in Proc. of the 2009 International Symposium on Collaborative Technologies and Systems (CTS 2009), Baltimore, MD, May 18-22, 2009.

SemSOS: Semantic Sensor Observation Service

Cory A. Henson, Josh K. Pschorr, Amit P. Sheth, and Krishnaprasad Thirunarayan

Kno.e.sis Center, Department of Computer Science and Engineering
Wright State University, Dayton, OH 45435
cory@knoesis.org, pschorr.2@wright.edu, amit@knoesis.org, t.k.prasad@wright.edu

Abstract

Sensor Observation Service (SOS) is a Web service specification defined by the Open Geospatial Consortium (OGC) Sensor Web Enablement (SWE) group in order to standardize the way sensors and sensor data are discovered and accessed on the Web. This standard goes a long way in providing interoperability between repositories of heterogeneous sensor data and applications that use this data. Many of these applications, however, are ill equipped at handling raw sensor data as provided by SOS and require actionable knowledge of the environment in order to be practically useful. There are two approaches to deal with this obstacle, make the applications smarter or make the data smarter. We propose the latter option and accomplish this by leveraging semantic technologies in order to provide and apply more meaningful representation of sensor data. More specifically, we are modeling the domain of sensors and sensor observations in a suite of ontologies, adding semantic annotations to the sensor data, using the ontology models to reason over sensor observations, and extending an open source SOS implementation with our semantic knowledge base. This semantically enabled SOS, or SemSOS, provides the ability to query high-level knowledge of the environment as well as low-level raw sensor data.

***Index Terms*— Semantic Sensor Web, Semantic Web, Sensor Observation Service, Sensor Web**

I. Introduction

What are the possible benefits of integrating the Sensor Web with the Semantic Web? Much can be said in answer to this question, including a more expressive graph-based representation that models relationships as first class objects, the use of Uniform Resource Identifier's that allows all concepts to be independently accessible throughout the Web, and a triple-pattern encoding scheme that provides for simplified integration of heterogeneous datasets [1][2]. While these are all important elements of the Semantic Web, in this paper we will focus on the need for inference on sensor data enabled by semantic modeling and what advantages this provides to standard SOS.

Reasoning is a useful tool for providing meaning to sensor data and presenting insight into an observed environment. The quantified nature of sensor data, however, is not well suited for logical inference. In order to reason over sensor observations the data must first be annotated with meaningful concepts that can be manipulated with an inference engine. These concepts are defined in an ontology which provides the logical framework for further inference. In the Semantic Web, the Web Ontology Language (OWL) fulfills this role of a meta-language for ontology development.

This collection of annotations and inferences within an ontology make up a knowledge base. The knowledge in this knowledge base can be accessed through a standard SOS request, making the sensor data useful for a wide range of applications that lack the facility to handle raw sensor data but are able to deal with high-level knowledge. On the other hand, supposing an application does have the capability to handle raw sensor data, the lack of a service providing a shared semantics of sensor observations, obligates the client to independently translate the raw sensor data into useful high-level knowledge. This approach may lead to interpretations of data that are exclusive to a single client application and incompatible with applications that may otherwise make use of such knowledge. By committing to the interpretation described within an ontology, applications may benefit from a shared semantics of sensor data, thus leading to improved interoperability.

This configuration of a Sensor Observation Service that provides access to ontological knowledge of sensor observations is termed Semantic SOS, or SemSOS. Figure 1 shows an implemented architecture of SemSOS.

The remainder of the paper is organized as follows. Section 2 presents background material on the Sensor Web, as defined by the OGC Sensor Web Enablement, and the Semantic Web. Sections 3 and 4 discuss ontology development and semantic annotation, respectively. Rule-based reasoning over sensor data is presented in section 5. Section 6 describes our implementation. Finally, conclusions and future work are discussed in section 8.

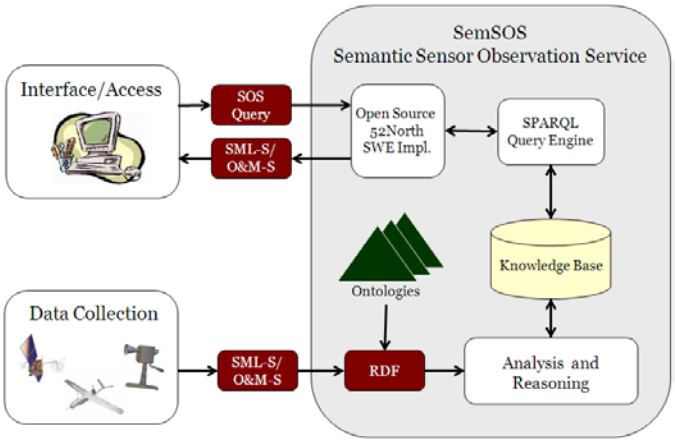

Figure 1. High-level view of SemSOS Architecture

from SAS and SPS web services and other elements of service workflows [1].

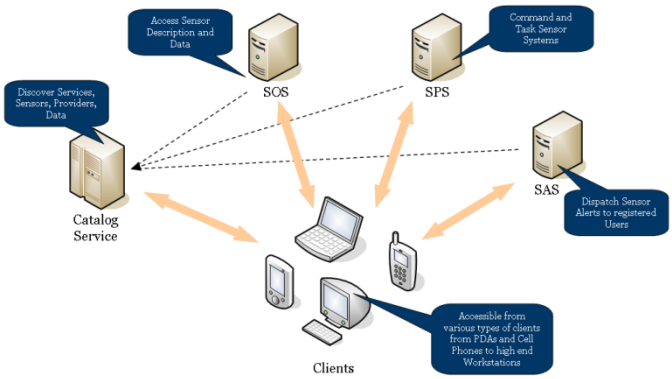

Figure 2. OGC Sensor Web Enablement Services

II. BACKGROUND

SemSOS is reliant on two sets of standardizations, (1) the Sensor Web Enablement languages and service interface specifications defined by the Open Geospatial Consortium (OGC), and (2) the Semantic Web languages defined by the World Wide Web Consortium (W3C).

