2013

Smart Sensor Webs For Environmental Monitoring Integrating Ogc Standards

William Trevor Wright
North Carolina Agricultural and Technical State University

Smart Sensor Webs for Environmental Monitoring Integrating OGC Standards

William Trevor Wright

North Carolina A&T State University

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Department: Energy & Environmental Systems

Major: Energy & Environmental Systems

Major Professor: Dr. Albert Esterline

Greensboro, North Carolina

2013

School of Graduate Studies
North Carolina Agricultural and Technical State University
This is to certify that the Doctoral Dissertation of

William Trevor Wright

hasmet the dissertation requirements of
North Carolina Agricultural and Technical State University

Greensboro, North Carolina
2013

Approved by:

Dr. Albert Esterline Major Professor	Dr. Keith Schimmel Committee Member
Rich Baldwin Committee Member	Dr. Cathy Connor Committee Member
Dr. Abdollah Homaifar Committee Member	Dr. Jinsheng Xu Committee Member
Dr. Keith Schimmel Department Chair	Dr. Sanjiv Sarin Dean, The Graduate School

Biographical Sketch

William Trevor Wright was born April 22, 1981 in Newark, New Jersey.He received his bachelor's of science in Computer Science and Physics from Francis Marion University in 2005. Soon after, William began graduate school at North Carolina A&T State University and completed his master's of science degree in Computer Science in 2008. William is currently a Ph.D. candidate in the Energy & Environmental Systems doctoral program.

Acknowledgements

I would, first and foremost, like to give thanks to my Lord and Savior, Jesus Christ, for through him, all things are possible. Also, I give many thanks to my advisor, Dr. Albert Esterline, for his wisdom, persistence, and encouragement throughout this journey. I would to thank my program director, Dr. Keith Schimmel for all his support and assistance in the Energy and Environmental Systems Program. Additionally, I would like to thank my dissertation committee (Rich Baldwin, Dr. Cathy Connor, Dr. Abdollah Homaifar, and Dr. Jinsheng Xu, along with Dr. Albert Esterline and Dr. Keith Schimmel) for their kindness, guidance, and inspiration. I would also like to give many thanks to mom & dad (Iva & Melvin Greene) for supporting my efforts to pursue education and strengthening my drive along the way. To my beautiful wife and lovely daughter, Christa & LeiLani, thank you for all the support and the extra push that I needed.

Table of Contents

List of Figures

List of Tables

Abstract

Sensor webs are the most recent generation of data acquisition systems. The research presented looks at the concept of sensor webs from three perspectives: node, user, and data. These perspectives are different but are nicely complementary, and all extend an enhanced, usually wireless, sensor network. From the node perspective, sensor nodes collaborate in response to environmental phenomena in intelligent ways; this is referred to as the collaborative aspect. From the user perspective, a sensor web makes its sensor nodes and resources accessible via the WWW (World Wide Web); this is referred to as the accessible aspect. From the data perspective, sensor data is annotated with metadata to produce contextual information; this is referred to as the semantic aspect. A prototype that is a sensor web in all three senses has been developed. The prototype demonstrates theability of managing information in different knowledge domains. From the low-level weather data, information about higher-level weather concepts can be inferred and transferred to other knowledge domains, such as specific human activities. This produces an interesting viewpoint of situation awareness in the scope of traditional weather data.

CHAPTER 1

Introduction

Data and information have become the primary currency of today's technology age. Today's applications connect clients to almost any stream of information. Until recently, most streams of information, or data sources, were human generated, but through some of today's most advanced technologies, such as wireless sensor nodes, humans have been extracted from the process of information generation. This leads to streams of information generated by environments and objects so that clients can get suitable information "from the horse's mouth", in other words, they get the most reliable information directly from the source.

Environments provide many dynamic components that could be monitored or possibly controlled. They present vast amounts of varying, but interesting, data for collection and analysis. Interesting real-world objects, such as mechanical structures, also present vast amounts of useful data through their complex, dynamic interactions of their components. Two principle areas of data collection and analysis include structural health monitoring and environmental monitoring. This research focuses on environmental monitoring, but most of the same concepts directly apply to structural health monitoring.

Most environments involve multiple systems that interact along with thousands of smaller components. For an example, consider a rain forest. The process of collecting and analyzing data manually in such an environment is an overwhelming task; with the assistance of multiple sensors networked together; the task is not as daunting. Although sensor networks have only been around for several years, there have been a number of advances in this area, and current directions largely point to sensor webs. The research reported here developed a sensor web inspired in part by UAS's (University of Alaska Southeast) experience with their

implementation of their SEAMONSTER (SouthEast Alaska MOnitoring Network for Science Technology Education and Research) sensor web (Heavner et al., 2007).

Throughout the comprehensive literature review, the term “sensor web” has been used in three different but related senses that always refer to an enhanced, usually wireless, sensor network. In the accessible sense, a sensor web makes resources accessible via the World Wide Web, and, in the collaborative sense, nodes collaborate in response to environmental observations. The semantic sense makes use of metadata that adds context to the environmental observations; this context provides awareness of the overall situation. The research developed a prototype that incorporates all three senses into a sensor web. The prototype also exhibits techniques of data fusion and situation awareness. Data fusion is a method of generating additional information and data from integrating multiple data sets. The generated data is usually more useful and accurate than the initial data sets. The introduction of more useful and accurate data leads to situation awareness. Situation awareness displays a certain level of understanding of an environment or object and its behaviors. With this understanding, reasonable assumptions about the future state of the environment can be made.

This dissertation presents the concept of a sensor web along with the accessible, collaborative, and semantic senses. These senses are developed through the discussion of their components. The following is a brief summary of the dissertation.

After the Introduction, the chapters that follow are the Background, Literature Review, Specific Aims & Objectives, Architecture, Results, Discussion, and the Conclusion. The Background and Literature Review chapters present the foundational body of knowledge behind the research. The Background chapter discusses the technologies used by the research while the Literature Review chapter discusses the techniques reported in the literature that utilize the

technologies presented in the Background chapter. The Literature Review chapter also presents what others have done in related areas. The Specific Aims & Objectives chapter presents the primary research goals to be achieved through the research. The Architecture chapter describes the architecture of the prototype. The Results chapter outlines the outcome of the developed prototype. The Discussion chapter presents the concepts along with their relevance and importance to the research community. The Conclusion chapter summarizes the topics discussed throughout the dissertation and presents possible future directions for the research.

CHAPTER 2

Background

The following sections discuss the concepts and technologies used to facilitate the prototype sensor web. The prototype realizes the overall concepts discussed in the research. This chapter starts by discussing the overarching concepts of data fusion and situation awareness. Data fusion integrates multiple sources of data to form a high level context of the situation. Then the multiagent system framework is discussed to get a foundation for agents, intelligent computer systems with reactive, proactive, autonomous, and social properties. JADE is a Java-based framework for developing agents that adhere to FIPA standards. Next, the Web service standards and technologies are discussed. Web services are self-contained procedures on the Web that allow for machine-to-machine interaction. SOAP facilitates communication between Web services and clients, while WSDL is needed for describing how to connect and communicate with Web services. REST is the latest Web service methodology and focuses on resources. WSIG is the JADE component that allows the agents to interact with Web services, while the Semantic Web adds meaning to data giving a context of the situation.

2.1 Sensor Networks & Sensor Nodes

A sensor network is a collection of spatially distributed sensor nodes that monitor certain aspects of an environment. The general purpose of a sensor network is to collect information pertaining to the environment. In addition, “[e]ach sensor node is capable of only a limited amount of processing. But when coordinated with the information from a large number of other nodes, they have the ability to measure a given physical environment in great detail” (Bharathidasan and Ponduru, 2002). Basically, individual sensor nodes provide limited capabilities, but when situated with a group, they provide amplified capabilities.

The sensor nodes, or motes, are the integral components of the sensor web. They are the perceptors through which the sensor web interacts with the environment. They also serve as the backbone of the infrastructure; through the integrated wireless capabilities the sensor nodes control the communication of the system. The latest technology of smart sensors adds new functionality to sensor node technology. Smart sensors introduce dynamic processing capabilities to the traditional sensor nodes. One of the most significant smart sensor capabilities involves the conversion of the signal to its human readable form before it is transmitted through the network. The sensor nodes used in the prototype of this dissertation are categorized as smart sensors since they all possess the signal conversion capability. The sensor nodes discussed here include the Berkeley Imote2s and Sun SPOTs. They will be briefly discussed in the following sections, but, beforehand, we will discuss the major components of sensor nodes along with the advantages and disadvantages of sensor nodes.

Most sensor nodes contain a common set of components: processor, transceiver, memory, power supply, and sensors. The processor handles the processing tasks of the nodes and controls the functionality of the other components. Several kinds of processors are used within sensor nodes, including common workstation processors, digital signal processors, and microprocessors. Workstation processors typically require more power than other processors, while digital signal processors perform more advanced signal processing than is generally needed for sensor webs. Microprocessors are the ideal compromise for the processing unit of sensor nodes. The transceiver is a device that has signal receiving and transmitting capabilities. Sensor node transceivers generally take advantage of radio frequency and infrared technologies for wireless communication. Sensor nodes also take advantage of memory storage units. For efficiency reasons, memory capacities are limited to a few megabytes. Sensor node memory combines

memory on the microprocessor chip and flash memory for program data. The power supply is needed to provide power to all the sensor node components. Most sensor nodes use standard AAA or AA batteries. The sensors are the devices that measure the phenomenon of interest, such as temperature, pressure, or radiance. Table 1 presents a summary of these technical specifications of the sensor nodes used in this research.

Table 1 Summary of Sensor Node Specifications.

	Imote2	**Sun SPOT**
Processor	Marvell PXA271 (13MHz-416MHz)	ARM 926ej-S (400MHz)
Radio	Chipcon CC2420 (2.4GHz)	Chipcon CC2420 (2.4GHz)
Memory	32M Flash, 32M SDRAM	8M Flash, 1M SRAM
Power Supply	3 'AAA' batteries or rechargeable Li-Ion battery	Li-Ion Rechargeable Battery
Sensors	3-axis Accelerometer, Temperature, Humidity, and Light	3-axis Accelerometer, Temperature, and Light
Programming	NesC (TinyOS)	Java
Input/Output	USB, 3xUART, 2xSPI, I2C, SDIO, GPIOs, I2S, AC97, and 1 LED	8 LEDs, 6 analog inputs, 2 momentary switches, and 5 general purpose I/O pins

There are several advantages of sensor nodes, along with some disadvantages. One advantage is the cost. In particular, off-the-shelf components are fairly inexpensive, especially for kits that contain multiple sensor nodes. Another advantage pertains to the sensor node's size. Most of the latest sensor nodes are about the size of two decks of cards or smaller. Also, every new generation of sensor nodes is smaller than the previous generation. The latest generations of sensor nodes use wireless technology; this adds to the benefits of sensor nodes the advantages of flexibility of location and ease of deployment. Wireless sensor nodes have a distinct advantage over wired sensor nodes due to the elimination of restrictions from attached wires. The sensor nodes may thus be positioned in a wider range of locations. There is also the advantage of ease of deployment as the sensor nodes can be positioned and deployed in flexible locations. Most

sensor nodes are housed inside of pods, or casings, to protect the vulnerable circuitry of the device. The housing for the sensor nodes adds durability to the list of advantages. It allows the sensor nodes to function properly given the harsh elements of most environments.

Along with the advantages of sensor nodes, there are a few disadvantages. One disadvantage pertains to the cost of sensor nodes. The cost of sensor nodes usually range from inexpensive to fairly expensive, but this is dependent on a number of factors such as the manufacturer or type of integrated sensors. Custom built sensor nodes are usually more expensive, resulting from the expenses of developing and manufacturing the hardware. Power is also another disadvantage of sensor nodes; just as with all mobile devices, power is a limiting resource. Sensor nodes are involved in a wide range of tasks, including wireless communication, which is a large consumer of the sensor node's power.

2.1.1 Berkeley Imote2. The Imote2 is a collaboration effort of Crossbow and Memsic, two state-of-the-art smart sensor companies (Imote2). The Imote2 is the latest generation mote used for adding wireless and low-power options to sensor networks. It is an advanced node platform that incorporates sensing and networking technologies. It incorporates a processor, external/internal memory, a radio transceiver, an antenna, and expansion connectors. There are a number of improvements to this sensor mote over the previous generations, such as the Mica, Mica2, and iMote motes.

The Imote2 was designed around the XScale Processor, PXA271, which operates between the frequencies of 13MHz and 416 MHz. With a 13 MHz low frequency, which requires low voltage, the Imote2 can achieve low power operation. The Imote2 also incorporates a MMX coprocessor for multimedia operations. The radio transceiver conforms to the 802.15.4 IEEE standard for wireless personal area networks. This is ideal for the low power requirements for

sensor nodes. The radio range is augmented by a 2.4 GHz surface mount antenna that achieves a range of nearly 30 meters. The Imote2 enables hardware expansion through a basic and advanced set of connectors. The basic set is meant for common, low cost sensor boards, while the advanced set is meant for accessing the advanced features of the Imote2's processor, such as the camera or audio interfaces. Figure 1 presents the Imote2 board in isolation as well as the Imote2 board integrated with the sensor board and the battery board beside a quarter for size comparison.

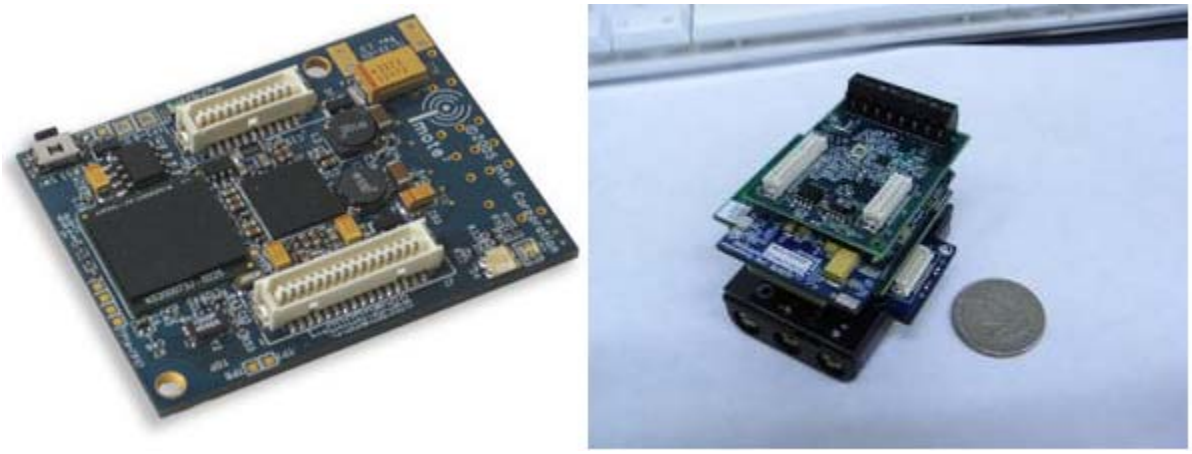

Figure 1 Imote2 Sensor Node.[1],[2]

2.1.2 TinyOS. TinyOS is an operating system designed specifically for resource constrained devices (Levis and Gay, 2009). TinyOS is used in a variety of applications from personal area networks to ubiquitous computing but is excellent for sensor nodes and sensor networks. It is open source so it allows for modification and extension for applications. Developers can take advantage of the software abstractions provided by TinyOS to access low level hardware components of devices.

[1] The first figure was referenced from http://cwnlab.ece.okstate.edu/images/facilitiesimg/imote2_4.jpg
[2] The second figure was referenced from http://imcl.comp.polyu.edu.hk/wiki/wsn/lib/exe/fetch.php?w=300&media=research:imote2_1.jpg

TinyOS is best suited for resource constrained devices that are controlled by microcontrollers. These types of devices are designed for very low power consumption and are intended to be integrated with sensor and networking hardware. These three features are also supported through TinyOS. TinyOS supports wireless networking through the 802.15.4 IEEE standard (Part 15.4), which is very common among wireless sensor nodes.

There are two main disadvantages of TinyOS, which relate to its programming model. The first disadvantage is that it is more difficult to learn than most other environments. TinyOS was specifically designed for devices with little memory, so it does not support process blocking. Without the capability to block processes, any process, or piece of code, should not run for long periods of time because other processes cannot run their code. Although this is necessary, most developers are accustomed to a programming style with blocking. The second disadvantage involves the difficulty of developing applications that are computationally intensive. Because TinyOS does not support process blocking, long computationally-intensive processes must be broken down into smaller chunks.

2.1.3 NesC. NesC is a programming language for networked embedded systems (Gay et al., 2009). It is ideal for sensor nodes and sensor networks that contain multiple low power devices. NesC provides a programming model that supports event-driven execution and component oriented design. NesC is the language used to develop TinyOS as well as with other sensor node and sensor network applications. Because of their wide use and popularity, TinyOS and NesC have become staples of most sensor network research.

The primary concept behind NesC is its system design, where the applications developed in NesC are closely tied to the device hardware. This presents three properties that developers must keep in mind. The first property is that the device resources are known to be constant. The

second property is that NesC applications are developed with reusable system components that integrate application-specific components. The third property relates to the importance of a flexible design because of the wide variety of supported hardware.

Regarding NesC's use in sensor nodes and sensor networks, there are a few challenges that it addresses. Sensor nodes collect data and control the environment, so the sensor nodes interact with the environment through events. NesC approaches this with its event-driven execution. NesC approaches the challenge of the limited resources of sensor nodes with its small footprint. NesC also attempts to mollify the challenge of reliability. Individual sensor nodes fail due to hardware failure; although this cannot be avoided, NesC reduces run-time errors to add a layer of reliability.

2.1.4 Sun SPOT. Sun SPOTs (Small Programmable Object Technology) are state-of-the-art sensing devices (sunspotworld.com). They are an experimental platform for accessing sensor hardware on a wireless device. The SPOT kit includes two wireless SPOT devices, a SPOT basestation and an environment for developing applications and deploying them to the SPOTs. The hardware of the Sun SPOT devices includes a 180 megahertz processor, 512 kilobytes of RAM, and four megabytes of flash memory. The device also has a 2.4 gigahertz radio with an integrated antenna. The hardware of the SPOT includes a three-axis accelerometer, a temperature sensor, a light sensor, eight LEDs, six analog inputs, two switches, and five input/output pins. The inputs and pins allow the SPOT hardware to be extended with additional sensors, actuators, or other technology, such as a GPS unit, humidity sensor, or solar cell. Figure 2 shows a Sun SPOT sensor node that has been disassembled for a clear view of its sensor board, processor board, and its battery. It also shows (with a quarter for reference) the SPOT's small size and compact design for its intricate hardware.

Unlike the previously discussed Imote2 sensor node, the SPOTs do not run on an operating system, such as TinyOS, but run on a special Java virtual machine (JVM), called the Squawk JVM. This JVM is an implementation of the Java MicroEdition of virtual machines. The Squawk JVM supports pJava and MIDP, both of which implement LEAP, the JADE framework for sensor nodes.

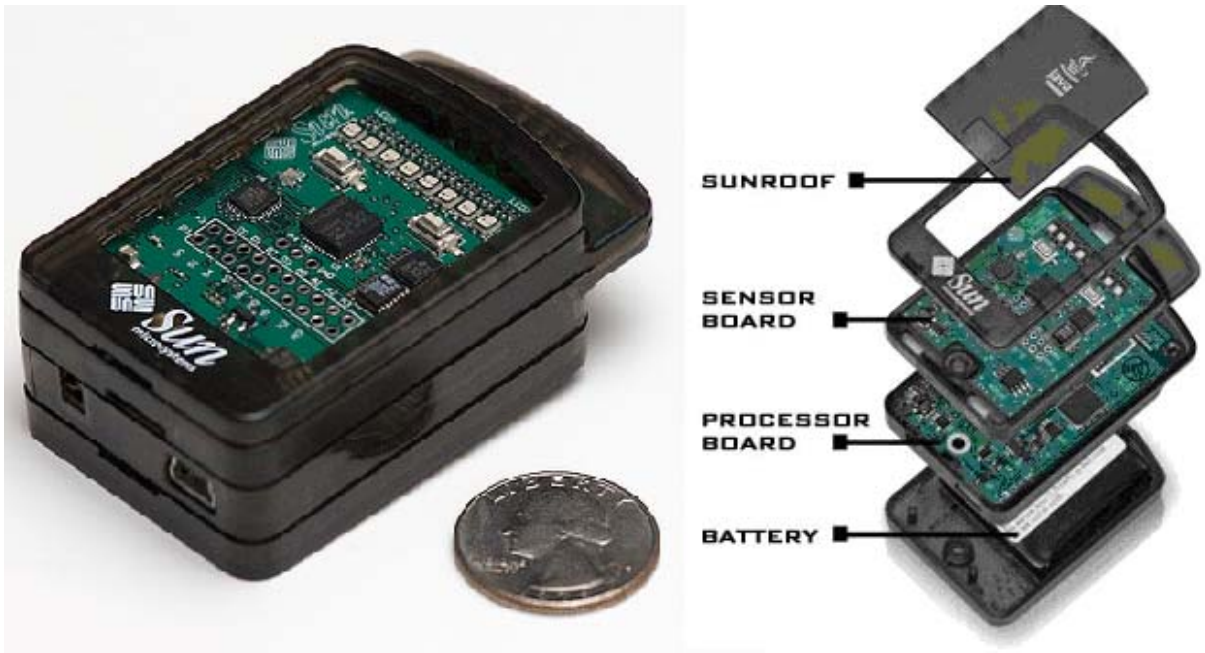

Figure 2 Overview and Disassembled View of Sun SPOT. (sunspotworld.com, 2008)

2.1.5 Davis VantagePro2. The Davis Vantage Pro2 is a wireless weather station (VantagePro2). It incorporates the Integrated Sensor Suite (ISS) that includes a rain collector, temperature and humidity sensors, and anemometer. The weather station communicates wirelessly to a console connected to a workstation. Figure 3 shows the Davis Vantage Pro2 weather station on top of McNair Hall at North Carolina A&T State University.

Figure 3 Davis VantagePro2 weather station on McNair Hall.

2.1.6 Miscellaneous Sensors. There is a variety of other sensor nodes that would provide useful functionality. Generic sensor nodes provide a standardized platform at an affordable cost, while custom sensor nodes provide the exact functionality needed by the client.

IRIS, Micaz, and TelosB are a few generic sensor nodes that could easily be interchanged with the Imote2 sensor nodes. The IRIS sensor nodes were designed for industrial, structural, and energy monitoring (IRIS). The Micaz sensor nodes were meant for indoor building monitoring (MICAz). The TelosB sensor nodes were designed for research development and wireless sensor network experimentation (TELOSB). These sensor nodes depend on the same hardware stack of

TinyOS and nesC, so they are highly interoperable. They also serve as some of the standard sensor nodes for educational and industrial use.

2.2 Multiagent Systems

An agent is an intelligent computer system able to act autonomously and to interact with other agents. A multiagent system is composed of multiple agents typically interacting by exchanging messages through some computer network infrastructure and acting on behalf of users with different goals and motivations (Wooldridge, 2002).

The field of multiagent systems is relatively new in computer science. It is influenced and inspired by multiple fields, including economics, philosophy, game theory, logic, ecology, and the social sciences. Intelligent agents in particular have three important characteristics: they are reactive, are proactive, and have social ability. An agent is reactive if it can perceive its environment and respond rapidly. It is proactive if it exhibits goal-driven behavior by taking the initiative. Lastly, an agent has social ability so that it can interact with other agents and humans. Agents interact with each other to create a multiagent system. In a multiagent system, agents may have different roles, yet typically all collaborate to maintain or to achieve the goals of the system. The interaction of agents allows a multiagent system to exhibit complex behaviors. The field of multiagent system also introduces interesting paradigms, such as Gaia and the Contract Net Protocol, for development and problem solving. Gaia (Wooldridge et al.2000) presents a methodology for designing multiagent systems, while the Contract Net Protocol (Alibhai, 2003) allows agents to incorporate problem solving capabilities through agent collaboration.

The following sections discuss the technologies and methodologies needed to build and extend multiagent systems. JADE and LEAP allow the development of multiagent systems, while Jess incorporates intelligence into multiagent systems through rules. Agent communication

languages facilitate communication within and among multiagent systems. FIPA defines standards for multiagent systems to facilitate interoperability among them.

2.2.1 ACL. Several agent communication languages (ACLs) have been defined to allow agents to communicate to perform complex tasks. An ACL is typically grounded in speech-act theory, which analyzes language use as consisting of acts, such as asserting, requesting, or commanding (Bellifemine et al., 2007). Certain verbs, such as “assert”, “request”, and “command” explicitly represent actions done with words; such verbs are called performatives. An older ACL for multiagent systems is KQML (Knowledge Query and Manipulation Language), which was developed as part of the ARPA Knowledge Sharing Initiative (Draft Specification). KQML defines various performatives and is intended for large-scale knowledge bases that are shareable and reusable (Java Agent Development). The content of a message is in KIF (Knowledge Interchange Format), essentially first-order logic in a LISP-like syntax. KIF is used to express properties of things in a domain and relationships between them as well as general properties of a domain.

In order for agents to communicate, they must agree on a common set of terms. An ontology is a conceptualization of a domain and provides the semantics for the terms in the domain, thus, allowing agents that share the ontology to communicate about the domain. An ontology typically describes relations among classes and the properties of elements of classes. In the Knowledge Sharing Initiative, Ontolingua was developed as a language for expressing ontologies.

A more recent ACL, FIPA-ACL, has been developed by the Foundation of Intelligent, Physical Agents (FIPA). FIPA is an organization that publishes standards for heterogeneous and interacting agents and agent based systems. Its purpose is to promote the success of emerging

agent-based applications, services, and equipment. FIPA-ACL is a proposed IEEE standard for agent communication. FIPA-ACL handles message exchange interaction protocols, speech act theory-based communicative acts, and content language representations. FIPA-ACL does not mandate the use of any particular language for expressing content.

