in describing sensor systems and in the processing and analysis of sensor observations. Observations recorded by sensor systems and encoded in the O&M specification can be encoded within SensorML and described as a SensorML process. SensorML is robust enough to handle the processing of data from virtually any sensor whether mobile, in-situ or remotely sensed, or active or passive.

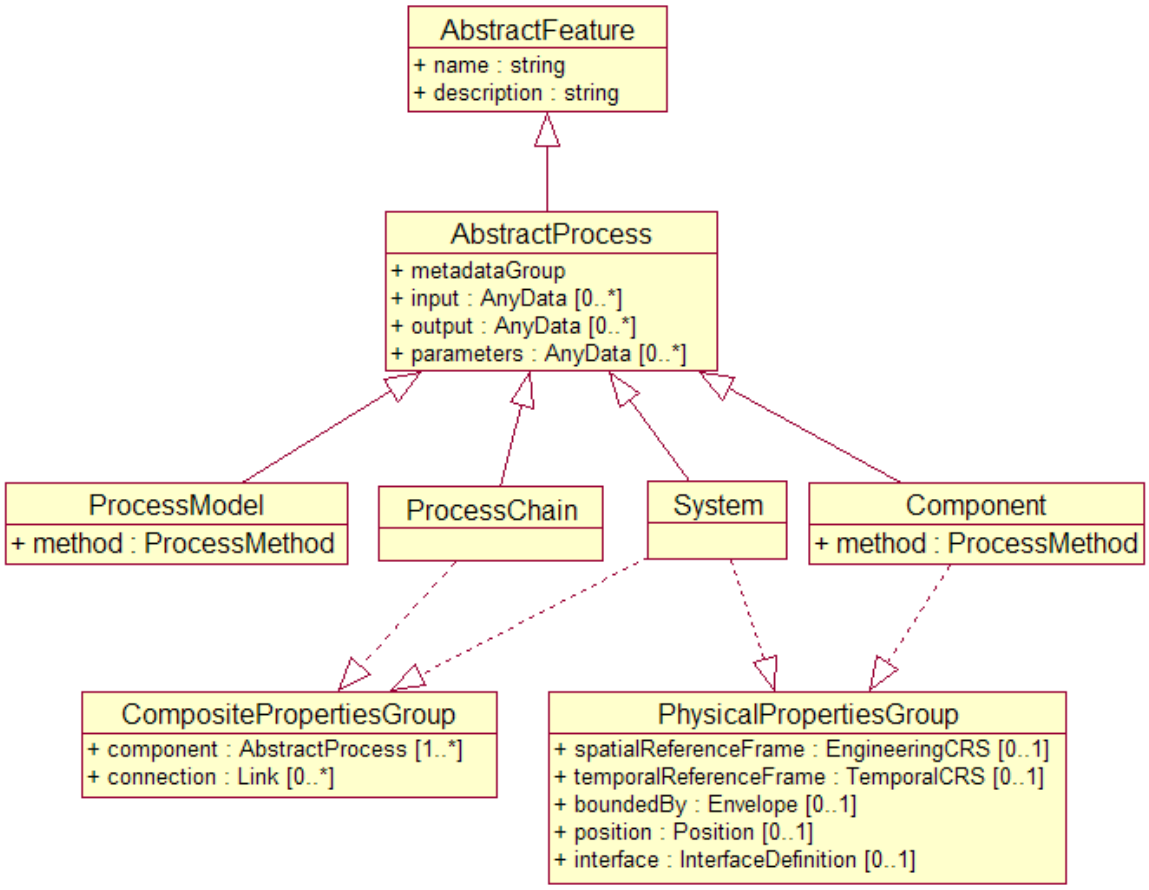

Figure 4. Conceptual Model for SensorML processes (Botts, 2007).

The SOS uses SensorML to describe the capabilities and metadata of any available sensor nodes. The SPS accepts user scheduling plans described as SensorML processes which it then executes. In OSWA Java objects defined by SensorML models are derived with the aid of XMLBeans. These objects are then used by the two services, to encode or decode the XML data. A SWE client may also use these objects as necessary.

Observations and Measurements Schema

The O&M schema is an encoding for observations and measurements retrieved from a sensor network by the SOS. The purpose of the O&M is to alleviate the need for sensor-specific or research independent data formats for describing data retrieved from sensor networks. An observation is any event which has a value that describes some phenomenon. The term measurement is used to identify a numeric quantity associated with the observation. The phenomenon is a property of an identifiable object, which is the feature of interest of the observation (Cox, 2007). For example, if a sensor network is deployed in a room to measure the light intensity, the observed property would be lux, the photometric unit for describing illuminance and the feature of interest would be light. An Observation model identifies the real-world observation target for which observations are made; an extract of this model is illustrated in Figure 5. This includes the value of the observed property and may include a description of the process used to generate the result. Using our light example the value recorded in an office might be 320 lux, the process could be the procedure used for recording light. Constructs exist for

describing the sampling time and result time for a time sequence of observations. An observation may have metadata associated with it, such as a geospatial location. Observations can be composed into collections, which share some commonalities such as the same sampling time or the same feature of interest.

The SOS is responsible for returning observational data encoded in the O&M specification, which can be real-time data retrieved from a sensor network or archived data. In OSWA, O&M objects are generated with the aid of XMLBeans, these are used by the SOS and by SWE clients.

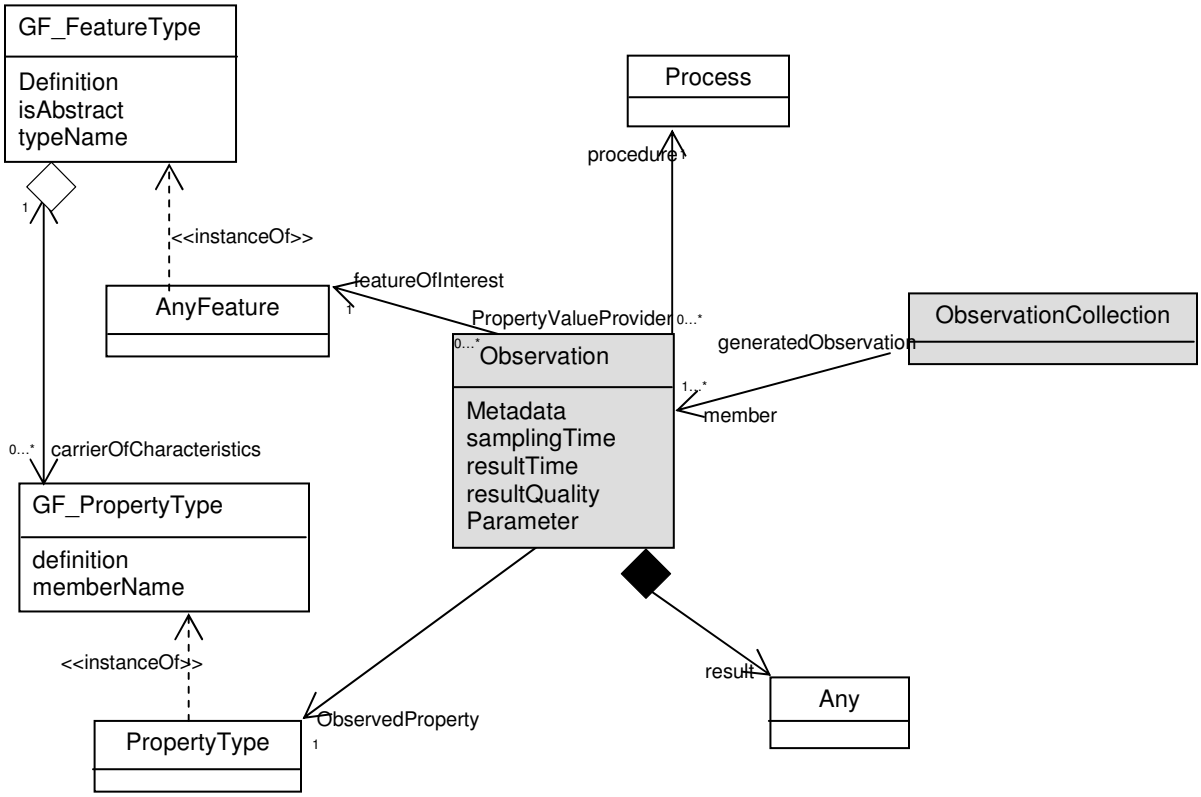

Figure5. O&M Model Extract (Na, 2007)

Transducer Markup Language

TML defines a set of models which are used in describing the data captured from transducers, along with methods for communicating real-time sensor data. TML includes information necessary for the post-processing of data by the eventual recipient. A transducer is typically a group of devices which can capture real-time data from multiple phenomena. Transducers work in two ways, they can sense data or data can be sent to them to produce some sort of predetermined result. An advantage of TML is that it makes it possible to share data across application domains. It can be used for retrieving data from live sources or archived sources.

