Machine reasoning about anomalous sensor data

Matt Calder, Robert A. Morris, Francesco Peri
University of Massachusetts Boston
To appear Ecological Informatics with doi:10.1016/j.ecoinf.2009.08.007 or see also http://dx.doi.org.ezproxy.lib.umb.edu/10.1016/j.ecoinf.2009.08.007
Figures in this preprint best viewed at 200%, or printed. This preprint differs from the publication principally in layout.

Abstract. We describe a semantic data validation tool that is capable of observing incoming real-time sensor data and performing reasoning against a set of rules specific to the scientific domain to which the data belongs. Our software solution can produce a variety of different outcomes when a data anomaly or unexpected event is detected, ranging from simple flagging of data points, to data augmentation, to validation of proposed hypotheses that could explain the phenomenon. Hosted on the Jena Semantic Web Framework, the tool is completely domain-agnostic and is made domain-aware by reference to an ontology and Knowledge Base (KB) that together describe the key resources of the system being observed. The KB comprises ontologies for the sensor packages and for the domain; historical data from the network; concepts designed to guide discovery of internet resources unavailable in the local KB but relevant to reasoning about the anomaly; and a set of rules that represent domain expert knowledge of constraints on data from different kinds of instruments as well as rules that relate types of ecosystem events to properties of the ecosystem. We describe an instance of such a system that includes a sensor ontology, some rules describing coastal storm events and their consequences, and how we relate local data to external resources. We describe in some detail how a specific actual event---an unusually high chlorophyll reading---can be deduced by machine reasoning to be consistent with being caused by benthic diatom re-suspension, consistent with being caused by an algal bloom, or both.

Key Words
Ecology, Measurement, Ontology, Sensor. Rules, OWL.

Introduction
Deciding whether a sensor reading is unexpected, and if so whether it represents a system failure or, instead, an event of unusual interest requires testing against not only design parameters of the instrument, but also testing against models of the expected behavior of the system under measurement. There is a long history of addressing this question in industrial control and safety-critical systems (Del Gobbo et al., 1998). Recently, with the rapidly decreasing cost of wireless sensor networks for ecological study, the issue has attracted the attention of computer scientists
(Peng, 2009). The high rates at which wireless sensor networks can deliver data and the great volume of data the scientific community will therefore be faced with, suggest a need for a software solution to the problem of distinguishing sensor failures from interesting observations. Our work demonstrates that reasoning based on formal ontologies can assist in that task. The approach allows scientists to put forth formal models for physical properties being measured, and decide whether one or another explanation is logically consistent with the model and the measurements.

Our software relies on several components that work together to form a domain-agnostic data reasoning system. Its current configuration comprises a system for reasoning about coastal storm events, and data about them gathered by a wireless sensor network.

Our original implementation treated only the simple case of validating hypotheses of potential sensor failure. A simple example of sensor failure might be the return of values known to be out of the range of possible values specified by the sensor manufacturer. For example, let us assume that a certain model of temperature sensor has the physical limitation of never being able to take a measurement lower than -30 °C. In widely used rule notation with variables denoted with ‘?’, we can express this formally as

\[
\begin{align*}
\text{model37 range check:} \\
(?s \text{ type Model37\_TemperatureSensor}), \\
(?s \text{ hasTakenMeasurement } ?m), \\
(?m \text{ value } ?v), \\
\text{lessThan}(?v, -30) \rightarrow \\
(?s \text{ hasOutOfRangeViolation } ?m) 
\end{align*}
\]

The above rule expresses to the reasoning system “If \(s\) is a Model37 Temperature sensor and \(s\) has taken measurement \(m\) and \(m\) has value \(v\) and \(v\) is less than minus 30, then \(s\) might be out of range for reporting \(m\).” Use of the word might here is not incidental. Reasoning systems such as we describe are essentially constrained to examining their internal consistency, the consistency of their data with their models, and—of central interest to this work—the consistency of additional hypotheses about those models and data. Thus, they form a useful framework for evidence-based examination of hypotheses and models.