A. Sensor Web Enablement

The Open Geospatial Consortium recently established the Sensor Web Enablement as a suite of specifications related to sensors, sensor data models, and sensor Web services that will enable sensors to be accessible and controllable via the Web [1]. The core suite of language and service interface specifications includes the following:

- *Observations & Measurements (O&M)* - Standard models and XML Schema for encoding observations and measurements from a sensor, both archived and real-time.
- *Sensor Model Language (SensorML)* - Standard models and XML Schema for describing sensors systems and processes; provides information needed for discovery of sensors, location of sensor observations, processing of low-level sensor observations, and listing of taskable properties.
- *Transducer Model Language (TransducerML)* - Standard models and XML Schema for describing transducers and supporting real-time streaming of data to and from sensor systems.
- *Sensor Observations Service (SOS)* - Standard web service interface for requesting, filtering, and retrieving observations and sensor system information. This is the intermediary between a client and an observation repository or near real-time sensor channel.
- *Sensor Planning Service (SPS)* - Standard web service interface for requesting user-driven acquisitions and observations. This is the intermediary between a client and a sensor collection management environment.
- *Sensor Alert Service (SAS)* - Standard web service interface for publishing and subscribing to alerts from sensors.
- *Web Notification Services (WNS)* - Standard web service interface for asynchronous delivery of messages or alerts

B. SWE Sensor Observation Service

The Sensor Observation Service (SOS) is an OGC-SWE standard which defines a web service interface for providing "access to observations from sensors and sensor systems in a standard way that is consistent for all sensor systems including remote, in-situ, fixed and mobile sensors [4]." SOS groups observations made by related sensor systems into *Observation Offerings*. An *Observation Offering* is a logical collection of sensors and sensor systems that, generally, are located in proximity to one another and sample their environment at shared intervals. *Observation Offerings* are characterized by the following parameters [4]:

- "Specific sensor systems that report the observations"
- "Time period(s) for which observations may be requested (supports historical data)"
- "Phenomena that are being sensed"
- "Geographical region that contains the sensors"
- "Geographical region that contains the features that are the subject of the sensor observations (may differ from the sensor region for remote sensors)"

SOS defines four service profiles: core, transactional, enhanced, and entire (which includes all functions from the previous three). For a standards compliant SOS service, only support for the core profile is mandatory, while all other profiles are optional. The core and enhanced profiles provide support for consumers of sensor data. A consumer client of sensor data requires methods for obtaining information about the service itself and requesting observations, sensor descriptions, features, etc. over some spatial and temporal context. This information is useful in applications such as visualization, data fusion, and situation awareness. The transactional profile supports publishers of sensor data. Such publisher clients are responsible for acting as intermediaries between sensor networks generating observations and the SOS service where it inserts sensor descriptions and observations.

The core profile includes three operations: *GetCapabilites*, *DescribeSensor*, and *GetObservation*. The *GetCapabilites* function provides a means to request a description of the service. This description includes information such as service identification (service name, keywords, etc.), provider, and most importantly, metadata that allows for the discovery of the

capabilities of the service. The capability description includes metadata about all supported functions of the service (including valid values and ranges for query parameters), filtering capabilities (logical operators that may be supplied with query parameters), and a full list of all *Observation Offerings* (including the aforementioned parameters: sensor systems, time, phenomenon, location, etc.) defined within the service. *DescribeSensor* allows the client to request information about a sensor. *DescribeSensor* is parameterized by the ID of the senor and returns a SensorML or TransducerML document describing the sensor and its capabilities. The *GetObservation* function 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*. *GetObservation* supports a multitude of parameters and filters, which give the client the ability to query over the sensor, time, location, phenomena, features, and measurement values of the observations. The response from *GetObservation* is encoded in O&M.

The transactional profile of SOS includes functions that allow a client to insert new sensors and observations, and is composed of two functions: *RegisterSensor* and *InsertObservation*. *RegisterSensor* allows a client to insert a new sensor into an SOS service, including the sensor's capabilities as described in a SensorML or TransducerML document. *InsertObservation* allows a client to insert a new observation into an SOS service. The new observation is provided to the SOS encoded as an O&M document.

The enhanced profile provides an assortment of less-frequently needed functions. *GetObservationById* returns an O&M observation based on the ID of the observation. *GetResult* provides a means for a client to obtain sensor data on a frequent basis using less bandwidth, by using a template O&M document from a previous call to *GetObservation*. *GetFeatureOfInterest* returns a description of a feature of interest whose ID was advertised by *GetCapabilities*. *GetFeatureOfInterestTime* describes the valid time periods for a feature. *DescribeFeatureType* yields an XML schema for a feature. *DescribeObservationType* returns the XML schema for an observation generated for a type of phenomenon. *DescribeResultModel* yields an XML schema which can further describe the format of results returned by the SOS and referenced in *GetCapabilities*.