TML is a recent addition to the SWE set of encodings, in our OSWA we are yet to implement it and have no access to a transducer device. The SOS is primary responsible for returning the sets of transducer results encoded in TML. It is the responsibility of the SOS to communicate with the transducer device.

Sensor Observation Service

The SOS is a service responsible for forwarding requests to the sensor network and retrieving the recorded observational results. It acts as an intermediary between the client and real time or archived sensor observation data. It provides a common interface to communicating with sets of heterogeneous networks and archived data sources. The SOS communicates with the sensor network via a base station node which acts as a bridge between the service and sensor nodes. Observational results retrieved from the sensors are returned to the client encoded in the O&M specifications. Metadata describing the sensor platform (hardware capacity, sensor types) is returned in the SensorML encoding. SWE enabled clients can connect directly to the SOS to retrieve near real-time data, or for complex queries the SPS can facilitate lifecycle management and coordinate data retrieval from multiple SOS instances simultaneously.

The SOS is composed of three core operations *DescribeSensor, GetObservation* and *GetCapabilities*. *GetObservation* is responsible for returning observations from a sensor network, *GetCapabilities* returns metadata information about the SOS service and *DescribeSensor* is responsible for returning metadata information about the sensor nodes. Other operations include *RegisterSensor*, and *InsertObservation,* which are used to support transactions along with six enhanced operations including *GetResult*, *GetFeaturesOfInterest*, *GetFeaturesOfInterestTime*, *DescribeFeatureOfInterest*, *DescribeObservationType* and *DescribeResultModel*.

In OSWA we implement all the core operations described in the specification, they include *GetObservation*, *DescribeSensor* and *GetCapabilities*. These interfaces provide sensor descriptions and observational data from a heterogeneous set of sensor networks which include TinyOS running on Mica2, MicaZ and Imote2, NICTOR sensors developed by NICTA running Linux and SunSPOT sensors running Java. The architecture of the SOS is illustrated in Figure 6.

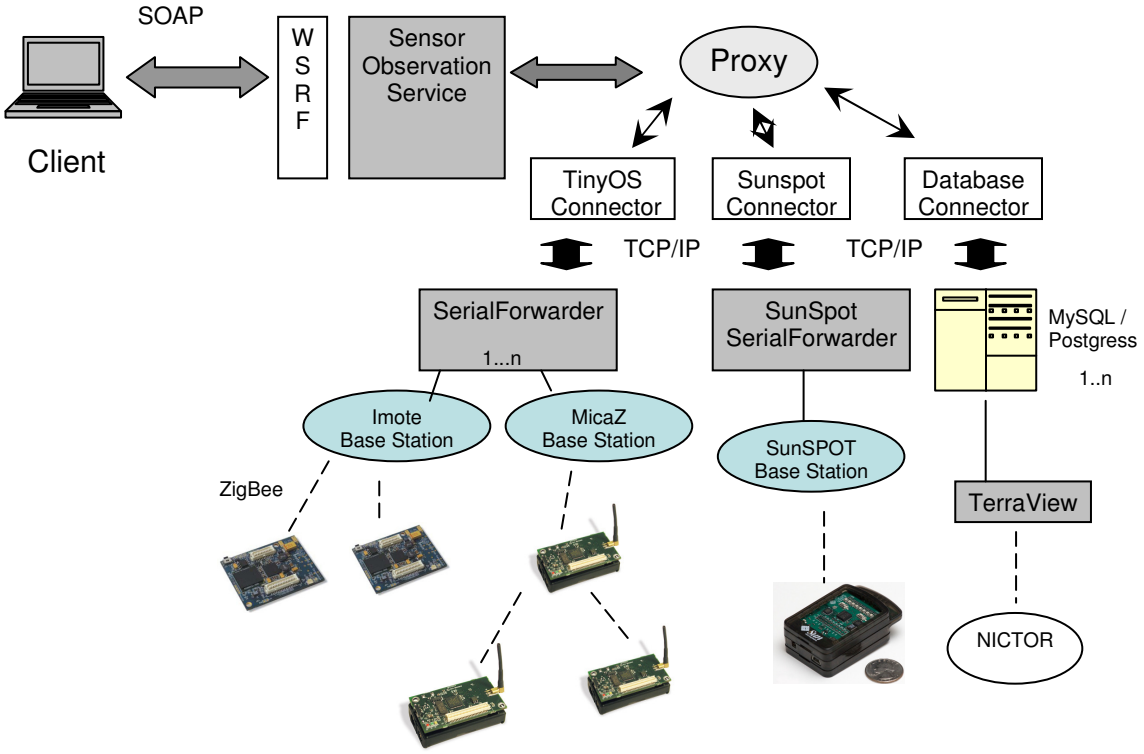

Figure 6. Architecture of the SOS

A client connects to the SOS interface through the Globus WSRF library. The client can be a user connecting with a SWE client or a service, such as the SPS, initiating the connection on behalf of a user executable plan. Once the connection has been negotiated calls to the *DescribeSensor* or *GetCapabilities* operations are directed through a proxy class to a database connector. The database connector communicates with a PostgreSQL database to retrieve metadata information describing the sensor hardware (*DescribeSensor*) and the offerings that are available from the SOS (*GetCapabilities*). This information is encoded in SensorML and returned to the client, which will use it to determine if the SOS service is capable of fulfilling an observational request. The client will send a *GetObservation* request to retrieve observational data from the sensor network. The request will contain a SQL-like syntax, information encapsulated in the syntax may include; the sensor network type (vendor), the location of the network, the observed phenomenon (light, temperature, acceleration, etc.), a threshold value (only values temperature values greater than 0 degrees Celsius), the duration for which to sense the data, the update frequency for observations, and the network ID's of sensors to be queried. This information may vary with each application context. The proxy will distribute the query request to the appropriate network connector. It is the responsibility of the connector to communicate with the base station and retrieve the observational results. In most cases this is facilitated by a daemon which forwards queries to the serial port that the base station is connected to. The observational data recorded by the sensors is then published on a TCP/IP port and is available for the connector class to retrieve. In some cases, such as for the NICTOR sensors, this interface occurs via a database. NICTORS are unique in that they publish their observational results directly to a MySQL database. Once the observation data has been collected it is encoded into the O&M specification and returned to the client.

As part of a continued effort to enhance the performance of the SOS we have introduced a cache mechanism into the SOS architecture. A bottleneck of the SOS has been the inability of sensor networks to handle more than one query at a time, without some special operators or middleware deployed onto the sensors. When an observational query is sent to a sensor network that query must return a result before a consecutive query can be fulfilled. In a system with multiple concurrent users, all users are interested in an immediate response, which can lead to a major performance bottleneck. To overcome this bottleneck, a cache mechanism has been developed that consists of a two-level cache chain incorporated with query aggregation rules and a partial matching scheme to improve accuracy and performance. The cache mechanism handles the parsing of cached queries and the predicting of results for current queries. Query results are stored in a local cache, incoming queries are checked against historical ones and if the query strings are similar and lie within a timeout the cached results are returned instead of sending a query to the sensor network.

The key components of the Cache mechanism are illustrated in Figure 7: where the cache interfaces with the proxy and connector components of the SOS architecture.