Modeling of sensor data with ontology has been approached before (Eid et al., 2006) but that work does not provide any framework for considering sensors as signalers of events. To describe events, the sensor data requires a range of values upon which arithmetic computations can be performed. However, because the now standard Web Ontology Language (OWL, 2004), is unable to express any constraints on numeric bounds of a data type field, a reasoner validating only against the ontology would fail to detect any inconsistency. This paper describes how we have added ontology-based rules to allow scientists to offer science-driven hypotheses about the cause of anomalous sensor network reports. To model hardware in use at the University of Massachusetts-Boston Center for the Study of Coastal Environmental Sensor Networks (CESN), the CESN ontology (CESN Sensor Ontology, 2008) provides concepts about sensors and their
deployments as seen by middleware responsible for database persistence. For this work, we have extended it to contain concepts about events that may occur during storms. The ontology is unconcerned with the sensor network logical or physical topology, or issues of intermediate aggregation within the sensor network. A second component is a local Knowledge Base of facts describing particular CESN instrument deployments as instances of classes defined in the ontology. A third, more novel, component is a collection of rule sets which represent the domain specific knowledge and hypotheses of scientists, in our case, oceanographers. Rules can be added to the software system dynamically and the Jena (Jena, 2009a) reasoner will signal whether they contradict the current state of knowledge in the system (i.e. the rules and facts). The final component is a KB supporting discovery and acquisition of data from resources other than our own, such as NOAA weather stations, guided by concepts from the ontology. For example, our simple ocean event ontology discussed later requires wind and wave data (or at least the value of attributes of the wind and waves at the time of the event), but our instruments provide no such information. (In the present implementation, we import this data statically, but the software is agnostic about its source, and future work will launch semantically based web searches to discover it.)

Our sensor systems push their collected data to the internet using a web service interface. Behind the web service our reasoning platform validates and applies domain knowledge rules on incoming data. Rules, with associated actions, applicable to the incoming data are supplied to the reasoner, which decides whether any conditions of the rules are met or fail to be met and trigger the associated action. Actions can include such things as tagging the data (e.g., as anomalous), putting it in a separate data store, or generating a notification to other software or to a human.

Background on Knowledge Representation

An ontology can be described as a representation of concepts within a certain domain of knowledge and the relationship between those concepts (Antoniuo and Harmelen, 2004). The notions that the OWL language uses to express an ontology are shared by most ontology languages. These include: classes, instances and properties (sometimes called attributes or predicates). Classes represent the concepts of a knowledge domain. Hierarchies among classes are represented by subclass relationships. Properties are used to establish relationships between concepts or between concepts and primitive data types such as numeric data. Individuals are instances of classes (Fig. 1). It might have been tempting to implement reasoning about sensor data purely with the use of an Object Oriented Programming (OOP) language such as Java. However, not only are there a number of reasoning engines available for OWL, but also, unlike in OOP, a class in OWL need not be defined by the sum of its properties. This makes ontology-based models particularly suitable for eco-informatics, because ecosystems—indeed most natural systems—exhibit many individual exceptions to the applicability of attributes that may form important parts of models.

A semantic reasoner is a software program capable of making or validating inferences based on sets of rules and facts expressed using the concepts of an ontology and the mathematical formalism known as first order logic. This has simple mechanisms for deducing whether statements are true or false, starting with a set of axioms and rules. This, too, corresponds closely to the way most scientific inquiry proceeds. The inquiry starts with a set of hypothetical facts—whose

![Fig. 1: Class and instance](image-url)
acceptance is bolstered by observation---and a set of rules for reasoning about those facts. The latter are really a kind of model, sometimes formally so. Within first order logic, a rule takes the general form of $P_1, \ldots, P_n \rightarrow Q$ where the $P_i$ and $Q$ are predicates (i.e. statements that are either true or false), and the arrow denotes logical implication. For example, a coastal ecosystem model might have a rule which says in plain language: “If water level is low and winds are high, sediment suspension will happen.” This can be represented logically as \( \text{low(waterlevel), high(winds)} \rightarrow \text{occurred(sediment suspension)} \). Given a set of facts and rules, a reasoner can find logical paths for new facts, validate those attributed to new observations, or even propose which existing rules and facts are inconsistent with the new observations. For use in machine reasoning or semantic modeling, there are special rule languages. Among the widely used ones are RuleML (2006), SWRL (Horrocks, 2004) and one that is utilized in our software from the Jena framework for semantic processing, Jena RL (Jena, 2007a).

**CESN: An Ontology of Sensors and their Measurements**

The purpose of the CESN sensor ontology is to describe the relationships between sensors and their measurements. The main concepts found in the CESN sensor ontology are similar to the terminology described in SensorML (2007) and to some of those emerging in the Marine Metadata Interoperability device ontology project (MMI, 2009), and CSIRO Sensor Ontology (CSIRO, 2009). These and several others are under scrutiny by the W3C Semantic Sensor Incubation Group (W3C, 2009a) with the active participation of their developers, and the entire subject of semantically enabled sensors is in its infancy (CEUR-WS, 2009; Sheth and Hanson, 2008).