C. Semantic Web

The Semantic Web, as described by the W3C Semantic Web Activity, is an evolving extension of the World Wide Web in which the semantics, or meaning, of information on the Web is formally defined [5]. Formal definitions are captured in ontologies, making it possible for machines to interpret and relate data content more effectively. The principal technologies of the Semantic Web include the Resource Description Framework (RDF) [6] data representation model, and the ontology representation languages RDF Schema (RDF-S) [7] and Web Ontology Language (OWL) [8]. In addition to these representation languages, an RDF query language called SPARQL [9] is now a W3C recommendation and the common method of querying ontological data. Many rule languages and rule engines are now capable of reasoning with Semantic Web data, including SWRL (Semantic Web Rule Language), RIF (Rule Interchange Format), and the general purpose rule engine for the Jena Semantic Web Framework [16].

III. Ontology Models

An ontology is a formal model that defines concepts and their relations in a standard language, commonly described as a "specification of a conceptualization [10]." In practice, the Semantic Web defines several ontology languages, RDF, RDF-S, and OWL. The Resource Description Format (RDF) is a graph-based language that allows data within a domain to be linked through named relationships. An RDF graph is encoded as a set of subject-predicate-object triples which resemble the subject, verb, and object of a sentence. The subject and object are nodes in the graph and the predicate is a directional named link between the subject and object. "This simple triple structure turns out to be a natural way to describe a large majority of the data processed by machines. The subjects, verbs and objects are each identified by a Universal Resource Identifier (URI)—an address just like that used for Web pages. Thus, anyone can define a new concept, or a new verb, by defining a URI for it on the Web [11]." RDF-S, or RDF Schema, adds the ability to define hierarchies of concepts to RDF. The Web Ontology Language (OWL) is built on top of RDF and adds a logical formalism to the language. OWL is based on a tractable subset of First Order Logic called Description Logic. The logical formalism provided by OWL, in combination with rule engines, is what allows inference over semantically annotated sensor observations. The ontologies dealt with in this paper are encoded in OWL.

A. Observations and Measurements Ontology

Observations and Measurements (O&M) is an OGC-SWE standard which defines an XML Schema for describing observations and features. Within this standard, an observation (*om:Observation*) is defined as an "act of observing a property or phenomenon, with the goal of producing an estimate of the value of the property," and a feature (*om:Feature*) is defined as an "abstraction of real world phenomenon [12]." (Note: *om* is used as a namespace for Observations and Measurements and will be placed, with a colon, before concepts defined in the O&M schema. All defined concepts are italicized). The major properties of an observation include feature of interest (*om:featureOfInterest*), observed property (*om:observedProperty*), sampling time (*om:samplingTime*), result (*om:result*), and procedure (*om:procedure*). Often these properties can be complex entities that may be defined in an external document. For example, *om:FeatureOfInterest* could refer to any real-world entity such as a coverage region, vehicle, or weather-storm, and *om:Procedure* often refers to a sensor or system of sensors defined within a SensorML document. Therefore, these properties are better described as relationships of an observation.

In order to encode relationships in XML, the OGC-SWE often make use of XLink, XML Linking Language, a markup language that "allows elements to be inserted into XML

documents in order to create and describe links between resources. XLink provides a framework for creating both basic unidirectional links and more complex linking structures. It allows XML documents to:

- Assert linking relationships among more than two resources
- Associate metadata with a link
- Express links that reside in a location separate from the linked resources" [13]

While XLink allows XML documents to break free of the standard tree-model and define relationships between entities, the triple-pattern approach of RDF provides a far more natural and useful approach to encoding relationships. In RDF and OWL, relationships are considered first-class objects which have many benefits over XLink, such as the ability to assign a URI to a relationship, to classify relationships into hierarchies (RDF-S and OWL), and place constraints on relationships (OWL).

For these reasons, we have developed an encoding of the Observations and Measurements language in OWL. In this ontology, we have defined the previous relations, and more, in a form that may be queried and reasoned over effectively in order to derive actionable knowledge of the environment from sensor observations. (Note that the ontology captures a subset of concepts in O&M. A few notable exemptions currently include concepts related to coverage and sampling feature). The translation between O&M in OWL and O&M in XML is straightforward and thus allows SemSOS to remain SOS compliant. (From this point forward, we will refer to O&M in OWL as O&M-OWL and refer to O&M in XML as O&M-XML). Figure 3 shows a diagram of the major concepts and relations in O&M-OWL.

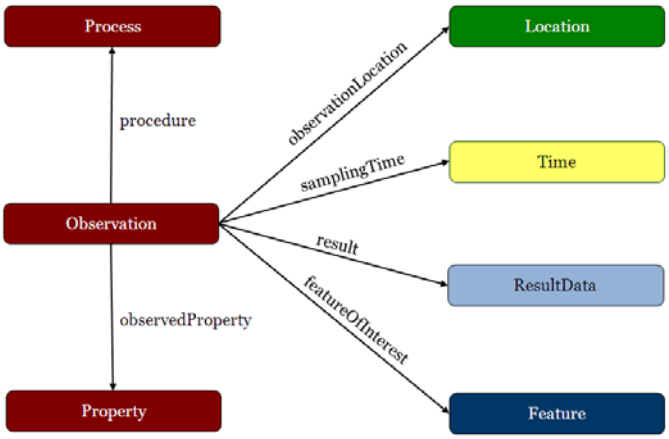

Figure 3. Subset of major concepts and relations in O&M-OWL

The following descriptions of relationships in O&M-OWL includes a running example of an observation from the domain of weather (concepts from weather ontology contain namespace "*w*"), encoded as a set of RDF triples. (Each line represents a triple, with the first term representing the subject, the second representing the predicate, the third representing the object, and ending with a period).

om:obs_1 rdf:type om:Observation .

- *om:featureOfInterest* is a "representation of the observation target, being the real-world object regarding which the observation is made [12]." Example includes a blizzard feature.

om:obs_1 om:featureOfInterest om:blizzard_1 .
om:blizzard_1 rdf:type w:Blizzard .
w:Blizzard rdfs:subClassOf om:Feature .