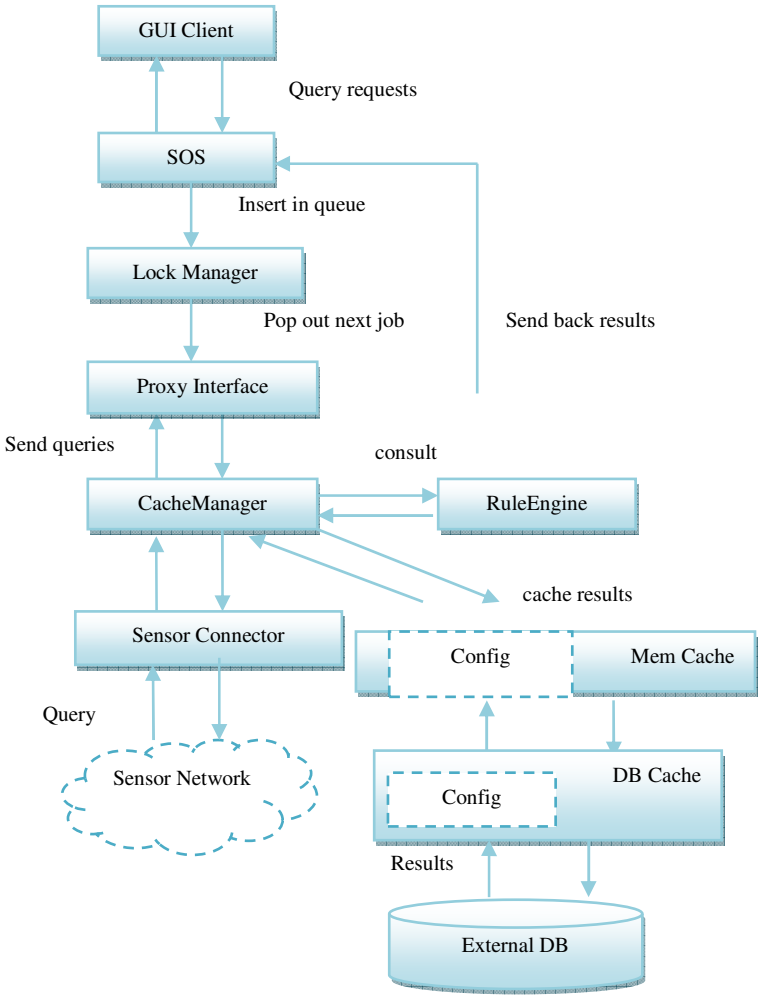

Figure 7. Architecture of Cache Mechanism

A CacheManager maintains a cache chain, which gives orders of precedence to the available caches. Upon receiving an observation request the CacheManager checks with the RuleEngine to determine if it should query the Cache. The RuleEngine maintains a series of parameters, which are designed to improve the accuracy of the Cache in order to maximize the cache hit rate. The query SQL string is used as a key for caches. The CacheManager checks each Cache in the cache chain and returns a hit if the SQL string exists. If the RuleEngine determines that the CacheManager is unlikely to retrieve a cache hit, i.e. if the Cache is full, or entries are expired or don't match the key, control is returned to the CacheManager which redirects the query to the physical sensor network. When the sensor network returns observational results the proxy will forward these to the CacheManager which will update the cache chain along with parameters in the RuleEngine.

The RuleEngine analyses the observational results and makes changes to the tolerance parameters of Estimate and Threshold. Estimate is a numeric value given to the rate of change observed in the environment by the sensor network. A small Estimate is given to a rapidly changing environment; a larger estimate is assigned to a stable environment. The Estimate is determined by analyzing the difference between consecutively recorded observations returned by the sensor network. The RuleEngine uses the Estimate to determine if a query should be checked against the Cache or not. If the current time exceeds the last update time of a cached result plus the Estimate, then the request is redirect to the sensor network, because the environment is

changing fast, thus any observation cached will already be out of date. This ensures that the Cache is queried only in circumstances where we are confident that a cache hit is as close as possible to a correct reflection of the physical environment. The Estimate is initialized in the configuration file and dynamically changed by the RuleEngine at runtime to reflect the changing environment. The Threshold is a dynamically changing parameter that adapts to the cache size, frequency of entries being cached and the values of entries.

The Caches are strung together to form a cache chain. Typically, the memory cache takes precedence over the database and is limited in size. Upon receiving an incoming observation request, the SOS calls the CacheManager. If a cache is hit, the cached result is returned as the observational result to the client. If a cache miss occurs, the CacheManager will insert the observation request into a queue of observational requests to be retrieved from the sensor network. When an item is removed from the queue, a second check is made by the CacheManager to determine if any observational results have been cached while the current request has been waiting in the queue. If a cache hit occurs, the result is returned to the client. If there is a miss, the SensorProxy will query the physical sensor network. The observational results retrieved are then written by the CacheManager to the Cache, following the precedence of the cache chain, and fed back to the RuleEngine. A cache hit can occur if two cache keys (query strings) are "similar", which means that they are exactly the same or their values lie within the tolerance Threshold. The responsibility of determining whether two keys are similar is given to the Comparer. If a cache miss occurs, the Comparer can still use existing cached entries to achieve a cache hit by using partial matching schemes. When writing a new entry to the cache, if a cache entry already exists with the same key, the new observational results will replace the stale data. Otherwise, a new key entry is allocated and a new result entry is placed into the cache. If the cache is full, we employ two eviction strategies, Least Recently Used (LRU) and least rank. Although other more complex cache eviction strategies exist, LRU is our primary choice because it works well in an environment where there is a high temporal locality of reference in the workload (that is, when most recently reference objects are most likely to be referenced again in the near future). After being stored in the cache, the result is feedback to the RuleEngine.

Sensor Planning Service

The SPS is responsible for providing a high level planning, scheduling, tasking, collection, processing, archiving of requests for all services. A SWE client can submit a SensorML encoded plan to the SPS, the plan must contain the observation request, location of the sensors, duration of the request and any other relevant metadata or post-measurement processing requirements. The SPS is responsible for discovering available SOS instances from a registry of services and capabilities, processing and scheduling the plan, managing subscription requests to the SAS and forwarding notifications to the WNS. Use of the SPS should be limited to circumstances where observational data from more than one SOS instances is required, where the connection duration may persist over some period of time, and where post-processing, such as data aggregation from several sensor networks or archived observation sources is required. We define these types of requests as complex observational requests.

In a similar fashion to the SOS there are both mandatory and optional operations which are required to be implemented. *GetCapabilities*, *DescribeTasking*, *Submit*, and *DescribeResultAccess* are all mandatory operations. *GetCapabilities* is responsible for returning metadata information regarding the capabilities of the SPS. *DescribeTasking* returns information about all parameters, which need to be set by a client, to perform a *Submit* request. The *Submit* operation submits the user plan for scheduling and execution by the SPS. *DescribeResultAccess* returns the SOS location that the SPS communicates with in order to access observational data

from a particular sensor. Optional operations which may also be implemented include *GetFeasability*, *GetStatus*, *Update* and *Cancel*. *GetFeasability* provides feedback to the client on the feasibility of executing the plan, this includes checking the validity of the parameters, and locating a SOS instance and checking the instance can fulfill the request. *GetStatus* returns the current status of the request. *Update* allows the client to update a previously submitted plan and *Cancel* terminates a plan.

In the OSWA we implement all the mandatory and optional operations. The architecture of our SPS is based heavily on earlier design work our team did in developing a Grid resource broker called Gridbus. The Gridbus Broker is a scheduler for distributed data-intensive applications on global grids (Venugopal, Nadiminti, Gibbins, Buyya, 2008). The architecture of the SPS is illustrated in Figure 8.