As shown in Fig. 2, the core concepts in the CESN sensor ontology are the physical sensor devices themselves, Sensor; the PhysicalProperty that a Sensor can measure; and the measurement that a sensor has taken, PhysicalPropertyMeasurement. Not shown are important constraints, expressed in OWL, on this core. For example, a Sensor object can measure only one physical property. Objects that can contain Sensors and so measure more than one physical property are modeled by a class named Instrument. In turn, an instrument is usually deployed on some kind of Platform, which typically constrains its relationship to the environment in which it is deployed. Also not shown is the class Deployment, which represents the deployment of an instrument at a particular time and place, and so can be used to relate instrument readings to expected or unexpected events putatively signaled by the data modeled by the structure sketched in Fig. 2. A more detailed depiction of the core ontology is shown in Fig. 7.

Deployment attributes of individual instruments is particularly important in the real world of movable instruments. For example, it might be critical to know in advance of deployment whether an instrument package can be deployed underwater and to what depth. However, in this work we have no discussion of the Instrument, Platform, or Deployment because our focus, and the present software and rules, are agnostic about the origin of the data. As we shall see shortly, the rules in this case have no terms involving those classes. (Strictly speaking, this alone does not guarantee logical independence from those
classes, but an explication would require detailed discussion of inferred vs. stated relationships, which is beyond the scope of this paper, other than for our motivating example.)

**Semantic Data Validation and Inferences**

Fig. 3 depicts the principal components of the CESN reasoner and the data flow between them. Gateway hosts G receive data from instruments I via radios R or other communications channels. These forward data to a Reasoning Invocation Host, G*, which invokes our system via a web interface. Incoming sensor data, data requested by the system from other web services, and inferred data produced by the Semantic Reasoner are persisted as facts in a MySQL database through Jena’s Model interface. The Model is a convenient programming abstraction for easily managing instances. The reasoner component of the system knows about sets of rules expressed in the Jena rule language and it is asked by the Jena software to perform validation and inferences on the Model. Doing so may trigger actions specified by the rules.

The data that the Jena persistence engine has available to it for its reasoning and to maintain the integrity of its resources over time is illustrated in Fig. 4, which may be thought of as the engine's view of those resources. The Knowledge Base comprises domain specific ontology and rules, together with the backend database that Jena manages. We next discuss how the system maintains this integrity, computes inferred facts, and signals actions required by the rules.

**Validation**

Validation is an important part of maintaining the integrity of the KB. Validation insures that the KB “conforms” to the ontology. The main job of validation is to check for inconsistencies with constraints set in the ontology. If the reasoner does find any violations of constraints it reports these problems back to the system to take appropriate action. Because our wireless sensor networks must be scalable to very large deployments, our software only considers a few basic mechanisms for determining when validation and inference should occur. The first is to trigger an inference evaluation after a certain number of observations or observations of particular type have passed through the system. The second is a configurable time interval which upon expiration, will trigger the evaluation. A third is a batch loading from...
archival sensor data. An example of validation is checking that a particular type of sensor, such as a temperature sensor, is taking measurements of only one physical property. The reasoner has the ability to enforce this because that constraint is expressed in the ontology.

Inferences

Inference is the mechanism in our system by which domain knowledge rules can be used to deduce domain specific knowledge or to generate an action. For example, the following naïve rule called winter illustrates what a domain scientist may want to happen if some set of conditions on sensor data has been met. Winter can be expressed in Jena RL as

\[
\text{winter: (}\text{?m type Average),} \\
\text{(}\text{?m measurementOf Temperature),} \\
\text{(}\text{?m value ?value),} \\
\text{lessThan(?value, 0) } \rightarrow \text{ (season isWinter true) } 
\]

This rule corresponds to the following statement, “If there exists an average temperature which is below zero then the season is winter.” Obviously this rule is grossly simplified with respect to spatial and temporal variables, but it illustrates how we can express the creation of new knowledge. The right hand side of the ‘→’ is known as the consequent and the expressions on the left are the antecedents. The definition and evaluation of predicates in rules can also be expressed in Java, in an extension mechanism that Jena calls—somewhat misleadingly—a built-in (Jena, 2008b). Built-ins are used in our system to aid in providing actions and notifications in response to the antecedents, and hence the consequent, of a rule being true. For example, if an ecologist wished to receive a notification every time the temperature of his pond drops below 0 degrees, the system would have a rule such as

\[
\text{notifyme:} \\
\text{(}\text{?m type TemperatureMeasurement),} \\
\text{(}\text{?m value ?value),} \\
\text{lessThan(?value, 0) } \rightarrow \text{ email(“scientist@domain.edu”, “water is freezing! check the pond!”) } 
\]

Here, email is a simple built-in function that we added to the system. ‘greaterThan’ and ‘lessThan’ are built-ins that come with the Jena reasoner.