- *om:observedProperty* "identifies or describes the phenomenon for which the observation result provides an estimate of its value. It must be a property associated with the type of the feature of interest [12]." Example includes a temperature observed property.

om:obs_1 om:observedProperty w:temperature .
w:temperature rdf:type om:Property .

- *om:samplingTime* is the "time that the result applies to the feature-of-interest [12]," or, in other words, it is the time when the phenomenon was measured in the real-world. Example includes a single instant sampling time at 5:00 am on Jan. 26, 2009.

om:obs_1 om:samplingTime om:time_1 .
om:time_1 rdf:type owl-time:Instant .
om:time_1 owl-time:date-time "20090126T05:00:00" .

- *om:observationLocation* is the location of an observation event; usually associated with the location of the sensor when an observation occurred (i.e., *om:samplingTime*). Example includes a single point observation location with latitude, longitude, and elevation coordinates.

om:obs_1 om:observationLocation om:location_1 .
om:location_1 rdf:type gml:Point .
om:location_1 gml:latitude "41.1915" .
om:location_1 gml:longitude "-111.8351" .
om:location_1 gml:elevation "6562.0" .

- *om:result* is an "estimate of the value of some property generated by a known procedure [12]." Example includes a temperature measurement result of 37 degrees Fahrenheit.

om:obs_1 om:result om:result_1 .
om:result_1 rdf:type om:ResultData .
om:result_1 om:value "37" .
om:result_1 om:uom w:Fahrenheit .

- *om:procedure* is a "description of a process used to generate the result. It must be suitable for the observed property [12]." Note that in this schema a sensor is defined as a type of process, along with other methods, algorithms, instruments, or systems of these. Example includes a temperature sensor as the procedure.

om:obs_1 om:procedure om:sensor_1 .
om:sensor_1 rdf:type w:TemperatureSensor .
w:TemperatureSensor rdfs:subClassOf om:Sensor .
om:Sensor rdfs:subClassOf om:Process .

B. Spatial, Temporal, and Thematic Ontologies

From Figure 3, you will notice concepts related to *om:Observation* such as *om:Location*, *om:Time*, and *om:Feature*. While these concepts are defined in O&M-OWL, they are also extended with more expressive descriptions from existing schemas, in the case of *om:Location* and *om:Time*, and from a domain specific ontology, in the case of *om:Feature*. Locations within O&M-OWL are described using concepts from GML, or Geography Markup Language [14]. In particular, we re-use common concepts such as *gml:Point*, *gml:Polygon*, and *gml:coordinates*.

Time within O&M-OWL is described using concepts from OWL-Time [15]. OWL-Time, a W3C recommended ontology based on temporal calculus, provides descriptions of temporal concepts such as *owl-time:instant* and *owl-time:interval*, which supports defining interval queries such as 'within', 'contains', and 'overlaps'. The logical framework provided by OWL-Time for reasoning over time intervals could be very useful when dealing with observations that require complex temporal models. For example, o*m:TimeSeriesObservation* is defined as a *om:CompoundObservation* "whose sampling time is the period encompassing all the member times" such that all "member observations have the same feature of interest, the same observed property, and different sampling times [12]."

The concept of *om:Feature* within O&M encompasses all real-world entities and thus can be best described through domain-specific thematic ontologies. For example, for use in the domain of weather, *om:Feature* is extended with a weather ontology describing concepts such as *w:SnowStorm*, *w:Blizzard*, and *w:SnowFlurry*.

IV. Semantic Annotation

While encoding sensor data in OWL is useful for advanced analysis and reasoning, the SOS specification requires observation data to be encoded in XML for several operations. The *InsertObservation* operation takes an O&M-XML document as input and adds the observations to the storage facility. Similarly, the *GetObservation* operation returns an O&M-XML document as response to the query. As previously stated, translating from O&M-XML to O&M-OWL, and vice-versa, is straightforward. However, it is often useful to also embed semantic terminology defined in an ontology model into an XML document. This technique is called semantic annotation and is used for greater semantic interoperability of data encoded in XML, which provides only syntactic interoperability. Ontology terms are embedded in XML documents through model references, or URIs of concepts defined in an ontology. The OGC-SWE standards already provide several mechanisms to reference concepts that are external to the document. Such concepts are either defined in another XML document and accessed through an XLink element or defined in a registry and accessed through the *swe:definition* attribute. Using either mechanism, we can embed a model reference that will provide more meaningful description and thus enhanced semantic interoperability. Semantically annotated O&M and SML are called O&M-S and SML-S, respectively. This technique is also applied within the *GetCapabilities* operation in order to embed high-level *om:Feature* concepts that may otherwise be unavailable in an SOS *GetCapabilities* response. This is necessary to inform a SemSOS client of the precise description of concepts that may be used to query the knowledgebase.

V. Rule-based Reasoning

To derive additional knowledge from semantically annotated sensor observations, it's necessary to define and use rules. To demonstrate rule-based reasoning over sensor observation data, we currently use the general purpose rule engine from the Jena Semantic Web Framework [16]. Such rules deduce new ontological assertions from known instances and class descriptions. This section provides an example of inference through rules in SemSOS.

In the weather domain, if a group of sensors provides observations regarding wind speed, visibility, and precipitation, then by using inference rules we can specify existing weather events in the environment, such as a blizzard. The following rule states that if wind speeds are high (*HighWinds*), visibility is low (*LowVisibility*), and it is snowing (*Snowfall*), then there is a blizzard event (*Blizzard*) [17].