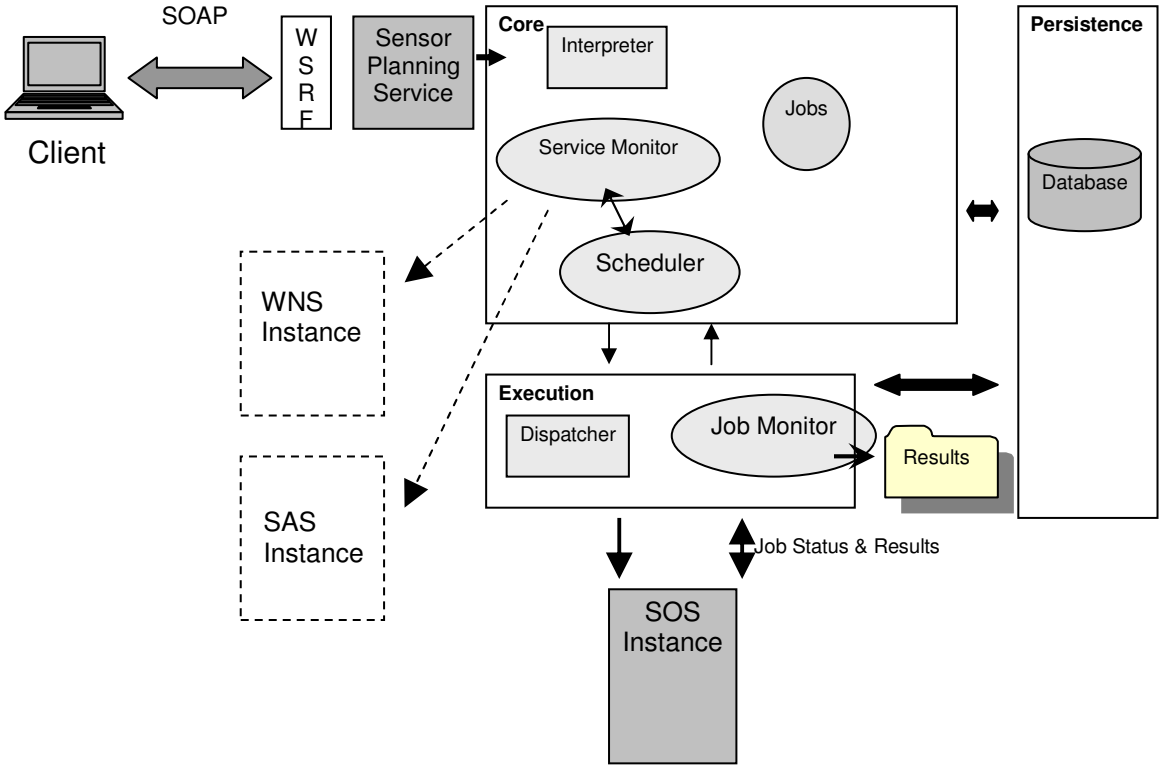

Figure 8. Architecture of the SPS

A client will initialize a SOAP connection with the SPS using Globus WSRF libraries. If it is the first time that the client is connecting to the SPS it may query the *GetCapabilities* operation in order to retrieve metadata information about the service. The operation may return details describing the hardware and software of the server, the organization responsible for operating the server, the accessible sensor systems (in the form of SOS URI's) along with the physical location and observational phenomenon recorded by the sensor networks. The client may send a *DescribeResultAccess* request to determine the SOS location responsible for a particular sensor it is interested in. This data can then be used in the construction of the plan. The client will also need to discover what parameters it needs to set in order to perform a *Submit* operation. It is the responsibility of the *DescribeTasking* operation to provide this information. The client uses *DescribeTasking* response to construct a user plan that will contain all the information necessary to execute a *GetObservation* operation on a SOS instance, along with any pre-processing, post

measurement processing, archiving, notification, duration and any other tasking it wishes to perform on the observational data.

When the client performs a *Submit* operation an interpreter decodes the user plan and constructs a job. A Job is an object that encapsulates all the content described in the user plan. If the user plan specifies the client to be notified of the Job completion via the WNS (email, SMS, Instant Message, phone call), the SPS registers the Job with the WNS. The state of the Job at anytime throughout its lifecycle is maintained by a Hibernate database. The Job is placed in a queue and scheduled for execution. A Service Monitor thread sits in the background and discovers any new SWE service instances that may be accessible on the network. When the Job is ready for execution a dispatcher subscribes to the SOS instance identified in the Job, the dispatcher calls the *GetObservation* operation which communicates with the sensor network and retrieves the observational results. Notifications are sent by the SOS back to the SPS as the Job executes, these are forwarded to the Job Monitor which updates the Job state in the database. Whenever the SPS receives a *GetStatus* request it retrieves the current state of the job from the database and returns it to the client. Once the observation is completed the SOS returns the results to the Job Monitor encoded in the O&M specification. The results are then written to the local file system and if any post processing requirements are included in the plan these are performed. Post-processing may include error checking, performing additional calculations or data transformations. Upon completion the data is returned to the client, for more complex plans which persist over time, or which may require alternative communication means (SMS, email etc..) the WNS is notified which in turn notifies the client over the clients preferred protocol.

Sensor Alert Service

The SAS specification provides an interface for sensor nodes to advertise and publish alerts. Clients can subscribe to data that matches specific criteria, for example when the battery is low or if an observation value is returned above or below a threshold value. When this data becomes available the SAS notifies the client. Intelligent sensors can connect to the SAS and make their resources available to clients for subscription. The SAS uses the Extensible Messaging and Presence Protocol (XMPP), a decentralized open XML-based protocol targeted at near real-time communication, to publish sensor data.

The SAS specification outlines ten operations that can be requested by a client and performed by a SAS server. The required operations include *GetCapabilities*, *Subscribe*, *CancelSubscription*, *RenewSubscription*, *DescribeAlert*, and *DescribeSensor*. The remainder of the operations are optional, they include *GetWSDL*, *Advertise*, *CancelAdvertisement* and *RenewAdvertisement*.

The SAS is a new addition to the SWE method and therefore it is yet to be implemented in the OSWA. We will briefly describe the proposed functionality of the interfaces, which are illustrated in Figure 9.

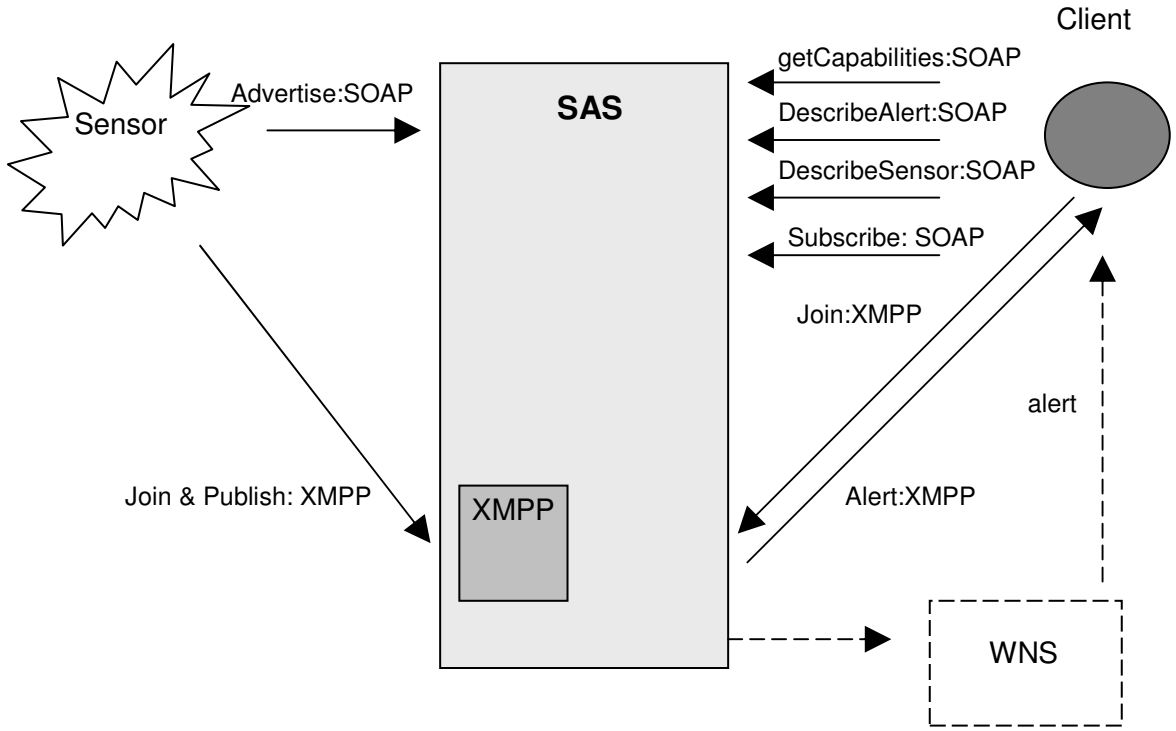

Figure 9. SAS overview (Simonis, 2007)

GetCapabilities returns metadata describing the abilities of the SAS implementation. The *Subscribe* operation allows clients to subscribe to the advertised capabilities. *CancelSubscription* terminates the subscription and *RenewSubscription* restarts the subscription. *DescribeAlert* returns the structure of the data observed by a particular sensor. This includes the physical phenomenon being observed and format of the recorded data. Upon receiving a return value the client has enough information to *Subscribe* to the alert. *GetWSDL* returns the WSDL description of the SAS interface. *Advertise* allows sensors to advertise their capabilities to a SAS instance. Sensors or data producers calling *Advertise* will be added to the sensor pool and are available to clients for subscription. *CancelAdvertisement* allows the advertising sensor source to terminate the relationship and be removed from the pool. *RenewAdvertisement* restarts the advertisement.