Study Case

Here we discuss a real unexpected event in a CESN instrument deployment, which was in fact the motivation for extending our data validation system to one that allows scientists to propose explanations for anomalous data. We present in some detail parts of an ontology, a Knowledge Base, and a set of rules that we have developed to support machine reasoning about ecosystem events in coastal embayments. It can support a wide variety of semantic modeling and hypothesis building about such events. We focus on the portions needed to address the kind of turbidity events that led to unusual chlorophyll readings in an instrument package deployed in a
Boston Harbor embayment named Savin Hill Cove. These rules are easily encoded for the Jena reasoner, which we deploy in the architecture described above. Albeit important for ecological and biological processes, ontological time modeling is quite complicated (See, e.g. the discussion of continuants in [Smith et al., 2005]). In the ontology described below, we ignore the fact that the properties of objects vary with time and we model an object as an instance of a class at a particular time. Also, the ranges (i.e. possible values) of class properties are greatly simplified for this discussion.

The motivating incident is reflected in the two graphs of Fig. 5. The lower shows temporally correlated chlorophyll and turbidity measurements from our instruments. On the same time scale the upper graph shows depth data at the instrument location, extrapolated from temporal wave height data in Boston Harbor, together with a simple model tracking the wave influence at a distance 1/4 wavelength below the wave. (Outside the deep ocean, the influence of waves on the bottom takes place mainly at depths less than 1/4 of the wavelength. More precisely, that influence decreases exponentially below that depth [Knauss, 2005]). From the figure, a human can easily see that the wave penetrates the bottom shortly before the chlorophyll and turbidity spike. In this section, we explore how our reasoning system can evaluate these data to signal the plausibility of several hypotheses to explain their correlations.

The implementation described in this paper is dedicated to events that take place in the near shore ocean or coastal estuaries. Fig. 6 shows our current ontology characterization of an OceanEvent, with the two subclasses about which the example below exhibits how reasoning proceeds to accept or reject that a particular OceanEvent instance may lie in one or both (or neither) of the subclasses AlgalBloom or BenthicResuspension. Whether or not an OceanEvent even occurred, and which, if either, type it might have been, is a question to be addressed by the data and rules, not the ontology. The question becomes: given semantically appropriate measurement values (Turbidity and Chlorophyll for an AlgalBloom or WaveLength and water Depth for a BenthicResuspension; See Fig. 7.), do those values support the possibility of an AlgalBloom (resp. a BenthicResuspension). In more detailed Appendices we offer some technical detail about how Jena helps us answer this question. Here we content ourselves with a less formal explanation.
According to the ontology details in Appendix 1, sensors that generate TurbidityMeasurements and ChlorophyllMeasurements must be of type OpticalBackScatterSensor and Fluorometer, respectively. Part of the ontology specifies that those types of sensor—and only those types—can generate the required type of measurement. Note also that measurement aggregations, such as statistical parameters, are treated as though they are themselves measurements. These measurements contribute to the data required by the algal bloom rule seen below.

[algal bloom rule: (?turb rdf:type cesn:StandardDeviation),
(?chlor rdf:type cesn:StandardDeviation),
(?turb cesn:aggregationOf cesn:TurbidityMeasurement),
(?chlor cesn:aggregationOf cesn:ChlorophyllMeasurement),
(?turb cesn:value ?turb_value), (?chlor cesn:value ?chlor_value),
greaterThan(?chl_value, “2.0”^^xsd:float),
greaterThan(?turb_value, “2.0”^^xsd:float) ->
email(`scientist@domain.edu`, `possible algal bloom`),
persist(`AlgalBloom`, now(), `Savin Hill Cove`) ]

This rule is capturing the fact that we are looking for turbidity and chlorophyll measurements that are two times greater than the standard deviation. When all the predicates of the rule are fulfilled then an email notification is generated and also the new knowledge of a possible AlgalBloom event is recorded in a data store. This is knowledge that an AlgalBloom could have happened at a certain time. It is not the data, nor the rules and ontology supporting the conclusion. However, Jena can be configured to log those also, during the invocation of its reasoning engine.

There are two measurement types that we use to model the re-suspension event; WavelengthMeasurement and DepthMeasurement. Both of these measurements are calculated from sensor data that comes from outside of the CESN network.