2.2.2 JADE. JADE (Java Agent DEvelopment framework) is an open-source Java package for developing FIPA-compliant agents and multiagent systems (Java Agent Development). The JADE framework provides basic middleware functionalities that are independent of the specific application and that simplify the realization of distributed multiagent systems (Bellifemine et al., 2007). JADE provides a runtime environment for agents, a Java library that provides ready-made pieces of functionality, abstract interfaces for application-dependent tasks, and graphical tools for administration and monitoring of agents (Bellifemine et al., 2007).

JADE's design is simple and influenced by the agent abstraction, which presents three properties. The first property is that the agent is autonomous and proactive. Thus, an agent cannot provide call-backs. Also, an agent must have its own thread of execution, using it to control its lifecycle and to decide autonomously when to perform which actions. This property aligns with the characteristics of an intelligent agent. The second property is that agents have the right to say 'No' and they are loosely coupled. Thus, a message is sent to a destination identified only by a name, which the message transport system resolves into a transport address. This allows multi-cast communication to be implemented as an atomic operation, and a message can be sent to a group identified by a description or (using a proxy agent) by a domain. Also, the receiver decides which messages to ignore and which to process with what priorities. Finally, communication is asynchronous; there is no temporal dependence between sender and receiver,

and the sender does not block until the receiver processes the message. The third property is that the system is peer to peer (P2P). Each agent is identified by a globally unique name. It can join and leave a host platform at any time and can discover other agents through both white-page services, which allow agents to search for other agents by name, and yellow-page services, used by an agent to register its services or to search for services. An agent can initiate communication with any other agent anytime and can equally be the object of an incoming communication anytime.

A JADE *platform* consists of agent *containers* that can be distributed over the network. Agents live in containers, which are the Java processes that provide the JADE run-time environment and all the services needed for hosting and executing agents. The main container, called “Main Container,” is the bootstrap point of a platform and is the first container that is launched. All other containers must join to the main container by registering with it. By default, these are named “Container-1,” “Container-2,” and so on, where the numbers increase in the order in which the containers thereby named are created. Figure 4 depicts the relationship between the main architectural elements: the Directory Facilitator (DF), the Agent Management System (AMS), Main Container, Container-1, Container-2, and several JADE agents. The DF, which by default is in the main container, is the agent providing the yellow-page service. It also accepts subscriptions from agents that wish to be notified whenever a service registration or modification is made that matches some specified criterion. The AMS, which is always in the main container, is the agent that supervises the entire platform, manages the life-cycles of all agents, and provides the white-page service. JADE has every agent register with the AMS to obtain an AID (Agent IDentifier).

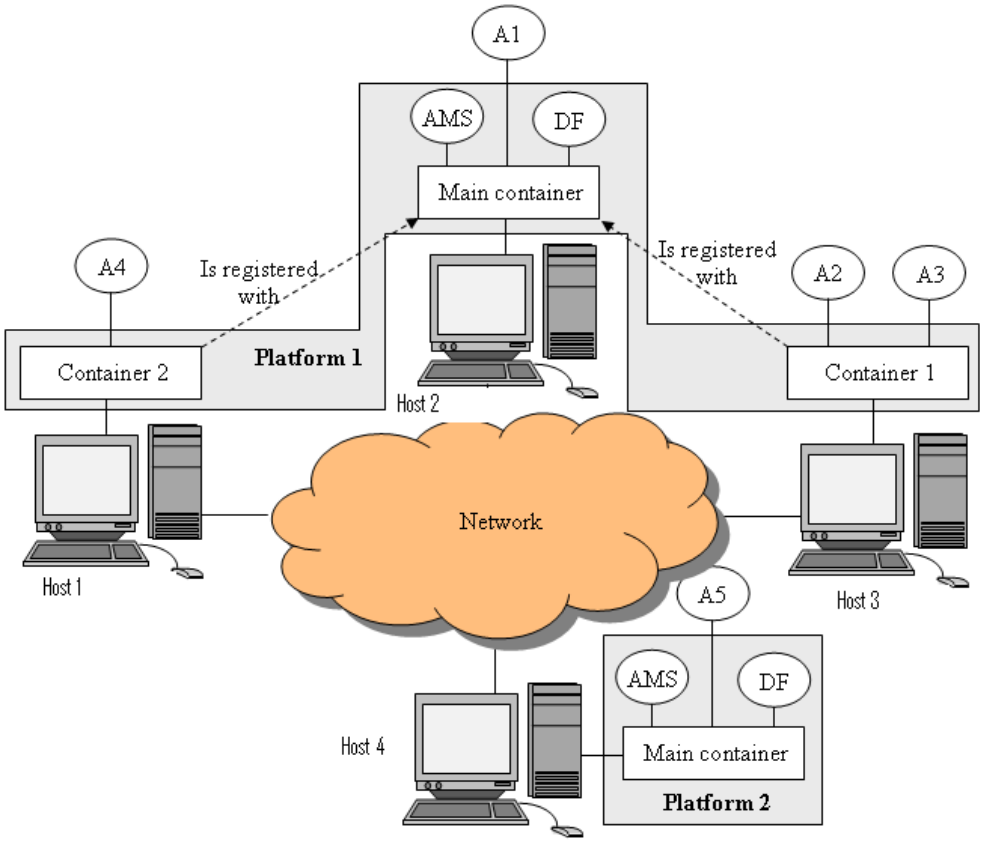

Figure 4 Architecture of JADE agent platform. (Bellifemine et al., 2007)

According to the FIPA specifications, a MTS (Message Transport Service) is one of the three most important services (besides those provided by the DF and the AMS) an agent platform is required to provide. It manages all message exchange within and between platforms. Any number of MTPs can be activated on a given JADE container. The GADT (Global Agent Descriptor Table) in the main container is the registry of all agents in the platform. To prevent the main container from becoming a bottle neck, JADE provides a cache of the GADT in each container. When a container looks for the location of a recipient of a message, it consults its LADT (Local Agent Descriptor Table); if this fails, it consults the main container and updates its cache of the GADT with the remote reference. The CT (Container Table) is the registry of the object references and transport addresses of all container nodes in the agent platform.

2.2.3 LEAP. LEAP (Light Extensible Agent Platform) (Bellifemine et al., 2007) is a modified version of the JADE framework that allows development of FIPA-compliant agents on resource constrained devices such as Java-enabled cell phones and personal digital assistants (PDAs). The primary motivation behind the development of LEAP has been to take advantage of the advances in hardware technology, particularly in the area of mobile devices. Although there have been many advances in hardware for mobile devices, there are a few limitations that LEAP addresses to allow JADE agents to function properly on such devices. These limitations of mobile devices, including hardware limitations, the amount of hardware resources required by the Java virtual machine, and network limitations, prevent the normal JADE framework from being deployed on mobile devices. The hardware limitations on mobile devices involve reduced processing power, limited memory capacities, and battery life. Mobile devices usually do not support 32-bit or 64-bit microprocessors, massive gigabytes of storage, and unlimited power from a wall outlet. These devices usually support 16-bit microprocessors, 640 megabytes to 2 or 4 gigabytes of storage, and a few active hours of battery life. The advances in mobile technology are closing the gap between the components supported in mobile devices and the components supported in desktops or laptops, but mobile devices will always continue to support components whose resources are limited compared to those of other devices.

The limitations on the hardware bring inherit limitations on the Java virtual machine (JVM), which is the software that interprets the Java bytecode, or intermediate computer language (Liang, 2005). Most desktop hardware configurations support J2SE (Java 2 Standard Edition), but the hardware limitations on mobile devices do not allow full-featured versions of Java to execute properly on these devices. Through the introduction of JME (Java Platform, Micro Edition), previously known as J2ME (Java 2 Micro Edition), a subset of the full

functionality offered by Java, mobile devices are able to execute Java applications more efficiently with their hardware limitations. There are also network or connectivity limitations with mobile devices. The JADE containers require constant connectivity, but mobile devices do not guarantee this type of connectivity.

LEAP supports mobile devices by addressing these limitations. LEAP supports four main Java configurations for mobile devices, namely, pJava (PersonalJava), MIDP (Mobile Information Device Profile), Microsoft.Net, and Android. The J2SE configuration is useful for desktops and servers that use a standard Java Development Kit (JDK) to execute LEAP. The pJava configuration is obsolete and was the predecessor of J2ME. MIDP is a profile of JME, more specifically, JME's configuration of Connected Limited Device Configuration (CLDC) thatis popular for mobile devices. The Microsoft .Net configuration is needed for interacting with the .Net infrastructure and Microsoft supported applications. The Android configuration allows deployment and interaction with devices that support the Android operating system.

LEAP provides the split execution mode to address the connectivity issues of mobile devices (Bellifemine et al., 2007). Remember that JADE is implemented with containers, which are not appropriate for mobile devices and their connectivity issues. Split execution mode divides a container into a front-end and back-end. The front-end provides a subset of a normal container's features and resides on the mobile device, while the back-end provides the rest of the features and resides with the main container's dedicated machine. The combination of the front-end and the back-end through the communication of a dedicated connection create the same functionality of a normal JADE container. The split execution mode provides a number of advantages that are beneficial to the connectivity issues. The front-end has a smaller footprint than a normal container and has a faster boot time. The IP address of the mobile device is not

used, so its change does not affect operation. The wireless link is also a dedicated connection between the front-end and the back-end, so it is optimized.

2.2.4 Jess. Jess (Java Expert System Shell) (Friedman-Hill, 2003) is a rule engine for the Java language that provides rule-based programming suitable for automating an expert system. Jess maintains a working memory where facts are stored. A rule is structured as an *if-then* statement. The *if* portion of the rule (the "left-hand side") contains the condition that must be matched against facts in working memory for the rule to fire. The *then* portion (the "right-hand side") specifies the actions that may be performed when the rule fires; these actions typically add or retract facts to or from working memory. When the left-hand sides of several rules match, they are placed on the agenda, and Jess uses a conflict strategy to prioritize the rules and to determine which fires. The conflict strategy takes into account the complexity and specificity of the rule for prioritization; also, the programmer can explicitly specify the priority of a rule. The Jess rule engine uses the Rete algorithm, which efficiently matches the left-hand sides of rules against facts in working memory. To use this algorithm, Jess builds a network of nodes, where each node corresponds to a pattern occurring in a rule. The Rete algorithm sacrifices some memory for a vast increase in speed.

It is reasonable to use Jess to endow JADE agents with intelligence. Policies can often be encoded as Jess rules, and Jess provides a natural way to enforce policies within a JADE-based multiagent system. The main difficulty with integrating Jess with JADE is that a JADE agent runs on a single thread, and the rule engine may require so much time as to prevent the agent from performing its normal behavior, such as communicating with other agents. The solution is to time-limit the execution of the rule engine.

2.3 Web Services

A Web service is a software system designed to support interoperable machine-to-machine interaction over the Web (Web Services). Web services expose and access interfaces described in the format of WSDL (Web Service Descriptor Language). A Web service interacts with other software systems according to the relevant WSDL's descriptions, and it communicates with SOAP (Simple Object Access Protocol) messages. Figure 5 presents the three primary parties of Web service implementations: the service providers, the service consumers, and the discovery agency. Service providers develop Web services and expose the Web service interfaces to service requesters via the Web. Additionally, they may register their services with a discovery agency so that requesters that do not know of a service may find it. The process of registering a Web service with a discovery agency is also known as publishing. The discovery agency, or service broker, maintains a registry of published Web services used to mediate the connection of service providers and service consumers. The discovery agency is also known as the yellow pages for its function of listing services. Service consumers find suitable service providers in the discovery agency and then contact a service provider to use its service, unless the consumer already knows of the service, in which case it can access it directly.

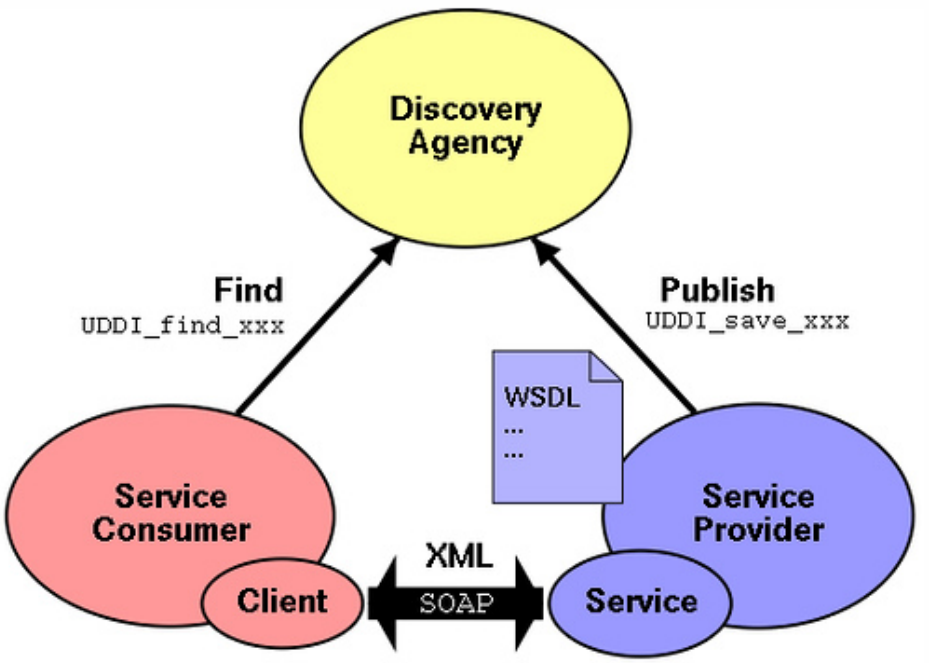

Figure 5 Service-Oriented Architecture of Web services.[3]

Web services are based on an architecture established by standards that enable connection, communication, description, and discovery. The W3C (World-Wide Web Consortium) (w3.org) and OASIS (Organization for the Advancement of Structured Information Standards) (oasis-open.org) are the primary organizations that promote standards and protocols for Web technologies, in particular, Web services.

The following sections discuss the underlying technologies needed for implementing Web services. The discussion begins with XML (eXtensible Markup Language), the backbone of communication for the Web. Then JAXB (Java Architecture for XML Binding) is discussed for an efficient method of binding XML content with Java objects. SOAP is a protocol for Web service communication between clients and providers thatis based in XML. WSDL is a specification language for describing how clients access and utilize the Web service. REST (Representational State Transfer) is a resource-oriented alternative to the standard service-

[3] This figure was referenced from http://www.flickr.com/photos/dullhunk/415645479/#/

oriented approach to Web services. WSIG (Web Services Integration Gateway) allows JADE agents to interact with Web services.

2.3.1 XML. XML (eXtensible Markup Language) is a W3C standard that facilitates the sharing of structured data across the Web (XML Schema). The ability to allow users to define elements and attributes leads to the extensible nature of XML. For an XML document to be useful, it must be not only *correct* but also *valid*. An XML document is said to be correct if it is well-formed, that is, if it respects the basic syntactic rules (such as that elements must be properly nested and may not overlap).

The principle components of an XML document include elements, attributes, data types, and namespaces. Elements consist of start and end tags (<element></element>) along with the enclosed data content. Element tags create a hierarchy that preserves a structure presented by the data, or content. Attributes present additional information about the elements they are embedded within (<element attribute="…"></element>). Data types restrict the particular data content held by the element's start and end tags, which can includemany types such as integers, strings, and even, dates. Namespaces provide a mechanism for distinguishing between identical names, as it is not unusual for different vocabularies to address an element or attribute with the same name.

An XML document is said to be valid if it conforms to a specified XML schema, which may be a DTD (Document Type Definition) or a document written in the XML Schema Definition Language. DTDs are an older standard and are written in an extended BNF notation. The XML Schema language (XML Schema) is more powerful than the DTD notation in part because it allows new data types to be defined and multiple namespaces to be handled easily. A document in the XML Schema language must be a correct XML document and, in fact, must be valid (in relation to the schema for the XML Schema language).

2.3.2 JAXB. XML is one of the backbone technologies of the Web. The ability to capture structure along with data content is a huge benefit of XML. Because of the specialized structure, XML content must be accessed in a specific way to preserve the data hierarchy or relationships between data elements.

XML parsing is required for accessing XML data content. DOM (Document Object Model) and SAX (Simple API for XML) are two methods of XML parsing. DOM accesses the XML data content using a tree-based method (DOM). SAX is a stream-based method that saves memory (Simple API for XML). SAX and DOM both require knowledge of XML and the specific documents being accessed.

JAXB (Java Architecture for XML Binding) (Laun) is a Java package for accessing XML content similar to SAX and DOM. JAXB makes it easy to access XML documents as Java data structures where classes correspond to element definitions in the corresponding XML schema. JAXB allows Java developers to access and process XML data without having to know XML structure or XML processing. It is also a two way process, so content can go from XML to Java objects and vice versa.

2.3.3 SOAP. SOAP is an XML-based messaging framework for the exchange of information over networked environments (Web Services). SOAP is the primary communication mechanism of Web services, where it serves as the foundational technology for development of Web services. SOAP is the successor of XML-RPC (Remote Procedure Call) (St. Laurent et al., 2001), which naturally used XML to encode calls to remote procedures over networks. The framework defines four specifications:

(1) the SOAP processing model defines the rules for processing a SOAP message,

(2) the SOAP extensibility model defines the SOAP modules and features,

(3) the SOAP underlying protocol binding framework specifies how to bind to an underlying protocol for exchanging SOAP messages, and

(4) the SOAP message construct defines the structure of a SOAP message.

2.3.4 WSDL. WSDL is an XML-based standard for describing how to interface with a Web service (Christensen et al., 2001). A WSDL specification includes the definitions of data types, input and output message formats, the operations provided by the service, network addresses, and protocol bindings. In particular, WSDL describes services as collections of ports, or service endpoints. A WSDL document contains abstract and concrete descriptions. The abstract descriptions define the datatypes referenced and the collection of the operations exposed by the Web service. The concrete descriptions define how the inputs and outputs of each operation map to the appropriate communication protocol and the collection of endpoints, or addresses of the bindings, that represent the overall Web service.

2.3.5 REST. In terms of Web services, Service-Oriented Architecture (SOA) has been the de facto approach for the past years, but Resource-Oriented Architecture (ROA) is quickly becoming a worthy competitor. ROA focuses on treating every web site and web application as a service that can be accessed through standard web technologies of HTTP, URIs, and XML (Richardson and Ruby, 2007). ROA is a fresh approach to distributed systems as well as a different approach to distributed programming. ROA introduces a set of principles for taking advantage of a simpler approach to the Web, which is referred to as REST (Representational State Transfer). Web services that conform to this set of principles are known as *RESTful* Web services.

REST revolves around the concepts of resources, connectors, representations, components, and URIs. Resources are accessible sources of information located on the Web,

such as HTML documents, XML documents, and search engines. Connectors are media that serve the request for these resources. Connectors include servers, caches, or even other clients. Representations are the returned data formats of the requested resources, such as HTML, XML, and plain text. Components are defined as the objects that communicate via a specific protocol, usually HTTP, to exchange a representation of a resource. URIs (Universal Resource Identifiers) are global identifiers for the resources located on the web. As a general note, when speaking of the Web, one usually refers to URLs (Universal Resource Locators). But URLs are a specific class of URIs, and "URI" is used to refer to a more generic case. In general, components request resources by their URIs on connectors. The responding component returns a representation of the resource to the requesting component.

Roy Fielding's dissertation (Fielding, 2000) draws on REST principles to identify drawbacks of SOA and possible solutions. REST defines the set of principles that try to address several issues including complexity, scalability, and interoperability. With the growing number of independent components, interoperability is becoming a more pressing issue. As the number of independent components grows, the number of interactions between them grows at an even greater rate, so scalability concerns must take into account interactions between components. In addition to the increasing number of components and component interactions, the number of component interfaces also adds to the complexity of the architecture. Referring to manageable complexity, scalability, and interoperability, Richardson and Ruby state:

> "Simplicity—that despised virtue of HTTP 0.9—is a prerequisite for all three. The more complex the system, the more difficult it is to fix when something goes wrong." (Richardson and Ruby, 2007)

REST simply defines a set of architectural principles for developing Web services. These principles include a client-server architectural style, stateless communication, cacheable communication, uniform interface, layered systems, and code-on-demand. The client-server

architectural style is the fundamental model of the World Wide Web. This style separates the client user interface from the server data storage, which introduces portability and scalability to the Web.

The stateless communication principle pertains to the absence of state information stored on the server, although it does allow for state information to be stored on the client, through such mechanisms as session cookies. The REST stateless communication principle encourages sending a request with all necessary information to service it properly. The cacheable communication principle states that data can be cached and reused if labeled as cacheable. The uniform interface principle specifies certain guidelines to which all interfaces must conform and requires that all components share a uniform interface.

The layered system principle refers to the hierarchical layers of a system; this restricts the layer's behavior to focus on its interactions with the next layer. The code-on-demand principle involves downloading and executing code that allows clients to extend their functionality.

2.3.6 UDDI. UDDI (Universal Description, Discovery and Integration) is an OASIS standard that defines the method for publishing and discovering Web services and their related information (uddi.org). UDDI implementations provide APIs (Application Programmer Interfaces) and Web services endpoints for publishing information and inquiring for information from a UDDI registry. To properly access a UDDI registry, a username and password are required, along with the appropriate UDDI URL for publish or inquiry actions. Information published to a UDDI registry is accessible by all users of the registry while inquiring allows users to learn about businesses, their Web services, and how to access any Web services of interest.

UDDI has three components (“pages”) for the classification of information: the white, yellow, and green pages. The white pages provide information about the businesses that provide Web services, the yellow pages provide information about the classification of Web services based on the services they provide, and the green pages provide information about the bindings of the Web service, in particular, WSDL information. Thus, a business with several Web services has several yellow page entries, one for each Web service, while there is only one white page UDDI entry for the business itself. Also, the Web service may have multiple green page entries for multiple interfaces, each corresponding to an interface of the Web service.

2.3.7 WSIG. As Web services are passive (they can only provide new information when invoked) while agents are autonomous and proactive, integrating agents and Web services would be the logical progression. WSIG (Web Services Integration Gateway) is a JADE add-on that allows integration between the Web services framework and the multiagent systems framework, in particular, the JADE platform (JADE Board, 2011). WSIG offers bidirectional discovery and remote invocation of Web services by JADE agents and of JADE agent services by Web services.

The WSIG architecture presented in Figure 6 contains the WSIG Servlet and WSIG Gateway Agent. The WSIG Servlet connects the WSIG system to the Web acting as a front-end that services requests from the Web and responses to Web clients, while the WSIG Gateway Agent connects the WSIG system to the multiagent system framework, acting as a back-end that fulfills the requests. The WSIG Gateway Agent manages the entire WSIG system.

The WSIG system includes four important functionalities. It receives agent service registrations from the JADE DF and translates them into the corresponding WSDL descriptions, which are registered with the UDDI yellow-pages. The system also receives Web service

registrations from the UDDI repository and registers them with the DF as agent services. The WSIG system receives a Web service invocation request from a JADE agent, finds the service in the DF, translates the request into SOAP, and sends it to the Web service; a service response is translated into ACL and sent to the requesting JADE agent. Lastly, the WSIG system receives an agent service request from a Web service client, finds the service in the UDDI repository, translates the request into ACL, and sends it to the requested agent, without interaction with the DF; an agent response is translated into SOAP and sent to the requesting Web service. Although the functionality of agents discovering Web services as agent services is a documented feature of WSIG, it is not fully implemented. A simple alternative would involve the agent directly invoking the Web service via Axis calls or SOAP messages.

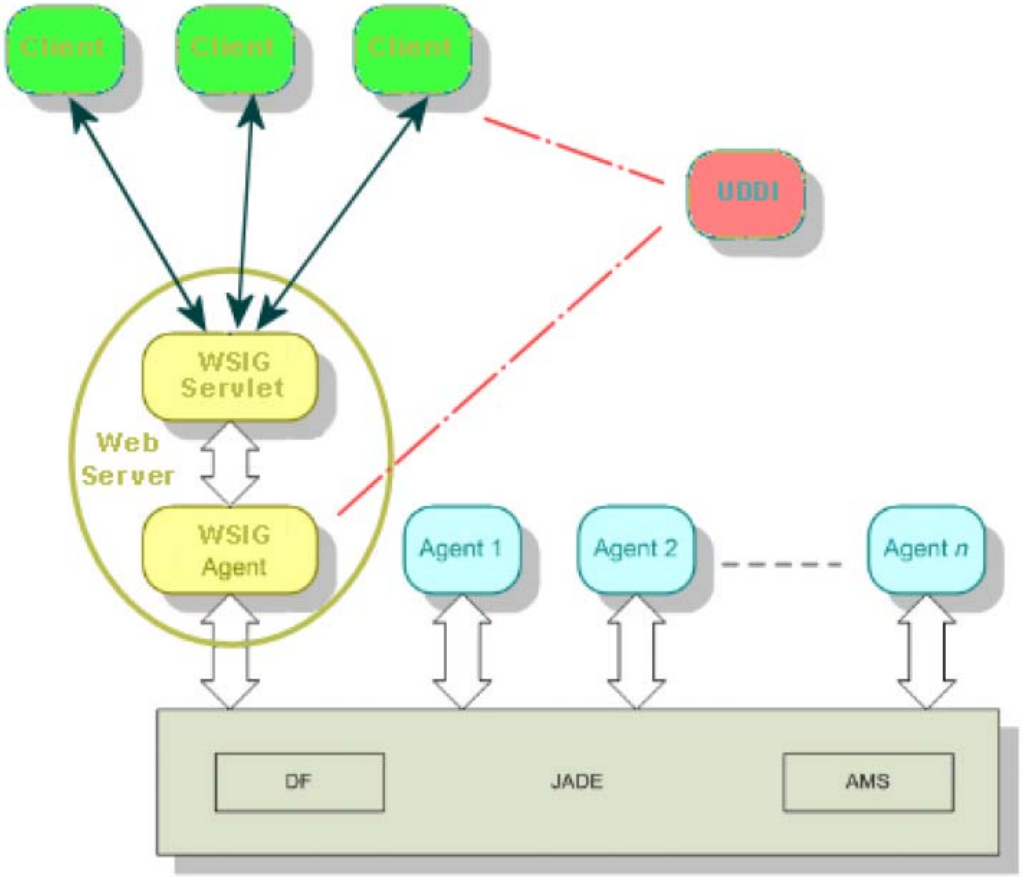

Figure 6 WSIG Architecture. (JADE Board, 2011)

2.4 Semantic Web

"Tim Berners-Lee has a two-part vision for the future of the Web. The first part is to make the Web a more collaborative medium. The second part is to make the Web understandable, and thus processable, by machines." (Daconta, 2003)

The Semantic Web is the next evolutionary step to the Web. The principal concept of the Semantic Web revolves around the word "semantic," which relates to meaning. The Web allows large volumes of data to be accessible to everyone. The Semantic Web allows a more efficient use of that data through the addition of meaning (Daconta, 2003). With the absence of meaning, users or programs must interpret this data to make meaningful decisions. With the advent of the Semantic Web, focus on the Web is shifting from application-centric to data-centric. The key goal of the Semantic Web is to create "smart data."