Web Notification Service

The WNS is an asynchronous messaging service whereby users can subscribe and receive notifications, over one of several protocols, on any interesting phenomena that may occur in any SWE service. Any service can call the WNS to send a notification. The WNS handles two notification methods, one-way, where notifications from services are forwarded to the client and two-way where a response is expected from the client. A variety of communication clients can be programmed into the WNS model, including email or SMS. This allows for users to program their mobile devices to accept notifications describing processing errors or completed SPS plan requests.

Mandatory operations defined for the WNS include *GetCapabilities*, *RegisterUser*, and *DoNotification*. Optional operations are *DoCommunication* and *DoReply*. *GetCapabilities* works in a similar fashion to previously mentioned services, returning metadata about the WNS. *RegisterUser* allows a client to register to receive notifcations and *DoNotification* initiates the

notification of the registered user. *DoCommunication* is called to initiate the communication with the user, and *DoReply* accepts a user response to a two-way notification.

In the OSWA we implement the mandatory operations described in the WNS specification. The mandatory operations are used to perform one-way communication whilst optional operations are only required for two-way notification. The architecture of the WNS is illustrated in Figure 10.

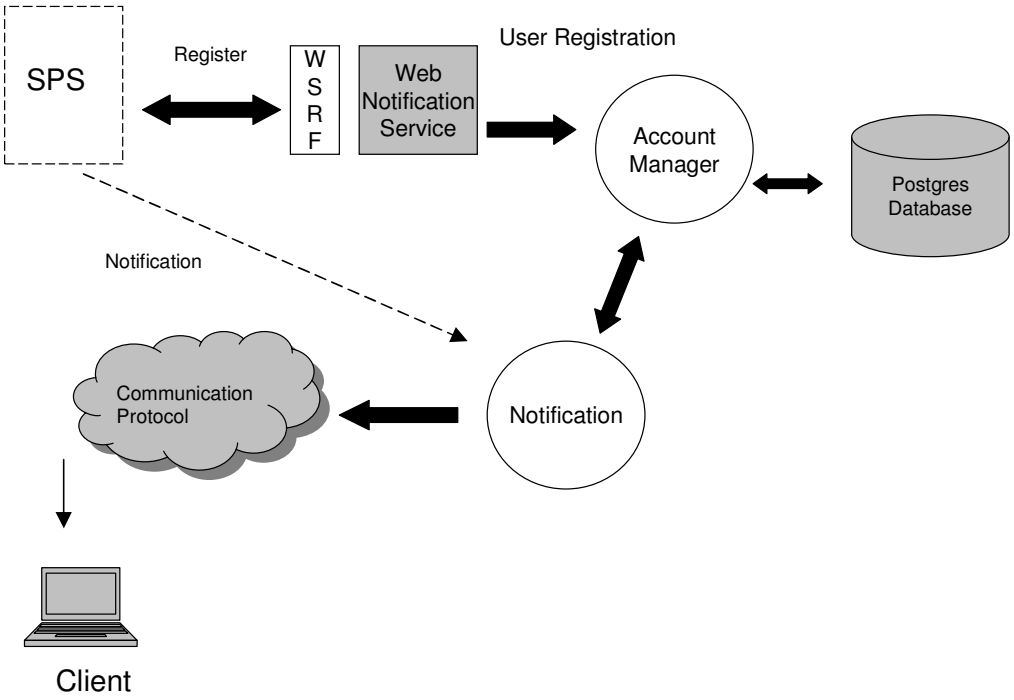

Figure 10. The architecture of the WNS as implemented in OSWA

Clients can discover the capabilities of the WNS by calling the *GetCapabilities* operation, this returns the available communication protocols implemented by the WNS. When a client calls the *RegisterUser* operation on the WNS the user is assigned a registration ID by the Account Manager which is then stored in a Postgres database. The client case can be any SWE service, however it will typically by the SPS, as this is responsible for managing the scheduling of user plans. When some interesting event occurs the SPS will send a *DoNotification* request to the WNS. This is handled by a Notification class which discovers the user details from the Account Manager and notifies an end-user client with an appropriate communication protocol. In the OSWA we implement email as the preferred protocol, although an interface exists so virtually any protocol can be easily added.

In the following section we present the problem of gesture recognition and build and deploy a gesture recognition application using the OSWA to access real-time observational data produced by SunSPOT sensors, transform the observational results and visualize the data.

CASE STUDY: A GESTURE RECOGNITION APPLICATION

From GUIs to multi-touch surface pads, speech to gesturing, the ways we interact with computers are diversifying more than ever before. To demonstrate the usability of the OSWA we

implemented a prototype arm gesture recognition system, trying to free the user from the keyboard and mouse and incorporate a more natural gesture user interface utilizing sensors, machine learning and sensor web. This requires the user to hold a sensor node in their hand and perform a gesture with their arm. Each unique gesture has a different semantic meaning, this may include a letter of the alphabet, moving to the next slide in a presentation or opening and closing a browser window. In this section we introduce the problem of gesture recognition; we outline the idiosyncrasies and challenges in building a gesture recognition system. We introduce the software components which we need to meet these challenges and use the SOS to collect and process gesture data, which we forward to a SWE client for visualization.

Human motion is an inherent continuous event and difficult to predict. Theoretically, the human motion recognition problem is similar to voice recognition which is well studied. The main difference is that human motion occurs in three dimensional space, which requires measurements to be recorded for at least three axes. To recognize human gestures first we need to capture gesture data and transmit it for further processing. This raw data can then be analyzed by recognition algorithms in order to extract some useful meaning or content.

SunSPOT (Sun Small Programmable Object Technology) is an open source software package and hardware sensor node developed by Sun Microsystems. Developers can customize both virtual machine source code and circuit board design to meet their own special requirements, using Java to write applications and deploy them on the physical devices. SunSPOT devices come with a light sensor, temperature sensor, and accelerometer integrated onto the sensor board.

There are two main challenges in for gesture recognition. The first challenge is segmentation, i.e., how to identify the beginning and end of a motion in a multi-attribute data stream. The second challenge is to recognize the segmented stream with a high level of accuracy. To fulfill these requirements there are several challenges which need to be addressed (Li, Zheng, & Prabhakaran, 2007):

- Similar motions may look different: Due to variance in speed and direction, similar motions can have variations in a multi-attribute data stream. Figure 11, illustrates the raw acceleration data produced by gestures of a small (10 centimeter) diameter circle and a larger diameter (60 centimeters) circle.

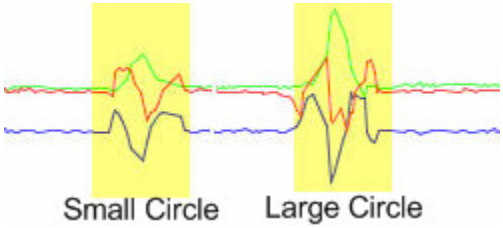

Figure 11. Variations in acceleration data from the X, Y and Z axis produced by similar motions

- Similar motions vary in duration: Different people may perform the same gesture in different ways. Even the same person can not perform exactly the same gesture at the same speed twice. The sensor sampling rate may differ as well. This data series is illustrated in Figure 12.

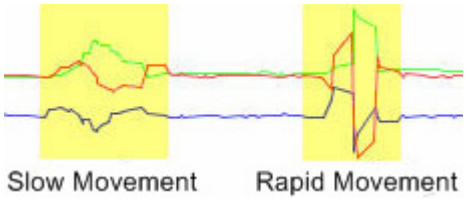

Figure 12. Variations in acceleration data from the X, Y and Z axis produced by similar gestures with different durations

- Similar motions may have different meanings. Illustrated in Figure 13 are the accelerometer readings from three gestures with similar motions, but with different semantic meanings. Complete motions are concatenated by brief transitions, and the motion candidates in a stream can contain these transitions. Hence, the difference between a complete motion and motion candidates with missing or extra segments needs to be captured.

