Semantic Access to Sensor Observations through Web APIs

Kevin R. Page, Alex J. Frazer, Bart J. Nagel, David C. De Roure, and Kirk Martinez
Intelligence, Agents, Multimedia Group, Electronics and Computer Science, University of Southampton, UK
{krp,ajf3,bjn,dder,km}@ecs.soton.ac.uk

***Abstract*—Sensor networks are often deployed with the purpose of providing data to large-scale information management and GIS systems, or to collect measurements for specific scientific experiments. The benefits of such use are clear and widely accepted. The reuse of observations in low-cost, lightweight, web applications and mashups is a further compelling use case for sensor networks, but requires provision of data through simple mechanisms, readily accessible, that are quick to develop with. To enable the latter while maintaining support for larger applications and, to increase information utility, links to and from other datasets, we propose a domain-driven approach that embodies REST and Linked Data principles using a common semantic framework that underpins a separation of concerns between domain models, sensor observation infrastructure, and Application Programming Interfaces (APIs) while maintaining information consistency. We describe a reusable, reconfigurable, web service that realises this design and can be deployed to provide access to multiple sources of sensor information, including databases and streaming data, with flexible semantic configuration of the API and domain mapping.**

I. Introduction

The ability to rapidly develop applications is one of the key challenges facing the semantic sensor web [1], encompassing the need to provide simple, consistent, interfaces to an ever growing quantity of sensed data, available in formats that provide enough context to be relevant and applicable to user applications.

Lightweight web applications and mashups are now an established phenomenon. They are typically constructed by re-purposing as much existing functionality and data as possible, developing new code to tie current systems together in interesting and insightful ways. This re-use relies on access to data and services through Application Programming Interfaces (APIs) that are easy to understand and simple to use.

That is not to say that the aggregate complexity of a mashup is necessarily lower than other development models once the services providing the APIs are also taken into account; it is rare that necessary functionally can be entirely removed (although it might be abstracted, segregated, and more efficiently scaled or applied). So while a good lightweight web API can significantly reduce the burden on an application or mashup developer, significant planning, design, and care during implementation must be employed by the service provider.

Self-describing and distributable data semantics, such as those developed for the Semantic Web, can be highly beneficial when combining data from different sources – the defining practice of any mashup. In this paper we explore the use of RDF and Linked Data for publishing sensor observations, presenting a suitable domain structure for this purpose. We develop a generic, reconfigurable, and adaptable service to provide high-level APIs for access to sensor observations. We show that the semantic domain model, once captured for data, can also be used to automate deployment of specific APIs, reducing administrative overheads while retaining a well-formed and structured API suitable for mashup development.

II. Background

A. Web Architecture, REST, and Linked Data

When surveying Web architecture and the means to access data through Web APIs, two prominent methodologies are found which both adopt an approach centred around the notion of *resources*.

The Linked Data movement has achieved considerable success constructing a semantic Web of Data [2]. While earlier research focussed on building a stack to enable reasoning and logic this more recent effort makes large scale distribution and linking a priority. Moving on from earlier assumptions that URIs would do nothing more than uniquely identify Things, the key thrust of Linked Data has been the re-adoption of HTTP URIs for retrieval of resource representations. This overloading of URIs requires a distinction between identifiers for real-world objects (non-information resources) and web resources containing (meta-)data (information resources), with mechanisms such as 303 redirects and content negotiation [3] to inform a client of a move from one to the other. The approach is often summarised by the four Linked Data 'rules' [4]: use URIs as names for things; use HTTP URIs so that people can look up those names; when someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL); and include links to other URIs, so that they can discover more things.

Representational State Transfer (REST) [5] is a set of design principles which have been popularly and successfully adopted in many ('RESTful') web services, and is typically framed as an alternative to 'heavyweight' web services. As with Linked Data, REST is founded upon the identification of resources for specific things that we wish to reference, and the referencing of these resources using URIs. Representations of resources – encoded in a particular format – are then accessed through the URI, usually by HTTP. In contrast to Linked Data's focus on a single form of data (RDF), REST encourages a plurality of representations with semantics encoded in link relations. REST can also be summarised by a set of principles: everything is a resource which is addressable; resources have multiple representations; relationships between resources are expressed through hyperlinks; all resources share a common interface with a limited set of operations; client server communication is stateless.

B. Models and Encodings for Sensor Data

A second consideration when designing an API are any existing services and specifications through which data can be served and accessed. Standardised encodings and service definitions from the Open Geospatial Consortium (OGC) are widely adopted across the sensor network industry. Earlier specifications introduced services to directly support Geospatial Information Systems (GIS), while more recent efforts have resulted in services defined as part of Sensor Web Enablement (SWE). GML is the common OGC encoding framework: an XML schema derived language in which several GML Application Schema are defined.

Earlier OGC standards used by GIS applications include Web Map Service (WMS) and Web Feature Service (WFS), the former returning information as a raster layer, the latter returning feature data as GML. Although Sensor Web Enablement is designed to provide for "Web-connected sensors", the approach taken by the included services is to run over Web protocols but not to adopt a Web Architecture through Resource Oriented services. While this is a valid and useful technique to extend GIS services into a more web-like platform, this specialisation of interfaces according to task (Sensor Observation, Alert, etc.) does not provide the kind of RESTful lightweight API desirable for mashup development.