The Semantic Web offers technologies and concepts that resolve issues of information technology, such as information overload and poor content aggregation. Information overload occurs when one is presented with so much data as to impair decision making. Content aggregation refers to combining data from separate sources. The Web primarily consists of unstructured content often consumed by human users, while the Semantic Web focuses on structured, formal statements to be consumed by machines. Web content also contains formatting instructions to structure information in a nice presentation, while Semantic Web content contains information for meaning and logic.

Figure 7 shows the Semantic Web Stack, the hierarchy of Semantic Web standards and concepts. The foundation consists of Unicode (What is Unicode, 2010), a standard for encoding and representation of text, and the URI (Universal Resource Identifier) (Connolly, 2006) standard for uniquely naming networked resources. The next layer incorporates the self-

descriptive standards of XML (eXtensible Markup Language), XML Schema, and namespaces. The next two layers, with RDF, RDF Schema, and ontology vocabulary, take advantage of XML's descriptive capabilities to describe resources and concepts. The logic layer presents a framework for writing rules. The proof layer evaluates the Logic layer's rules based on the RDF and ontology data. The trust layer adds a decision making mechanism from the basis of the proof layer.

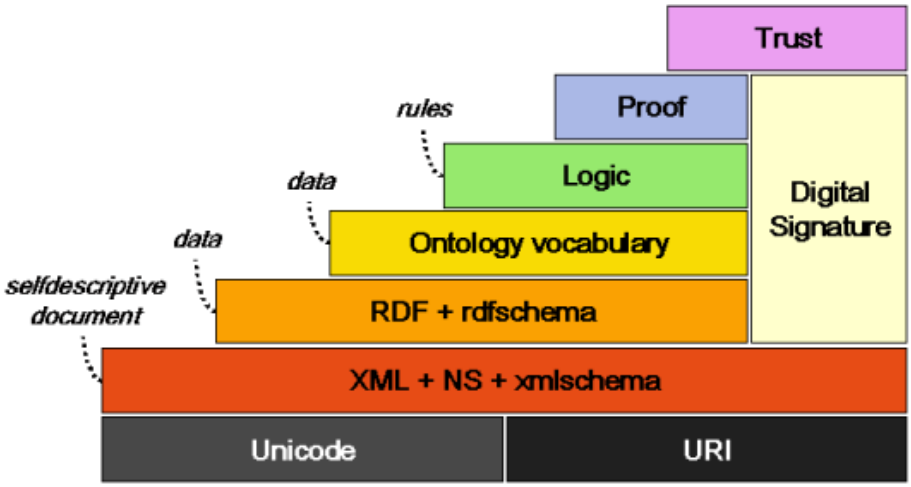

Figure 7 The Semantic Web Stack. (Daconta, 2003)

Through a brief explanation of RDF, ontologies, the Jena Semantic Web framework, and the Pellet reasoner, the concept of the Semantic Web and its use in this research can be understood.

2.4.1 RDF. Resource Description Framework (RDF) is a language used to describe resources and the relationships between other resources (Hebeler et al., 2009). The definition of a resource is quite broad, but it includes anything accessible via the Web, A resource could be anything available on the Web including HTML documents, audio or video files, data files, or persons. The resource can be accessed by its Uniform Resource Locator (URL), or more specifically its Uniform Resource Identifier (URI). All URLs are URIs because a URL is a portion of a URI but not all URIs are URLs. RDF captures metadata about resources. For

instance, RDF metadata on a document could include the information on the author, the creation date, and the type. The metadata is expressed with statements called triples. A triple is a three-part statement that consists of a subject, a predicate, and an object. This structure allows triples to clearly model English language and syntax. The subject is what the metadata is describing. The predicate is the relationship between the subject and the object. The object could be anything including another resource that describes the subject or has a particular relationship with the subject. An RDF document represents a labeled directed graph, which is the actual data model from the point of view of the Semantic Web. A triple represents a labeled edge in the graph: the subject is the source node, the predicate is the label on the edge, and the object is the destination node.

RDF is used in a variety of serializations, or formats, four of which are widely used, namely, RDF/XML (Hebeler et al., 2009), N3 (Notation 3) (Berners-Lee and Connolly, 2008), Turtle (Terse RDF Triple Language) (Beckett and Berners-Lee, 2008), and N-Triples (N-Triples). RDF/XML is an XML-based RDF serialization and is the only standard syntax for RDF serialization, so all “true” Semantic Web applications must support it. Since an RDF/XML serialization of an RDF graph is generally not unique, different tools can produce different serializations of the same graph. Unlike RDF/XML, the other serializations are not based in XML. N3 was designed to be more concise and human-readable than RDF/XML. Turtle is another RDF serialization that is a subset of N3. The primary difference between Turtle and N3 is that Turtle does not go beyond the RDF graph model class. N-Triples is another RDF serialization that is a subset of Turtle and uses the same syntax as Turtle but incorporates some restrictions such as not supporting certain directives, like @prefix, or shorthand statements. N-

Triples' simplicity is useful for streaming data and serializing RDF because of its single line format.

Resource Description Framework in attributes (RDFa) allows RDF statements to be included in ordinary HTML/XHTML/XML files using formally defined attributes (Hebeler et al., 2009). The essence of RDFa is to provide a set of attributes that can be used to carry metadata in an XML language. Two common attributes include `about` and `property`. The `about` attribute specifies the resource the metadata describes while the `property` attribute specifies the RDF property of the content.

2.4.2 Ontologies and OWL. An ontology is a formal representation, or specification, of knowledge through the definition of specific concepts (Hebeler et al., 2009). It uses a predefined set of terms, or reserved keywords, to define a set of concepts of a domain, along with the relationships between the concepts. Generally, an ontology is a logic-based model for describing knowledge or, in a more restricted sense, a vocabulary or a taxonomy. A vocabulary is simply a listing of defined concepts, or terms, while a taxonomy is a classification of concepts. Ontologies allow the semantics behind terms and relationships to be expressed, along with the context of their usage.

OWL (Ontology Web Language) (McGuinness and Harmelen, 2004) allows ontologies to be defined. It builds additional resources onto the RDFS vocabulary to allow more detailed ontologies for the Web. OWL also incorporates RDF and XML Schema datatypes.

The namespace URL for OWL is `http://www.w3.org/2002/07/owl#`, which is conventionally associated with the `owl` prefix. OWL2 (OWL 2, 2009) is the latest version of the OWL vocabulary, which extends the original OWL vocabulary. The `owl` namespace prefix is also conventionally used with OWL2. The namespace URL for `OWL2` is

`http://www.w3.org/ns/owl2-xml`, while the `owl` namespace prefix is also conventionally used.

The OWL specification includes three sub-languages of OWL. They include OWL Lite, OWL DL, and OWL Full, which increase in expressiveness. OWL Lite provides the basic functionalities of the OWL language for classification hierarchies and simple constraints. OWL DL (Description Logic) provides all the constructs of the OWL language. It was designed to provide a level of complete expressiveness. OWL DL guarantees computational completeness so all relations are guaranteed to be computed. OWL Full provides all the constructs of the OWL language, like OWL DL, but with the compatibility of RDF. It also provides a level of complete expressiveness, but unlike OWL DL, it does not guarantee computational completeness. Ontologies are the core of the Semantic Web, where ontologies include far more than the structure of an RDF graph or a list of defined concepts. The resources constituting the traditional Web and the resource descriptions constituting the Semantic Web are both inherently distributed. Since OWL extends RDF, OWL directly supports this distributed nature, because RDF allows the descriptions of resources that may be local or remote. OWL also allows an ontology to import other ontologies, which is another aspect of the distributed nature of the Semantic Web.

The Semantic Web attempts to make valid inferences on this distributed knowledge, but, to accomplish this, two important assumptions must be met: the open world assumption and the no unique names assumption. The open world assumption states that not knowing a statement is true does not justify taking it to be false. In contrast, the closed world assumption states that what is not known to be true is false. The open world assumption assumes that information is incomplete and forces information to be additive (or “monotonic”): if something follows from what was previously known, it still follows when something is added to what is known.

The no unique names assumption states that, unless explicitly stated otherwise, one cannot assume that resources identified by different URIs are distinct. In the distributed environment of the Web, it is unreasonable to assume that everyone uses the same URI to identify the same resource. It is common for resources on the Web to have multiple URIs. The open world assumption and the no unique names assumption impact inference capabilities relating to the completeness of information and the uniqueness of resources.

Suggested Upper Merged Ontology (SUMO) is one of the largest formal ontologies publicly available to the research community (SUMO). SUMO is an upper ontology, where ontology focuses on a set of concepts and relationships in a particular domain; an upper ontology focuses on concepts that are more general and abstract. Upper ontologies include broad concepts that could be extended into more specific domain areas, so other ontologies can be built from upper ontologies. SUMO is ideal for ontological concept re-use where SUMO includes over 1000 concepts, 4000 axioms, and 750 rules. It is maintained by the IEEE and a candidate for the standard upper ontology.

The NASA Jet Propulsion Laboratory (JPL) has developed a set of upper ontologies for the Earth system domain. These ontologies are NASA/JPL's Semantic Web for Earth and Environmental Terminology (SWEET) ontologies (SWEET). The ontologies include several thousand concepts from several domains such as the Earth realm, substances, etc. The Earth realm ontology includes Earth-related concepts such as atmosphere or ocean. The substances ontology includes concepts for the basic components of nature such as particles or chemical compounds. Figure 8 shows the high-level SWEET ontologies and some of the major lower-level ontologies. The SWEET ontologies are highly modularized for reusability so the high-level ontologies are easily composed of the lower-level ontologies.

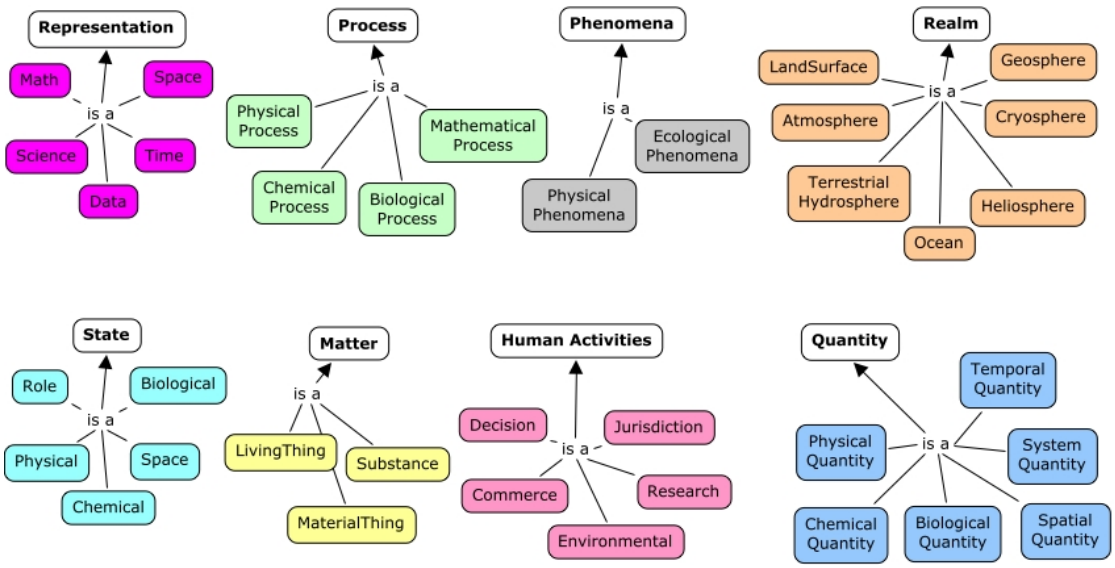

Figure 8 Overview of SWEET Ontologies. (SWEET)

2.4.3 Protégé. Protégé (Getting Started with Protégé, 2011) is an open-source software package for creating and editing ontologies and knowledge bases. It is independent of platform and domain, where it has proved useful from the biomedical field to pure computer science. Protégé's primary tool is the ontology editor, but it also integrates reasoners, which will be discussed in a later section. Protégé also has a very active user community which includes the open-source development of Protégé, online documentation, and forums for additional assistance.

Protégé provides a simple user interface for its ontology editor. The ontology editor provides separate tab views for the major component of the ontology that include entities, classes, object properties, data properties, and individuals. An entity is a component of the ontology, which includes the classes, object properties, data properties, and individuals. A class is an abstract concept that describes a level in a hierarchy. An object property relates an individual to another individual while a data property relates an individual to a data value. An

individual is a member, or an instance, of a class. Each tab view displays the corresponding hierarchy of entities, the corresponding properties, and their relations.

2.4.4 Jena. Jena is an open-source framework for the development of Semantic Web applications (McBride, 2010). It is a Java-based software package that directly supports RDF, RDFS, OWL, and SPARQL ("SPARQL Protocol and RDF Query Language") (Prud'hommeaux and Seaborne, 2008). It also supports the ability to read and write RDF/XML, N3, and Turtle serializations. Jena uses an abstract model to represent an RDF graph. The abstract model methods that allow models to be added, removed, and queried using its methods. This allows RDF graphs to be created with Jena and RDF files.

SPARQL is a query language for RDF that allows for patterns, conjunctions, and disjunctions. A SPARQL endpoint serves as a service for SPARQL queries to interact with the knowledge base. SPARQL contains four query constructs that interact with a SPARQL endpoint. The SELECT query returns raw values in a table format. The CONSTRUCT query returns valid RDF. The ASK query returns a True or false result. The DESCRIBE query returns an RDF graph. SPARQL only supports querying RDF triples, not non-RDF storage constructs such as relational databases. There are alternatives that allow SPARQL querying outside of RDF, like the D2RQ platform (D2RQ). The D2RQ platform is a system for accessing relational databases as virtual RDF graphs. It offers RDF-based access to the content of relational databases. D2RQ allows querying a relational database using SPARQL.

2.4.5 SWRL. SWRL (Semantic Web Rule Language) (Horrocks, 2004) is a rule-based language for ontologies and knowledge bases. A rule is a statement that determines a consequence based on a current state. Rules take the form of "If …, Then …" statements. The "If" portion is the antecedent that matches the rule, while the "Then" portion is the consequent

that determines the outcome of the rule. SWRL integrates the OWL Lite and OWL DL along with a sub-set of RuleML. RuleML (Boley and Tabet) is an XML-based markup language for defining forward and backward rules.

SWRL provides an abstract and two concrete syntaxes. The abstract syntax is basically a sub-set of Extended Backus-Naur Form. The XML Concrete syntax combines the XML syntax of RuleML with the OWL XML Presentation syntax. The RDF Concrete syntax duplicates the XML Concrete syntax but with RDF/XML.

2.4.6 OWL Reasoners. An OWL reasoner, or a semantic reasoner, is a software package that uses asserted facts to make inferences (W3C: Reasoners). It formulates new facts from what it infers from the OWL ontology. First-order predicate logic is the primary means of a reasoner's inference, although probabilistic logic reasoners are fairly common. OWL reasoners support forward-chaining and backward-chaining inferences. Forward-chaining involves progressing from the facts to a consequence, or end goal, while backward-chaining involves starting from the end goal.

Pellet is an OWL reasoner available as an open-source, Java package (Sirin et al., 2007). It can be easily integrated into the Jena Semantic Web framework and will be used in this research. Pellet offers better performance than the pre-packaged reasoners included with Jena. It also offers an added benefit over Jena's reasoners, which is the support of OWL DL (OWL Description Logic). Pellet also includes a consistency checker and syntax checker for OWL. It also supports SWRL. Pellet provides three different interfaces, which include an interface for an RDF parser, a SPARQL parser, and an interface that supports three input formats from Jena, OWL, or DIG (Decentralized Information Group), an interface for access to Description Logic Reasoners.

2.4.7 Data Fusion & Situation Awareness. Data fusion combines data from multiple sources to infer more detailed data and information (Liggins et al., 2008). Data fusion provides a distinct advantage over single data sources and improves the accuracy and quality of the integrated applications. These applications range from medical diagnosis to battlefield enemy tracking. The Joint Directors of Laboratories developed one of the most popular data fusion models. The JDL model categorizes data fusion processes into five separate levels based on their functionality. The levels include signal refinement, object refinement, situation refinement, threat refinement, and process refinement. Signal refinement preprocesses the signal for further work; this is called level 0. Object refinement estimates an object's position, identity, and characteristics; this is level 1. Situation refinement determines relationships between multiple objects and events in the environment; this is level 2. Threat Refinement involves future projection of the objects and the state of the environment; this is level 3. Process Refinement involves improving the entire data fusion process by monitoring the previous levels; this is level 4. Figure 9 shows the JDL model and the connections between the five levels.

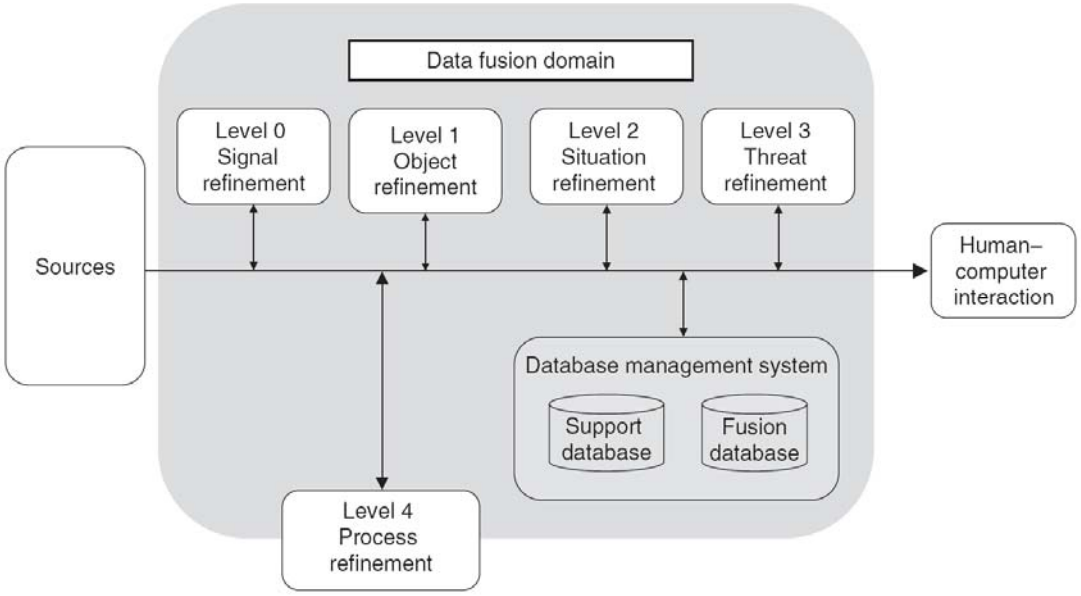

Figure 9 JDL Data Fusion Model. [4]

Situation awareness is the fundamental understanding of an environment and its interacting elements. Situation awareness plays a vital role in decision making where sensor networks and sensor webs have become a significant proponent. In any of these scenarios, fully understanding the current state of the environment significantly improves decision making and increases the possibility of achieving future goals in that environment. Endsley (2004) states that situation awareness provides "the primary basis for subsequent decision making and performance in the operation of complex, dynamic systems...(Endsley)". Endsley presents a three level model of situation awareness. Level 1 involves perception of interacting elements and components of the environment. Level 2 pertains to comprehending the environment while level 3 involves projection of the environment into the future. Data fusion and situation awareness are significant areas that while they directly support one another, they also overlap.

[4] This figure was referenced from http://www-dsp.elet.polimi.it/ispg/ASTUTE/wiki/lib/exe/detail.php?id=d22%3Ainformation_retrieval_and_fusion%3Amultimodal_information_fusion&media=d22:information_retrieval_and_fusion:jdl-process-model.png

CHAPTER 3

Literature Review

The following sections present a concise literature review on the advancements in the area of sensor webs. The discussion begins with sensor networks and their fundamental elements, sensor nodes. Sensor networks are then extended into sensor webs where the collaborative, accessible, and semantic senses are explored.

3.1 Sensor Networks

Archana Bharathidasan and Vijay Ponduru present the idea that sensor nodes are “small, inexpensive, low-power, distributed devices, which are capable of local processing and wireless communication” (Bharathidasan and Ponduru, 2002). The addition of wireless technology has introduced wireless sensor networks. Wireless sensor networks integrate wireless sensors and components that incorporate radios for communication without the hassle of wires. Sensor networks generally integrate sensors directly wired, or tethered, to workstations, hubs, or routers for connectivity. Bharathidasan and Ponduru add that “… nowadays, the focus is more on wireless, distributed, sensing nodes. But, why distributed, wireless sensing? When the exact location of a particular phenomenon is unknown, distributed sensing allows for closer placement to the phenomenon than a single sensor would permit” (Bharathidasan and Ponduru, 2002). Wireless sensors add greater flexibility through less restricted sensor placement (Lewis, 2004). Wireless sensor networks benefit from using radio waves as a communication medium, which are less susceptible to disruption by animals, nature, and other environmental factors than are cables and wires.

Research on sensor networks integrates concepts from several research areas (Chong and Kumar, 2003). Sensing research focuses on the component of sensor networks that transforms a

physical attribute into an electrical signal. A sensor node incorporates sensors for monitoring aspects of the environment, such as temperature, pressure, or humidity, along with additional hardware such as memory and microprocessors. Communication research focuses on efficient and timely data transfer among nodes. Communication in sensor networks has a major impact on energy consumption, so efficiency in this area is significant. Bharathidasan and Ponduru state that "[i]n most cases, the environment to be monitored does not have an existing infrastructure for either energy or communication. It becomes imperative for sensor nodes to survive on small, finite sources of energy and communicate through a wireless communication channel" (Bharathidasan and Ponduru, 2002). General computing research integrates a variety of concepts from hardware, software, and algorithms.

There is a variety of attributes of a sensor network that impact its development and efficiency (Chong and Kumar, 2003). The architecture of sensor networks can be centralized, distributed, or a hybrid, along with wired or wireless communication of a wide range of choices of sensor nodes.

3.2 Sensor Webs

The term "sensor web" is used in three different but related senses. In one sense, it refers to a sensor network where the nodes are not simply passive but also collaborate to respond in intelligent ways to events in the environment. In another sense, "sensor web" refers to a sensor network that produces and consumes Web services. Then, in a third sense, "sensor web" refers to a sensor web that generates contextual information by the addition of metadata. Sensor webs in the first sense are being actively pursued by the Advanced Information Systems Technology (AIST) Program (Suri et al., 2007) of NASA's Earth Science Technology Office (ESTO). Sensor webs in the second sense are being standardized by the Open Geospatial Consortium (OGC)

(opengeospatial.org). Sensor webs in the third sense are primarily being researched by Wright State University (Sheth et al., 2008). We shall refer to a sensor web in the first sense as a collaborative sensor web, a sensor web in the second sense as an accessible sensor web, and a sensor web in the third sense as a semantic sensor web. Clearly, these three senses are compatible and complementary. Some research integrates at most two senses, mostly the collaborative and accessible senses. The AIST Program incorporates aspects of sensor webs in the second sense, as is natural, given that it involves satellite-based sensors among other kinds of sensors. This research presents a prototype integrating all three senses.

3.2.1 Collaborative Sensor Webs. A sensor network that incorporates components that collaborate in collecting data and adapting in response to certain environmental observations will be referred to as a collaborative sensor web. Collaborative sensor webs offer autonomous operation through reconfiguration by the adaptive and reactive components. The advancements in wireless capabilities make collaborative sensor webs an ideal platform for wireless sensor nodes.

Certain issues must be addressed with the collaborative sensor web concept, including ad hoc deployment of sensor nodes, autonomous operation of the sensor web, sensor node reconfiguration and mobility, and limitations imposed by power sources and the communication infrastructure (Bharathidasan and Ponduru, 2002). No overall configuration is imposed by the deployment of the sensor nodes, so the location and connectivity of sensor nodes is ad hoc, left to the individual nodes and their operators. The operation of the sensor web must be autonomous, functioning properly without human intervention. In particular, sensor nodes must reconfigure themselves in response to changes in the environment. One way for nodes to

reconfigure themselves is by migrating. Finally, the sensor nodes must not be restricted by power sources or communication links.

NASA's AIST program has funded numerous projects with research interests in this area of collaborative sensor webs. According to Karen Moe of NASA's Earth Science Technology Office, "[o]ne of the most promising aspects of this family of networks is their ability to reassemble themselves to fit changing environments – this is the Sensor Web" (Moe, 2007). The AIST program has emphasized several concepts, including *ad hoc* integration, self-organizing nodes, distributed control, and embedded intelligence, which contribute to the overall goal and vision of the sensor web (Moe, 2007). These concepts meld into a network that is decentralized and non-orchestrated, where node failure does not catastrophically affect the overall sensor web performance (Suri et al., 2007).

SEAMONSTER (SouthEast Alaska MOnitoring Network for Science Technology Education and Research) (Heavner et al., 2007) is a sensor web developed by the University of Alaska Southeast in collaboration with Microsoft as part of NASA's AIST Program; it is also associated with the NOAA Interdisciplinary Scientific Environmental Technology (ISET)Cooperative Science Center. SEAMONSTER is a test platform for the newest sensor technology. The initial implementation of the SEAMONSTER sensor web focused on the Lemon Creek watershed, near Juneau, Alaska. In this area, lakes form behind ice dams on top of the glaciers. When an ice dam breaks, the lake floods into the area downstream. A pressure transducer placed in the lake behind the dam detects the sudden drop in pressure associated with the lake drainage, which serves as an alert for a flood in the watershed. When the pressure transducer detects the event, it sends messages to other sensors, which are appropriately reconfigured. Figure 10 shows a collaborative sensor web where its nodes are smaller sensor

webs and sensor networks. The nodes collaborate via message passing to respond to certain events in the environment, for example, the firetowers sensor web (also a node of the ecosystem monitoring sensor web) communicates with another node to allow emergency services to respond to the wild fire.