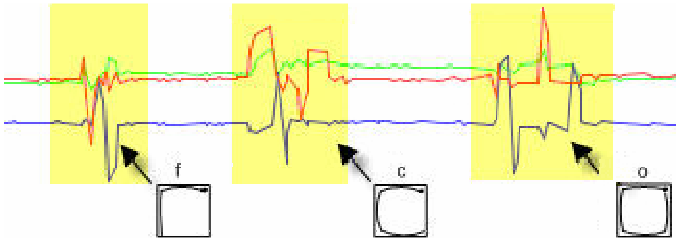

Figure 13. Variations in acceleration data from the X, Y and Z axis produced by the motion of three similar gestures with unique character outcomes

- Different motions may follow similar trajectories but in different directions: For example Figure 14 illustrates a clockwise circle and a counter clockwise circle which follow a similar trajectory but may produce two different results.

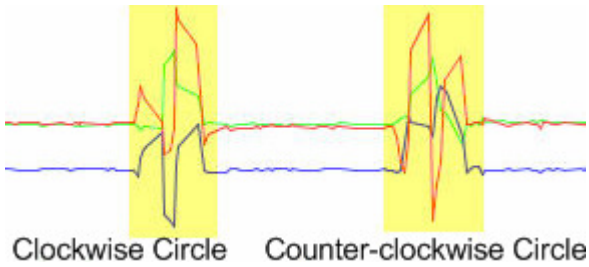

Figure 14. Variations in acceleration data from X, Y & Z axis produced by similar trajectories in different directions

These challenges show that a solution to the gesture recognition problem is non-trivial. For a gesture recognition system we first need to segment the data, i.e., we need to identify the start and end of the data stream. We can achieve this manually with the SunSPOT nodes by holding and releasing a button to explicitly mark the beginning and end of a gesture. To recognize the human motion in the stream we can use a Hidden Markov Model (HMM) (Baum & Petrie, 1966). Hidden Markov is defined as a set of states of which one state is the initial state, a set of output symbols, and a set of state transitions. Each state transition is represented by the state from which the transition starts, the state to which transition moves, the output symbol generated, and the probability that the transition is taken. HMM are especially known for their applications in temporal pattern recognition such as speech (Rabiner, 1989). In the context of gesture recognition, each state could represent a set of possible hand positions. The HMM which holds the highest probability of state transitions could be determined as the user's most likely gesture. HMM need to be trained before they can be used for recognition. It is important to determine the appropriate number of states for each gesture to maximize accuracy and performance.

We implemented the gesture recognition system in the SOS component of the OSWA. We only use the SOS because we are interested in near-real time observational data from the sensor

network. We have no need to schedule the data so we don't use the SPS. Likewise there are no notifications to be sent so we don't use the WNS. The architecture of gesture recognition system with relation to the SOS and its components is illustrated in Figure 15.

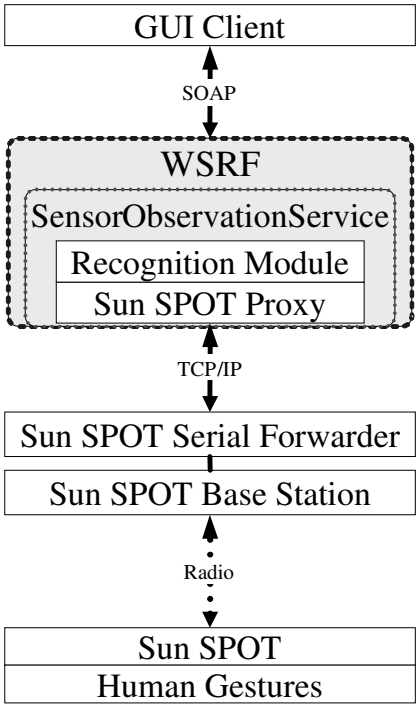

Figure 15. Architecture of the gesture recognition system with relation to existing SOS components

We develop a small application in Java which we deploy on the SunSPOT module, the application uses the onboard accelerometer to capture arm movement and forward it to a base station node. Acceleration data from all X, Y and Z axes (Figure 16) is significant as is tilt on all 3 of these axes. These 6 parameters are later analyzed by the Recognition Module. The base station simply acts as a relay, forwarding packets to the SunSPOT sensor and forwarding the observational results back to a serial port.

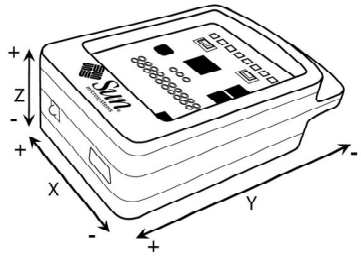

Figure 16. X, Y and Z axes on Sun SPOT accelerometer (SunSPOT, 2008)

One problem with directly using the serial port is that only one application can interact with at any time. A solution to this is the Sun SPOT Serial Forwarder which opens a packet source and lets applications connect over a TCP/IP socket.

To interface with the SOS we implement a connector (SunSPOTConnector) and a recognition (Recognizer) module. The architecture of these components with relation to existing SOS components is illustrated in Figure 17. The SunSPOTConnector interfaces with the

SunSPOT Serial Forwarder. The recognition module is invoked by the SunSPOTProxy to analyze the data series observed by the SunSPOT sensors. The SunSOPTObservationFormatter encodes the observational result returned from the sensors into the O&M format, which is later returned to the SWE client. The Recognizer performs the HMM transformation on the raw observation data. Two open source components are used by the Recognizer, the Gesture and Activity Recognition Toolkit (GART) (GART, 2008) and the Hidden Markov Model Toolkit (CU-HTK) ("HTK Speech Recognition", 2008). GART is a prototyping toolkit for the rapid creation of gesture-based applications, developed by the Contextual Computing Group at the Georgia Institute of Technology. It attempts to minimize the complexity of underlying machine learning algorithms and encapsulates functions provided by CU-HTK. The CU-HTK is a portable toolkit for building and manipulating HMM, it is primarily used in speech recognition research. CU-HTK was developed in partnership with the Machine Intelligence Laboratory at Cambridge University and Microsoft.

Prior running the experiment, we use the SunSPOT sensors to produce a segmented sample of acceleration data for the HMM. We do this by running a small host side application and repeating a set of predefined training gestures. The resulting training data is maintained as a set of XML files that hold all raw gesture samples along with their names and configuration arguments. All consecutive observational data produced during the use of the system is used to improve on the initial gesture library and build up an experience set. This training need only be done once, during execution the CU-HTK loads the experience set from disk and compares it to the recorded gesture data. The identified gesture data is encapsulated in the O&M encoding by SunSOPTObservationFormatter and returned to the SWE client

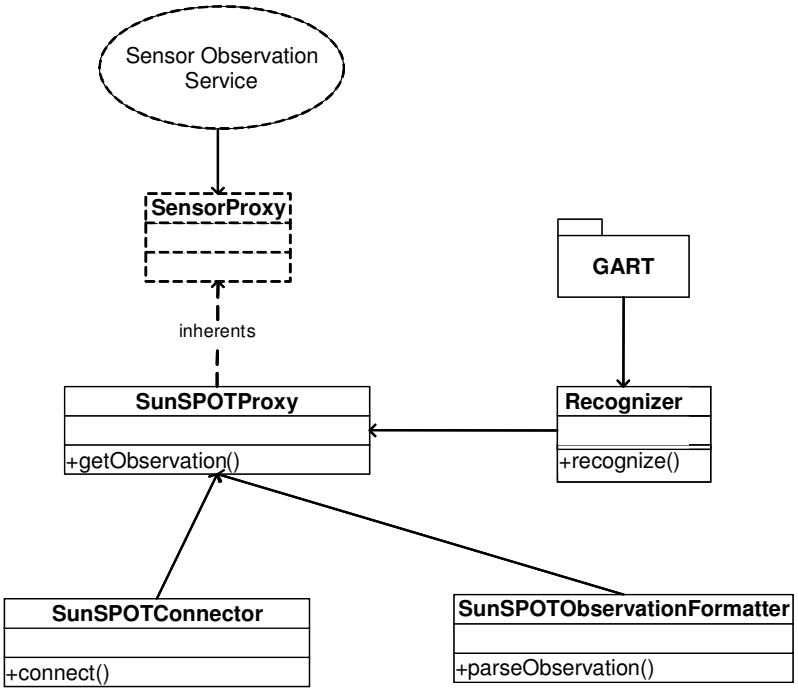

Figure 17 Architecture of the SunSPOT proxy and connector

The GUI client is a simple SWE client deployed as a Java desktop application that interfaces with user. The client uses WSRF to connect to the SOS and when a gesture is identified it prints the result in a text box. The gesture result consist of a letter of the alphabet which

mapped to a particular motion recorded by the sensors. Figure 18 is a screenshot of system in action. For illustration purposes the GUI client, SOS instance and Sun SPOT Serial Forwarder are running on a single machine. One gesture is performed and the raw acceleration data series is printed out in the console. The identified gesture is returned and printed in GUI client.