The OGC data models and schemas (used to transfer information between server and client through interface calls) are of more interest since they are based on a thorough and comprehensive domain analysis without any presupposition of architectural style. Within SWE this is manifest as two complementary perspectives over the data:

A *provider-centric* approach orientated around and describing the processes undertaken by sensors, structured networks of sensors, and constituent elements of sensors. Data is a product of the described sensor network. In OGC standards this approach is adopted by the SensorML GML application schema and the SWE Sensor Planning Service.

A *consumer-centric* approach orientated around and primarily describing the observations and measurements – i.e. the data, the results – captured by sensors rather than the sensors themselves (although the provenance of observations is modelled through an associated process). The OGC Observations and Measurements (O&M) [6] model and GML application schema apply this approach, in turn used by the SWE Sensor Observation Service (SOS).

Recent work by the W3C Semantic Sensor Networks Incubator Group has incorporated these perspectives, including a derivative of the O&M Observation model, in an OWL ontology[7] which forms the basis for the data model developed in Section IV.

C. Related Work

In addition to an early prototype of the API design revisited in Section IV [8], there have been several proposals for exposing of sensor related information as Linked Data, each with differing motivations and foci: automated conversion from OGC standards and services [9], sensor data resource addressability [10], alignment with foundation ontologies [11], publishing linked sensor locations and attributes [12], sensor discovery over Linked Data [13], and integration from multiple sensor sources into a single service [14].

While the work presented here touches on issues raised in several of these works, our primary focus is instead on the design and deployment of APIs that are accessible and relevant to a developer working in the domain, on linking the semantics of the domain to observations so they can be reused in web applications and mashups, and on utilising the semantics of the domain model to simplify the configuration and deployment of services.

III. API Approach

The following design principles, derived from an evaluation and comparison of REST and Linked Data [15] informs the development of the High-Level API introduced in the next section:

1. Agile development of lightweight mashups is best supported by Resource Oriented service architectures. Complexity for mashup developers is reduced through the simplification of access methods espoused by REST. To develop a good API of this type requires careful identification and design of resources by the service provider.

2. Identification of resources must be undertaken within the context of the domain of the data. Use Domain Driven Design as a flexible and suitable methodology to ensure that the knowledge of domain experts is drawn upon during an iterative design and development process.

3. Use Semantic Web data structures and ontologies (RDF, RDFS, and OWL) for canonical representations of resources; they share a common architectural heritage that makes them particularly suitable for use with REST. This enables development of a common domain model with self-describing link semantics beyond the relatively simple structures found in traditional REST deployments.

4. Identify resources to support not only the domain model, but also the API and its use. Provide Linked Data through content negotiation and a SPARQL endpoint, but also identify resources to enable RESTful applications where hypertext is the engine of application state.

5. RESTfully provide other representations, derived from the domain model, to enrich the service for easy application development and compatibility with existing tooling, as identified through the Domain Driven Design process.

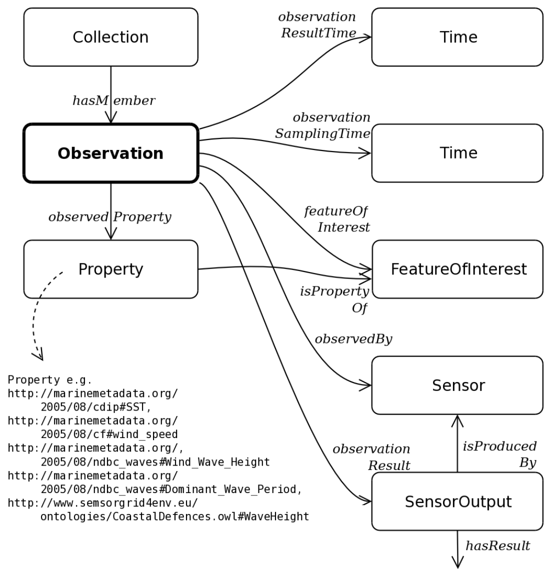

Collection
observation ResultTime
Time
hasMember
Observation
observation SamplingTime
Time
observedProperty
featureOf Interest
Property
FeatureOfInterest
isProperty Of
observedBy
Sensor
observation Result
isProduced By
SensorOutput
hasResult
Property e.g.
http://marinemetadata.org/2005/08/cdip#SST,
http://marinemetadata.org/2005/08/cf#wind_speed
http://marinemetadata.org/,2005/08/ndbc_waves#Wind_Wave_Height
http://marinemetadata.org/2005/08/ndbc_waves#Dominant_Wave_Period,
http://www.semsorgrid4env.eu/ontologies/CoastalDefences.owl#WaveHeight

Figure 1. Observation class from the SSN ontology and associated properties.

IV. Design of a High-level API for Observations

In this section we introduce the design of a High-Level API for Observations, suitable for use by lightweight semantic web clients. The API design is illustrated in the context of the Channel Coastal Observatory (CCO) data management centre, which provides real-time data from the largest network of coastal sensors in the UK. An earlier design following the same basic structure is described in more detail by Page et al. [8]; here we focus on those elements that reflect the principles outlined in Section III, particularly the use of a common semantics to link the domain model with the API.