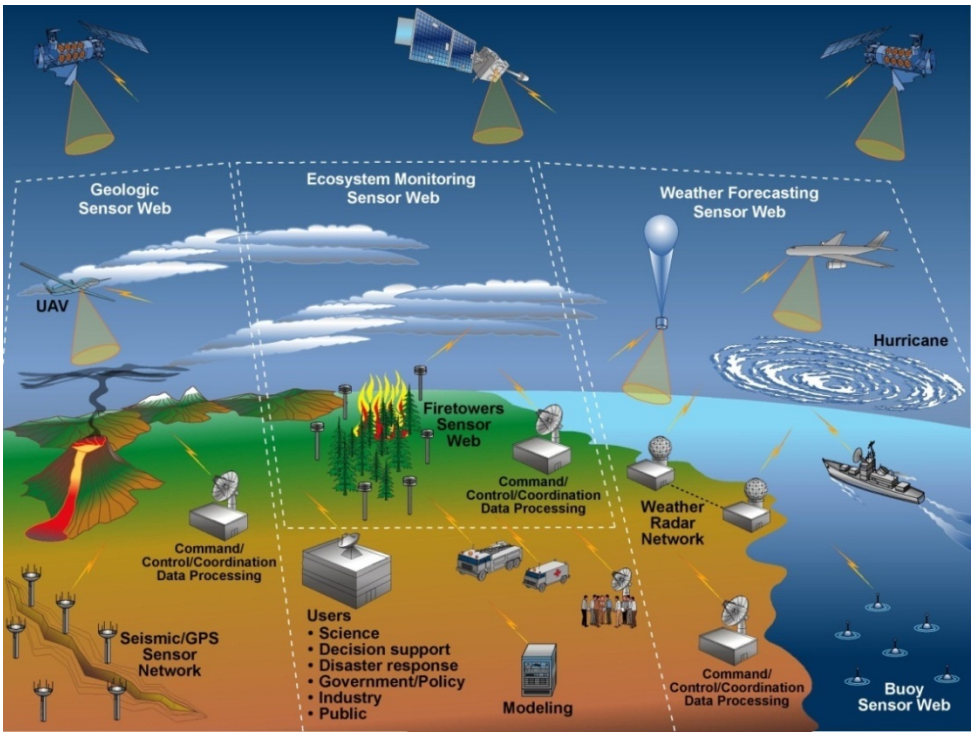

Figure 10 Collaborative Sensor Web.[5]

Recall that a multiagent system allows multiple agents to interact and collaborate. They are ideal for the collaborative aspect of sensor webs. The agents communicate via an efficient mechanism of message passing that directly results in collaboration. Several wireless sensor network implementations have incorporated a multiagent system for different functionalities. Several of these implementations are briefly discussed.

Network management in wireless sensor networks and sensor webs has recently become a popular task for multiagent systems. Today's networks have rapidly expanded into massive

[5] This figure was referenced from http://cert.ics.uci.edu/sesa2011/images/SensorWebImageForEnewsJuly2.jpg

entities. These networks require additional resources for the sole purpose of management to keep the network's nodes connected. Network management pertains to the monitoring and administration of a network so that it can perform optimally for its users. As networks grow, more resources must be allocated to monitor and optimize them. Multiagent systems have proven to be an ideal candidate for the management of networks due to their characteristics of autonomicity, proactivity, reactivity, and socialability. Bieszczad, Pagurek, and White (1998) present an OSI (Open Systems Interconnection) (Tanenbaum, 2002) network management model as the generalization of applications that are ideal for multiagent systems (Bieszczad, 1998). Their management model categorizes the major areas of network management into the management areas of faults, accounting, configuration, performance, and security. Wang and Tianfield (2005) of Glasgow Caledonian University integrate the SNMP (Simple Network Management Protocol) (Tanenbaum, 2002) with MAS research through to develop the OAA (Open Agent Architecture) (Wang and Tianfield, 2005). The OAA organization directly positions agents in a network to control communication and monitor network resources. Along with the tasks of the collaborative aspect, the multiagent system in this research should perform similar tasks.

Herbert, O'Donoghue, Ling, Fei, and Fok have integrated wireless sensor technology with the multiagent system approach for a medical application called the Wireless Patient Sensor Network (WPSN) (Herbert et al., 2006). The WPSN allows patient vital information to be collected and processed remotely. This research is of particular importance to the research reported here. The multiagent system approach of the WPSN attempts to combine the same agent technology used here.

3.2.2 Accessible Sensor Webs. Recall that, in a second sense, “sensor web” refers to a sensor network that produces and consumes Web services. This is the more common sense of “sensor web,” which centers on having the functionality provided by the sensor network and its components available as resources on the World Wide Web. According to the OGC, “A Sensor Web refers to web accessible sensor networks and archived sensor data that can be discovered and accessed using standard protocols and application program interfaces (APIs)” (Botts, 2007). Again, according to Chu and Buyya, “The Sensor Web is an emerging trend which makes various types of web-resident sensors, instruments, image devices, and repositories of sensor data discoverable, accessible, and controllable via the World Wide Web” (Chu and Buyya, 2007).

Figure 11 depicts a sensor web as accessible via the Web. The sensor web client can access the sensors, cameras, and other resources remotely over the Web. This would be accomplished through Web services that exploit the sensor web resources and expose their methods. Clients can directly access the resources by consuming the Web services.

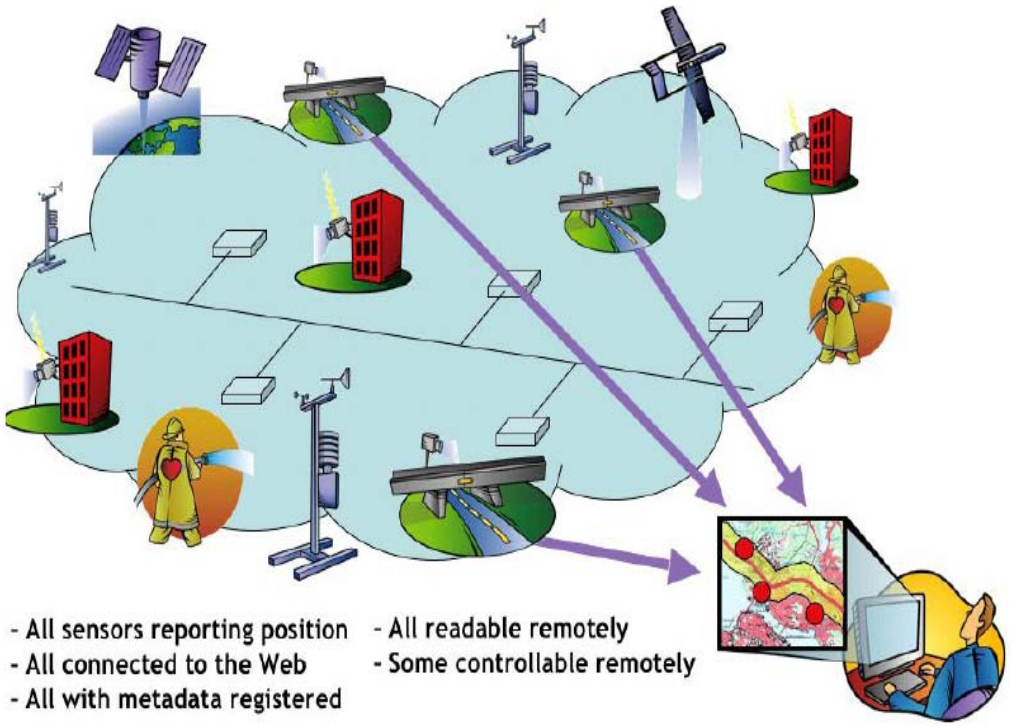

Figure 11 Sensor Web accessible from the WWW. (Botts, 2007)

This sense of sensor web essentially conforms to the OGC's (Open Geospatial Consortium) standards, known as the SWE (Sensor Web Enablement) standards (Reichardt, 2005). The OGC is a standards organization that defines geospatial standards and specifications. It is a voluntary organization that includes over 300 organizations worldwide. The SWE standards are a set of specifications for a framework of Web services offered by (accessible) sensor webs and XML encodings for information communicated by these services. The OGC intends the services to provide a certain set of sensor web functionalities. These functionalities provide a means for the user to discover useful sensors that meet his or her specific needs, a standard interface for data access, a standard interface for assigning tasks to resources, and a way for users to subscribe to alerts published from specific sensors when certain events occur. The following discussion first describes the encodings and then summarizes the Web services.

3.2.2.1 OGC Information Models. The OGC defines three information models, or encodings, within its Sensor Web Enablement initiative. The information models are the Observations & Measurements Schema (O&M), the Sensor Model Language (SensorML), and the Transducer Markup Language (TransducerML). XML Schema is the principal standard used to define the structure and content of the information models (XML Schema).

The O&M Schema (Cox, 2007) is the information model for representing observations, measurements, procedures, and metadata of sensor systems. An observation is a process of associating a value or a sequence of values with a given phenomenon. A measurement is the result, usually a numerical value, obtained from an observation. The procedure is the process used to generate the result, while the metadata is the information associated with the observation, so it is the data about the data. The O&M Schema is required for the Sensor Observation Service because it structures the information exchanged between clients and the service.

SensorML is the information model and XML encoding for presenting important information for discovering sensors and assigning sensors tasks. It is also used for describing sensor processes as well as observation locations of interest to the clients (Chu, 2005). SensorML allows a standard view, or model, of the overall sensor system for the clients.

3.2.2.2 OGC Web Services. The OGC is exploring the development and uses of standard Web services that allow clients to have easy access to sensor web resources and functionality (Botts, 2007). These Web services follow the previously mentioned information models for describing and exchanging data between clients and Web services. Several Web services assist a client in obtaining sensor data, namely, the Sensor Observation Service (formerly known as the Sensor Collection Service) (opengeospatial.org). The Sensor Observation Service provides an interface for clients to request observations and sensor data. It provides a standard access

mechanism for clients to interface with fixed, remote, and mobile sensors. This Web service is an intermediary between the clients and the sensors. The core profile includes three operations: GetCapabilites, DescribeSensor and GetObservation. The GetCapabilites operation provides the means to request a description of the service. DescribeSensor allows the client to request information about a sensor. The GetObservation operation is the heart of the SOS, allowing the client to request observation data generated by a sensor or sensor system contained in a specified observation offering. An observation offering is a logical collection of sensors located in proximity to one another.

The Sensor Repository Service provides storage for useful observations and measurements that have been requested by clients (Chu, 2005). There is also a Sensor Data Grid Service, which provides a means for maintaining large amounts of sensor data (Chu and Buyya, 2007).

The Sensor Registry Service (Chu, 2005) allows a client to discover sensors and services. The Sensor Registry Service catalogs the information about the sensors and the services such as their WSDL addresses. The Sensor Registry Service works in conjunction with the Sensor Observation Service (or, previously, the Sensor Collection Service) and the Sensor Planning Service to satisfy client requests. The Sensor Registry Service finds suitable sensors and services that satisfy the client's search criteria while the Sensor Observation Service process the client's request for data.

There are several Web services that assist in the use of other Web services and sensors, including the Sensor Alert Service, the Sensor Planning Service, and the Web Notification Service. The Sensor Alert Service allows clients to publish and subscribe to alerts from sensors (Botts, 2007). The Sensor Planning Service supports the planning of requests that primarily

involve observation data from the Sensor Collection Service (Chu, 2005). The Web Notification Service delivers asynchronous messages for the Sensor Alert Service and the Sensor Planning Service (Botts, 2007).

Figure 12 depicts the layered architecture of the sensor web. The sensor web's overall architecture includes the application layer, application development layer, application services layer, and the sensor environment layer. The application layer applies to the applications that the user interacts with to access the sensor web and its features. The application development layer applies to the APIs (Application Programming Interfaces) used for creating applications that access the services of the sensor web. The application services layer applies to the OGC services discussed above. The sensor environment layer applies to the hardware setup of sensors, actuators, and other resources.

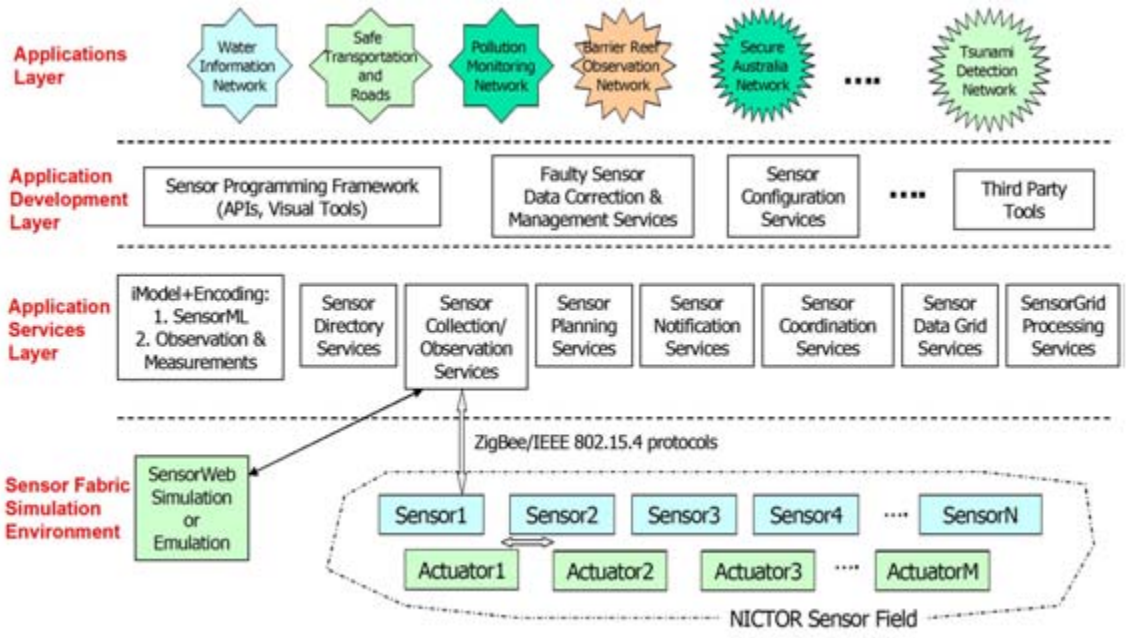

Figure 12 Sensor Web Architecture. (Chu, 2005)

Again according to Chu and Buyya, "Bringing the idea of SOA to sensors and sensor networks is a very important step forward to presenting the sensors as reusable resources which

can be discoverable, accessible and where applicable, controllable via the World Wide Web" (Chu and Buyya, 2007). The popularity of the OGC's Sensor Web Enablement has steadily increased over the past few years. The attraction of the Sensor Web Enablement lies in the standardization of sensor web resources and services, so a user can access different resources or services on different sensor webs in the same manner. The growing number of sensor webs indicates the importance of the Sensor Web Enablement that has been an excellent resolution to the pressing need presented by this growth. Numerous implementations incorporate components of the Sensor Web Enablement in a similar manner used by this research.

3.2.3 Semantic Sensor Webs. The primary function of the sensor web is data collection, which produces vast amounts of data. The Semantic Sensor Web attempts to cultivate the data into more useful information. The developers who take advantage of the Semantic Sensor Web framework can make logical inferences from the relationships between sensors and resources to determine what the data really means.

> "Having greater numbers of accurate, real-time data from sensors of all types will enable people to make better informed decisions about things in the world around them." (Sheth et al., 2008)

The Semantic Sensor Web extends the primary goal of the Semantic Web, where it tries to make data into more useful information. Computers do not really understand anything, so they cannot make sense of anything. So, to achieve this ability to make sense of data, additional data, or metadata, can annotate the data to help create logical connections of the data and infer useful information.

Extending the concept of the sensor web with the semantic sense seems to be a natural fit. This is clearly shown by comparing the semantic connections possible among the thousands of types of available sensors and their limited number of readings the sensors could measure against the endless number of semantic connections for associating the information of the WWW.

Figure 13 depicts what the Semantic Sensor Web tries to accomplish using the progression from natural phenomena to raw sensor data to semantic annotation of Sensor Web Enablement to ontological knowledge of space, time, and theme.

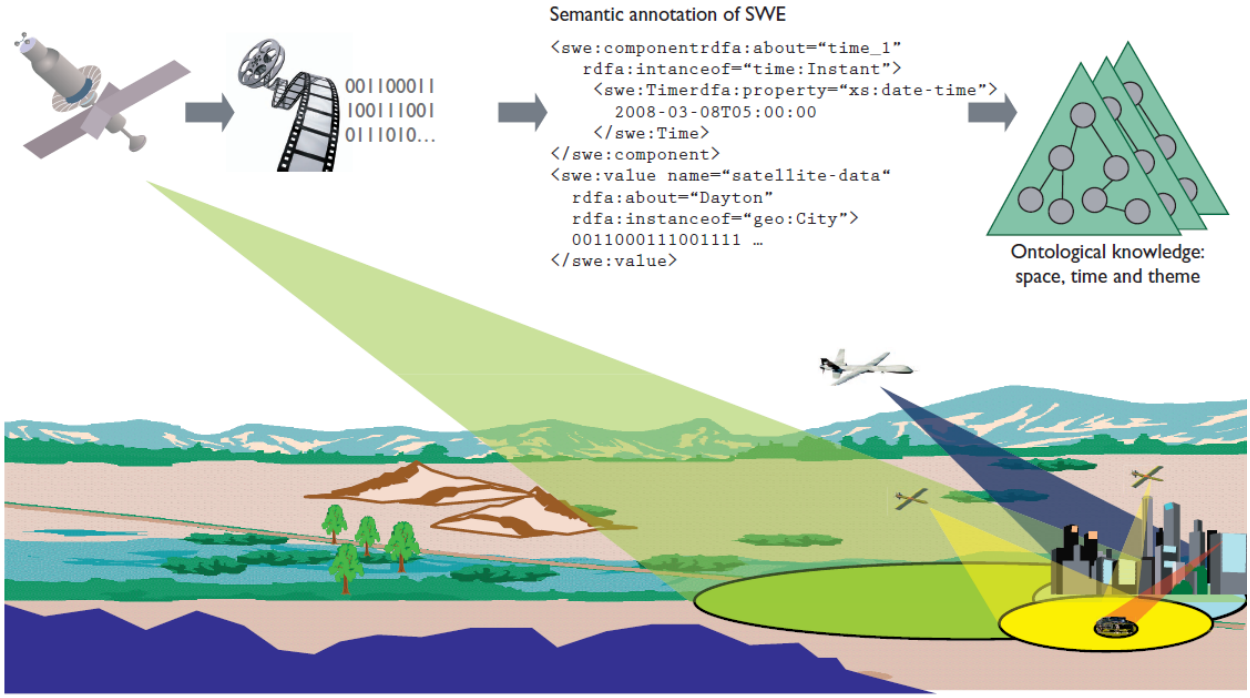

Figure 13 Semantic Sensor Web. (Sheth et al., 2008)

The Semantic Sensor Web extends the OGC's Sensor Web Enablement initiative with the Semantic Web's knowledge modeling abilities of RDF and OWL. Essentially, sensor web data is enhanced with spatial, temporal, and thematic metadata. Spatial metadata incorporates location information, temporal incorporates time, while thematic incorporates descriptions of real-world events.

Wright State University has been the primary research force in the area of the Semantic Sensor Web. Sheth, Henson, and Sahoo of Wright State University have effectively integrated the Semantic Web into the OGC's Sensor Observation Service to create the Semantic Sensor Observation Service (Semantic SOS) (Sheth et al., 2008). Semantic SOS allows users to create

complex queries that are executed over weather readings from the sensors of Ohio's Department of Transportation traffic monitoring system. The general example presented of the Semantic SOS employs a query of freezing or blizzard conditions. The freezing condition would involve only a temperature sensor with a threshold rule specifying freezing temperature but the blizzard condition would involve temperature, wind, and precipitation sensors. The situational awareness presented by the inferences over the semantic metadata and sensor data would allow such complex queries.

"In the same way that the Semantic Web offers to make internet searches and web content more intelligent, the Semantic Sensor Web will be the backbone of establishing semantic relationships across sensors and data around the world." (Dickinson, 2010)

CHAPTER 4

Specific Aims

Remember that the primary objectives of sensor networks and sensor webs involve the collection of data from its environment. With more advanced sensor networks and sensor webs, data collection has become easier and more efficient, while the data is being collected in higher volumes and has become more accurate. Higher volumes of data do not lead to better end results or even more informed decisions from the end users. This issue is the primary challenge of this research. This research seeks to cultivate high-level, useful information from the low-level data that would be of more importance to an end-user for decision making.

The overall goal of this research is to improve end-user decision making by improving the presentation of sensor web data. This improved presentation involves presenting higher-level, easier to understand concepts to the end-user, along with the detailed data captured in the environment. The end-user can easily align their knowledge with the presented high-level concepts of the sensor web to improve decision making.

This research has several specific aims that support the goal and challenge just presented. They also relate the significance of the research to the user as well as to the research community in a way that expands the potential of these technologies. These specific aims are briefly listed then described in more detail below.

- Develop a sensor web system that collects observational data.
- Deliver useful information along with observational data to make results more meaningful.
- Connect specific, low level details to higher level concepts of interest to the user.
- Develop a deeper understanding of the context of the observational data.

- Integrate a user-friendly approach to minimize complexity to the user.
- Develop a modularized sensor web implementation to support reusability.

Specific Aim: Develop a sensor web system that collects observational data.

Hypothesis: The primary motivation behind sensor web research for environmental monitoring focuses on collecting useful data. By reliably getting useful data to the user, the system will be formally classified as a sensor web.

Specific Aim: Deliver useful information along with observational data to make results more meaningful.

Hypothesis: Besides the observational data, the sensor web should also deliver information that is more useful to the user. This information should aid the user in decision making in their current situation.

Specific Aim: Connect specific, low level details to higher level concepts of interest to the user.

Hypothesis: Through the inclusion of a multiagent system framework, the collaborative aspect of sensor web can be realized. The agents will provide the logical capability to connect separate bodies of knowledge and connect similar concepts in different domains.

Specific Aim: Develop a deeper understanding of the context of the observational data.

Hypothesis: Through the inclusion of data fusion and semantic web techniques, the user can have a deeper understanding of the overall situation given the low level details. The cultivated data and information adds context to the observational data which improves decision making for the user.

Specific Aim: Integrate a user-friendly approach to minimize complexity to the user.

Hypothesis: To alleviate needless complexity to the user, custom graphical user interfaces should allow the users to quickly and easily access the resources of the sensor web. Along with the custom user interfaces, the integrated user processes will create a more fluid user experience.

Specific Aim: Develop a modularized sensor web implementation to support reusability.

Hypothesis: Separate components from the sensor web implementation may be of interest to separate users. Most importantly, the software would be common, open-source modules that can be easily acquired. The modules could be of interest to users such as the National Climatic Data Center (NCDC), the Earth Systems Research Laboratory (ESRL), or UAS's SEAMONSTER.

CHAPTER 5

Architecture & Implementation

The prototype of the sensor web is used to realize the goal of presenting the end-user with useful information and improving decision making. The general understanding of the architecture and operation of the prototype leads to a better understanding of the primary goal and objectives of this research. The following sections include detailed discussions about the materials and methodology needed to realize the prototype. The materials section presents the software and hardware components utilized for the prototype. The methodology section discusses how the components fit together into one coherent system and how the components interact to accomplish tasks. The design of the prototype is described in separate layers. The operation of the prototype is detailed along with the conceptual and real-world situations where the prototype would apply.

5.1 Materials

The prototype incorporates a variety of software and hardware components. The following descriptions briefly detail the necessary components and their purpose for inclusion in the prototype.

General workstations: Three workstations (two desktops and one laptop) were utilized for the prototype. The specifications of the workstations were all comparable. One of the desktop workstations and the laptop workstation contained almost identical specifications with an Intel i5 processor, 4 gigabytes of RAM, 240-480 gigabytes of data storage, and the Windows 7 operating system. The other desktop workstation contained an Intel Core2 processor, 2 gigabytes of RAM, 80 gigabytes of data storage, and the Ubuntu 12.04 operating system. Although the specifications of the i5 processor workstations and the Core2 processor workstation significantly vary, they are

still comparable performance-wise. Note that we are not exactly performing CPU-intensive computations but serving Web requests and storing data. The operating system differences account for software needs, in particular, Wview and Postgresql 8.3, which will be discussed later.

Apache Tomcat is a web server package that supports web servlets and web applications. Apache Tomcat (or a similar web server) is required to expose WSIG and the semantic version of the Sensor Observation Service (Semantic SOS) to the web.

Apache Axis2 (Apache eXtensible Interaction System) is an open-source SOAP engine (Web Services – Axis). It is a framework for constructing Web service providers and consumers. Axis2 runs as a servlet that processes the incoming SOAP message and extracts information from the message headers and payloads. Apache Axis2 is incorporated into the Semantic SOS service to facilitate communication through SOAP messages.

SQLite is a minimal SQL database engine. It comes preconfigured with the Wview software. An instance of a SQLite database is connected to the local weather station for data persistence.

PostgreSQL is a full-featured SQL database engine. It provides a graphical interface, administrative services, a SQL editor, and other tools for managing databases. An instance of the PostgreSQL database is used for persistence of the observations from all sensor nodes.

JADE is the framework for developing agents and multiagent systems. It provides a runtime environment for the agents, applications for administering and monitoring the agents, and other useful tools and libraries. The JADE framework is used here to develop the variety of agents that interact with the users.

LEAP is an extension to the JADE framework which allows the development of agents for resource constrained devices. LEAP is used here to develop JADE agents that are deployed to the Sun SPOT sensor nodes.