In the informal (and oceanographically simplified) statement below, we model the antecedents of a possible re-suspension event by asking whether a candidate wave has been signaled within the last 10 minutes at a location where the current water depth is less than 1/4 the

Fig 6: Ocean Event Structure

Fig 7. Partial measurement ontology
wavelength. As for the algal bloom event, when all requirements are fulfilled for the rule, then a notification email is posted and the possible event is recorded. Informally, the re-suspension rule is

\[
\text{if (wave was within last 10 minutes) and (depth < wavelength/4) then a re-suspension event was possible.}
\]

The expression of this in the Jena Rule Language requires considerably more complexity than this (see Appendix 2), particularly to set up the arguments to the arithmetic and time comparison predicates that we had to add, requiring a total of 17 antecedents to reach the conclusion and trigger a notice that conditions were supportive of a re-suspension event. In turn, were we operating the system to be triggered by a turbidity spike, an investigator could put forth hypotheses of both a benthic sediment re-suspension event and an algal bloom, either or both of which might be supported by the evidence as determined by the respective rules.

**Lessons Learned**

The CESN ontology was built for this project and has not been carefully examined for its generality. We are unlikely to pursue this, since a better path is to adopt the MMI Device Ontology or the W3C Semantic Sensor Network Incubation Group ontology as each emerges. Reflecting on the CESN ontology for this paper, we encountered modeling decisions familiar to more experienced knowledge modelers, and which we would not repeat (and probably would find impossible with the emerging standard ontologies). The most common of these is an excess of classes, some of which are too specific. For example, each instrument in our deployment is an instance of a class identifying the manufacturer and model, an unscalable model that could just as well have been satisfied by assigning those details to properties of more generic classes. In addition, the most straightforward mapping of relational data to OWL instance documents is to model tables as classes (Allemang and Hendler, 2008), and it is even clearer that one table per manufacturer/model is an unlikely organization for most scientific use cases.

A second issue deserves mention. That is, our modeling conflates the observation data, i.e. the measurements of the sensors during the events we reasoned about, with the properties of the sensors themselves. A separation of these two knowledge domains would be reasonable: there can be physical measurements in which the measurement instruments are irrelevant or unknown, and there can be instruments, knowledge of whose properties is useful independent of any data taken by them. Separating these is not likely to have much impact on our approach short of adding a second namespace, and the efforts of MMI and W3C intend to keep these domains separate.

Our ontology classifies different physical properties that sensors might measure, and specifies that a sensor can only measure one kind of physical property. However, it does not currently model physical properties as distinct. The OWL language includes relatively simple mechanisms to place such distinctiveness restrictions on concepts, but in this case we do not make use of them. Arguably, doing so is a more realistic model and could lead to more robust data. For example, nothing presently prohibits an object of type TemperatureMeasurement from also being an object of type HumidityMeasurement, although OWL is capable of that
expressivity. This could impede semantic data integration, where it may be important to
distinguish not only when classes can overlap, but when they cannot.

Future Work

Often the data available from the local network is insufficient to make certain complex
inferences because sensor networks tend to be very specific in what physical properties they are
measuring. In order to fulfill the data requirements of certain rules, a sensor network might need
to get information outside of its spatial, temporal and knowledge domains. In recent years there
have been several ongoing efforts to develop and standardize interoperable data interfaces to
sensor networks (Tilak et al., 2007; Sensor Observation Service, 2007; Sensor Planning Service,
2007). Coupled with the rising popularity of web services and service oriented architecture in the
scientific community, this gives the potential for sensor data systems to validate and make
inferences beyond the scope of their own knowledge base.

We intend to examine the performance scalability of operating the architecture in a real
time environment. Several questions come to mind for such an effort: (a) what is the impact of
our current over-coupling of the general and specific parts of the CESN ontology; (b) can rule
sets be automatically decomposed or reordered in such a way as to cause validation failure to
happen very quickly, e.g. because certain antecedents are the most likely to be the cause of
rejection of the consequent.

Energy conservation is a central concern for the remote nodes of a sensor network, so
operations not relating directly to a physical measurement should be put off elsewhere. All but
simple inference, such as programming “if-then-else” statements, is computationally expensive.

The most intuitive placement for sensor data inference is at or directly in front of a
natural aggregation point in the sensor network, where higher computational power is required
anyway. For many sensor networks these aggregation points are known as gateways. In general,
a gateway for a sensor network is usually a larger device with more computing power, storage,
and bandwidth than what is available to the sensor or instrument nodes. The gateway makes an
appropriate host for our system because the computational requirements our software take no
consideration of the resource sensitivity of sensor nodes.