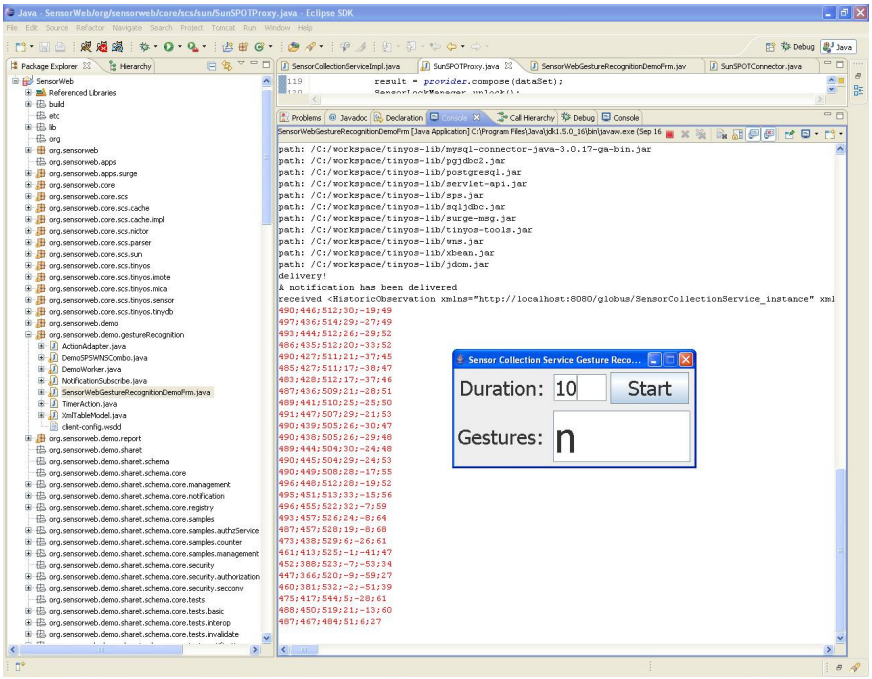

Figure 18. Gesture recognition system in action

To evaluate the performance of the gesture training and recognition system, test cases were chosen from EdgeWrite ("EdgeWrite Text", 2008) a unistroke text entry method developed by University of Washington. Its benefits include increased physical stability, tactility, accuracy, and the ability to function with minimal sensing. Instead of testing a whole character set in this experiment, we selected two sets of characters. As illustrated in Figure 19, each set has its own characteristics. In set S1, the shape and track of each gesture is unique. The Recognizer should be able to identify these characters easily. In set S2 the six gestures share similarities among each other. When compared to S1 we expect the level of recognition accuracy for S2 to be lower. We want to know the difference of accuracy between these two gesture sets.

Figure 19: Two character sets

Because the HMM is a supervised learning algorithm more training samples will lead to a higher recognition accuracy. So for each gesture set we train two versions of the sample sets:

- Version 1(V1): 5 samples per gesture. User specific, trained by one developer
- Version 2(V2): 10 samples per gesture. Non user specific (relative), trained by two individuals each taking turns in producing gestures

Compared with V1, V2 contains more training samples, so it should get higher accuracy result. When we run each of these sample sets on the two gesture sets we get a total of four training sets, these are illustrated in Table 1.

	S1	S2
V1	N1	N3
V2	N2	N4

Table 1. Improved Sample matrix

10 gestures were performed on each of the training sets to evaluate the performance of the system. Table 2 depicts a summary of the accuracy after the experiment. Generally, the recognition accuracy of S1 is higher than S2, which is to be expected because characters in S1 are unique. V2 achieves higher accuracy for both character sets because it contains more training samples than V1.

	S1	S2
V1	85%	75%
V2	96%	78%

Table 2. Summary of accuracy

Tables 3, 4, 5 & 6 are confusion matrices which expand on the recognition accuracy detail for each of the four training tests in Table 2. A confusion matrix is used in machine learning to illustrate correct and incorrect classification results. For each confusion matrix the x-axis represents the actual gesture performed and the y-axis represents the recognized gesture by system. For example, in Table 3, cell **nn** (shadowed area) is 70% which means 70% of gesture **n** was correctly recognized by system. On the other hand, gesture **n** has been incorrectly recognized as **a** and **g** at 20% and 10% respectively.

85%	**a**	**g**	**n**	**t**	**x**	**s**
a	**70%**	30%	0	0	0	0
g	0	**100%**	0	0	0	0
n	20%	10%	**70%**	0	0	0
t	0	20%	0	**80%**	0	0
x	0	10%	0	0	**90%**	0
s	0	0	0	0	0	**100%**

Table 3. Confusion matrix of N1

96%	**a**	**g**	**n**	**t**	**x**	**s**
a	**100%**	0	0	0	0	0
g	0	**100%**	0	0	0	0
n	0	0	**100%**	0	0	0
t	0	0	0	**100%**	0	0
x	10%	0	0	0	**90%**	0
s	0	10%	0	0	0	**90%**

Table 4. Confusion matrix of N2

75%	**o**	**c**	**f**	**g**	**q**	**Ø**
o	**70%**	0	0	0	20%	10%
c	0	**30%**	0	50%	0	10%
f	0	0	**90%**	10%	0	0
g	0	0	0	**100%**	0	0

q	10%	0	0	40%	**60%**	0
Ø	0	10%	0	0	0	**100%**

Table 5. Confusion matrix of N3

78%	**o**	**c**	**f**	**g**	**q**	**Ø**
o	**80%**	0	0	0	20%	0
c	0	**90%**	10%	0	0	0
f	0	0	**90%**	0	10%	0
g	0	0	0	**40%**	60%	0
q	10%	0	0	20%	**80%**	0
Ø	0	10%	0	0	10%	**90%**

Table 6. Confusion matrix of N4

Statistically, the HMM recognition engine works well with a minimum of 75% accuracy. Using more training samples, it easily reaches 96%. As we predicted, the higher the level of training samples, the greater the level of accuracy we can expect from the system. The more motion is recorded about one particular user, the more the system can recognize their actions. In this example we use letters of the alphabet because they offer a suitable amount of complexity in gesture variance. The accuracy of results produced by alphabet set gives us confidence in the performance of our system. The core concept of gesture recognition can be expanded to a variety of deployment scenarios. A user can use perform hand gestures during a presentation which may result in changing slides, or initializing a multimedia component. The user does not need physical access to the computing system and is free to be mobile and interact with the audience during the course of the presentation.

In this experiment, we successfully implemented a gesture recognition system using SunSPOT sensors, a machine learning algorithm and OSWA. The SOS was used to retrieve real time observation data from the SunSPOT sensors. The raw observation results were transformed using a Gesture Recognition and a HMM toolkit and forwarded to a SWE client for visualization. Additional development work which was required to implement the application consisted of: introducing an application specific Connector class to the SunSPOT proxy interface, a data formatter class to encapsulate the application specific acceleration data, a recognition module to perform post-processing on the raw observational data, and the visualization capabilities in the form of a text box on the SWE client. Besides these modifications the SOS provided all the necessary architecture components to fulfill our objective. With the additional components added to the architecture we could run our experiment and measure the accuracy of the gesture recognition algorithm. The algorithm saw improved performance in accuracy with the help of more training samples, reaching 96% accuracy at its peak. The combination of OSWA and gesture recognition has the potential to free the user physical access to a computing system and provide them with an accurate alternative.