A. Domain Model

While a provider-centric approach to sensor data might be applicable to provisioning, deploying, and managing sensor network devices, developers of lightweight web applications (and their users) engage in activities which instead require manipulation of the *data* collected by the sensor network. We therefore adopt the data-(consumer-)centric approach introduced in the Observations and Measurements (O&M) model, and represented by the Observation class and associated properties in the SSN Ontology (Figure 1).

Note, in particular, observedProperty and featureOfInterest, which provide the crucial link between the observations and the domain concepts being observed (and therefore links to domain resources beyond the API).

B. Resources

As a RESTful Linked Data system, the high-level API is defined by its resources and the representations of those resources – in this case by observations of the phenomena measured by the CCO. In defining a resource we must create a globally unique identifier for it – a URI – in an act frequently referred to as “minting”. The URI for the resource should be treated as an opaque string when it is accessed through the API; while the implied structure within the URI is helpful when designing and maintaining the web service (and perhaps for developers when writing clients), client applications must navigate to and between resources using links. Use of the API must not rely on encoding semantics within the URI – this clearly violates REST principles.

Identification and structuring of resources can be highly dependent on the data (the resources) being exposed. For example, primary observation resources for the CCO are of the form:

http://id.semsorgrid.ecs.soton.ac.uk/observations/cco/boscombe/Hs/20110101#140500

where the individual observation is dereferenced by first retrieving the resource (which is an observation collection):

http://id.semsorgrid.ecs.soton.ac.uk/observations/cco/boscombe/Hs/20110101

In this case, the observation of wave height (Hs) made by the Boscombe sensor on 01/01/2011 at 2.05pm is asserted within an observation collection of all wave height measurements from the Boscombe sensor on 01/01/2011. This strikes a balance between the size of the retrieved resource representation and the number of links the client must retrieve for this specific data set. In this case the observations of Hs at the Boscombe sensor are taken half-hourly, so the resource that must be retrieved to dereference any one observation will contain 48 observations (all the observations for the day). This grouping of resources (and the associated dereferencing) would not be as practical if there were many more observations per second.

Once more, note that while semantics have been used to structure the minting of the URIs (by our design), they are not exposed through, or necessary for the operation of, the API. Relationships between resources must be expressed in the representations returned by the API, not within the syntax of the URI. A key principle of this Resource Oriented approach is to link to other resources within the API, and to external resources with compatible semantic representations.

Although there must be a canonical statement of an Observation, this does not preclude its inclusion in other observation collections – an RDF model can be declared and reused across several resources by linking between statements. A collection of all measurements of wave height across the sensor network on 01/01/2011 might be identified by the resource:

http://id.semsorgrid.ecs.soton.ac.uk/observations/Hs/all/20110101

and the earlier canonical observation would be linked by reference.

C. Representations

For each non-information observation resource (such as the wave height observation introduced above) the API provides several representations through a common information resource, and further (non-common) representations through a separate information resource, e.g. for backwards compatibility with existing GIS systems. This is implemented through

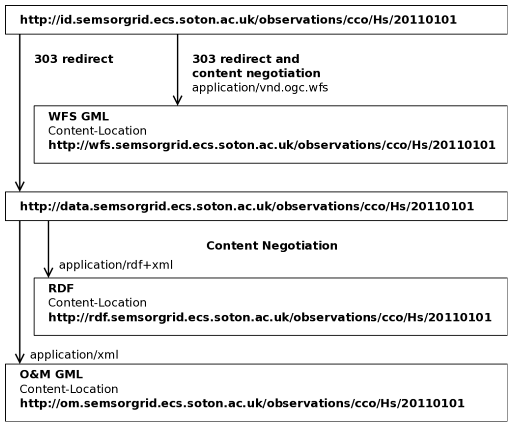

http://id.semsorgrid.ecs.soton.ac.uk/observations/cco/Hs/20110101
303 redirect
303 redirect and content negotiation
application/vnd.ogc.wfs
WFS GML
Content-Location
http://wfs.semsorgrid.ecs.soton.ac.uk/observations/cco/Hs/20110101
http://data.semsorgrid.ecs.soton.ac.uk/observations/cco/Hs/20110101
Content Negotiation
application/rdf+xml
RDF
Content-Location
http://rdf.semsorgrid.ecs.soton.ac.uk/observations/cco/Hs/20110101
application/xml
O&M GML
Content-Location
http://om.semsorgrid.ecs.soton.ac.uk/observations/cco/Hs/20110101

Figure 2. Dereferencing an Observation resource.

linked data dereferencing and content negotiation as illustrated in Figure 2.

The primary representation is RDF/XML using the observations ontology; this representation is also added to a triplestore for provision of a SPARQL endpoint. The second representation conforms to the O&M GML schema. While the XML returned is very similar to that provided by the SOS GetObservation function, here we support a RESTful interaction by navigation between resources. This is made possible by the extensive support for XLink in GML and an underlying object-property-value model which closely resembles RDF. The third representation is in HTML and is a human browsable hyperlinked interface to the observation resources.