WSIG is an extension to the JADE framework that allows the development of agents that expose their operations as agent Web services. Web service clients can interact with WSIG agents through traditional SOAP communication. The prototype incorporates WSIG agents to further interact with Semantic SOS results.

The OGC is an international committee for geospatial standards. The OGC develops standards for encoding geospatial data and services for accessing geospatial data. The prototype focuses on several OGC standards which include Observations & Measurements, SensorML, and the Sensor Observation Service.

Protégé is a package for creating and editing ontologies and knowledge bases. Protégé provides an ontology editor for easy creation and development of ontologies. Here, Protégé is used to develop several ontologies that are integrated into the Semantic SOS service and multiagent system.

Jena is a framework for the developing Semantic Web applications that support RDF, RDFS, and OWL. It also supports the ability to query and infer over RDF triples. Jena is incorporated into the 52 North Sensor Observation Service to develop the semantic version of the service (Semantic SOS).

D2RQ is a software package for accessing relational databases as RDF graphs. With D2RQ, data in relational databases do not have to be duplicated with an RDF triple store. The prototype incorporates D2RQ to map SPARQL queries to the PostgreSQL relational database.

The Davis weather station is a device that collects observational data from the environment. It is a professional-grade sensor system that wirelessly communicates its observed data to a remote console connected to a basestation that stores the data for other purposes such as displaying current data on the web for users. The Davis weather station is used here to retrieve the current environmental observations of temperature, pressure, relative humidity, precipitation, wind speed, and wind direction.

Wview is a software package that interacts with a variety of weather stations. Wview includes a collection of processes that include collecting weather station data, configuring the weather station, exposing the weather station data to the web, and other tasks. It coordinates the collection of data from the weather station with other tasks that involve the data. Wview is used here to consistently store the weather station observational data into the SQLite database and properly configure the weather station.

Imote2s are smart sensor nodes that support the TinyOS operating system and the NesC programming language. Imote2s were initially designed for structural health monitoring applications but they are equally effective for environmental monitoring applications. Here the Imote2 sensor nodes collect temperature and humidity observations

Sun SPOTs are smart sensor nodes that support the Squawk JVM. These sensor nodes are compatible with LEAP, which allows JADE agents to be deployed onto resource-constrained devices. The Sun SPOTs serve as connections to data storage for the Imote2 sensor nodes and they take temperature observations.

5.2 Methodology

The prototype integrates a variety of software and hardware components. This section presents the design of the prototype and how the components fit together into one functional system. Figure 14 presents a layered architecture of the prototype.

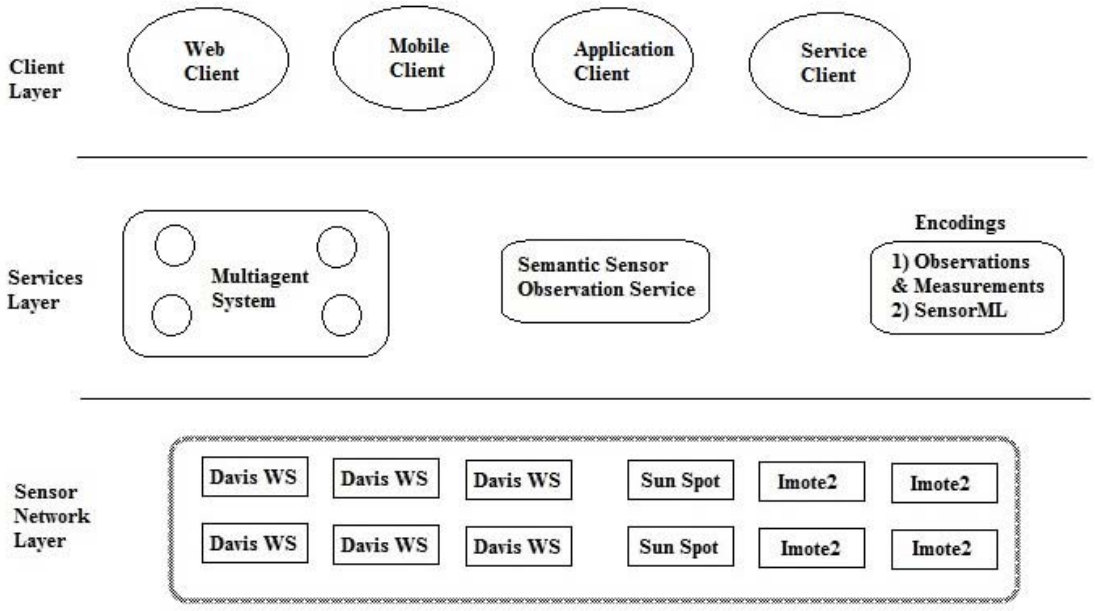

Figure 14 Layered Architecture of the Prototype.

The following presents the high-level layers of the prototype along with their functionality and integration into the overall system. Referring to the above figure, the high-level layers include the sensor network layer, the service layer, and the client layer.

5.2.1 Sensor Network Layer. The sensor network layer integrates the sensor nodes with the sensor network, so this layer is purely responsible for data acquisition. Here our sensor nodes serve as the perceptors to the environment, so they represent our fingers that provide useful information about the environment of interest. The sensor nodes include the previously discussed Sun SPOT and Imote2 smart sensor nodes along with the Davis Vantage Pro2 weather station for

observational data collection. The sensor network layer includes a database for storage of sensed data.

The first component of the sensor network layer involves several Davis weather stations. As previously mentioned, the Davis weather station provides observational data on temperature, relative humidity, pressure, precipitation, wind speed, and wind direction. The Greensboro area hosts several Davis weather stations along with the primary Davis weather station mounted on McNair Hall of North Carolina A&T State University's campus. Figure 15 shows the distribution of Davis weather stations around the Greensboro area that publish observational data to wunderground.com (Wunderground, 2013).

Figure 15 Accessible Davis weather stations of Greensboro, NC. (Wunderground, 2013)

Data from five of the above weather stations (at locations 1, 3, 4, 6, and 9) are used in the prototype. Weather station 1 (station ID KNCGREEN30) is the Westerwood neighborhood of central Greensboro, station 3 (ID KNCGREEN39) is the Pinecroft neighborhood of southern Greensboro, station 4 (ID KNCGREEN49) is a neighborhood of northern Greensboro, station 6 (ID KNCGREEN22) is the Meadowood neighborhood of western Greensboro, station 9 (ID KNCGREEN25) is the Jamestown neighborhood of southwestern Greensboro. From the Wunderground site, data from the five weather stations are updated at slightly different intervals,

but the average interval is five minutes. The McNair weather station was also configured for five minute updates. The Wunderground site provides API access to each weather station through its station ID. So a service call to

http://api.wunderground.com/weatherstation/WXCurrentObXML.asp?ID=KNCGREEN30

would result in observational data from that particular weather station in XML format. Figure 16 shows a sample query of the Westerwood weather station with the station ID of KNCGREEN30.

The sample XML result contains all weather station information and available observational data, along with observational data in different formats such as temperature in degrees Celsius or Fahrenheit. To obtain the observational data from the weather stations available on Wunderground, a Java application was developed that, given a station ID such as KNCGREEN30, queried the Wunderground API for that weather station's latest observational data. The XML result was then parsed and each separate observational value of temperature, relative humidity, pressure, precipitation, wind speed, and wind direction was inserted into the PostgreSQL database, which will be discussed later. The local McNair weather station is not accessible through Wunderground, so another approach was taken to obtain data from the weather station. To obtain the observational data from the McNair weather station, a Java application was developed that queried the SQLite database used by Wview. The query returned the latest observational data inserted, then the temperature, relative humidity, pressure, precipitation, wind speed, and wind direction observational values were inserted into the PostgreSQL database. To synchronize and simplify all weather station data insertions into the PostgreSQL database a cron job was scheduled to run the Java application for the McNair weather station and the Java application for Wunderground five separate times with the above station IDs.

```
<current_observation>
    <credit>Weather Underground Personal Weather Station</credit>
    <credit_URL>http://wunderground.com/weatherstation/</credit_URL>
    <image> … </image>
    <location>
       <full>Westerwood, Greensboro, NC</full>
       <neighborhood>Westerwood</neighborhood>
       <city>Greensboro</city>
       <state>NC</state>
       …
    </location>
    <station_id>KNCGREEN30</station_id>
    <observation_time>Last Updated on May 11, 5:43 PM EDT</observation_time>
    <observation_time_rfc822>Sat, 11 May 2013 21:43:11 GMT</observation_time_rfc822>
    <temperature_string>85.1 F (29.5 C)</temperature_string>
    <temp_f>85.1</temp_f>
    <temp_c>29.5</temp_c>
    <relative_humidity>56</relative_humidity>
    <wind_string>Calm</wind_string>
    <wind_dir>SW</wind_dir>
    <wind_degrees>221</wind_degrees>
    <wind_mph>0.0</wind_mph>
    <wind_gust_mph>5.0</wind_gust_mph>
    <pressure_string>29.93" (1013.4 mb)</pressure_string>
    <pressure_mb>1013.4</pressure_mb>
    <pressure_in>29.93</pressure_in>
    <dewpoint_string>67.7 F (19.8 C)</dewpoint_string>
    <dewpoint_f>67.7</dewpoint_f>
    <dewpoint_c>19.8</dewpoint_c>
    <heat_index_string>88 F (31 C)</heat_index_string>
    <heat_index_f>88</heat_index_f>
    <heat_index_c>31</heat_index_c>
    <windchill_string/><windchill_f/>
    …
    <precip_1hr_string>0.00 in (0.0 mm)</precip_1hr_string>
    <precip_1hr_in>0.00</precip_1hr_in>
    <precip_1hr_metric>0.0</precip_1hr_metric>
    <precip_today_string>0.61 in (1.5 cm)</precip_today_string>
    <precip_today_in>0.61</precip_today_in>
    <precip_today_metric>1.5 cm</precip_today_metric>
    …
</current_observation>
```

Figure 16 Sample Wunderground API query of Westerwood weather station.

The second local component of the sensor network layer involved the deployment of the Sun SPOT and Imote2 sensor nodes. The sensor nodes were deployed on the front lawn of North Carolina A&T State University's Edward Fort Interdisciplinary Research Center (IRC) building. Figure 17 shows a front view of the IRC building with deployed sensor nodes, while Figure 18 shows a side view of the same layout with deployed sensor nodes. Figure 19 depicts the overall layout with sensor node locations.

Figure 17 Sensor Layout in front of IRC building.

Figure 18 Sensor Layout from side view.

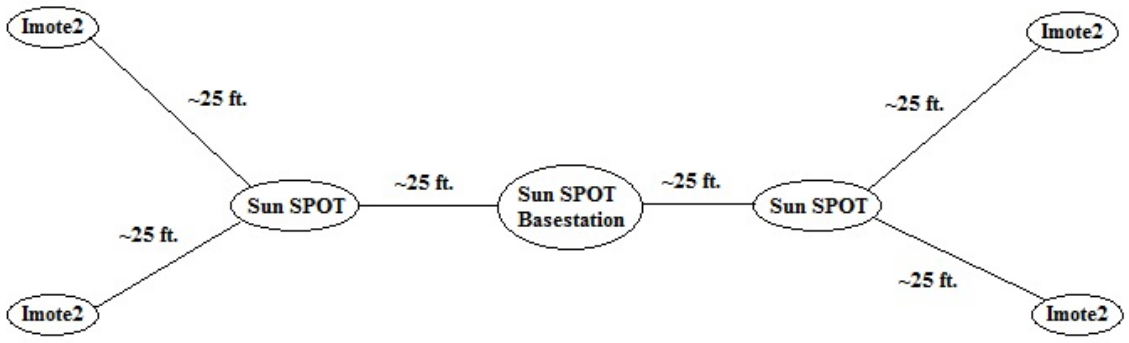

Figure 19 Sensor Layout with distances.

The sensor node layout consisted of two free-range Sun SPOT sensor nodes, one base station Sun SPOT sensor node, and four Imote2 sensor nodes. With the approximate radio ranges of the Sun SPOT and Imote2 sensor nodes being 100 meters and 30 meters, respectively, the sensor node layout was designed for a location near the McNair weather station. From Figure 19, the four Imote2 sensor nodes served as the outer leaf nodes. These sensor nodes take temperature and humidity observations at five minute intervals. The free-range Sun SPOT sensor nodes have LEAP agents deployed onto them. The LEAP agents take temperature observations at five minute intervals and send the observational data to a JADE agent on the workstation, which is the machine connected to the base station Sun SPOT sensor node. This JADE agent inserts the observational data from all incoming messages into the PostgreSQL database. The LEAP agents also continuously listen for incoming messages from the Imote2 sensor nodes and forward the data to the workstation JADE agent.

The final component of the sensor network layout is the PostgreSQL database. The database design is modified from 52 North's Sensor Observation Service package. Figure 20 shows the database schema of the PostgreSQL database.

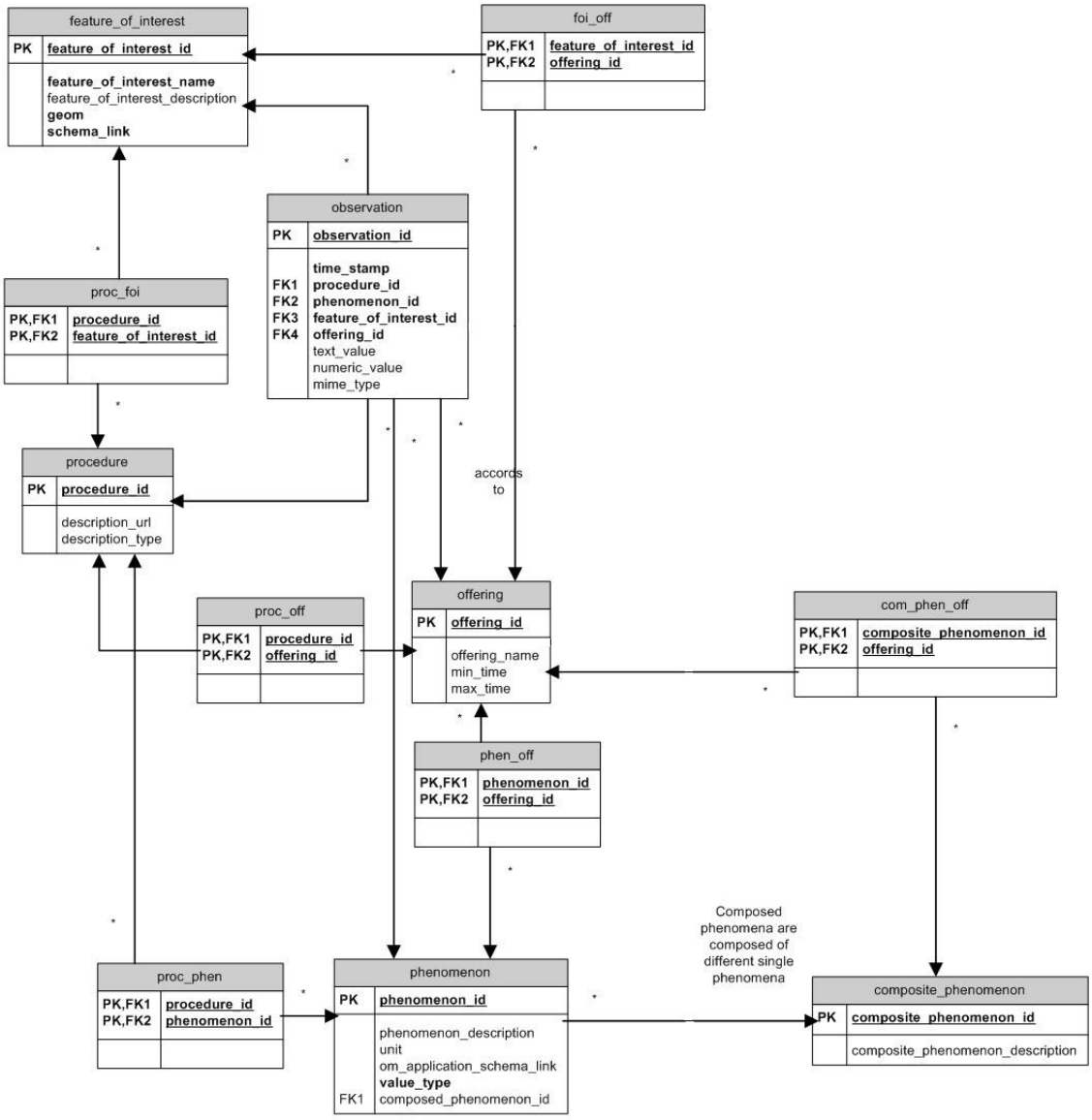

Figure 20 Database schema for Semantic Sensor Observation Service.

This design includes tables that correspond to the important OGC concepts of feature of interest, procedure, offering, phenomenon, and observation. The feature of interest table stores the data relating to the geographical locations of the sensor nodes and weather stations that take observational data. The procedure table stores data about the sensor nodes and weather stations. The offering tables stores data about the observational data that is offered through the sensor nodes and weather stations. The offerings include the temperature, relative humidity, pressure,

precipitation, wind speed, wind direction-in general, any observational data that could be observed through any sensor node or weather station in the sensor network. The phenomenon table stores data about the physical phenomena observed in the environment. The observations table stores the actual observational data. The feature-of-interest, procedure, offering, and phenomenon tables are pre-loaded with descriptive data about the environment and sensor nodes. This data is primarily used as IDs for the observational data stored in the observation table. Every observational value is stored with corresponding IDs for its feature of interest, procedure, offering, and phenomenon.

5.2.2 Services Layer. The services layer is the intermediary between the client and sensor network layers. The services layer allows simple access to sensor nodes and observational data through the structured implementation of the service of interest. The service layer consists of the Semantic SOS service and the multiagent system. The following describes the Semantic SOS service and the multiagent system developed for this prototype.

This implementation of the Semantic SOS was based from 52 North's standard implementation of the Sensor Observation Service. Figure 21 shows the design of 52 North's SOS service.

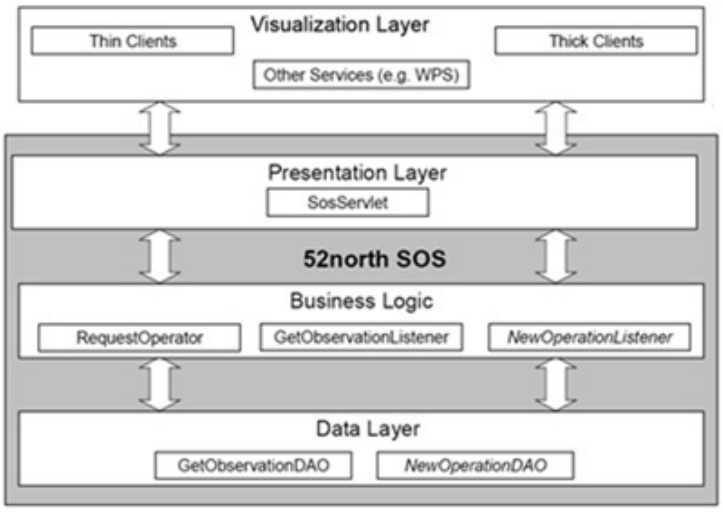

Figure 21 52 North's Semantic Sensor Observation Service.

52 North's SOS service is composed of three main components, the presentation layer, the business logic layer, and the data layer. The presentation layer is responsible for the handling the incoming requests from clients and sending back the appropriate response. The business logic layer forwards requests from the presentation layer to the appropriate operation listener. Listeners perform the requested operation. The data layer encapsulates access to the database for the requested operation.

The prototype extends the 52 North's SOS service to create the Semantic Sensor Observation Service (Semantic SOS). The Semantic SOS service adds the Semantic Web capabilities of querying over the structured data of RDF triples. The Semantic SOS service also preserves this query's results by annotating the results with metadata. The primary operations of the SOS service are the GetObservation, GetCapabilities, and DescribeSensor operations. The Semantic SOS service modifies the GetObservation operation. This operation is of most importance because it returns the observational data. Because of the extensive work needed to modify one operation, the GetObservation operation was the only operation modified to contain

Semantic Web capabilities. The semantic additions to the SOS service involved modifying the listener and data access object of the GetObservation operation.

The listener for the GetObservation operation receives incoming O&M queries. The O&M query specifies the parameter of interest to the user, such as features of interest and procedures. From the O&M query, the SOS service constructs a SQL query for the data access object to retrieve relevant data. The Semantic SOS service constructs a SPARQL query for the data access object. Since the focus is on the benefits of the Semantic Web, we use RDF triples to convey information. SQL does not have the capability to query over these constructs but SPARQL can perform queries over RDF triples.

The semantic component of the Semantic SOS service depends on concepts defined in the O&M schema. Here, the ontology plays a vital role of formalizing concepts and their relations for semantic purposes. The O&M schema defines several concepts for an observation, such as feature of interest and observed property. An ontology version of the O&M schema is fundamental to the Semantic SOS service where the standard SOS service requires the O&M schema, the semantic version of the SOS service requires the O&M concepts defined in an ontology.

The University of Muenster's Institute for Geoinformatics has developed the first O&M ontology. Wright State University has made a more recent attempt at developing an O&M ontology. Figure 22 shows the structure of the major concepts and their relationships.

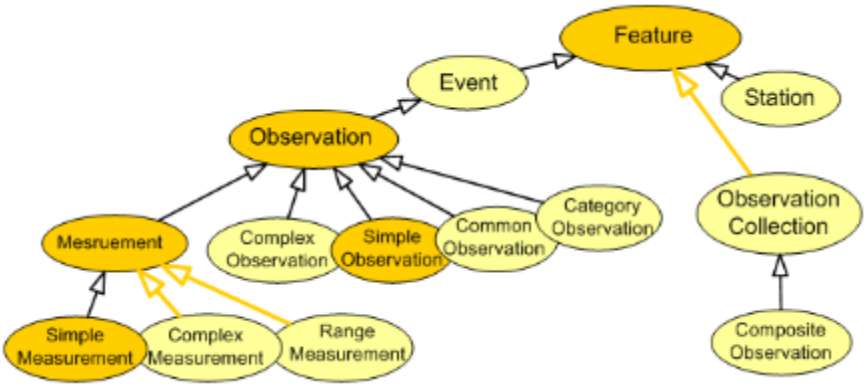

Figure 22 Structure of concepts used in O&M ontology. (Probst, 2006)

For this research, a simplified O&M ontology was developed based on the above structure. Figure 23 shows the general SPARQL query generated from an O&M query.

```
PREFIX om: <./observationsMeasurements.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?offering ?offeringID ?proc ?obs ?foi ?foiName ?foiDesc ?foiGeom
?foiType ?foiSchemaLink ?instant ?phen ?phenUnit ?valuetype ?result ?loc ?lat ?long
WHERE {
        ?offering rdf:type om:System .
        ?offering om:offID ?offeringID .
        ?offering om:systemComponentProcedure ?proc .
        ?proc om:generatedObservation ?obs .
        ?obs om:featureOfInterest ?foi .
        ?foi om:foiName ?foiName .
        ?foi om:foiDesc ?foiDesc .
        ?foi om:observationLocation ?foiGeom .
        ?foi om:foiType ?foiType .
        ?foi om:foiLink ?foiSchemaLink .
        ?obs om:samplingTime ?instant .
        ?obs om:observedProperty ?phen .
        ?phen om:phenUnit ?phenUnit .
        ?phen om:phenValue ?valuetype .
        ?obs om:floatValue ?result .
        ?foi om:observationLocation ?loc .
        ?foi om:lat ?lat .
        ?foi om:long ?long .
        FILTER ( ?offering=<TEMPERATURE>) .
        FILTER ( ?phen=<./urn:ogc:def:phenomenon:OGC:1.0.30:AirTemperature>) .
}
```

Figure 23 SPARQL query Generated from O&M Query.

All SPARQL queries generated from O&M queries are roughly the same; the only difference involves the FILTER statements at the end. The SPARQL query returns the parameters in the SELECT clause that match the corresponding triples in the WHERE clause. The FILTER statements limit the results to values that match the given parameters. Given a different O&M query, only the FILTER statements of the SPARQL query would change. All generated SPARQL queries are sent to the data access object.

The data access object receives and executes the SPARQL query. The data access object of the SOS service expects SQL queries, so the data access object for the Semantic SOS service must be modified for SPARQL queries. The Semantic SOS service incorporates a PostgreSQL

relational database which is incompatible with Semantic Web querying technology. To use a relational database with RDF querying capabilities, a mapping is needed. D2RQ provides a mapping between SPARQL queries and relational databases. D2RQ produces a default mapper that translates the database tables to RDFS (RDF Schema) classes. The default mapper is customized with concepts from the ontology. Figure 24 shows the customized D2RQ mapping of the offering table. This mapping was customized by replacing default values with ontology concepts. In the offering table, the class map of map:offering was modified to om:System, which is defined in the ontology. The property bridge of map:offering_offering_id was modified to om:offID, also defined in the ontology.

```
# Table offering
map:offering a d2rq:ClassMap;
      d2rq:dataStorage map:database;
      d2rq:uriPattern "@@offering.offering_id@@";
      d2rq:class om:System;
      d2rq:classDefinitionLabel "offering";
      .
map:offering__label a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:offering;
      d2rq:property rdfs:label;
      d2rq:pattern "offering #@@offering.offering_id@@";
      .
map:offering_offering_id a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:offering;
      d2rq:property om:offID;
      d2rq:propertyDefinitionLabel "offering offering_id";
      d2rq:column "offering.offering_id";
      .
map:offering_offering_name a d2rq:PropertyBridge;
      d2rq:belongsToClassMap map:offering;
      d2rq:property vocab:offering_offering_name;
      d2rq:propertyDefinitionLabel "offering offering_name";
      d2rq:column "offering.offering_name";
      .
```

Figure 24 D2RQ mapping of the offering table.

This customized D2RQ mapping links the concepts of the ontology to the relational database. Now the SPARQL query can be executed against the database. The query results are loaded into an SOS observation object and transformed into an O&M response. The O&M response is annotated with RDFa attributes. This is returned to the listener and then to the client.

The agent component of the services layer includes the multiagent system. This is included in the services layer because a few agents are exposed as agent Web services so the client layer can access the agents for additional Web service-like functionality. Figure 25 shows the overview architecture of the multiagent system.