Blizzard ← *HighWinds* & *LowVisibility* & *Snowfall*

Each of these conditions described above is associated with a single time and location, derived from the time and location of the corresponding observations. Subsequently, the *Blizzard* condition is associated with the same time and location as the component weather conditions. The terms *HighWinds* and *LowVisibility* are also derived through rules.

HighWinds ← *WindSpeed* >= 35 MPH
LowVisibility ← *Visibility* <= ¼ mile

Within O&M-OWL, we begin with a quantified observation (*om:Observation*) and data result (*om:ResultData*) and translate this into additional qualified knowledge that can also be used within a reasoning engine. For example, the following set of RDF triples represents data about a wind speed observation.

om:windspeed_1 rdf:type w:WindSpeedObservation .
om:windspeed_1 om:samplingTime om:time_1 .
om:windspeed_1 om:observationLocation om:location_1 .
om:windspeed_1 om:result om:result_1 .
om:result_1 om:value "37" .
om:result_1 om:uom w:MPH .

From this set of RDF triples we can infer that observation *w:windspeed_1* can also be defined as an instance of class *w:HighWindSpeedObservation*. This new assertion is added to the original set of RDF triples (new triple in bold).

om:windspeed_1 rdf:type w:WindSpeedObservation .
om:windspeed_1 om:samplingTime om:time_1 .
om:windspeed_1 om:observationLocation om:location_1 .
om:windspeed_1 om:result om:result_1 .

om:result_1 om:value "37" .
om:result_1 om:uom w:MPH .
om:windspeed_1 rdf:type w:HighWindSpeedObservation .

The rule used to generate this new knowledge, titled *HighWindSpeedObservationRule*, is specified below (in the Jena rule syntax [16]).

[HighWindSpeedObservationRule:
(?w_obs rdf:type w:WindSpeedObservation)
(?w_obs om:samplingTime ?time)
(?w_obs om:observationLocation ?location)
(?w_obs om:result ?result)
(?result om:uom w:MPH)
(?result om:value ?value)
greaterThan(?value 35)
→(?w_obs rdf:type w:HighWindSpeedObservation)]

A low visibility observation (*w:LowVisibilityObservation*) is deduced similarly, and together with a snowfall precipitation observation (*w:SnowfallObservation*) we can infer a blizzard event (*w:Blizzard*) at the same time and location. The rule used to generate this new knowledge is titled *BlizzardObservationRule*.

[BlizzardObservationRule:
(?w_obs rdf:type w:HighWindSpeedObservation)
(?w_obs om:samplingTime ?time)
(?w_obs om:observationLocation ?location)
(?v_obs rdf:type w:LowVisibilityObservation)
(?v_obs om:samplingTime ?time)
(?v_obs om:observationLocation ?location)
(?p_obs rdf:type w:SnowfallObservation)
(?p_obs om:samplingTime ?time)
(?p_obs om:observationLocation ?location)
makeTemp(?blizzard)
→(?blizzard rdf:type w:Blizzard)
(?blizzard om:eventTime ?time)
(?blizzard om:eventLocation ?location)
(?w_obs om:featureOfInterest ?blizzard)
(?v_obs om:featureOfInterest ?blizzard)
(?p_obs om:featureOfInterest ?blizzard)]

Note that the *makeTemp(?blizzard)* function in the body of the rule generates a new instance in the knowledge base. Subsequently, we then supply this instance of *om:Blizzard* with relations in the head of the rule. In this example, such relations include *rdf:type*, *om:eventTime*, *om:eventLocation*, and *om:featureOfInterest*. The final set of RDF triples is shown below (ellipses used to truncate set of triples, and new triples in bold).

om:windspeed_1 rdf:type w:WindSpeedObservation .
om:windspeed_1 om:samplingTime om:time_1 .
om:windspeed_1 om:observationLocation om:location_1 .
…
om:windspeed_1 rdf:type w:HighWindSpeedObservation .
om:visibility_1 rdf:type w:VisibilityObservation .
…
om:visibility_1 rdf:type w:LowVisibilityObservation .
om:precipitation_1 rdf:type w:SnowfallObservation .
…
om:blizzard_1 rdf:type w:Blizzard .
om:blizzard_1 om:samplingTime om:time_1 .
om:blizzard_1 om:observationLocation om:location_1 .
om:windspeed_1 om:featureOfInterest om:blizzard_1 .
om:visibility_1 om:featureOfInterest om:blizzard_1 .
om:precipitation_1 om:featureOfInterest om:blizzard_1.

In this manner, we can infer features within the environment, of a particular type, at a specific time and place, and then generate *om:featureOfInterest* relations between the original observations and the new features. These new *om:featureOfInterest* relationships can be used to query for high-level feature concepts in SemSOS.

VI. SemSOS Implementation

In order to validate the framework discussed above, we have constructed a prototype of SemSOS. Our SemSOS extends the open source implementation of SOS from 52North [18] with an ontological knowledge base in order to provide inference over sensor data and queries of high-level features. For this prototype, the sensor observation data used to populate our ontologies was collected from MesoWest, a repository of weather data at the University of Utah [19]. MesoWest continually collects data from over 20,000 sensor systems within North America, and stores archives since 2002.

A. 52North SOS

52North's SOS implementation is designed to be highly modular, and adaptable to arbitrary suitable sensor data sources, transport protocols, etc. The larger enclosed box in

Figure **4** shows the high-level architecture of the 52North SOS.