The fourth representation conforms to the WFS GML schema (XML). This representation provides compatibility with existing web GIS mapping tools (e.g. OpenLayers). The nature of these tools requires all the grouped data points (observations) to be provided in a single "layer" which can be overlaid onto a map; this flattened data structure is incompatible with a the other representations so is provided as a separate information resource. The fifth representation, GeoJSON, is widely accepted by client side mapping tools and widgets.

Since non-information resources are shared a client application can move seamlessly between RDF and GML representations, taking advantage of the semantic linking provided, while being able to retrieve established encodings for Web GIS applications when required. Conversely an application can use a GML identifier as a jumping off point into the linked data web.

D. Web API

The domain model (in RDF), resources, and representations form the core of the High-Level API for Observations as applied to the CCO. Thus far this presents a Linked Data interface with additional representations specific to the domain, but only exposes resources as structured by the domain model. To complete the API, attention should be paid to resources which might aid RESTful client use (when changing application state by navigation between resources). The following additional resources and links are included, and as with all resources provided through the API semantic consistency is maintained through the common domain model:

- /latest : relative within each observation collection, a resource that is always the most recent observation.
- "next" and "previous" : for each observation, a reference to the prior and following observations of that class.
- links from constituent collections to the broader collections ("up"; isMemberOf in RDF) to enable better navigation through the data.
- /summary – for each observation collection, a summary resource containing information about that collection, e.g. maximum/minimum values, frequency, averages, units of measurements, and descriptive metadata (this can be used by clients to calibrate visualisations and provide annotations).
- /sensors – a collection of links to all sensors that generate observations.
- For each sensor resource, links to the temporally broadest observation collections generated by that sensor platform.

V. HLAPIO Service Implementation

A. Overview of Design

While an earlier proof-of-concept [8] validated a limited prototype of the RESTful Linked Observation API, the implementation was limited. A bespoke instantiation for a specific data set, built upon a web platform originating in OGC standards and services, the software proved brittle and difficult to maintain. Iterative modification of the domain model and REST interfaces to support evolving user requirements frequently introduced errors in derivative representations, while more wide-ranging changes – for example deployment over a different dataset – were tantamount to re-implementation. The service also lacked a Linked Data query interface.

To address these shortcomings, and to assist the design and deployment of domain-driven semantic APIs (Section III) a new service platform has been developed – the High-Level API for Observations (HLAPIO).

The HLAPIO provides a flexible implementation of the API design introduced in Section IV such than an installation can offer web application developers RESTful access to linked sensor data sourced from a wide variety of sensor networks. By focussing on a core underlying semantic data structure – the Observation model – the HLAPIO realises a domain-driven design for this API and extends this approach into the design and configuration of the service software.

The overall composition of the HLAPIO is shown in Figure 3. The core of the service is implemented in Java, using Jena[1] RDF models throughout, and serialising to an Apache web server with additional configuration and scripting for content negotiation and Linked Data dereferencing. 4Store[2] is used to provide the SPARQL endpoint. The design builds upon on the semantics of four sections of functionality within the service and the relationships between them:

[1]http://jena.sourceforge.net/
[2]http://4store.org/

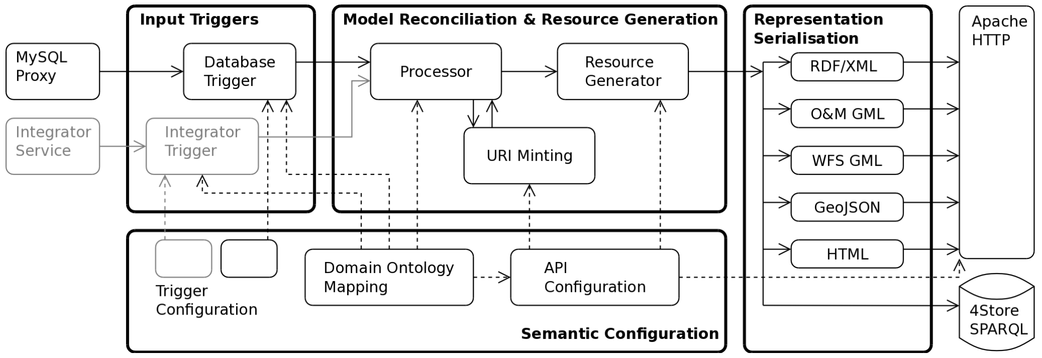


Figure 3. General structure of the HLAPIO platform with reference implementation triggers and serialisations.

1) The Observation model, a semantic backbone to the operations of the HLAPIO, hard-coded into the operations of the software and through which other data models are created and referenced. As data is ingested into the HLAPIO it is reconciled with the observation and domain models to produce the resources which comprise the API.
2) The extended domain model, which encodes the specifics of data from the particular sensor network over which the HLAPIO is deployed. This includes observed properties and sensors plus links to data sources describing these (e.g. to environmental ontologies, location information).
3) Import mechanisms through which the HLAPIO gathers Observations, covering both the software interface and the mapping of data from the input trigger to the extended domain model and Observations. The current implementation supports two of these input triggers.
4) The identification and generation of resources that constitute the API following the general design principles introduced in Section IV and using terms from the Observation and extended domain models.