There are instances where we would like to expand the geographic scope of reasoning.
Suppose that in Savin Hill Cove we were able to accept either of our two hypotheses, an algal
bloom or re-suspension. These events at the moment are local to a particular part of Boston
Harbor. To offer strength, or to generate further hypotheses, we would like to see if our
inferences hold after performing the same reasoning, or parts thereof, across similar sensor
networks available in Boston Harbor. We hope to incorporate such distributed reasoning in the
future.

Another interesting possibility is to allow rules to influence how the sensor network
behaves based on the discovery of an event. A use case for this might be that on detection of an
algal bloom in one geographic location in the network, it may be important to increase sampling
frequency in areas surrounding the existing location. This allows the network to determine when
it is appropriate the expend more energy for better measurements of interesting events and also it
leads to the possibility of being able to track a moving event across a region.

Lastly, well-designed user interface is needed to enable domain scientists to express their
knowledge. First order logic expressions in Jena RL are too cumbersome for users. A friendly
user interface could provide a high level graphical tool for scientists to construct rules.

Summary
We have developed an ontology and a Knowledge Base for sensor networks which observe coastal ecosystems. It includes mechanisms that can validate sensor observations, provide ways for scientists or decision makers to test hypotheses about anomalous sensor network observations and examine the impact on their models of accepting those observations as valid. The machine reasoning parts of this system are in place and tested with rules for validating data supplied by heterogeneous instrument packages according to rules relating expected correlations between the individual sensors and rules describing necessary conditions for the occurrence of specific ecological events.

Acknowledgements. This work was supported in part by U.S. Department of Energy Our thanks also to Bernie Gardner for explanation of the models of wave influence on bottom sediments. We appreciate a number of suggestions made by the reviewers.

Appendix 1. Jena, RDF, and OWL
This section sketches Jena and its use in somewhat more detail than above. For further details, consult Jena (2009b, 2009c).

RDF and its related languages RDFS (2009) and OWL describe resources by identifying them with Universal Resource Identifiers (“URIs”) and relations between them. Formally, RDF has two equivalent definitions. First, it is a set of triples <subject, predicate, object>, where the subject and object are URIs that identify some resources that are being described, and the predicate identifies a relation between them. Triples themselves can be declared to be resources, allowing relationships among triples to be described. This process is called reification, loosely following terminology from the linguistics discipline. To the extent we should think about a triple as part of a description of its subject, reification allows the formation of descriptions of descriptions. In turn, this allows descriptions not only of resources, but also of abstractions about them, i.e. classes of resources and properties of resources expressed without regard to any particular explicit resources. That is the role of RDFS and OWL.

A set of triples naturally gives rise to a directed labeled graph, whose nodes are resources occurring in the triple set, and directed edges from subject to object labeled with the predicate URI. Conversely given such a graph, we can produce a set of triples whose subject is the (URI of the) edge source, predicate is the edge label, and object is the (URI of the) target of the edge. Such a graph provides an equivalent definition of RDF. We have oversimplified these definitions especially in that RDF includes a rudimentary type system, which is especially important with the introduction of RDFS, which is an RDF vocabulary that adds classes to the basic notions of RDF. Thus, a triple <A, rdf:type, B> where B is a class defined in RDFS can be interpreted as saying that A is a member of B.

Sometimes one of these two expressions of RDF provides the modeler with a better view than the other. This makes the W3C RDF Validator (W3C, 2009b) a particularly helpful tool for exploring RDF knowledge models, because it can display both forms.

Finally, RDF has several serializations, including one in XML, called RDF/XML. This is convenient mainly due to widespread familiarity with XML and availability of many tools to
manipulate it. Unlike the graph or triple representations, it often fails to provide human readers
with insights into subtle issues in a model.