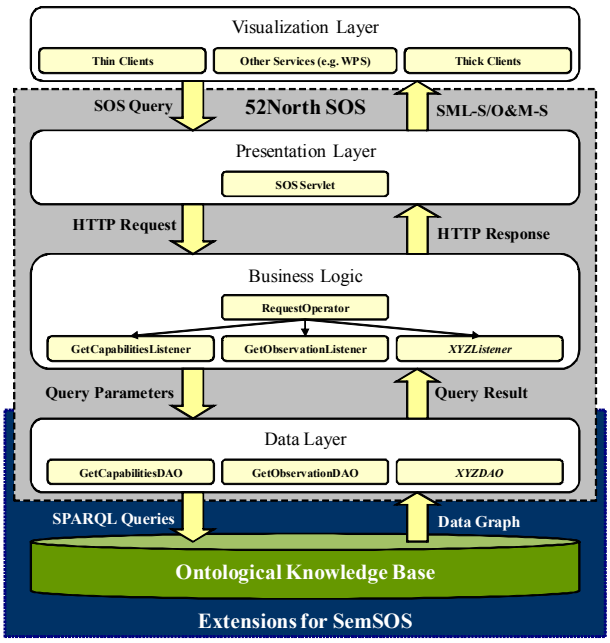

Figure 4. 52North SOS Architecture, extended with an ontological knowledge base

The Visualization Layer shown in Figure **4** is not part of the SOS itself, but rather corresponds to external clients that interact with the SOS. These can be either publishers or consumers of sensor data, and may also be other web services.

The Presentation Layer of 52North's architecture defines the SOS's interface to the outside world. The default implementation has a Servlet interface that accepts requests and communicates responses via HTTP. If another transport mechanism or protocol is required, this level can be replaced without affecting the other layers of the SOS.

The next level is the Business Layer, which receives requests from the Presentation Layer, handles them as appropriate, and returns a response. The Business Layer contains the logic for decoding requests and encoding responses. The main entry-point from the Presentation Layer is the *RequestOperator* object, which validates incoming requests, determines the type of request, and dispatches accordingly. Each operation supported by the SOS (*GetCapabilities*, *GetObservation*, etc.) is embodied by a Listener object which handles the corresponding incoming request (resp. *GetCapabilitiesListener*, *GetObservationListener*, etc.). The Listener objects may be configured externally during deployment of the service. The individual Listeners handle high-level translation of the request into an internal format which is then used to query the respective object in the Data Layer and compose the response.

The final layer of the 52North architecture is the Data Layer. The Data Layer is an abstraction of a sensor data source through Data Access Objects (DAO). Each DAO represents a particular interface to the sensor data from the point of view of one of the SOS's operations. For each Listener object in the Business Logic Layer, there is a corresponding DAO object in the Data Layer. The DAO objects are used by their respective Listener objects to obtain the data pertaining to a query. The abstraction provided by the DAOs and the Data Layer is what allows the 52North's SOS implementation to be so easily adapted to new sources of sensor data. For each operation that must be supported, all that is required is a new DAO that works with the data source. The default implementation shipped with 52North uses a PostGIS database with a custom database schema to store observation data, while sensor descriptions are stored on the file system in XML files (using SensorML or TransducerML).

B. SemSOS extensions to 52North

The box surrounding the bottom third of Figure **4** denotes the extensions made to 52North's SOS in order to implement SemSOS. The modular nature of the 52North implementation allowed us to leave the request routing, encoding/decoding, and similar details in place, while replacing the data access implementation with our own. The DAOs for all three operations specified in the SOS core profile (*GetCapabilities*, *GetObservation*, and *DescribeSensor*) were replaced with implementations that support data access to an O&M-OWL knowledge base.

Specifically, SemSOS uses the Jena Semantic Web Framework [16] to store and access the O&M-OWL ontology. The stored ontology is then accessed via SPARQL queries that are generated from the incoming SOS query parameters [9]. In producing the SPARQL queries, the syntactic form of the SOS query parameters (such as date, time, magnitude, etc.) are transformed into appropriate formats for semantic querying over O&M-OWL. Likewise, query filters (such as location, comparison operators, etc.) must be transformed into SPARQL-style filters and relational operations.

Evaluating a SPARQL query results in a set of triples representing an RDF graph, with data annotated in O&M-OWL. This graph is then transformed into the internal 52North result structure and returned to the Business Logic Layer. Here, the previous translation to convert SOS queries into SPARQL must be performed in reverse. O&M-OWL concepts instantiated within a set of RDF triples are translated into O&M-XML.

The results of SemSOS client queries are thus valid SOS results. SemSOS also provides richer semantic interoperability for clients that are semantically-aware through semantic annotation of the O&M-XML result document. This is achieved by using model references, or URIs of concepts defined in an ontology, as identifiers within O&M-XML.

C. Example SemSOS Query Processing

The first step the SemSOS DAOs must take in serving an SOS request is to translate the incoming SOS query into a SPARQL query which may be run against the knowledge base. Figure 5 shows an example SOS query asking for all observations that:

- are generated by procedures (sensors) that are part of offering (sensor constellation) 'BRAU1'
- fall within the time span of 2003-04-03T20:00:00-05 to 2003-04-04T02:00:00-05 (a six-hour interval)
- correspond to one of four specific observed properties:
    - Air Temperature
    - Precipitation
    - Wind Speed
    - Wind Gust

```
<GetObservation xmlns=... service="SOS" version="1.0.0"
                srsName="urn:ogc:def:crs:EPSG:4326">
<offering>BRAU1</offering>
<eventTime>
   <ogc:TM_During>
      <ogc:PropertyName>urn:ogc:data:time:iso8601</ogc:PropertyName>
      <gml:TimePeriod>
         <gml:beginPosition>2003-04-03T20:00:00-05</gml:beginPosition>
         <gml:endPosition>2003-04-04T02:00:00-05</gml:endPosition>
      </gml:TimePeriod>
   </ogc:TM_During>
</eventTime>
<observedProperty>http://.../weather.owl#_AirTemperature</observedProperty>
<observedProperty>http://.../weather.owl#_Precipitation</observedProperty>
<observedProperty>http://.../weather.owl#_WindSpeed</observedProperty>
<observedProperty>http://.../weather.owl#_WindGust</observedProperty>
<responseFormat>text/xml;subtype=&quot;om/1.0.0&quot;</responseFormat>
</GetObservation>
```