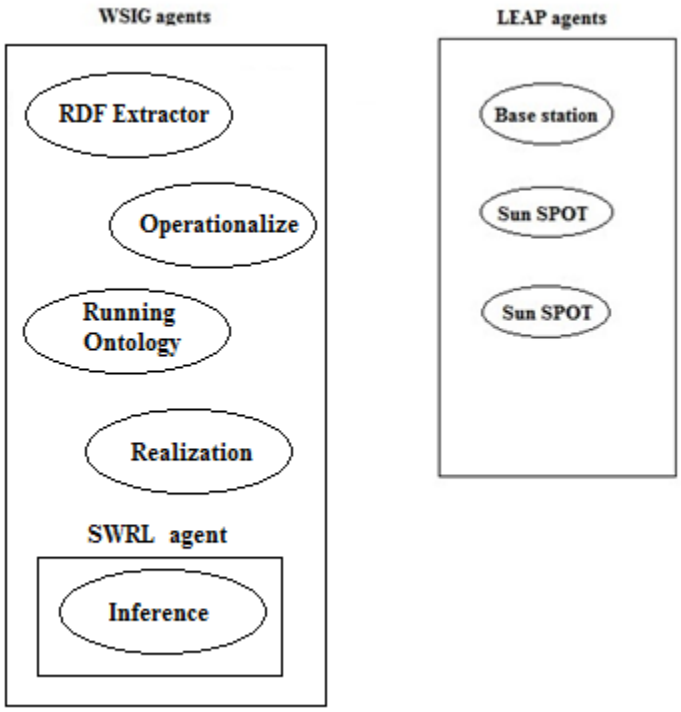

Figure 25 Multiagent Architecture of the Prototype.

The agents are divided into three categories based on their functionality: WSIG agents, SWRL agents, and LEAP agents. WSIG agents are JADE agents that expose their operations as Web services, SWRL agent uses the Jess rule engine to execute SWRL rules, and LEAP agents communicate over mobile, resource constrained devices.

The WSIG agents of the prototype include a RDF-Extractor, Operationalize, Running Ontology, Inference, and Realize agents. The RDF-Extractor agent takes an RDFa-annotated O&M response and extracts the RDF triples, which are returned to the client. Figure 26 shows a portion of the O&M response annotated with RDFa attributes while Figure 27 shows the extracted triples from the RDF-Extractor agent. Several concepts from the O&M ontology are integrated into this O&M response. In the XML segment, the `typeof` attribute of the `SamplingPoint` element has a value of `w:Feature` and the `property` attribute of the `gml:name` element has a value of `w:foiName`. Through java-rdfa api, triples are extracted based on certain properties associated with elements (RDFa). Figure 27 shows the resulting triples from the RDFa embedded in Figure 26's XML segment.

```
<om:featureOfInterest property="w:featureOfInterest" content="…/sosMap.ttl#foi_103">
        <sa:SamplingPoint gml:id="…/sosMap.ttl#foi_103" about="…/sosMap.ttl#foi_103" typeof="w:Feature">
                <gml:description property="w:foiDesc">Pinecroft</gml:description>
                <gml:name property="w:foiName">KNCGREEN39</gml:name>
                <sa:sampledFeature/>
                <sa:position>
                <gml:Point gml:id="point_sf_0">
                        <gml:pos srsName="…:EPSG::4326" property="w:location">-79.844 36.02</gml:pos>
                </gml:Point>
                </sa:position>
        </sa:SamplingPoint>
</om:featureOfInterest>
```

Figure 26 RDFa-annotated O&M response.

```
<…/sosMap.ttl#foi_103>      a       w:Feature ;
                w:foiDesc "Pinecroft" ;
                w:foiName "KNCGREEN39" ;
                w:location "-79.844 36.02" .
```

Figure 27 Extracted RDF Triples.

The Running Ontology agent accepts a set of RDF triples and returns a new set of RDF triples, given the new ontology it incorporates. The Running Ontology agent aligns the Running ontology with the ontology of the given RDF triples. Figure 28 shows a simplified version of

Running Ontology in Protégé with its major concepts. This local ontology shares identical concepts with the global weather ontology. The alignment of identical concepts allows translation between the two ontologies. The yellow circles represent classes and the blue rectangles represent properties of certain classes. This ontology's concepts of Measurement and Phenomena are identical to the weather ontology's Result and Observation concepts. RDF triples can now be expressed in terms of the new ontology's concepts. The Running ontology was based from a similar ontology from the University of Maryland's Human-Computer Interaction Lab. The Running ontology used in this research incorporated the same concepts with integration of concepts from the SWEET ontologies.

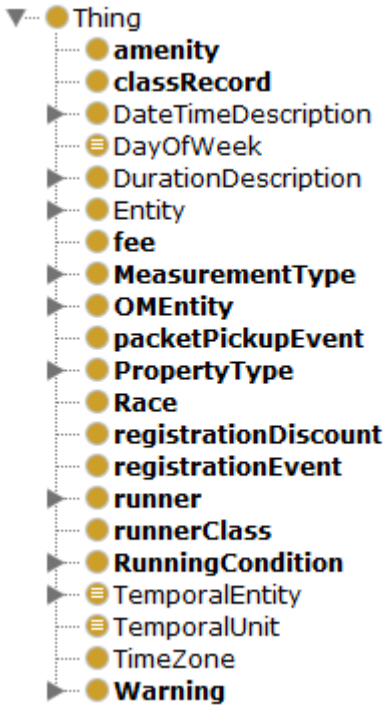

Figure 28 Simplified Running ontology in Protege.

Ontology alignment allows a concept in one ontology to become equated to a concept in another ontology. This leads to inferences within the new ontology between related concepts. Figure 29 shows the alignment between concepts of the O&M ontology and the Running

ontology. The ontology alignment is accomplished by adding additional triples to the working model that expressed the equivalent concepts. Here, we equate the `Feature` class from the O&M ontology to the `FeatureEntity` class from the Running ontology. We also equate the `hasAvgTemp` and `hasAvgPrec` properties of the O&M ontology with the `raceTemp` and `racePrec` properties, respectively, from the Running ontology.

```
String rNS = "./running.owl#";
schema.setNsPrefix( "r", rNS );
Resource resource =
schema.createResource("./observationsMeasurements.owl#Feature");
Property prop =
schema.createProperty("http://www.w3.org/2002/07/owl#equivalentClass");
Resource obj = schema.createResource("r:FeatureEntity");
schema.add(resource,prop,obj);
resource = schema.createResource("w:hasAvgTemp");
prop =
schema.createProperty("http://www.w3.org/2002/07/owl#equivalentProperty");
obj = schema.createResource("r:raceTemp");
schema.add(resource,prop,obj);
resource = schema.createResource("w:hasAvgPrec");
prop =
schema.createProperty("http://www.w3.org/2002/07/owl#equivalentProperty");
obj = schema.createResource("r:racePrec");
schema.add(resource,prop,obj);
```

Figure 29 Ontology Alignment using Jena.

The SWRL agents of the prototype include an Inference agent. Given the RDF triples aligned with a local ontology, the Inference agent can infer new RDF triples. For example, given a certain set of RDF triples, one or more triples could be inferred to generate new RDF triples. Figure 30 shows a sample SWRL rule. The rule infers that a variable is a `w:Freezing` type if it is a `w:Temperature` type with a value of its result less than 32.

```
[FreezingRule:
(?temp rdf:type w:Temperature)
(?temp om:result ?result)
(?result om:value ?value)
lessThan(?value 32)
→ (?temp rdf:type w:Freezing)]
```

Figure 30 Sample SWRL Rule.

The LEAP agents include two Sun SPOT agents and a Base station agent. As previously mentioned the Sun SPOT agents receive observational data from the Imote2 sensor nodes and forwards this data to the Base station agent. The Sun SPOT agents also forward their own observational data to the Base station agent. The Base station agent receives observational data from the Sun SPOT agent and inserts the data into the PostgreSQL database.

5.2.3 Client Layer. The client layer is a layer of applications that take advantage of the other layers. This layer provides client applications that primarily access the exposed services of the prototype, which services encapsulate the other functionality of the system. Through the modularity of the system, the prototype's client user interface, or any user interface, has easy access to the Semantic SOS service provided and essentially to the sensor network's observational data.

Here, the current client is a simple web page developed with HTML and JavaServer Pages (JSP). JSP is a web technology that allows for creating dynamically generated HTML web pages (JSP). The prototype's test client allows the user to select their options of interest and then the client generates the appropriate O&M request for access to the service. Screenshots of the test client's user interface can be seen in the next chapter.

The client could easily be extended and other clients could be developed to access the service. A few interesting client user interface options would include desktop applications, mobile device applications, or other Web services that take advantage of the Semantic SOS. The standardized interface of the Web service would allow easy integration of the service's functionality into other applications.

5.3 Implementation

The implementation section generally describes how the previous architectural components fit together to perform a comprehensive operation. The overall system includes many components and services that provide separate functionalities that depend on each other. Although, the overall system is comprised of many separate components, there are specific components that stand out in the overall operation of the system. These components include the O&M ontology, the Running ontology, the ontology alignment, and the inference rules. The O&M ontology defines the low-level concepts of the environment. The Running ontology defines the high-level concepts of an activity in terms of its own set of low-level concepts. The ontology alignment equates concepts between the O&M ontology and the Running ontology. The inference rules determine new information based on current statements that match defined patterns of statements. We'll describe these components in more detail, along with how they relate to each other.

The O&M schema defines the principle concepts used within the environmental domain. Here, the primary focus is the measurement of environmental phenomena that includes temperature, atmospheric pressure, relative humidity, precipitation, wind speed, and wind direction. Additionally, the O&M schema defines generalized, higher level concepts that abstract away low-level details and can be reused within most environmental domains. These concepts include the observation, feature of interest, and observed property. The observation is a measurement, or series of measurements, the feature of interest is a particular location of interest that provides observations, and the observed property is a particular phenomenon that can be measured as an observation. The O&M ontology is a simplified, direct translation of the O&M schema so the major concepts and relations are preserved from the O&M schema. Figure 31

displays the O&M ontology in Protégé. The classes are shown in the orange section while the properties are shown in the blue. The important classes and properties used here are circled.

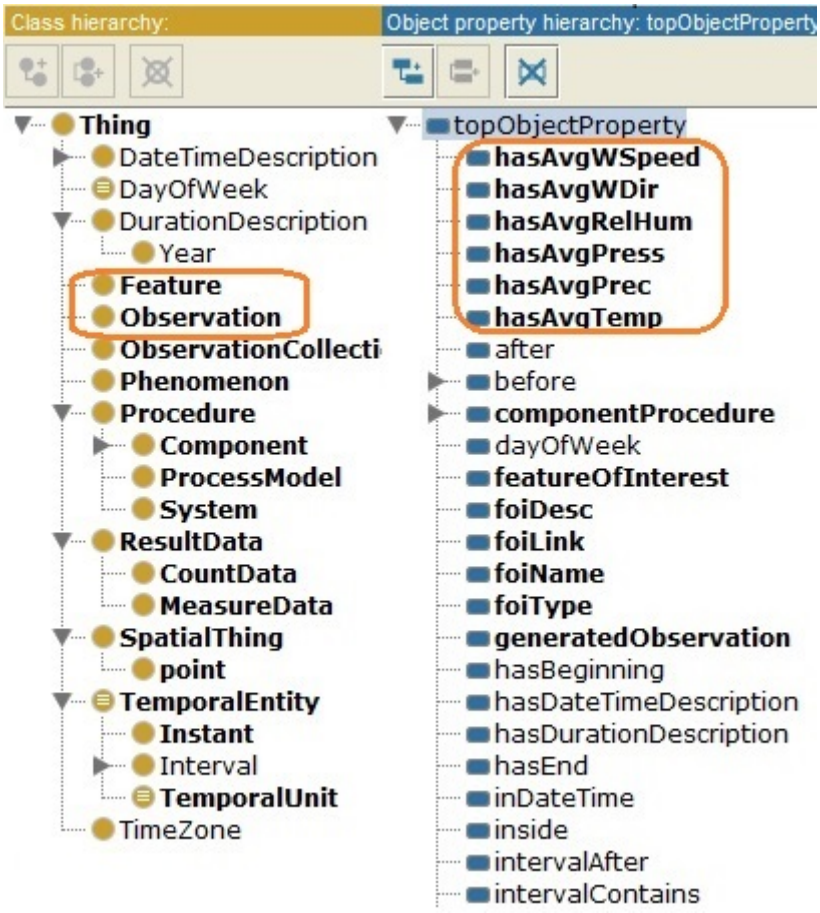

Figure 31 Class and Property Concepts of the O&M Ontology.

The O&M ontology is the cornerstone of the semantic capabilities of the Semantic SOS service. The inclusion of the ontology allows the data to be transformed into a format usable by semantic processes. The O&M ontology in the Semantic SOS service allows the RDFa attributes to be embedded into the XML-based O&M response. With the embedded RDFa attributes, extracted RDF triples will preserve the integrity of the observations while using a more readable format that can be used for semantic purposes.

The Running ontology defines specialized, high-level concepts for a specific activity while also including low-level details. This ontology focuses on concepts related to running,

which include features of interest, running conditions, and warnings. The feature of interest is a particular location of interest; a running condition describes an environmental state that is conducive to running, and a warning pertains to a dangerous environmental state. Figure 32 displays the Running ontology in Protégé while highlighting important concepts.

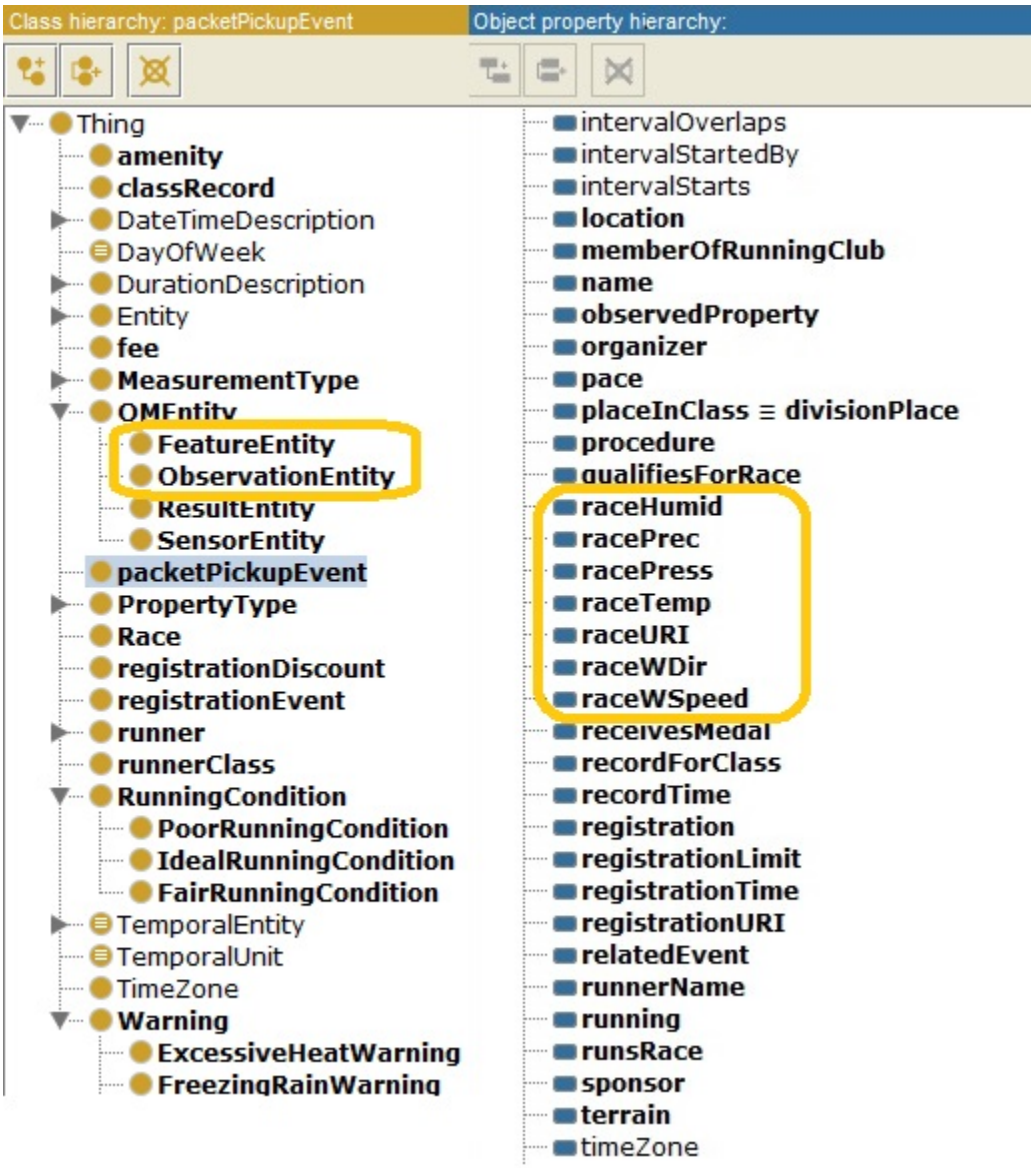

Figure 32 Class and Property Concepts of the Running Ontology.

The ontology alignment equates concepts between the O&M ontology and the Running ontology. The O&M ontology focuses on generalized environmental concepts for a variety of

environmental domains while the Running ontology focuses on specialized concepts for a particular activity domain. Although these ontologies focus on separate domains the specialized Running ontology has underlying concepts that equate to the generalized O&M ontology. The ontology alignment here focuses on a common set of concepts that are shown in Table 2. Figure 33 shows how ontology alignment is accomplished in the Jena framework. The `Feature` class of the O&M ontology and `FeatureEntity` class of the Running ontology both refer to a location of interest. The `hasAvgTemp` property of the O&M ontology and `raceTemp` property of the Running ontology both refer to the temperature measurement of a feature. The `hasAvgPrec` property of the O&M ontology and `racePrec` property of the Running ontology both refer to the precipitation measurement of a feature. The `hasAvgPress` property of the O&M ontology and `racePress` property of the Running ontology both refer to the atmospheric pressure measurement of a feature. The `hasAvgRelHum` property of the O&M ontology and `raceHumid` property of the Running ontology both refer to the relative humidity measurement of a feature. The `hasAvgWSpeed` property of the O&M ontology and `raceWSpeed` property of the Running ontology both refer to the wind speed measurement of a feature. The `hasAvgWDir` property of the O&M ontology and `raceWDir` property of the Running ontology both refer to the wind direction measurement of a feature.

Table 2 Comparison of equivalent O&M and Running ontology concepts.

O&M ontology	**Running ontology**
`w:Feature`	`r:FeatureEntity`
`w:hasAvgTemp`	`r:raceTemp`
`w:hasAvgPrec`	`r:racePrec`
`w:hasAvgWSpeed`	`r:raceWSpeed`
`w:hasAvgWDir`	`r:raceWDir`
`w:hasAvgPress`	`r:racePress`
`w:hasAvgRelHum`	`r:raceHumid`

```
String rNS = ".../running.owl#";
schema.setNsPrefix( "r", rNS );
Resource resource = schema.createResource(".../observationsMeasurements.owl#Feature");
Property prop = schema.createProperty("http://www.w3.org/2002/07/owl#equivalentClass");
Resource obj = schema.createResource("r:FeatureEntity");
schema.add(resource,prop,obj);

resource = schema.createResource("w:hasAvgTemp");
prop = schema.createProperty("http://www.w3.org/2002/07/owl#equivalentProperty");
obj = schema.createResource("r:raceTemp");
schema.add(resource,prop,obj);

resource = schema.createResource("w:hasAvgPrec");
prop = schema.createProperty("http://www.w3.org/2002/07/owl#equivalentProperty");
obj = schema.createResource("r:racePrec");
schema.add(resource,prop,obj);

resource = schema.createResource("w:hasAvgWSpeed");
prop = schema.createProperty("http://www.w3.org/2002/07/owl#equivalentProperty");
obj = schema.createResource("r:raceWSpeed");
schema.add(resource,prop,obj);

resource = schema.createResource("w:hasAvgWDir");
prop = schema.createProperty("http://www.w3.org/2002/07/owl#equivalentProperty");
obj = schema.createResource("r:raceWDir");
schema.add(resource,prop,obj);

resource = schema.createResource("w:hasAvgPress");
prop = schema.createProperty("http://www.w3.org/2002/07/owl#equivalentProperty");
obj = schema.createResource("r:racePress");
schema.add(resource,prop,obj);

resource = schema.createResource("w:hasAvgRelHum");
prop = schema.createProperty("http://www.w3.org/2002/07/owl#equivalentProperty");
obj = schema.createResource("r:raceHumid");
schema.add(resource,prop,obj);

Reasoner reason = ReasonerRegistry.getOWLReasoner();
reason = reason.bindSchema(schema);
InfModel iModel = ModelFactory.createInfModel(reason,model);
```

Figure 33 Ontology Alignment between O&M and Running ontology concepts.

The inference rules expand the knowledgebase by adding new inferred information. With the general if-then format, when a set of triples are matched, new statements can be added to the knowledgebase. The new statements generally connect concepts to higher-level concepts. Figure 34 shows the rules used with the Running ontology.

```
[rule1:
        (?feat <rdf:type><r:FeatureEntity>), (?feat <r:raceTemp> ?temp),greaterThan(?temp,105.0)
        ->
        (?feat<rdf:type><.../running.owl#ExcessiveHeatWarning>)]

[rule2:
        (?feat <rdf:type><r:FeatureEntity>),(?feat <r:raceTemp> ?temp),lessThan(?temp,32.0),
        (?feat<r:raceWSpeed> ?wspeed),greaterThan(?wspeed,35.0)
        ->
        (?feat<rdf:type><.../running.owl#WindChillWarning>)]

[rule3:
        (?feat <rdf:type><r:FeatureEntity>),(?feat <r:raceTemp> ?temp),lessThan(?temp,32.0),
        (?feat<r:racePrec> ?prec),greaterThan(?prec,2.0)
        ->
        (?feat<rdf:type><.../running.owl#FreezingRainWarning>)]

[rule4:
        (?feat <rdf:type><r:FeatureEntity>),(?feat <r:raceTemp> ?temp),lessThan(?temp,32.0),
        (?feat<r:racePrec> ?prec),greaterThan(?prec,2.0),
        (?feat<r:raceWSpeed> ?wspeed),greaterThan(?wspeed,35.0)
        ->
        (?feat<rdf:type><.../running.owl#BlizzardWarning>)]

[rule5:
        (?feat<rdf:type><r:FeatureEntity>),(?feat <r:raceTemp> ?temp),
        lessThan(?temp,65.0),greaterThan(?temp,45.0),
        (?feat<r:racePrec> ?prec),equal(?prec,0.0),
        (?feat<r:raceWSpeed> ?wspeed),lessThan(?wspeed,5.0)
        ->
        (?feat<rdf:type><.../running.owl#IdealRunningCondition>)]

[rule6:
        (?feat<rdf:type><r:FeatureEntity>),(?feat <r:raceTemp> ?temp),
        lessThan(?temp,100.0),greaterThan(?temp,35.0),
        (?feat<r:racePrec> ?prec),lessThan(?prec,1.0),
        (?feat<r:raceWSpeed> ?wspeed),lessThan(?wspeed,15.0)
        ->
        (?feat<rdf:type><.../running.owl#FairRunningCondition>)]

[rule7:
        (?feat <rdf:type><r:FeatureEntity>),(?feat <r:raceTemp> ?temp),lessThan(?temp,32.0)
        ->
        (?feat<rdf:type><.../running.owl#PoorRunningCondition>)]

[rule8:
        (?feat <rdf:type><r:FeatureEntity>),(?feat <r:raceTemp> ?temp),greaterThan(?temp,100.0)
        ->
        (?feat<rdf:type><.../running.owl#PoorRunningCondition>)]

[rule9:
        (?feat <rdf:type><r:FeatureEntity>),(?feat <r:racePrec> ?prec),greaterThan(?prec,1.0)
        ->
        (?feat<rdf:type><.../running.owl#PoorRunningCondition>)]

[rule10:
        (?feat <rdf:type><r:FeatureEntity>),(?feat <r:raceWSpeed> ?wspeed),greaterThan(?wspeed,15.0)
        ->
        (?feat<rdf:type><.../running.owl#PoorRunningCondition>)]
```

Figure 34 SWRL Rules for Inference.

The low-level details are used to determine higher-level concepts about the environment. The rules combine details to distinguish cases of interest to a particular type of user. Here, our users show interest in the domain of running, which is why we align our general environmental ontology to our specialized activity ontology about running. Inference over the specialized running ontology allows running-oriented users to determine specific information based on their concerns, not just additional environmental information.

CHAPTER 6

Results

This chapter views several sample data instances within two runs of the prototype system. The architecture, described above, was tested and shown to explore the presented specific aims. The test client was used to display the output results. Table 3 shows the sample data used in Run 1.

Table 3 Sample Data for Run 1.

Feature of Interest	Observed Property	Value
foi_101	Temperature	29.6
foi_101	Relative Humidity	50.0
foi_101	Atmospheric Pressure	30.12
foi_101	Precipitation	0.0
foi_101	Wind Speed	42.0
foi_101	Wind Direction	45.0
foi_102	Temperature	25.8
foi_102	Relative Humidity	55.0
foi_102	Atmospheric Pressure	29.45
foi_102	Precipitation	3.25
foi_102	Wind Speed	0.0
foi_102	Wind Direction	45.0
foi_103	Temperature	18.5
foi_103	Relative Humidity	87.0
foi_103	Atmospheric Pressure	30.2
foi_103	Precipitation	0.0
foi_103	Wind Speed	45.0
foi_103	Wind Direction	45.0
foi_104	Temperature	42.6
foi_104	Relative Humidity	87.0
foi_104	Atmospheric Pressure	30.17
foi_104	Precipitation	0.0
foi_104	Wind Speed	25.0
foi_104	Wind Direction	45.0
foi_105	Temperature	37.8
foi_105	Relative Humidity	82.0
foi_105	Atmospheric Pressure	30.17
foi_105	Precipitation	0.0
foi_105	Wind Speed	0.0
foi_105	Wind Direction	45.0
foi_106	Temperature	44.3
foi_106	Relative Humidity	88.0
foi_106	Atmospheric Pressure	30.17
foi_106	Precipitation	0.0
foi_106	Wind Speed	10.0
foi_106	Wind Direction	45.0

Figure 35 shows the initial O&M query submitted to the Semantic SOS service. This query refers to an offering of observation data labeled “ALL” that included all the observed properties of temperature, relative humidity, pressure, precipitation, wind speed, and wind direction. This query is submitted to the Semantic SOS service and handled by the “GetObservation” operation of the service.