These semantics are tailored to each installation through *configuration*, where each of model is aligned with a clear domain of user knowledge: the subject expert (for the extended domain model), the sensor network administrator (import trigger configuration), and the service administrator (API configuration). This separation of concerns enables a focus on each of the distinct issues at hand and modular re-use of configuration.

The remainder of this section describes implementation details of the domain and API configuration, and their application within the HLAPIO engine.

B. Semantic Configuration

The mechanism for using the domain model to configure a HLAPIO instance is centred around two files: the Domain Ontology Mapping and the API Configuration.

1) Domain Ontology Mapping: The domain ontology mapping translates between input data sources, the Observation model, domain ontologies, and terms associated with the observation (e.g. observed properties). It does so by mapping input data features (for databases: row data, table names, column headings) onto classes that form part of an observation. The mapping syntax is an extension of the D2RQ [16] language using the Turtle RDF notation.

```
map:Folkestone_met_Air_pressure a d2rq:ClassMap;
d2rq:dataStorage map:database;
d2rq:class obs:observation;
d2rq:classDefinitionLabel "observation";
.
map:Folkestone_met_Air_pressure_dateTimeBeginning a d2rq:PropertyBridge;
d2rq:belongsToClassMap map:Folkestone_met_Air_pressure;
d2rq:property time:hasBeginning;
d2rq:belongsToPropertyBridge map:Folkestone_met_Air_pressure_timeInterval;
d2rq:timestamp "envdata_Folkestone_met.0";
.
map:Folkestone_met_Air_pressure_observedProperty a d2rq:PropertyBridge;
d2rq:belongsToClassMap map:Folkestone_met_Air_pressure;
d2rq:property obs:observedProperty;
d2rq:columnHeading "envdata_Folkestone_met.5";
d2rq:substitute "http://marinemetadata.org/2005/08/cf#air_pressure_at_sea_level";

.
```

Figure 4. Excerpts from a Domain Ontology Mapping file.

In the these files, ClassMap objects are used to represent classes from the target ontology. For each ClassMap, a number of PropertyBridges are used to express ontology properties of the classes the ClassMap represents. In the example shown in Figure 4, the d2rq:property predicate shows that the PropertyBridge represents the time:hasBeginning property of its parent ClassMap, while the d2rq:timestamp variable informs the HLAPIO engine how to construct the value of this property.

While the mapping file format is based on the D2RQ syntax, several additions have been made. Typically D2RQ is used to create RDF that reflects the structure of the originating database, however the HLAPIO must be capable of mapping arbitrary sensor data sources into a fixed Observation structure. Most sensor networks are engineered around producer-centric assumptions; the HLAPIO converts these into consumer-centric resources. For example, a database row might contain readings for several different observed properties which the HLAPIO must map into multiple instances of the Observation class; this cannot be declared using standard D2RQ syntax. This is overcome by allowing the HLAPIO engine to process different ClassMaps to the same ontology class gracefully, and through the addition of the belongsToPropertyBridge property which allows assertion of relational semantics that are not necessarily expressed by the database structure (example in Figure 4).

The event driven nature of sensor readings, where each row should be processed on arrival, leads to further modification.

```
[printsubs]
http://id.semsorgrid.ecs.soton.ac.uk/sensors/cco/folkestone_met=folkestone_met
[global]
hostname="@@FORMAT.semsorgrid.ecs.soton.ac.uk"
NIR="FORMAT=id"
application/rdf+xml="FORMAT=rdf"

[canonicalValues]
type=canonical
path="/observations/cco/[obs:observedBy]/[obs:observedProperty]/
   [obs:observationResultTime{yyyyMMdd}]#[obs:observationResultTime{HHmmss}]"
formats="application/rdf+xml,application/xml,application/vnd.ogc.wfs,text/html,
   application/json"
[eachSensorAllProps]
type=collection
path="/observations/cco/[obs:observedBy]/all/
   [obs:observationResultTime{yyyyMMddHH}]::(perDay)"
formats="application/rdf+xml,application/xml,application/vnd.ogc.wfs,text/html,
   application/json"
[perDay]
type=metacollection
pathSub="[obs:observationResultTime{yyyyMMdd}]"

formats="application/rdf+xml,application/xml,text/html"
```

Figure 5. Excerpts from an API Configuration file.

The columnHeading and tableName properties enable the HLAPIO engine to match column headings and table names, which in many cases encode significant semantic information (e.g. sensor name/location). When such values are matched, the d2rq:substitute variable enables a second value (typically a URI) to be inserted into the observation model. Finally, the timestamp and alteredtimestamp variables instruct the HLAPIO processor to translate e.g. Unix timestamps values (often used in sensor records) into the XML dateFormat used by the Observation model (Figure 4).

2) API Configuration: While the Domain Ontology Mapping provides the semantic structure that enables the HLAPIO to manipulate data according to the domain model, the Observations created must also be granted identifiers (URIs) before being published as Linked Data. To provide a useful API to developers, the HLAPIO must identify which Observations should be included in *collections*, create URIs for these collection resources, and provide linking between resources to enable RESTful state transitions for clients (Section IV). Finally, the HLAPIO must know which resources should be made available in which representations.