CONCLUSION AND FUTURE WORKS

In this chapter we have introduced Sensor Web and the OGC SWE method. Sensor Web provides a conceptual framework where geographically distributed services can provide access to heterogeneous sensor resources regardless of the deployment scenario. The SWE method outlines a set of common data description formats and service interfaces which when implemented can realize the vision of a Sensor Web. Application independent data description formats are important for sharing data from heterogeneous sensor resources among independent deployment

scenarios. A common set of Service descriptions encourages the development of services by research organizations and businesses to communicate with one another in order to achieve cross-organizational collaboration, mutually benefiting stakeholders. OSWA is one implementation of the SWE Method which implements services as stateful web services using WSRF. OSWA is developed in Java and implements all the mandatory operations defined for the SOS, WNS and SPS, along with encodings for SensorML and O&M schema. The SOS provides access to a set of heterogeneous sensors and sensor operating systems including hardware developed by Crossbow running TinyOS, SunSPOT's running Java and NICTOR sensors running Linux. The SPS is built on the architecture of the Gridbus Broker, a mature broker application. A gesture recognition application has been presented in order to demonstrate the functionality of the SOS, a major OSWA component. This case study illustrates the ability of OSWA to meet the needs of almost any deployment scenario. Future developments which we intend to commit resources to include:

- An operator service, capable of hiding hardware implementation details from users. Users could use the operator service to automatically deploy and update applications on the sensor nodes without the need to physically access the sensor network. An overlay network would be deployed on the sensors which would be hardware transparent and capable of fulfilling the demands of the operator service. The overlay network would manage energy efficiency, security and automatic network configuration.
- A GUI IDE providing access to service operations and allowing users to visually construct applications, service plans and sensor deployments. Users could drag-and-drop GUI elements which would result in the generation of code that could automatically be deployed on the sensor network by the operator service. Users could visually construct SPS plans and describe service interactions.
- Data driven workflows, which could be deployed on the overlay network and across services. Sensor observations could automatically trigger service interactions and perform complex tasks.
- An implementation of the SAS along with the TML encoding.

REFERENCES

Barr, R., Bicket, J,C., Dantas, D,S., Du, B., Kim, T,W,D., Zhou, B., Sirer, E, G., (2002) *On the Need for System-Level Support for Ad hoc and Sensor Networks*. Operating Systems Review, 36(2):15.

Baum, L, E., & Petrie, T. (1966) *Statistical Inference for Probabilistic Functions of Finite State Markov Chains*. Ann. Math. Statist. Volume 37, Number 6.

Botts,M. (2007). OpenGIS Sensor Model Language (SensorML) Implementation Specification, Open Geospatial Consortium Inc. Retrieved on November 17, 2008 from http://portal.opengeospatial.org/files/?artifact_id=21273

Botts, M., Percivall, G., Reed, C., & Davidson, J. (2007). *OGC Sensor Web Enablement: Overview And High Level Architecture*, Open Geospatial Consortium Inc. Retrieved on November 3, 2008, from http://portal.opengeospatial.org/files/?artifact_id=25562

Cox, S. (2007). *Observations and Measurements – Part 1 – Observations schema*, Open Geospatial Consortium Inc. Retrieved on November 5, 2008, from http://portal.opengeospatial.org/files/?artifact_id=22466

(2008) EdgeWrite Text Entry, Retrieved June 6, 2008 from http://depts.washington.edu/ewrite

Fok, C., Roman, G., Lu, C. (2005). Mobile agent middleware for sensor networks: An application case study. In Proc. *The 4th int Conf. Information Processing in Sensor Networks* (pp. 382-387), UCLA, Los Angles, California, USA.

(2008) GART. Retrieved on June 6, 2008 from http://wiki.cc.gatech.edu/ccg/projects/gt2k/gt2k.

Gaynor, M., Moulton, S., Welsh, M., LaCombe, E., Rowan, A., & Wynne, J. (2004). *Integrating WSN with the Grid*, IEEE Internet Computing 8:32-39

Heinzelman, W, B., Murphy, A,L., Carvalho, H, S., & Perillo, M,A., (2004) *Middleware to support sensor network applications*, IEEE Network, 18(1):6-14.

(2008) HTK Speech Recognition Toolkit, Retrieved June 6, 2008 from http://htk.eng.cam.ac.uk

Li, C., Zheng, S. Q., & Prabhakaran, B. (2007) *Segmentation and recognition of motion streams by similarity search.* ACM Trans. Multimedia Comput. Commun. Appl. 3, 3.

Li, S., Son, S., Stankovic, J. (2003). Event Detection services using data service middleware in distributed sensor sensor networks. *In Proc. The 2nd Int. Workshop Information Processing in Sensor Networks* (pp.502-517), Palo Alto, California, USA.

Moodley, D., & Simonis, I. (2006). *New Architecture for the Sensor Web: the SWAP-Framework*, Semantic Sensor Networks Workshop 2006, 5th International Semantic Web Conference ISWC 2006, Athens, Georgia, USA.

(2008) Open Geospatial Consortium Inc. Retrieved 15 December 2008, from http://www.opengeospatial.org

Na, A. (2007). *Sensor Observation Service*, Open Geospatial Consortium Inc. Retrieved on November 25, 2008 from http://portal.opengeospatial.org/files/index.php?artifact_id=26667&passcode=xk3nxmxma23st1y6g6hh

Rabiner, L.,R., (1989) *A tutorial on hidden Markov models and selected applications inspeech recognition,* Proceedings of the IEEE, Volume 77, Issue 2.

Sawada, H., Hashimoto, S. (1997*) Gesture recognition using an acceleration sensor and its application to musical performance control.* Electronics and Communications in Japan 80:5.

Simonis, I. (2004). Sensor Webs: A RoadMap. In *Proc. of the 1st Goettinger GI and Remote Sensing Days*, Institute for Geoinformatics, University of Muenster.

Simonis, I. (2007). *OGC Sensor Alert Service Implementation Specification*, Open Geospatial Consortium Inc. Retrieved on November 25, 2008 from http://portal.opengeospatial.org/files/index.php?artifact_id=24780&version=1&format=pdf

Suman, N., Jie, L., Feng, Z. (2006). *Challenges in Building a Portal for Sensors World-Wide*. Paper presented at the First Workshop on World-Sensor-Web: Mobile Device Centric Sensory Networks and Applications, Boulder, CO, USA.

(2008) SunSpotWorld - Home of Project Sun SPOT, Retrieved June 6, 2008 from http://www.sunspotworld.com

Tao, V., Liang, S., Croitoru, A., Haider, Z. & Wang, C. (2004). GeoSWIFT: Open Geospatial Sensing Services for Sensor Web. In: Stefanidis, A., Nittel, S. (eds), *GeoSensor Networks* (pp.267-274), CRC Press.

Tham, CK., Buyya, R. (2005). *SensorGrid: Integrating sensor networks and grid computing*, CSI Communications 29:24-29.

Venugopal, S., Nadiminti, K., Gibbins, H. & Buyya, R. (2008). *Designing a Resource Broker for Heterogeneous Grids in Software: Practice and Experience* (pp. 793-825), Volume 38, Issue 8, ISSN: 0038-0644, Wiley Press.

KEY TERMS & DEFINITIONS

Sensor Web
The combination of sensor networks and a service oriented architecture, so that sensors are viewed as resources which can be controlled and accessed over the World Wide Web.

SensorML
A set of standard models and XML schema as defined in the Sensor Web Enablement method by the Open Geospatial Consortium for describing sensor systems and processes.

Observations & Measurements
A set of standard models and XML schema as defined in the Sensor Web Enablement method by the Open Geospatial Consortium for describing physical phenomena observed by sensor systems.

TML
An XML schema and encoding as defined in the Sensor Web Enablement method by the Open Geospatial Consortium for describing real-time streaming data recorded by transducers.

Sensor Observation Service
A web service interface definition as defined in the Sensor Web Enablement method by the Open Geospatial Consortium for requesting observations from sensor networks and observation repositories.

Sensor Planning Service
A web service interface definition as defined in the Sensor Web Enablement method by the Open Geospatial Consortium for scheduling and planning observational requests to sensor networks.

Web Notification Service
A web service interface definition as defined in the Sensor Web Enablement method by the Open Geospatial Consortium for the transmission of messages between SWE services.

Sensor Alert Service

A web service interface definition as defined in the Sensor Web Enablement method by the Open Geospatial Consortium for publishing and subscribing to alerts from sensors.