Figure 5. Example SOS Query

The SOS query is then transformed into the SPARQL query depicted in Figure 6, which expresses the same constraints as

the original, but in the language of O&M-OWL. Note that the event time specification in the SOS query becomes a SPARQL filter, as do the observed property specifications. Other SOS query relational operations and filters, such as location or feature of interest, are handled similarly.

```
SELECT DISTINCT ?offering ?offeringID ?proc ?obs ?phen ?resultDataType ?floatValue
    ?intValue ?booleanValue ?date ?foi ?foiType ?loc ?locType ?lat ?long ?elevation

WHERE {
   ?offering rdf:type observation:System .
   ?offering observation:ID "BRAU1" .
   ?offering observation:ID ?offeringID .
   ?offering observation:systemComponentProcess ?proc .
   ?proc observation:generatedObservation ?obs .
   ?obs observation:instFeatureOfInterest ?foi .
   ?obs observation:featureOfInterest ?foiType .
   ?obs observation:samplingTime ?inst .
   ?inst xsd:datetime ?date .
   ?obs observation:observedProperty ?phen .
   ?obs observation:result ?result .
   ?result rdf:type ?resultDataType .
   {
      {?result observation:floatValue ?floatValue . }
      UNION {?result observation:intValue ?intValue . }
      UNION {?result observation:booleanValue ?booleanValue . }
   }
   ?obs observation:observationLocation ?loc .
   ?loc rdf:type ?locType .
   ?loc observation:latitude ?lat .
   ?loc observation:longitude ?long .
   ?loc observation:elevation ?elevation .
   FILTER(?phen=<http://.../weather.owl#_AirTemperature>
      || ?phen=<http://.../weather.owl#_Precipitation>
      || ?phen =<http://.../weather.owl#_WindSpeed>
      || ?phen=<http://.../weather.owl#_WindGust> ) .
   FILTER ( ?date > "2003-04-03T20:00:00"^^xsd:dateTime
      && ?date < "2003-04-04T02:00:00"^^xsd:dateTime ) .
}
```

Figure 6. Example SPARQL Query

The table in Figure 7 displays one row of the result from the query in Figure 6. The row contains information pertaining to a single air temperature reading generated by a sensor that is a member of the offering specified in the original SOS query. The result value of the reading is present *(?floatValue*), along with the location (*?loc*, *?locType*, *?lat*, *?long*, *?elevation*) and a related feature of interest (*?foi*, *?foiType*), in this case an instance of freezing rain. The full result of the SPARQL query contains many more rows including observations from the same sensor at different times, and observations from other sensors contained in the same offering, which may have observed different phenomena and relate to different features.

?offering	<http://.../observation.owl#system_BRAU1>
?offeringID	BRAU1
?proc	<http://.../observation.owl#TemperatureSensor_46>
?obs	<http://.../observation.owl#observation_BRAU1_2003_04_04_01_00_00_AIRTEMPERATURE>
?phen	<http://.../weather.owl#_AirTemperature>
?resultDataType	<http://.../observation.owl#MeasureData>
?floatValue	2.0
?date	2003-04-04T01:00:00
?foi	<http://.../weather.owl#FreezingRain_562>
?foiType	<http://.../weather.owl#_FreezingRain>
?loc	<http://.../observation.owl#point_BRAU1>
?locType	<http://.../observation.owl#Point>
?lat	40.8844
?long	-110.8292
?elevation	8536.0

Figure 7. Example SPARQL Query Results

The result of the SPARQL query is then used to construct an SOS response document, as show in Figure 8.

```
<om:ObservationCollection ...>
 <gml:boundedBy>
  <gml:Envelope>
   <gml:lowerCorner>-110.8292007446289 40.8843994140625</gml:lowerCorner>
   <gml:upperCorner>-110.8292007446289 40.8843994140625</gml:upperCorner>
  </gml:Envelope>
 </gml:boundedBy>
 <om:member>
  <om:Observation>
   <om:samplingTime>
    <gml:TimePeriod xsi:type="gml:TimePeriodType">
     <gml:beginPosition>2003-04-03T20:00:00-05:00</gml:beginPosition>
     <gml:endPosition>2003-04-03T20:00:00-05:00</gml:endPosition>
    </gml:TimePeriod>
   </om:samplingTime>
   <om:procedure xlink:href="http://.../observation.owl#TemperatureSensor_46"/>
   <om:observedProperty>
    <swe:CompositePhenomenon gml:id="cpid0" dimension="2">
     <gml:name>resultComponents</gml:name>
     <swe:component xlink:href="urn:ogc:data:time:iso8601"/>
     <swe:component xlink:href="http://.../weather.owl#_AirTemperature"/>
    </swe:CompositePhenomenon>
   </om:observedProperty>
   <om:featureOfInterest>
    <gml:FeatureCollection>
     <gml:featureMember>
      <sa:SamplingPoint gml:id="FreezingRain_562">
       <gml:name>FreezingRain_562</gml:name>
       <sa:position>
        <gml:Point>
         <gml:pos srsName="urn:ogc:def:crs:EPSG:4326">
                -110.8292007446289 40.8843994140625</gml:pos>
        </gml:Point>
       </sa:position>
      </sa:SamplingPoint>
     </gml:featureMember>
    </gml:FeatureCollection>
   </om:featureOfInterest>
   <om:result>
    <swe:DataArray>
     <swe:elementCount>
      <swe:Count><swe:value>1</swe:value></swe:Count>
     </swe:elementCount>
     <swe:elementType name="Components">
      <swe:SimpleDataRecord>
       <swe:field name="Time">
        <swe:Time definition="urn:ogc:data:time:iso8601"/>
       </swe:field>
       <swe:field name="feature">
        <swe:Text definition="urn:ogc:data:feature"/>
       </swe:field>
       <swe:field name="weather.owl#_AirTemperature">
        <swe:Quantity definition="http://.../weather.owl#_AirTemperature">
         <swe:uom code="http://.../observation.owl#fahrenheit"/>
        </swe:Quantity>
       </swe:field>
      </swe:SimpleDataRecord>
     </swe:elementType>
     <swe:encoding>
      <swe:TextBlock decimalSeparator="." tokenSeparator="," blockSeparator="@@"/>
     </swe:encoding>
     <swe:values>2003-04-03T20:00:00-05,FreezingRain_562,2.0@@</swe:values>
    </swe:DataArray>
   </om:result>
  </om:Observation>
 </om:member>
```