The API Configuration file encodes the necessary information to achieve this by way of URI structures for the resources, collections, and representations that are to be published. Variable substitution uses terms from the domain model (via the Domain Ontology Mapping) – the domain model and API are inextricably linked. In this way the API Configuration directly encodes semantics in the URI string which is, of course, counter to both Linked Data principles and RESTful design (Section III). It is important to note, therefore, that while domain model semantics are used to create ("mint") the URIs and construct the resource representations within the HLAPIO server, clients of the API do not make use of any residual semantics that can be inferred from the URIs – clients receive domain semantics and state transitions through the representations dereferenced and retrieved from the URIs. This separation could be completely enforced through obfuscation of the URIs after use by the HLAPIO engine, but before publication through the API, however this seems an necessary removal of what can be a useful debugging reference for client developers.

The syntax of an API Configuration, illustrated in Figure 5, is as follows:

– a [printsubs] section containing short sub-strings to replace full URIs when a domain term is included within a URI.

– a [global] section, defining the hostname component of URIs and the sub-string replacement to be used for each representation. The hostname may be a simple string (i.e. identical for all URIs provided by the HLAPIO) or include substitutions. In this example the @@FORMAT substitution indicates that the hostname component of the URI will be used to distinguish between representations (e.g. rdf.semsorgrid.ecs.soton.ac.uk). This substitution could equally be placed within the path. The *NIR* format is a reserved configuration label used for minting the canonical non-information URI for each resource.

– an unlimited number of named *resource set* sections, each instructing the HLAPIO to generate resources for the API, and consisting of three resource types: observation resources (type=canonical), observation collections containing observation members (=collections) and observation collections with members consisting of other observation collections (=metacollections).

– each resource set includes a path for that resource (appended to the hostname to create a complete URI) which can contain substitutions for terms, signified by square brackets, retrieved from the observation model (e.g. obs:observedProperty to substitute available observed properties).

– each resource set includes the representation formats which should be generated for instances of that resource.

– temporal properties (e.g. result time, sample time) can be further sub-matched using ISO 8601 syntax in curled braces.

– metacollections (indicated by ::() syntax) instruct the HLAPIO to create secondary derivative Observation collections according to the named metacollection pattern referenced. e.g. where a collection groups Observations by hour, a metacollection can then be used to group these hourly collections by day.

The eachSensorAllProps resource set in Figure 5 would, for example, generate an hourly observation collection containing all observations for all observed properties from each distinct known sensor (obs:observedBy) with URIs of the form:

http://id.semsorgrid.ecs.soton.ac.uk/observations/cco/boscbome/all/2011021415

and metacollections grouping these collections, by day, with a smaller number of representations, at:

http://id.semsorgrid.ecs.soton.ac.uk/observations/cco/boscbome/all/20110214

C. Service Engine

The core of the HLAPIO service engine constructs and manipulates generic observation graphs. This is supplemented by Input Triggers and Serialisers to handle specific data sources and output representations respectively.

1) Input Triggers: To support the continuous flows of data typical of sensor networks the HLAPIO implements an event driven design which, by default, processes incoming measurements into observations as and when they are received. Data sources interface with the HLAPIO through *Input Triggers*, which convert streams into RDF graphs using the observation and domain model ontologies. The modular design of the service allows for any number of input trigger implementations; two have been developed and deployed for specific use cases thus far.

The first input trigger reference implementation ingests from database sources and was originally developed to gather readings from the CCO sensor network. The CCO stored data service records various observed properties (e.g. wave height, wind speed, air temperature) from a number of different sensing platforms. A separate database table represents each sensing platform, and each row of a table contains the set of all measurements recorded by the sensing platform at a given time. The HLAPIO and its trigger are installed in parallel to the existing database using a MySQL proxy. As an INSERT statement is sent to the database from the sensor network, it is caught by MySQL Proxy, and in turn forwarded to a TriggerHandler which splits multiple inserts before sending each one to the DBInsertTrigger. For each extracted column value, the DBInsertTrigger checks the Domain Ontology Mapping file to determine whether that value is recognised by the HLAPIO. If a match is found, the ClassMap is added to the list of observations to be built and, once the table name and all column values have been checked, the list of applicable ClassMaps is sent to the Processor to be turned into an observation RDF graph.

The second reference implementation demonstrates a method for receiving observations generated by other semantic components external to the HLAPIO. A Semantic Integration Service [17] is accessed through the SemSorGrid4Env reference architecture [18] which, based upon a query, combines multiple sensor data sources into a single observation structure, with a mapping document aligned to the HLAPIO Domain Ontology. The resulting graphs are passed by the IntegratorTrigger to the HLAPIO processor.

2) Model Reconciliation and Resource Generation: The core of the service engine performs three tasks, semantically aligning the input data and configurations, then preparing the data structures for publication through the API:

Model Reconciliation. Graphs received from the Input Triggers are reconciled with Observation instances. RDF is used as the principle internal representation as it is the most flexible and fully featured data model; all other output representations are derived from RDF at the point of serialisation. All serialised representations are semantically consistent, with any changes to the underlying RDF model continuing to generate valid secondary representations. Properties and their corresponding values are retrieved from the incoming graphs and asserted against the Observation; temporal values are adjusted according to any timestamp or alteredtimestamp modifiers. In preparation for WFS serialisation co-ordinates are retrieved by dereferencing the sensor URI asserted by the observedBy property.