Semantic SOS O&M XML Query

Service URL: http://localhost:8080/52nSOSv3.5.0/sos

```
<?xml version="1.0" encoding="UTF-8"?>
<GetObservation xmlns="http://www.opengis.net/sos/1.0"
        xmlns:ows="http://www.opengis.net/ows/1.1"
        xmlns:gml="http://www.opengis.net/gml"
        xmlns:ogc="http://www.opengis.net/ogc"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.opengis.net/sos/1.0
         http://schemas.opengis.net/sos/1.0.0/sosGetObservation.xsd"
        service="SOS" version="1.0.0" srsName="urn:ogc:def:crs:EPSG::4326">

        <offering>ALL</offering>
        <observedProperty>urn:ogc:def:phenomenon:OGC:1.0.30:AirTemperature</observedProperty>
        <observedProperty>urn:ogc:def:phenomenon:OGC:1.0.30:RelativeHumidity</observedPropert
        <observedProperty>urn:ogc:def:phenomenon:OGC:1.0.30:AtmosphericPressure</observedProp
        <observedProperty>urn:ogc:def:phenomenon:OGC:1.0.30:Precipitation</observedProperty>
        <observedProperty>urn:ogc:def:phenomenon:OGC:1.0.30:WindSpeed</observedProperty>
        <observedProperty>urn:ogc:def:phenomenon:OGC:1.0.30:WindDirection</observedProperty>
        <responseFormat>text/xml;subtype="om/1.0.0"</responseFormat>
</GetObservation>
```

Figure 35 O&M Query to Semantic SOS service.

Figure 36 shows the resulting O&M response from the Semantic SOS service. The XML response conforms to the O&M Schema and also annotated with RDFa attributes that can be extracted later. Figure 37 shows a portion of the actual output from Figure 36.

Semantic SOS O&M Response

```
<om:ObservationCollection xmlns:om="http://www.opengis.net/om/1.0" xmlns:gml="http://www.op
  <gml:boundedBy>
    <gml:Envelope srsName="urn:ogc:def:crs:EPSG::4326">
      <gml:lowerCorner>-79.896 36.02</gml:lowerCorner>
      <gml:upperCorner>-79.776 36.151</gml:upperCorner>
    </gml:Envelope>
  </gml:boundedBy>
  <om:member>
    <om:Observation gml:id="go_1381712321610" about="obs_1381712322565" typeof="w:Observat
      <om:samplingTime>
        <gml:TimeInstant xsi:type="gml:TimeInstantType">
          <gml:timePosition>2013-10-13T20:57:21.666-04:00</gml:timePosition>
        </gml:TimeInstant>
      </om:samplingTime>
      <om:procedure rel="w:procedure" resource="file:///C:/apache-tomcat-7.0.42/bin/sosMap
      <om:observedProperty>
        <swe:CompositePhenomenon gml:id="CompositePhenomenon_file:///C:/apache-tomcat-7.0.
          <gml:name>CompositePhenomenon_file:///C:/apache-tomcat-7.0.42/bin/sosMap.ttl#urn
          <swe:component xlink:href="http://www.opengis.net/def/property/OGC/0/SamplingTime
          <swe:component xlink:href="file:///C:/apache-tomcat-7.0.42/bin/sosMap.ttl#urn:ogc
          <swe:component xlink:href="file:///C:/apache-tomcat-7.0.42/bin/sosMap.ttl#urn:ogc
```

Figure 36 Screenshot of O&M Response from Semantic SOS service (Run 1).

```
<om:ObservationCollection … xmlns:w="./observationsMeasurements.owl#">
<gml:boundedBy>
<gml:Envelope srsName="urn:ogc:def:crs:EPSG::4326">
<gml:lowerCorner>-79.896 36.02</gml:lowerCorner>
<gml:upperCorner>-79.776 36.151</gml:upperCorner>
</gml:Envelope>
</gml:boundedBy>
<om:member>
<om:Observation gml:id="go_1381712321610" about="obs_1381712322565" typeof="w:Observation">
<om:samplingTime>
<gml:TimeInstant xsi:type="gml:TimeInstantType">
<gml:timePosition>2013-10-13T20:57:21.666-04:00</gml:timePosition>
</gml:TimeInstant>
</om:samplingTime>
<om:procedure rel="w:procedure" resource="…:feature:Station:IFGI:at-davis-weatherstation-3"/>
<om:observedProperty>
<swe:CompositePhenomenon gml:id="…:feature:Station:IFGI:at-davis-weatherstation-3" dimension="7">
<gml:name>…:feature:Station:IFGI:at-davis-weatherstation-3</gml:name>
<swe:component xlink:href="http://www.opengis.net/def/property/OGC/0/SamplingTime"/>
<swe:component xlink:href="…/sosMap.ttl#urn:ogc:def:phenomenon:OGC:1.0.30:Precipitation"/>
<swe:component xlink:href="…/sosMap.ttl#urn:ogc:def:phenomenon:OGC:1.0.30:RelativeHumidity"/>
<swe:component xlink:href="…/sosMap.ttl#urn:ogc:def:phenomenon:OGC:1.0.30:WindDirection"/>
<swe:component xlink:href="…/sosMap.ttl#urn:ogc:def:phenomenon:OGC:1.0.30:AtmosphericPressure"/>
<swe:component xlink:href="…/sosMap.ttl#urn:ogc:def:phenomenon:OGC:1.0.30:AirTemperature"/>
<swe:component xlink:href="…/sosMap.ttl#urn:ogc:def:phenomenon:OGC:1.0.30:WindSpeed"/>
</swe:CompositePhenomenon>
</om:observedProperty>
<om:featureOfInterest property="w:featureOfInterest" content="…/sosMap.ttl#foi_103">
<sa:SamplingPoint gml:id="…/sosMap.ttl#foi_103" about="…/sosMap.ttl#foi_103" typeof="w:Feature">
<gml:description property="w:foiDesc">Pinecroft</gml:description>
<gml:name property="w:foiName">KNCGREEN39</gml:name>
<sa:sampledFeature/>
<sa:position>
<gml:Point gml:id="point_sf_0">
<gml:pos srsName="urn:ogc:def:crs:EPSG::4326" property="w:location">-79.844 36.02</gml:pos>
</gml:Point>
</sa:position>
</sa:SamplingPoint>
</om:featureOfInterest>
<om:result>
<swe:DataArray>
<swe:encoding>
<swe:TextBlock decimalSeparator="." tokenSeparator="," blockSeparator=";"/>
</swe:encoding>
<swe:values property="w:result">2013-10-13T20:57:21.666-04:00,0.0,87.0,45.0,30.2,18.5,45.0;</swe:values>
</swe:DataArray>
</om:result>
</om:Observation>
</om:ObservationCollection>
```

Figure 37 Partial Output of O&M Response (Run 1).

Figure 38 shows the extracted RDF triples from the previous O&M XML-based response. Using the RDFa attributes embedded into the O&M response, RDF triples could be formed that preserved the useful data of the response but in a more user-friendly vocabulary. Figure 39 shows some of the triples from the output shown in Figure 38.

Figure 38 Screenshot of Extracted RDF Triples (Run 1).

```
<...#foi_104>      a       w:Feature ;
            w:foiDesc "Greensboro" ;
            w:foiName "KNCGREEN49" ;
            w:location "-79.789 36.151" .
<.../obs_1381712322634>      a       w:Observation ;
            w:featureOfInterest "...#foi_104" ;
            w:procedure <...#urn:ogc:object:feature:Station:IFGI:at-davis-weatherstation-4> ;
                w:result "2013-10-13T20:57:21.666-04:00,42.6,87.0,45.0,30.17,25.0,0.0;" .
<.../obs_1381712322565>      a       w:Observation ;
            w:featureOfInterest "...#foi_103" ;
            w:procedure <...#urn:ogc:object:feature:Station:IFGI:at-davis-weatherstation-3> ;
                w:result "2013-10-13T20:57:21.666-04:00,0.0,87.0,45.0,30.2,18.5,45.0;" .
<...#foi_103>      a       w:Feature ;
            w:foiDesc "Pinecroft" ;
            w:foiName "KNCGREEN39" ;
            w:location "-79.844 36.02" .
```

Figure 39 Partial Output of Extracted RDF Triples (Run 1).

Figure 40 shows the operationalized form of the above RDF triples. Triples were added and modified, or operationalized, to allow for easier access and use of the values. Figure 41 shows some of the operationalized RDF triples from Figure 40.

Semantic SOS O&M (Operationalized) RDF Triples

```
@prefix w:       <file:///C:/jade/add-ons/wsig/app/classes/observationsMeasurements.owl#>
@prefix xn:      <file:///C:/jade/add-ons/wsig/app/classes/xmlns> .
@prefix sa:      <http://www.opengis.net/sampling/1.0> .
@prefix om:      <http://www.opengis.net/om/1.0> .
@prefix xlink:   <http://www.w3.org/1999/xlink> .
@prefix gml:     <http://www.opengis.net/gml> .
@prefix swe:     <http://www.opengis.net/swe/1.0.1> .
@prefix xsi:     <http://www.w3.org/2001/XMLSchema-instance> .
<file:///C:/apache-tomcat-7.0.42/bin/sosMap.ttl#foi_104>      a       w:Feature ;
              w:foiDesc "Greensboro" ;
              w:foiName "KNCGREEN49" ;
              w:location "-79.789 36.151" ;
              <w:hasAvgPrec> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
              <w:hasAvgPress> "30.17"^^<http://www.w3.org/2001/XMLSchema#double> ;
              <w:hasAvgRelHum> "87.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
              <w:hasAvgTemp> "42.6"^^<http://www.w3.org/2001/XMLSchema#double> ;
              <w:hasAvgWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
              <w:hasAvgWSpeed> "25.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
              <w:lat> "36.151" ;
              <w:long> "-79.789" .
<file:///C:/jade/add-ons/wsig/app/classes/obs_1381712322627>      a       w:Observation ;
```

Figure 40 Screenshot of Operationalized RDF Triples (Run 1).

```
<...#foi_104>      a       w:Feature ;
            w:foiDesc "Greensboro" ;
            w:foiName "KNCGREEN49" ;
            w:location "-79.789 36.151" ;
            <w:hasAvgPrec> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgPress> "30.17"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgRelHum> "87.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgTemp> "42.6"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWSpeed> "25.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:lat> "36.151" ;
            <w:long> "-79.789" .
<.../obs_1381712322634>      a       w:Observation ;
            w:featureOfInterest "...#foi_104" ;
            w:procedure <...#urn:ogc:object:feature:Station:IFGI:at-davis-weatherstation-4> ;
                w:result "2013-10-13T20:57:21.666-04:00,42.6,87.0,45.0,30.17,25.0,0.0;" .
<.../obs_1381712322565>      a       w:Observation ;
            w:featureOfInterest "...#foi_103" ;
            w:procedure <...#urn:ogc:object:feature:Station:IFGI:at-davis-weatherstation-3> ;
                w:result "2013-10-13T20:57:21.666-04:00,0.0,87.0,45.0,30.2,18.5,45.0;" .
<...#foi_103>      a       w:Feature ;
            w:foiDesc "Pinecroft" ;
            w:foiName "KNCGREEN39" ;
            w:location "-79.844 36.02" ;
            <w:hasAvgPrec> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgPress> "30.2"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgRelHum> "87.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgTemp> "18.5"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWSpeed> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:lat> "36.02" ;
            <w:long> "-79.844" .
```

Figure 41 Partial Output of Operationalized RDF Triples (Run 1).

Figure 42 shows the aligned RDF triples. These triples were aligned with the Running ontology. Figure 43 shows some of the aligned RDF triples from Figure 42.

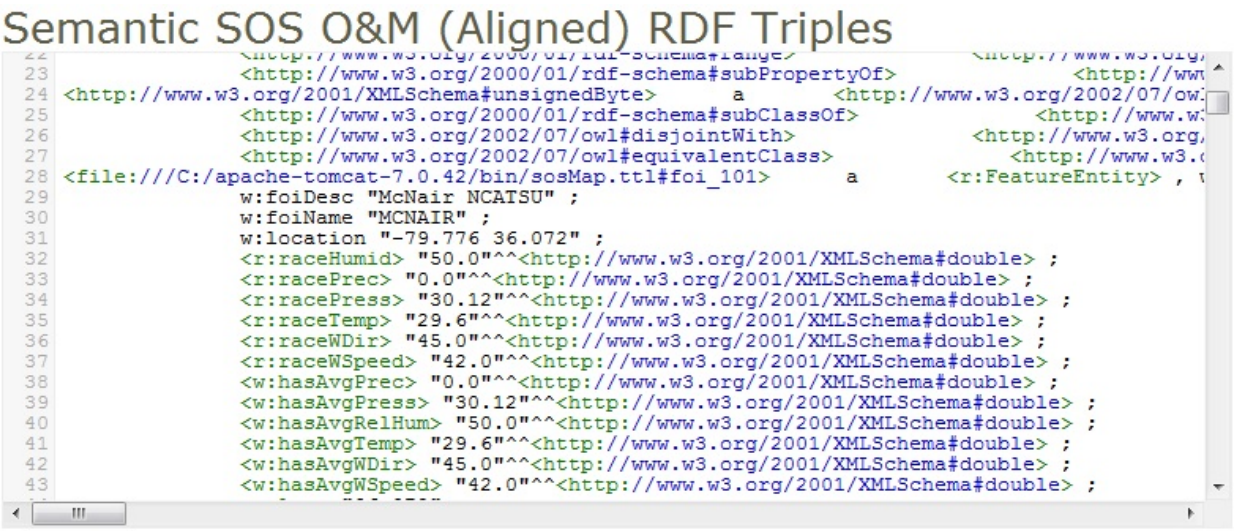

Figure 42 Screenshot of Aligned RDF Triples (Run 1).

```
<file:///C:/apache-tomcat-7.0.42/bin/sosMap.ttl#foi_103>      a       <r:FeatureEntity> , w:Feature ;
            w:foiDesc "Pinecroft" ;
            w:foiName "KNCGREEN39" ;
            <r:raceHumid> "87.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:racePrec> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:racePress> "30.2"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceTemp> "18.5"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceWSpeed> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgPrec> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgPress> "30.2"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgRelHum> "87.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgTemp> "18.5"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWSpeed> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
<file:///C:/apache-tomcat-7.0.42/bin/sosMap.ttl#foi_104>      a       <r:FeatureEntity> , w:Feature ;
            w:foiDesc "Greensboro" ;
            w:foiName "KNCGREEN49" ;
            <r:raceHumid> "87.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:racePrec> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:racePress> "30.17"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceTemp> "42.6"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceWSpeed> "25.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgPrec> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgPress> "30.17"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgRelHum> "87.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgTemp> "42.6"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWSpeed> "25.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
```

Figure 43 Partial Output of Aligned RDF Triples (Run 1).

Figure 44 shows all of the RDF triples including the inferred triples. Figure 45 shows some of the inferred RDF triples from Figure 44. Figure 46 shows the realized information from the inferred RDF triples. The realized information is the inferred information in simplified form.

Figure 44 Screenshot of Inferred RDF Triples (Run 1).

```
<…#foi_103>     a       <r:FeatureEntity> , w:Feature , r:PoorRunningCondition , r:WindChillWarning ;
            w:foiDesc "Pinecroft" ;
            w:foiName "KNCGREEN39" ;
            w:location "-79.844 36.02" ;
            <r:raceHumid> "87.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:racePrec> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:racePress> "30.2"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceTemp> "18.5"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceWSpeed> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgPrec> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgPress> "30.2"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgRelHum> "87.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgTemp> "18.5"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWSpeed> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:lat> "36.02" ;
            <w:long> "-79.844" .
<…#foi_104>     a       <r:FeatureEntity> , w:Feature , r:PoorRunningCondition ;
            w:foiDesc "Greensboro" ;
            w:foiName "KNCGREEN49" ;
            w:location "-79.789 36.151" ;
            <r:raceHumid> "87.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:racePrec> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:racePress> "30.17"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceTemp> "42.6"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceWSpeed> "25.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgPrec> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgPress> "30.17"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgRelHum> "87.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgTemp> "42.6"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWSpeed> "25.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:lat> "36.151" ;
            <w:long> "-79.789" .
```

Figure 45 Partial Output of Inferred RDF Triples (Run 1).

Semantic SOS O&M (Realized) Information

	Location	Condition
1	Location	Condition
2	Pinecroft	PoorRunningCondition
3	GSO/Jamestown	FairRunningCondition
4	Westerwood	FreezingRainWarning
5	Westerwood	PoorRunningCondition
6	McNair NCATSU	WindChillWarning
7	Meadowood	FairRunningCondition
8	Pinecroft	WindChillWarning
9	McNair NCATSU	PoorRunningCondition
10	Greensboro	PoorRunningCondition
11		

Figure 46 Screenshot of Final Information (Run 1).

The following presents the second sample run of the system with the data from Table 4. Figures 47 through 56 show the screenshots and partial outputs of Run 2.

Table 4 Sample Data for Run 2.

Feature of Interest	Observed Property	Value
foi_101	Temperature	40.6
foi_101	Relative Humidity	50.0
foi_101	Atmospheric Pressure	30.12
foi_101	Precipitation	0.5
foi_101	Wind Speed	3.0
foi_101	Wind Direction	45.0
foi_102	Temperature	49.8
foi_102	Relative Humidity	55.0
foi_102	Atmospheric Pressure	29.45
foi_102	Precipitation	0.0
foi_102	Wind Speed	0.0
foi_102	Wind Direction	45.0
foi_103	Temperature	61.5
foi_103	Relative Humidity	87.0
foi_103	Atmospheric Pressure	30.2
foi_103	Precipitation	0.0
foi_103	Wind Speed	45.0
foi_103	Wind Direction	45.0
foi_104	Temperature	108.6
foi_104	Relative Humidity	87.0
foi_104	Atmospheric Pressure	30.17
foi_104	Precipitation	0.0
foi_104	Wind Speed	25.0
foi_104	Wind Direction	45.0
foi_105	Temperature	37.8
foi_105	Relative Humidity	82.0
foi_105	Atmospheric Pressure	30.17
foi_105	Precipitation	0.0
foi_105	Wind Speed	0.0
foi_105	Wind Direction	45.0
foi_106	Temperature	44.3
foi_106	Relative Humidity	88.0
foi_106	Atmospheric Pressure	30.17
foi_106	Precipitation	0.0
foi_106	Wind Speed	0.0
foi_106	Wind Direction	45.0

Semantic SOS O&M Response

```
<om:ObservationCollection xmlns:om="http://www.opengis.net/om/1.0" xmlns:gml="http://www.o
  <gml:boundedBy>
    <gml:Envelope srsName="urn:ogc:def:crs:EPSG::4326">
      <gml:lowerCorner>-79.896 36.02</gml:lowerCorner>
      <gml:upperCorner>-79.776 36.151</gml:upperCorner>
    </gml:Envelope>
  </gml:boundedBy>
  <om:member>
    <om:Observation gml:id="go_1381714246703" about="obs_1381714247527" typeof="w:Observat
      <om:samplingTime>
        <gml:TimeInstant xsi:type="gml:TimeInstantType">
          <gml:timePosition>2013-10-13T21:28:54.937-04:00</gml:timePosition>
        </gml:TimeInstant>
      </om:samplingTime>
      <om:procedure rel="w:procedure" resource="file:///C:/apache-tomcat-7.0.42/bin/sosMap
      <om:observedProperty>
        <swe:CompositePhenomenon gml:id="CompositePhenomenon_file:///C:/apache-tomcat-7.0.
          <gml:name>CompositePhenomenon_file:///C:/apache-tomcat-7.0.42/bin/sosMap.ttl#urn
          <swe:component xlink:href="http://www.opengis.net/def/property/OGC/0/SamplingTim
          <swe:component xlink:href="file:///C:/apache-tomcat-7.0.42/bin/sosMap.ttl#urn:og
          <swe:component xlink:href="file:///C:/apache-tomcat-7.0.42/bin/sosMap.ttl#urn:og
```

Figure 47 Screenshot of O&M Response from Semantic SOS service (Run 2).

Semantic SOS O&M RDF Triples

```
@prefix w:       <./observationsMeasurements.owl#> .
@prefix xn:      <xmlns> .
@prefix sa:      <http://www.opengis.net/sampling/1.0> .
@prefix om:      <http://www.opengis.net/om/1.0> .
@prefix xlink:   <http://www.w3.org/1999/xlink> .
@prefix gml:     <http://www.opengis.net/gml> .
@prefix swe:     <http://www.opengis.net/swe/1.0.1> .
@prefix xsi:     <http://www.w3.org/2001/XMLSchema-instance> .
<file:///C:/apache-tomcat-7.0.42/bin/sosMap.ttl#foi_104>      a       w:Feature ;
              w:foiDesc "Greensboro" ;
              w:foiName "KNCGREEN49" ;
              w:location "-79.789 36.151" .
<file:///C:/jade/add-ons/wsig/app/classes/obs_1381714247527>      a       w:Observation ;
              w:featureOfInterest "file:///C:7apache-tomcat-7.0.42/bin/sosMap.ttl#foi_103"
              w:procedure <file:///C:/apache-tomcat-7.0.42/bin/sosMap.ttl#urn:ogc:object:fe
              w:result "2013-10-13T21:28:54.937-04:00,0.0,87.0,45.0,30.2,61.5,45.0;" .
<file:///C:/jade/add-ons/wsig/app/classes/obs_1381714247611>      a       w:Observation ;
              w:featureOfInterest "file:///C:7apache-tomcat-7.0.42/bin/sosMap.ttl#foi_104"
              w:procedure <file:///C:/apache-tomcat-7.0.42/bin/sosMap.ttl#urn:ogc:object:fe
              w:result "2013-10-13T21:28:54.937-04:00,108.6,87.0,45.0,30.17,25.0,0.0;" .
<file:///C:/jade/add-ons/wsig/app/classes/obs_1381714247587>      a       w:Observation ;
```

Figure 48 Screenshot of Extracted RDF Triples (Run 2).

```
<…/obs_1381714247643>      a       w:Observation ;
            w:featureOfInterest "…#foi_102" ;
            w:procedure <…#urn:ogc:object:feature:Station:IFGI:at-davis-weatherstation-2> ;
                w:result "2013-10-13T21:28:54.937-04:00,45.0,29.45,49.8,55.0,0.0,0.0;" .
<…#foi_101>      a       w:Feature ;
            w:foiDesc "McNair NCATSU" ;
            w:foiName "MCNAIR" ;
            w:location "-79.776 36.072" .
<…#foi_102>      a       w:Feature ;
            w:foiDesc "Westerwood" ;
            w:foiName "KNCGREEN30" ;
            w:location "-79.812 36.077" .
<…/obs_1381714247574>      a       w:Observation ;
            w:featureOfInterest "…#foi_101" ;
            w:procedure <…#urn:ogc:object:feature:Station:IFGI:at-davis-weatherstation-1> ;
                w:result "2013-10-13T21:28:54.937-04:00,3.0,0.5,30.12,50.0,40.6,45.0;" .
```

Figure 49 Partial Output of Extracted RDF Triples (Run 2).

Semantic SOS O&M (Operationalized) RDF Triples

```
1  @prefix w:        <file:///C:/jade/add-ons/wsig/app/classes/observationsMeasurements.owl#>
2  @prefix xn:       <file:///C:/jade/add-ons/wsig/app/classes/xmlns> .
3  @prefix sa:       <http://www.opengis.net/sampling/1.0> .
4  @prefix om:       <http://www.opengis.net/om/1.0> .
5  @prefix xlink:    <http://www.w3.org/1999/xlink> .
6  @prefix gml:      <http://www.opengis.net/gml> .
7  @prefix swe:      <http://www.opengis.net/swe/1.0.1> .
8  @prefix xsi:      <http://www.w3.org/2001/XMLSchema-instance> .
9  <file:///C:/apache-tomcat-7.0.42/bin/sosMap.ttl#foi_104>      a       w:Feature ;
10              w:foiDesc "Greensboro" ;
11              w:foiName "KNCGREEN49" ;
12              w:location "-79.789 36.151" ;
13              <w:hasAvgPrec> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
14              <w:hasAvgPress> "30.17"^^<http://www.w3.org/2001/XMLSchema#double> ;
15              <w:hasAvgRelHum> "87.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
16              <w:hasAvgTemp> "108.6"^^<http://www.w3.org/2001/XMLSchema#double> ;
17              <w:hasAvgWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
18              <w:hasAvgWSpeed> "25.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
19              <w:lat> "36.151" ;
20              <w:long> "-79.789" .
21 <file:///C:/jade/add-ons/wsig/app/classes/obs_1381714247527>      a       w:Observation ;
```

Figure 50 Screenshot of Operationalized RDF Triples (Run 2).

```
<…#foi_101>     a       w:Feature ;
            w:foiDesc "McNair NCATSU" ;
            w:foiName "MCNAIR" ;
            w:location "-79.776 36.072" ;
            <w:hasAvgPrec> "0.5"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgPress> "30.12"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgRelHum> "50.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgTemp> "40.6"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWSpeed> "3.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:lat> "36.072" ;
            <w:long> "-79.776" .
<…#foi_102>     a       w:Feature ;
            w:foiDesc "Westerwood" ;
            w:foiName "KNCGREEN30" ;
            w:location "-79.812 36.077" ;
            <w:hasAvgPrec> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgPress> "29.45"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgRelHum> "55.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgTemp> "49.8"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWSpeed> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:lat> "36.077" ;
            <w:long> "-79.812" .
```

Figure 51 Partial Output of Operationalized RDF Triples (Run 2).