Figure 8. Example SOS Response

VII. Conclusion and Future Work

A synthesis of the Sensor Web Enablement standards defined by the OGC and the Semantic Web languages defined by the W3C provides a platform for integration and reasoning over sensor observations in order to attain shared knowledge of an environment. This platform is broadly termed the Semantic Sensor Web [1], of which SemSOS is a principal component. In the preceding sections we have described how this is accomplished by modeling the domain of sensors and sensor observations in a suite of ontologies, adding semantic annotations to the sensor data, using the ontology models to reason over sensor observations, and extending an open source SOS implementation with our semantic knowledge base.

In the future, we hope to incorporate an abductive reasoning engine [20] as well as expand the Semantic Sensor Web platform. Abductive reasoning is often described as *inference to the best explanation*. In the sensors domain, a phenomenon is an effect that could have been caused (or could be explained) by many possible features, or real-world objects and events. An abductive reasoning engine would provide the

ability to reason from sensor observations of phenomena to possible hypothesis, or possible features, of the environment. Through an implementation of the SOS transactional profile (*RegisterSensor*, *InsertObservation*), and translation from O&M-XML to O&M-OWL, standard implementations of SOS may take advantage of the abductive reasoning capabilities of SemSOS in a modular, distributed, and standards-based environment.

In addition, we are planning on extending the Semantic Sensor Web platform beyond O&M-OWL and SemSOS. Such plans include developing an OWL version of Sensor Model Language (SML-OWL) and a semantically enabled Sensor Planning Service (SemSPS) and Sensor Alert Service (SemSAS). It is our belief that the addition of semantics to the OGC Sensor Web Enablement standards provides an improved platform for discovering, accessing, controlling, and reasoning over sensors and sensor observation data on the Web.

ACKNOWLEDGMENT

This research was supported in part by The Dayton Area Graduate Studies Institute (DAGSI), AFRL/DAGSI Research Topic SN08-8: "Architectures for Secure Semantic Sensor Networks for Multi-Layered Sensing." We also thank Harshal Patni and Hemant Purohit for their contribution towards the development of this project.

REFERENCES

[1] Amit Sheth, Cory Henson, and Satya Sahoo, "Semantic Sensor Web," IEEE Internet Computing, July/August 2008, p. 78-83.
[2] K. Thirunarayan, and J. K. Pschorr, "Semantic Information and Sensor Networks," In: Proceedings of the 24th Annual ACM Symposium on Applied Computing (ACM SAC 2009), March 2009.
[3] Mike Botts et al., "OGC Sensor Web Enablement: Overview and High Level Architecture (OGC 07-165)," Open Geospatial Consortium white paper, 28 Dec. 2007.
[4] Sensor Observation Service, http://www.opengeospatial.org/standards/sos
[5] W3C Semantic Web Activity, http://www.w3.org/2001/sw/
[6] Resource Description Framework (RDF), http://www.w3.org/TR/rdf-concepts/
[7] RDF Schema (RDF-S), http://www.w3.org/TR/rdf-schema/
[8] Web Ontology Language (OWL), http://www.w3.org/TR/owl-ref/
[9] SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/
[10] Tom Gruber, "A Translation Approach to Portable Ontology Specifications." Knowledge Acquisition, **5**(2), p. 199-220, 1993.
[11] Nigel Shadbolt and Tim Berners-Lee, "Web Science: Studying the Internet to Protect Our Future," Scientific American, September, 2008, http://www.sciam.com/article.cfm?id=web-science
[12] Observations and Measurements (O&M), http://www.opengeospatial.org/standards/om
[13] XML Linking Language (XLink), http://www.w3.org/TR/xlink/
[14] Geography Markup Language (GML), http://www.opengeospatial.org/standards/gml
[15] Time Ontology in OWL (OWL-Time), http://www.w3.org/TR/owl-time/
[16] Jena Semantic Web Framework, http://jena.sourceforge.net/
[17] National Oceanic and Atmospheric Administration's (NOAA) National Weather Service, Glossary, http://www.nws.noaa.gov/glossary/
[18] 52North Sensor Web Community, http://52north.org
[19] MesoWest, http://www.met.utah.edu/mesowest/
[20] K. Thirunarayan, C. Henson, and A. Sheth, "Situation Awareness via Abductive Reasoning from Semantic Sensor Data: A Preliminary Report," International Symposium on Collaborative Technologies and Systems (CTS2009), Workshop on Collaborative Trusted Sensing, Baltimore, Maryland, 2009. (Submitted)