URI Minting. URIs are minted for triples that have been created afresh by the HLAPIO (i.e. not properties from the Domain Ontology Map, values, or links to external resources). Since these URIs also form part of the Linked Data interface of the API, they are matched within the API Configuration file and string substitutions are made to ensure uniqueness of the Observation resources. Temporal properties of the observation are transposed into the ISO 8601 format specified in the API Configuration.

Resource Generation. Minting URIs completes a non-information Observation resource, ready to form a part of the API. The HLAPIO must also identify secondary resources: Observation Collections of which the Observation is a constituent and information resources for the secondary representations. Once again these are generated by substituting the Observation properties into the API Configuration structure and generating resources for those that match. Whenever collections are generated links to enable RESTful navigation are also added (precedes, follows, and isMemberOf).

3) Representation Serialisation: Five reference serialisers are provided by the HLAPIO implementation for five representations: RDF, O&M GML, GeoJSON, WFS GML, and HTML. For each resource identified and generated in the previous step, a serialised resource is constructed (for canonical observations) or updated (for collections and metacollections) using the URI minted according to the API Configuration file. The principle internal RDF representation is serialised first, then re-used to generate the other output formats appropriately. The RDF representation is also added to the 4Store triplestore which provides a SPARQL endpoint. As described in Section IV, while some representations share a common information resource with RDF, others require additional dereferencing of linked resources to create a 'flattened' representation.

The serialisations are served by an Apache web server. Dereferencing and content negotiation is implemented using PHP scripting automatically configured by the API Configuration; this also handles the "/lastest" pointer for RESTful navigation.

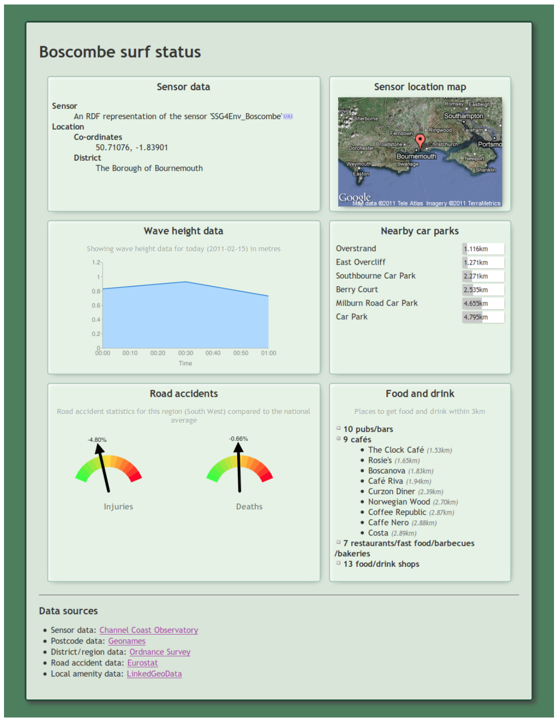
VI. Example mashup use

While the deployment of an HLAPIO for the CCO sensor network has provided observation data for GIS-sytle use cases (e.g. flooding emergency response [18]), it is unintended re-use of semantically enriched data in lightweight web applications that is a strong motivation for developing and deploying the HLAPIO.

The CCO network includes a sensor near the town of Boscombe, Dorset, which is the location of the first artificial surf reef to be installed in the Northern Hemisphere[3]. Figure 6 shows an illustrative mashup, combining linked data sources and wave height data from the CCO HLAPIO to provide a useful status page for surfers travelling to the reef, where:

- the current wave height (Hs) reading for the Boscombe sensor is obtained from the HLAPIO using the "/latest" pointer; historical wave heights are retrieved by RESTfully navigating to preceding observations.

[3]http://en.wikipedia.org/wiki/Boscombe_Surf_Reef


Boscombe surf status

Sensor data

Sensor
An RDF representation of the sensor 'SSG4Env_Boscombe' URI
Location
Co-ordinates
50.71076, -1.83901
District
The Borough of Bournemouth

Sensor location map

Wave height data

Showing wave height data for today (2011-02-15) in metres

Nearby car parks

Overstrand	1.116km
East Overcliff	1.271km
Southbourne Car Park	2.271km
Berry Court	2.535km
Milburn Road Car Park	4.655km
Car Park	4.795km

Road accidents

Road accident statistics for this region (South West) compared to the national average

Food and drink

Places to get food and drink within 3km

- **10 pubs/bars**
- **9 cafés**
  - The Clock Café *(1.53km)*
  - Rosie's *(1.65km)*
  - Boscanova *(1.83km)*
  - Café Riva *(1.94km)*
  - Curzon Diner *(2.39km)*
  - Norwegian Wood *(2.70km)*
  - Coffee Republic *(2.87km)*
  - Caffe Nero *(2.88km)*
  - Costa *(2.89km)*
- **7 restaurants/fast food/barbecues /bakeries**
- **13 food/drink shops**

Data sources

- Sensor data: Channel Coast Observatory
- Postcode data: Geonames
- District/region data: Ordnance Survey
- Road accident data: Eurostat
- Local amenity data: LinkedGeoData

Figure 6. An illustrative mashup comprising wave height observations and linked data sources for surfers.