The Jena framework provides a Java API for the manipulation of, and reasoning about, RDF graphs. It supports the representation of these in memory using java objects, on files using various serialization forms, and in relational databases. It does this based on a unified Java interface called a Model and provides factory methods for constructing various kinds of Model sub-interfaces and implementations. One Model implementation is a Jena class named ModelRDB, which provides, via JDBC, an RDF triple store represented in a relational database, in our case MySQL. Jena can read an ontology and create the database as a single table with attributes a subject, predicate and object, each represented by a URI. (A fourth attribute provides a local identifier for the graph, which we ignore here). In our application, cesn.owl, the main ontology, models only the sensor and measurement abstractions, which are suitable for persistent storage, to be reloaded by Jena at the time the reasoning is done. By contrast, a small oceanevents.owl, which models events such as resuspensions, is loaded when reasoning is to be done, along with rules and data. Indeed, our current implementation is a web service which can take these runtime resources as arguments. Jena’s treatment of the rules is somewhat different from its treatment of ontologies, a point we take up in Appendix 2. Here we remark that the Fig. 8 below indicates a model in which cesn.owl was decomposed into a general upper ontology and a more specific lower ontology modeling particular types of sensors. In fact, for our prototype, both the general and specific, as well as some instance representations (i.e. assertions about specific sensors and deployments) in a single ontology, cesn.owl. This is not a scalable model as we described earlier. We have since refactored the ontology to essentially what Fig. 8 represents (with the additional sensor types) and have begun to develop a corresponding refactorization of the Java application in order to generalize it.
CESN upper and lower ontology. In Fig. 8, the upper portion of the diagram depicts the structure of an abstract Sensor class and that of a PhysicalPropertyMeasurement, which is what a Sensor can measure. Undepicted are representations of Deployment, Instrument, and Platform which describe how sensors are managed. Those are unused in the reasoning discussed in this paper. The lower part of the diagram models particular kinds of sensors as subclasses of Sensor. They have restrictions on what they can measure, and that is restricted to a single thing, which is an instance of PhysicalProperty. Instances of PhysicalProperty represent actual data, the central feature of which is a floating-point number identified by the value RDF property of the PhysicalProperty. In the upper diagram, are five subclasses of PhysicalProperty which are in fact part of the lower ontology, but repeated in the upper diagram for display convenience. Likewise,

Figure 8. Partial CESN Ontology
PhysicalProperty appears in the lower ontology for a similar reason. Consequently, without merging the repetitions, the illustration does not precisely represent an RDF triple graph. Square boxes are instances, in these cases of class PhysicalProperty.
The actual ontology presently has several other Sensor types, which measure, e.g. dissolved oxygen, temperature, and other physical properties.

Below we illustrate a portion of the MySQL table produced by Jena

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>cesn:ChlorophyllMeasurement:</td>
<td>rdf:type:</td>
<td>owl:Class:</td>
</tr>
<tr>
<td>cesn:Fluometer:</td>
<td>rdf:type:</td>
<td>owl:Class:</td>
</tr>
<tr>
<td>cesn:Fluometer:</td>
<td>rdfs:subClassOf:</td>
<td>cesn:Sensor:</td>
</tr>
<tr>
<td>7fbf</td>
<td>rdf:type:</td>
<td>owl:Restriction:</td>
</tr>
<tr>
<td>cesn:Fluometer:</td>
<td>rdfs:subClassOf:</td>
<td>7fbf</td>
</tr>
<tr>
<td>7fbf</td>
<td>owl:allValuesFrom:</td>
<td>cesn:ChlorophyllMeasurement:</td>
</tr>
<tr>
<td>7fbf</td>
<td>owl:onProperty:</td>
<td>cesn:hasTaken:</td>
</tr>
<tr>
<td>7fbe</td>
<td>rdf:type:</td>
<td>owl:Restriction:</td>
</tr>
<tr>
<td>cesn:Fluometer:</td>
<td>rdfs:subClassOf:</td>
<td>7fbe</td>
</tr>
<tr>
<td>cesn:Chlorophyll:</td>
<td>rdf:type:</td>
<td>cesn:PhysicalProperty:</td>
</tr>
</tbody>
</table>