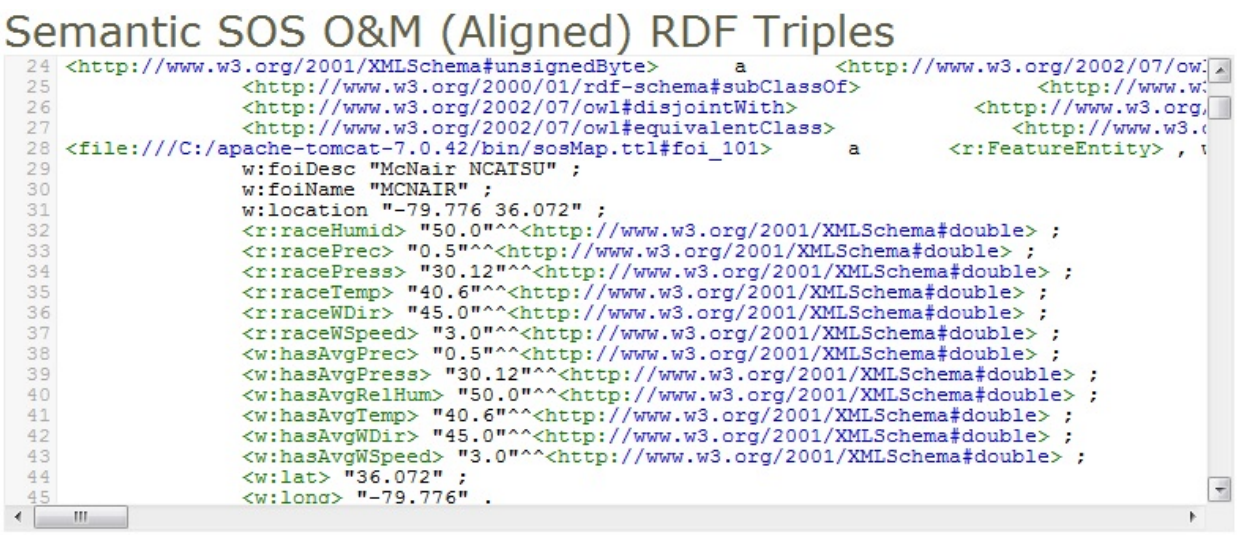

Figure 52 Screenshot of Aligned RDF Triples (Run 2).

```
<…#foi_101>     a       <r:FeatureEntity> , w:Feature ;
            w:foiDesc "McNair NCATSU" ;
            w:foiName "MCNAIR" ;
            w:location "-79.776 36.072" ;
            <r:raceHumid> "50.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:racePrec> "0.5"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:racePress> "30.12"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceTemp> "40.6"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceWSpeed> "3.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgPrec> "0.5"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgPress> "30.12"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgRelHum> "50.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgTemp> "40.6"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWSpeed> "3.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:lat> "36.072" ;
            <w:long> "-79.776" .
<…#foi_102>     a       <r:FeatureEntity> , w:Feature ;
            w:foiDesc "Westerwood" ;
            w:foiName "KNCGREEN30" ;
            w:location "-79.812 36.077" ;
            <r:raceHumid> "55.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:racePrec> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:racePress> "29.45"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceTemp> "49.8"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceWSpeed> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgPrec> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgPress> "29.45"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgRelHum> "55.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgTemp> "49.8"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWSpeed> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:lat> "36.077" ;
            <w:long> "-79.812" .
```

Figure 53 Partial Output of Aligned RDF Triples (Run 2).

Semantic SOS O&M (Inferred) RDF Triples

```
@prefix w:       <file:///C:/jade/add-ons/wsig/app/classes/observationsMeasurements.owl#>
@prefix xn:      <file:///C:/jade/add-ons/wsig/app/classes/xmlns> .
@prefix r:       <http://www.semanticweb.org/ontologies/2013/6/Ontology1375116837350.owl#>
@prefix sa:      <http://www.opengis.net/sampling/1.0> .
@prefix om:      <http://www.opengis.net/om/1.0> .
@prefix xlink:   <http://www.w3.org/1999/xlink> .
@prefix gml:     <http://www.opengis.net/gml> .
@prefix swe:     <http://www.opengis.net/swe/1.0.1> .
@prefix xsi:     <http://www.w3.org/2001/XMLSchema-instance> .
r:registrationURI      a       <http://www.w3.org/2002/07/owl#ObjectProperty> ;
               <http://www.w3.org/2000/01/rdf-schema#domain>              r:Race .
<http://www.w3.org/2001/XMLSchema#double>      a       <http://www.w3.org/2000/01/rdf-sche
<http://www.w3.org/2001/XMLSchema#unsignedByte>      a       <http://www.w3.org/2002/07/ow
               <http://www.w3.org/2000/01/rdf-schema#subClassOf>              <http://www.w
               <http://www.w3.org/2002/07/owl#disjointWith>              <http://www.w3.org/
               <http://www.w3.org/2002/07/owl#equivalentClass>              <http://www.w3.
<file:///C:/apache-tomcat-7.0.42/bin/sosMap.ttl#foi_101>      a       <r:FeatureEntity> , 
               w:foiDesc "McNair NCATSU" ;
               w:foiName "MCNAIR" ;
               w:location "-79.776 36.072" ;
               <r:raceHumid> "50.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
```

Figure 54 Screenshot of Inferred RDF Triples (Run 2).

```
<…#foi_101>     a       <r:FeatureEntity> , w:Feature , r:FairRunningCondition ;
            w:foiDesc "McNair NCATSU" ;
            w:foiName "MCNAIR" ;
            w:location "-79.776 36.072" ;
            <r:raceHumid> "50.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:racePrec> "0.5"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:racePress> "30.12"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceTemp> "40.6"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceWSpeed> "3.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgPrec> "0.5"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgPress> "30.12"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgRelHum> "50.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgTemp> "40.6"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWSpeed> "3.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:lat> "36.072" ;
            <w:long> "-79.776" .
<…#foi_102>     a       <r:FeatureEntity> , w:Feature , r:IdealRunningCondition , r:FairRunningCondition ;
            w:foiDesc "Westerwood" ;
            w:foiName "KNCGREEN30" ;
            w:location "-79.812 36.077" ;
            <r:raceHumid> "55.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:racePrec> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:racePress> "29.45"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceTemp> "49.8"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <r:raceWSpeed> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgPrec> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgPress> "29.45"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgRelHum> "55.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgTemp> "49.8"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWDir> "45.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:hasAvgWSpeed> "0.0"^^<http://www.w3.org/2001/XMLSchema#double> ;
            <w:lat> "36.077" ;
            <w:long> "-79.812" .
```

Figure 55 Partial Output of Inferred RDF Triples (Run 2).

Semantic SOS O&M (Realized) Information

```
1  Location          Condition
2  Pinecroft         PoorRunningCondition
3  Greensboro        ExcessiveHeatWarning
4  Westerwood        IdealRunningCondition
5  GSO/Jamestown     FairRunningCondition
6  McNair NCATSU     FairRunningCondition
7  Meadowood         FairRunningCondition
8  Westerwood        FairRunningCondition
9  Greensboro        PoorRunningCondition
10
```

Figure 56 Screenshot of Final Information (Run 2).

CHAPTER 7

Discussion

The prototype serves as a means to test the specific aims of this research. The first specific aim was to develop a sensor web system that collects observational data. The prototype directly addresses this specific aim by acquiring observational data and accurately servicing the client. The second specific aim focuses on delivering useful information along with observational data to make results more meaningful. The prototype addresses this by annotating the observational data with metadata. The use of RDFa allows the O&M response to be annotated with simple attributes such as “about”, “property”, and “typeof”. The third specific aim connects the specific, low level details to higher level concepts of interest to the user. This specific aim is addressed through the alignment of ontologies. The global ontology defines specific, low level details that are used throughout many domains, while the local ontologies define higher level concepts with some low level details that are identical to their global ontology counterparts. The fourth specific aim involves developing a deeper understanding of the context of the observational data. The multiagent system framework provides additional functionality that directly handles and manipulates the data and information. Observational data is fused into more accurate, useful information. This information allows the user to better understand the context of the situation and environment. The fifth specific aim integrates a user-friendly approach to minimize complexity to the user. An integrated user interface relieves the user of needless complexity, which increases time and resources for other areas such as decision making. The final specific aim focuses on developing a modularized sensor web implementation to support reusability. The development of the prototype resulted in separate components that could easily be integrated into other applications and domains. The components range from the

database schema and ontologies to collaborative agents and OGC service. The specific aims focus on creating a user experience that generates situation awareness.

This research explores sensor web technology by exploiting the technologies of distributed architectures. Most sensor webs focus on a particular sensor web concept, generally accessibility via OGC standards or sensor node collaboration. The sensor web here focuses on the three previously addressed senses of accessible, collaborative, and semantic sensor webs.

This research aligns well with the research of the sensor web community by seeking to improve data acquisition in dynamic environments as well as to provide accurate data to clients in a convenient, user-friendly manner. The research also adds value to the individual components needed to realize the sensor web vision, which includes multiagent systems, Web services, and the Semantic Web. Multiagent systems present an interesting approach to communication which leads naturally to collaboration. The primary advantage of machine-to-machine interaction via Web services extends the usability of sensor webs to another level while the search and inference capabilities of the Semantic Web makes sensor web more useful. This research develops a unique application of exciting technologies in the area of environmental monitoring.

CHAPTER 8

Conclusion

The primary goal of this research was to implement a prototype sensor web integrating the three aspects identified for a sensor web. Recall that a collaborative sensor web has nodes that collaborate in response to interesting environmental observations. An accessible sensor provides Web services that allow clients access to the sensor web resources and consumes Web services remotely. A Semantic Sensor Web adds contextual information to create a better understanding of the situation. The prototype displayed all three aspects.

The prototype sensor web and its products pose significance to agencies such as NOAA and NASA, as well as other organizations for environmental and structural health monitoring. This research contributes to the individual research communities of the major components needed to realize the prototype sensor web. The sensor web encourages an architecture for heterogeneous sensor nodes that can be used in “plug and play” and so could serve as a testbed for sensor node research. Multiagent systems exhibit efficient communication and problem-solving capabilities. The inference and query capabilities of the Semantic Web exploit another dynamic of intelligence, especially when paired with the intelligent agents of the multiagent systems. The OGC’s Sensor Web Enablement (SWE) integrates standardization and interoperability between sensor web implementations, resources, and data. Along with providing additional resources for SWE-compliant clients, the prototype sensor web allows reuse through its modular design.

The prototype sensor web could be easily extended for future work with a variety of possible directions. Interesting future work would incorporate additional OGC Web services or knowledge alignment in different domains. OGC Web services such as the Sensor Alert Service

or Sensor Registry Service would greatly benefit from the addition of semantic capabilities. Users can discover potentially useful sensors and resources through possible semantic additions.

References

Aalst, W. V. D. and Hee, K. V. (2002).*Workflow Management: Model, Methods & Systems*. Cambridge, Massachusetts: MIT Press.

Alhir, S. S. (2003). *Learning UML*.Sebastopol, CA: O'Reilly Media.

Alibhai, Z. (2003). *What is Contract Net Interaction Protocol?* Retrieved from http://www2.ensc.sfu.ca/research/iDEA/courses/files/Contract%20Net%20Protocol1.pdf

Bauer, B., Müller, J. P., and Odell, J. (2001).Agent UML: A Formalism for Specifying Multiagent Interaction. *Agent-Oriented Software Engineering*, P. Ciancarini and M. Wooldridge (Eds.), Berlin: Springer-Verlag, 91-103.

Beaujardiere, J. (Ed.) (2004*). OGC Web Map Service Interface*, OpenGIS White Paper, OGC 03-109r1, Open Geospatial Consortium Inc. http://portal.opengeospatial.org/files/?artifact_id=4756

Beckett, D. and Berners-Lee, T. (2008).*Turtle - Terse RDF Triple Language*. Retrieved from http://www.w3.org/TeamSubmission/turtle/

Bellifemine F. L., Caire, G.,and Greenwood, D. (2007).*Developing multi-agent systems with JADE*. Hoboken, NJ: John Wiley & Sons, Inc.

Bellifemine, F. (2001).*JADE: Java Agent Development Framework*. Retrieved from http://www.cril.univ-artois.fr/~leberre/MI32001/JADEEtaps2001.ppt

Berners-Lee, T. and Connolly, D. (2008).*Notation3 (N3): A readable RDF syntax*. Retrieved from http://www.w3.org/TeamSubmission/n3/

Bharathidasan, A. and Ponduru, V. (2002).*Sensor Networks: An Overview*, Department of Computer Science, University of California, Davis, CA. http://wwwcsif.cs.ucdavis.edu/~bharathi/sensor/survey.pdf

Bieszczad, A., Pagurek, B.,and White, T. (1998).*Mobile Agents for Network Management*.IEEE Communications, 1(1).

Boley, H.& Tabet, S. (Co-Chairs) *The Rule Markup Initiative*. Retrieved from http://ruleml.org/

Botts, M., Percivall, G., Reed, C., and Davidson, J. (Eds.) (2007).*OGC Sensor Web Enablement: Overview and High Level Architecture, Ver. 3*, OpenGIS White Paper, OGC 07-165, Open Geospatial Consortium Inc. http://portal.opengeospatial.org/files/?artifact_id=25562

Caire, G. (2011). *Wade User Guide*. Retrieved from http://jade.tilab.com/wade/doc/WADE-User-Guide.pdf

Chong, C.-Y.and Kumar, S. P. (2003). *Sensor Networks: Evolution, Opportunities, and Challenges*. IEEE, 91(8).

Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (2001). *Web Services Description Language (WSDL) 1.1*. Retrieved from http://www.w3.org/TR/wsdl

Chu, X. (2005).*Open Sensor Web Architecture: Core Services,* MS Project thesis, Dept. of Computer Science and Software Engineering, University of Melbourne, Australia. http://www.gridbus.org/reports/OSWA-core%20services.pdf

Chu, X. and Buyya, R. (2007).*Service Oriented Sensor Web,* Sensor Network and Configuration: Fundamentals, Standards, Platforms, and Applications, Springer, Berlin.http://www.gridbus.org/papers/**SensorWeb**Chapter.pdf

Ciancarini, P. and Wooldridge, M.(Eds.) (2001). *Agent-Oriented Software Engineering*. Berlin: Springer-Verlag Lecture Notes.

Collinot, A.and Drogoul, A. (1998).*Using the Cassiopeia method to design a robot soccer team.Applied Artificial Intelligence*, United Kingdom: Taylor & Francis, 127-147.

Connolly, D. (2006). *Naming and Addressing: "URIs, URLs, ..."* Retrieved from http://www.w3.org/Addressing/

CRICKET-KIT: MCS Mote Developer's Kit. Retrieved from http://www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=132%3Acricket-kit

D2RQ. Retrieved from http://d2rq.org/

Daconta, M. C., Obrst, L. J., and Smith, K. T. (2003).*The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management*. Hoboken, NJ: Wiley.

Dickinson, J. (2010).*The Semantic Sensor Web: Making Sense of the "Internet of Things"*. Retrieved from http://91-514-201-s2010.wiki.uml.edu/file/view/Dickinson_SemanticSensorWeb.pdf

Document Object Model (DOM). Retrieved from http://www.w3.org/DOM/

DRAFT: Specification of the KQML Agent-Communication Language plus example agent policies and architectures. Retrieved from http://www.csee.umbc.edu/csee/research/kqml/kqmlspec/spec.html

Endsley, M.R. (2004). *Situation Awareness: Progress and Directions*. S. Banbury & S. Tremblay (Eds.), Ashgate Publishing.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*.Doctoral Dissertation.University of California, Irvine.

Fok, C.-L., Roman, G.-C., and Lu, C. (2009). Agilla: A Mobile Agent Middleware For Self-AdaptiveWireless Sensor Networks.*ACM Transactions on Autonomous and Adaptive Systems*.4(3).

Friedman-Hill, E. (2003).*Jess in Action: Rule-Based Systems in Java*. Greenwich, Connecticut: Manning Publications Company.

Fuentes-Fernández, R., Guijarro M.,and Pajares, G. (2009).A Multi-Agent System Architecture for Sensor Networks, *Sensors*, 9.

Gay, D., Levis, P., Culler, D., and Brewer, E. (2009).*nesC 1.3 Language Reference Manual*. Retrieved from http://nesl.ee.ucla.edu/fw/torres/home/projects/tossim_gumstix/root/nesc-1.3.1/doc/ref.pdf

Getting Started with Protege 4.x OWL. (2011). Retrieved from http://protegewiki.stanford.edu/wiki/Protege4GettingStarted

Ghezzi, C., Jazayeri, M., and Mandrioli, D. (2002).*Fundamentals of Software Engineering, 2nd Ed*. Upper Saddle River, NJ: Prentice Hall.

Gomez-Sanz, J. J., Fuentes, R., Pavón, J., and García-Magariño, I. (2008). INGENIAS development kit: a visual multi-agent system development environment. *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: demo papers*. Richland, SC, 1675-1676.

Havens, S. (Ed.) (2007). OpenGIS Transducer Markup Language Implementation Specification, OpenGIS White Paper, OGC 06-010r6, Open Geospatial Consortium Inc.http://www.transducerml.org/downloads/TMLspec_100.doc

Heavner, M., Fatland, D., Hood, E., and Connor, C. (2007). *SEAMONSTER: A Sensor Web Technology Implementation and Testbed in Southeast Alaska*. University of Alaska Southeast, Juneau, AK.http://esto.nasa.gov/conferences/nstc2007/papers/**Heavner_Matt**_A1P2_NSTC-07-0069.pdf

Hebeler, J., Fisher, M., Blace, R., and Perez-Lopez, A. (2009).*Semantic Web Programming*. Hoboken, NJ: Wiley.

Herbert, J., O'Donoghue, J., Ling, G., Fei, K., and Fok, C.-L.(2006). Mobile Agent Architecture Integration for a Wireless Sensor MedicalApplication.*Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., and Dean M. (2004). *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. Retrieved February, 2011 from http://www.w3.org/Submission/SWRL/

How to Become a FIPA Member. Retrieved from http://www.fipa.org/subgroups/

Imote2: High-performance Wireless Sensor Network Node. Retrieved from http://wsn.cse.wustl.edu/images/e/e3/Imote2_Datasheet.pdf

IRIS: Wireless Measurement System. Retrieved from http://www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=135%3Airis

JADE Board (2011).*JADE Web Services Integration Gateway (WSIG) Guide*. Retrieved from http://jade.tilab.com/doc/tutorials/WSIG_Guide.pdf

Java Agent Development Framework. Retrieved from http://jade.tilab.com/

JavaServer Pages Technology Retrieved from http://www.oracle.com/technetwork/java/javaee/jsp/index.html

Juan, T., Sterling, L., and Winikoff, M. (2002). Assembling Agent Oriented Software Engineering Methodologies from Features. *Proceedings of the 3rd International*

Workshop on Agent-Oriented Software Engineering. Bologna, Italy: Springer-Verlag, 198-209.

Lange, D. B. and Oshima, M. (1999).*Seven Good Reasons for Mobile Agents*.Communications of the ACM.42(3).

Laun, W. *A JAXB Tutorial*. Retrieved from http://jaxb.java.net/tutorial/

Levis, P. and Gay, D. (2009).*TinyOS Programming*. Cambridge: Cambridge University Press.

Lewis, F. L. (2004).*Wireless Sensor Networks,* Automation and Robotics Research Institute, University of Texas at Arlington, Ft. Worth, Texas.

Liang, Y. D. (2005).*Introduction to Java Programming: Comprehensive Version. 5th Ed*. New Jersey: Prentice Hall.

Liggins, M. E., Hall, D. L. and Llinas, J. (2008).*Handbook of Multisensor Data Fusion: Theory and Practice, 2nd Ed*. CRC Press.

Lindörfer, F. (2010).*Semantic Web Frameworks*. Retrieved from http://informatik.unibas.ch/lehre/hs10/cs341/_Downloads/Workshop/Reports/2010-HS-DIS-F_Lindoerfer-Semantic_Web_Frameworks-Report.pdf

McBride, B. (2010). *An Introduction to RDF and the Jena RDF API*. Retrieved from http://jena.sourceforge.net/tutorial/RDF_API/index.html

McGuinness, D. L. and Harmelen, F. V. (2004).*OWL Web Ontology Language Overview*. Retrieved from http://www.w3.org/TR/owl-features/

MICA2: Wireless Measurement System. Retrieved from https://www.eol.ucar.edu/rtf/facilities/isa/internal/CrossBow/DataSheets/mica2.pdf

MICAz: Wireless Measurement System. Retrieved from http://www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=148%3Amicaz

Moe, K. (2007). *NASA Focus on Earth Science Sensor Web Technology*.Retrieved from http://wgiss.ceos.org/meetings/wgiss24/Tech-and-Services/SWTT/WGISS24_SensorWeb_NASA_KMoe.ppt

N-Triples: W3C RDF Core WG Internal Working Draft. Retrieved from http://www.w3.org/2001/sw/RDFCore/ntriples/

oasis-open.org. *OASIS*. Retrieved from http://www.oasis-open.org/home/index.php

opengeospatial.org. *Open Geospatial Consortium, Inc.* Retrieved from http://www.opengeospatial.org/

OWL 2 Web Ontology Language Document Overview. (2009). Retrieved from http://www.w3.org/TR/owl2-overview/

Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs). Retrieved from http://standards.ieee.org/getieee802/download/802.15.4d-2009.pdf

Probst, F. (2006) An Ontological Analysis of Observations and Measurements. 4th. International Conference on Geographic Information Science Münster, Germany.

Prud'hommeaux, E. and Seaborne, A. (2008).*SPARQL Query Language for RDF*. Retrieved from http://www.w3.org/TR/rdf-sparql-query/

RDFa Parser for java. Retrieved from https://github.com/shellac/java-rdfa

Reichardt, M. (2005).*Sensor Web Enablement: An OGC White Paper*. Open Geospatial Consortium (OGC), Inc.

Richardson, L. and Ruby, S. (2007). *Restful Web Services*.Sebastopol, CA: O'Reilly.

Sardini, E. and Serpelloni, M. (2011).Self-Powered Wireless Sensor for Air Temperature and Velocity Measurements with EnergyHarvesting Capability.*IEEE Transactions on Instrumentation and Measurement*,60(5).

Semantic Web for Earth and Environmental Terminology (SWEET). Retrieved from http://sweet.jpl.nasa.gov/ontology/

Sheth, A., Henson, C., and Sahoo, S. S. (2008). Semantic Sensor Web.*IEEE Internet Computing*.78-83.

Simon Cox (Ed.) (2007). *OGC Observations and Measurements – Part 1 - Observation schema*, OpenGIS White Paper, OGC 07-022r1, Open Geospatial Consortium Inc. http://xml.coverpages.org/OGC-07-022r1-ObservationsMeasurementsPart1-Schema.pdf

Simple API for XML. Retrieved from http://docs.oracle.com/javaee/1.4/tutorial/doc/JAXPSAX.html

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., and Katz, Y. (2007).Pellet: A Practical OWL-DL Reasoner, *Journal of Web Semantics*, 5(2).

St. Laurent, S., Johnston, J., and Dumbill, E. (2001).*Programming Web Services with XML-RPC*. Sebastopol, CA: O'Reilly.

Suggested Upper Merged Ontology (SUMO). Retrieved from http://www.ontologyportal.org/

sunspotworld.com. (2008).*Project SUN SPOT: Sun Small Programmable Object Technology*. Sun Microsystems, Inc.

Suri, D., Howell, A., Schmidt, D., Biswas, G., Kinnebrew, J., Otte, W., and Shankaran, N. (2007). “A Multi-Agent Architecture provides Smart Sensing for the NASA Sensor Web” *Proceedings of the 2007 IEEE Aerospace Conference*, 1-9.

Tan, Y. K. and Panda, S. K.(2011).Self-Autonomous Wireless Sensor Nodes with WindEnergy Harvesting for Remote Sensing ofWind-Driven Wildfire Spread.*IEEE Transactions on Instrumentation and Measurement*, 60(4).

Tanenbaum, A. S. (2002).*Computer Networks, 4th Ed*.Upper Saddle River, NJ: Prentice Hall.

TELOSB: TelosB Mote Platform. Retrieved from http://www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=152%3Atelosb

The Foundation for Intelligent Physical Agents. Retrieved from http://www.fipa.org

uddi.org.*OASIS*. Retrieved from http://www.uddi.org/pubs/uddi_v3.htm

Vantage Pro2 Console Manual. Retrieved from http://www.davisnet.com/product_documents/weather/manuals/07395-234_IM_06312.pdf

"VAST: Davis Weather Station." University of Alabama Huntsville, VAST. Retrieved from http://vast.uah.edu/index.php?view=article&catid=60%3Aweather-sensors&id=101%3Adavis-weather-station&option=com_content&Itemid=75

w3.org. *W3C: World Wide Web Consortium*. Retrieved from http://www.w3.org/

W3C: Reasoners. Retrieved from http://www.w3.org/2007/OWL/wiki/Implementations

Wang, Z. and Tianfield, H. (2005).A study of multi-agent system for network management.*Proceedings of International Conference on High Performance Computing and Its Applications*, Shanghai, China, 489-493.

Weather Underground. (2013). Retrieved from http://www.wunderground.com/

Web and XML Glossary.*HTTP Extension Framework*. Retrieved from http://dret.net/glossary/httpef

"Web Services – Axis."*Apache Web Services Project*. Retrieved from http://ws.apache.org/axis/

Web Services Summary. Retrieved from http://www.w3schools.com/webservices/ws_summary.asp

What is Unicode? (2010). Retrieved from http://unicode.org/standard/WhatIsUnicode.html

Wooldridge, M. (2002).*An Introduction to Multiagent Systems*. Chichester, England: John Wiley & Sons.

Woolridge, M., Jennings, N., and Kinny, D. (2000).The Gaia Methodology for Agent-Oriented Analysis and Design.*Autonomous Agents and Multi-Agent Systems*, Netherlands: Kluwer Academic Publishers, 3, 285-312.

XML Schema Summary. Retrieved from http://www.w3.org/XML/Schema