- a SPARQL query is used to find other sensor resources that can provide wave height observations.
- the Boscombe sensor RDF representation (via observedBy) is dereferenced to provide location information.
- linked data is then retrieved, through postcodes[4] and district information[5], for local facilities[6] (car parking, food & drink) and road safety[7].

VII. Summary and Future Work

Development of the High-Level API for Observations has demonstrated how a common semantic domain model can be applied to support two symbiotic use-cases: the simplified configuration and deployment of lightweight, self-descriptive APIs over varied sources of sensor data; and the rapid development of novel web applications that re-use and combine data in new and previously unforeseen ways. Semantic structures, representative of the use-case domain, are key to development of successful services and applications. Once established, RDF and ontologies can be used to apply a common model to both to provision the service and, through the API, drive application state: while made explicit in the API, this domain-driven design runs deep in the design of both server and application software.

Future work might involve further automation and assistance through use of data semantics: a frequency analysis of data sources using the Domain Ontology Mapping may produce default API configurations; a graphical representation of domain ontologies could form the basis of visual tools for creating the Domain Ontology Mapping.

As seen in Section VI, the utility of the Semantic Web for writing mashups is dependent on the availability of links between resources and their domain models. In this paper we have demonstrated how linked sensor data and domain models can be published using the Observation pattern; the same approach might be successfully applied to comparable models outside of sensor networks research.

[4]http://publishmydata.com/datasets/postcodes
[5]http://api.talis.com/stores/ordnance-survey/services/sparql
[6]http://linkedgeodata.org/sparql/
[7]http://www4.wiwiss.fu-berlin.de/eurostat/sparql

Acknowledgements

Thanks to our colleagues in the GeoData Institute, University of Southampton, for the help developing early prototype implementations of the API and access to the CCO data feeds. The work in this paper was supported by the IST STREP Programme of the Commission of the European Communities as project number FP7-223913 "SemSorGrid4Env: Semantic Sensor Grids for Rapid Application Development for Environmental Management".

References

[1] O. Corcho and R. García-Castro, "Five challenges for the semantic sensor web," *Semantic Web Journal*, vol. 1, no. 1-2, pp. 121–125, 2010.
[2] C. Bizer, T. Heath, and T. Berners-Lee, "Linked Data - The Story So Far," *Special Issue on Linked Data, International Journal on Semantic Web and Information Systems (IJSWIS)*, 2009.
[3] L. Sauermann and R. Cyganiak, "Cool URIs for the Semantic Web," W3C SWEO Interest Group Note, 2008.
[4] T. Berners-Lee, "Linked Data, Design Issues," http://www.w3.org/DesignIssues/LinkedData.html, 2006.
[5] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, Information and Computer Science, University of California, Irvine, California, USA, 2000.
[6] S. Cox, "Observations and Measurements - Part 1 - Observation schema (OpenGIS Implementation Standard OGC 07-022r1)," Open Geospatial Consortium Inc., Tech. Rep., 8 Dec. 2007.
[7] L. Lefort and C. Henson, "Semantic sensor network incubator report," "http://www.w3.org/2005/Incubator/ssn/wiki/Incubator_Report", October 2010.
[8] K. R. Page *et al.*, "Linked sensor data: Restfully serving rdf and gml," in *Semantic Sensor Networks 2009 (SSN09)*, October 2009.
[9] H. Patni, C. Henson, and A. Sheth, "Linked sensor data," in *2010 International Symposium on Collaborative Technologies and Systems (CTS)*. IEEE, 2010, pp. 362–370.
[10] K. Janowicz *et al.*, "Towards meaningful uris for linked sensor data," in *Towards Digital Earth: Search, Discover and Share Geospatial Data. Workshop at Future Internet Symposium*, 2010.
[11] K. Janowicz and M. Compton, "The stimulus-sensor-observation ontology design pattern and its integration into the semantic sensor network ontology," in *3rd International workshop on Semantic Sensor Networks (SSN10)*, 2010.
[12] P. Barnaghi, M. Presser, and K. Moessner, "Publishing linked sensor data," in *3rd International Workshop on Semantic Sensor Networks (SSN10)*, 2010.
[13] J. Pschorr *et al.*, "Sensor discovery on linked data," in *7th Extended Semantic Web Conference (ESWC2010)*, 2010.
[14] D. Le-Phuoc and M. Hauswirth, "Linked open data in sensor data mashups," in *Semantic Sensor Networks 2009 (SSN09)*, October 2009.
[15] K. R. Page, D. C. De Roure, and K. Martinez, "REST and Linked Data: a match made for domain driven development?" in *Second International Workshop on RESTful Design*, March 2011.
[16] C. Bizer and A. Seaborne, "D2rq-treating non-rdf databases as virtual rdf graphs," in *3rd International Semantic Web Conference (ISWC2004)*, Nov. 2004.
[17] J. Calbimonte, O. Corcho, and A. Gray, "Enabling ontology-based access to streaming data sources," *9th International Semantic Web Conference (ISWC2010)*, pp. 96–111, 2010.
[18] A. J. G. Gray *et al.*, "A Semantically Enabled Service Architecture for Mashups over Streaming and Stored Data," in *9th Extended Semantic Web Conference (ESWC2011)*, May 2011.