The first four of these are the easiest to understand. The first two together say that ChlorophyllMeasurement is a Class which is a subClass of PhysicalPropertyMeasurement. The second two similarly describe Fluorometer as a subclass of Sensor. The remainder of this set of records is devoted to expression of some of the semantics of ChlorophyllMeasurement, namely that a Fluorometer can measure it and only it (i.e. if a Fluorometer hasTaken a measurement, then that measurement is necessarily a ChlorophyllMeasurement. The use of the OWL "restriction" mechanism to accomplish this is somewhat arcane, including the assignment of URIs to anonymous classes (here 7fbf and 7fbe). The reader unacquainted with it may consult Allemand and Hendler (p. 179ff). Finally, we remark that following common practice, in the interests of readability we have severely abbreviated the URIs that Jena inserts. The above triples in the relational database correspond to this portion of the RDF/XML expression of the owl ontology:

```xml
<owl:Class rdf:about="#ChlorophyllMeasurement"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasTaken"/>
  <owl:allValuesFrom rdf:resource="#ChlorophyllMeasurement"/>
</owl:Restriction>
</rdfs:subClassOf>
<owl:Class>
  <rdfs:subClassOf rdf:resource="#Sensor"/>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#canMeasure"/>
  </owl:Restriction>
  <owl:Restriction>
    <owl:hasValue rdf:resource="#Chlorophyll"/>
  </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```
Appendix 2. Rules

[resuspensionrule:

(?wavelength rdf:type http://www.cs.umb.edu/~mcalder/sensor/cesn.owl#WavelengthMeasurement),
(?depth rdf:type http://www.cs.umb.edu/~mcalder/sensor/cesn.owl#DepthMeasurement),

(?wavelength http://www.cs.umb.edu/~mcalder/sensor/cesn.owl#timestamp ?wave_time),
(?depth http://www.cs.umb.edu/~mcalder/sensor/cesn.owl#timestamp ?depth_time),

dateToLong(?wave_time, ?wave_time_long),
dateToLong(?depth_time, ?depth_time_long),
now(?now),
dateToLong(?now, ?now_long),
difference(?now_long, "600000"^^http://www.w3.org/2001/XMLSchema#long, ?ten_minutes_ago),
greaterThan(?wave_time_long, ?ten_minutes_ago),
greaterThan(?depth_time_long, ?ten_minutes_ago),
lessThan(?wave_time_long, ?now_long),
lessThan(?depth_time_long, ?now_long),

(?wavelength http://www.cs.umb.edu/~mcalder/sensor/cesn.owl#value ?wave_value),
(?depth http://www.cs.umb.edu/~mcalder/sensor/cesn.owl#value ?depth_value),

quotient(?wave_value, "4"^^http://www.w3.org/2001/XMLSchema#integer, ?wave_pen),

lessThan(?wave_pen, ?depth_value) ->

print('resuspension rule matched'),
email('scientist@domain.edu', 'possible resuspension event'),
persist('ResuspensionEvent', now(), 'Savin Hill Cove')]

Informally, this might be expressed as

let

?wavelength be a WavelengthMeasurement and
?depth be a DepthMeasurement and
?wavelength have timestamp wave_time and
?depth have timestamp depth_time and

?wave_time_long = dateToLong(?wave_time) and
?depth_time_long = dateToLong(?depth_time) and

?now be the current time and
?now_long = dateToLong(?now) and
?ten_minutes_ago = ?now_long - 600000 and

#wave within last ten minutes
?ten_minutes_ago > wave_time_long and
?ten_minutes_ago > depth_time_long and
?wave_time_long < now_long and
?depth_time_long < now_long and

15
Jena has two subinterfaces of the Model java interface mentioned in Appendix 1. The first, OntModel, is useful for java representations of RDF ontologies. Jena's ModelRDB class implements OntModel. This class has methods to produce and access a triple store in relational database. We use it at system initialization to store cesn.owl and oceanevents.owl in the persistent triple store represented in the MySQL database. When the reasoning system is invoked, observation data is acquired and added to the persistent triple store. Both the actual measurement value and it's type (e.g. ChlorophyllMeasurement) will be represented in the triple store. CESN reasoning consists of instantiating a second Jena interface, called InfModel ("Inference Model") which has access to both the RDF graph now representing both data and ontologies, as well as to the Jena rules chosen at run time, e.g. the resuspension rule illustrated above. Invoking the InfModel's validate() causes Jena to determine the logical consistency of the aggregate RDF graph comprising the ontology triples, the data triples, and the triples inferred by applying the rules. In the resuspension rule case, validity will result in notification that the event is consistent with resuspension. In our demonstration software, the observation data is loaded all at once from legacy data before the reasoning is invoked. However, it is simple to write a reasoning invocation loop that adds data in real time from an operating sensor network and tests the rule on each data acquisition. Indeed, we have also implemented a web service that can accept new data and invoke the validation in response to that arrival. Performance considerations that we have not yet explored almost certainly would mitigate against repeated access to the persistent triple store in that case, particularly for problems where a large amount of observation data is accumulating rapidly.
Fig 9 Dataflow

Fig. 9 shows a simplified UML sequence diagram that shows the internal architecture of the Jena-based components of the system. It omits any indication of whether observation data should remain only in memory, or remain serialized in the MySQL triple store. That is purely a configuration question, and depends only on whether a use case has need of re-use of the observations.

References


http://jena.sourceforge.net/inference/index.html#RULEextensions


Liu, Y., Hill, D., Rodriguez, A., Marini, L., Kooper R., Myers J., Wu, X., Minsker, B.,


OWL, 2004. Web Ontology Language http://www.w3.org/TR/owl-features/


