

# From RESTful to SPARQL: A Case Study on Generating Semantic Sensor Data

Heiko Müller, Liliana Cabral, Ahsan Morshed, and Yanfeng Shu

Intelligent Sensing and Systems Laboratory, CSIRO, Hobart, Australia  
{heiko.mueller, liliana.silvacabral, ahsan.morshed, yanfeng.shu}@csiro.au

**Abstract.** The recent years have seen a vast increase in the amount of environmental sensor data that is being published on the web. Semantic enrichment of sensor data addresses the problems of (re-)use, integration, and discovery. A critical issue is how to generate semantic sensor data from existing data sources. In this paper, we present our approach to semantically augment an existing sensor data infrastructure, in which data is published via a RESTful API as inter-linked JSON documents. In particular, we describe and discuss the use of ontologies and the design and development of SERAW, a system that transforms a set of JSON documents into an RDF graph augmented with links to other resources in the Linked Open Data cloud. This transformation is based on user-provided mappings and supported by a library of purpose-built functions. We discuss lessons learned during development and outline remaining open problems.

## 1 Introduction

Environmental sensors play an important role in applications such as weather forecasting, bush fire monitoring, and irrigation scheduling. Recent years have seen a vast increase in the amount of environmental sensor data that is being published on the web. To make the most of the available data, it is important to enhance our ability to discover, integrate, and contextualise the data. Standardization efforts like the Open Geospatial Consortium (OGC) Sensor Web Enablement (SWE) [3], a collection of web service interfaces and data encodings for interoperable data access, are a first step in this direction. Semantic enrichment of sensor data, referred to as *semantic sensor web* [14], *linked sensor data* [11,2,7], or *linked stream data* [13,8], further improve (re-)use, integration, and discovery of the available data. The main ideas are to use ontologies for semantic description of data, and to publish data following linked data principles [6].

The Commonwealth Scientific and Industrial Research Organisation (CSIRO) collects and archives data from a large number of terrestrial and marine sensors that measure parameters such as rainfall, temperature, and salinity. Parts of the data are made available via an in-house sensor data infrastructure, referred to as *Sensor Cloud*. One of the main components of this infrastructure is a Web API implemented following RESTful principles [5]. This RESTful API, referred to as

SC-API in this paper, uses HTTP requests to access and manipulate information about resources such as sensing devices, sensor calibrations, and sensor observations. Data is published as inter-linked documents in JSON format <sup>1</sup>. Publishing data as JSON documents has proved to have a positive impact on development of applications that access sensor data. However, a lack of well-defined semantics and limited query capabilities hamper our ability to discover data in the Sensor Cloud that is fit for particular purposes. A typical query that currently cannot be answered automatically is a search for all rainfall sensors within a specific region that have been calibrated in the past six months. The Sensor Cloud contains the necessary information to answer this query, i. e., location, observed phenomena, and calibration history. This information, however, is up to users' interpretation. Furthermore, the information is spread across different documents, which means that, to answer the query, a user has to manually browse documents by following links between them or write a dedicated piece of software to do so automatically.

To overcome these limitations, we propose a conceptual framework to semantically augment the Sensor Cloud. Generating semantic sensor data from our existing Sensor Cloud infrastructure involves several steps. First, we model sensor data through the Sensor Cloud Ontology (SCO), which extends existing ontologies including the Semantic Sensor Network Ontology (SSNO) [15], and aligns with standardised vocabularies to represent common entities such as units of measurement and observed phenomena. We then construct mappings that describe document elements in terms of concepts and properties of SCO. These mappings form the basis for transforming JSON documents into RDF. The whole mapping and translation process is done via SERAW (Semantic RESTful API Wrapper). SERAW crawls documents accessible through the SC-API by following links between them. The set of documents is transformed into a single RDF graph. We further enhance the resulting RDF graph with links to resources in the Linked Open Data cloud (LOD) by using a set of purpose-built functions. As a result, we obtain a semantically-enriched dataset and have the full query capability provided by SPARQL [12].

In this paper we discuss the main components of our framework to semantically augment the Sensor Cloud, named *Semantic Sensor Cloud* and in this context we present our approach to generate RDF from JSON, implemented in SERAW. Given that there is a widespread use of APIs that serve data as JSON documents (e. g., geospatial data <sup>2</sup>), we believe that this approach can be generalised to other applications e. g., the API provided by the centralised sensor data repository *Xively* <sup>3</sup>.

In summary, the main contributions of this paper are:

- We create an ontology for the Sensor Cloud, which extends standard ontologies to model environmental sensor data.

<sup>1</sup> <http://www.json.org>

<sup>2</sup> <http://www.opengeospatial.org/projects/groups/gservrestswg>

<sup>3</sup> <https://xively.com>

- We develop SERAW, a system that transforms a set of inter-linked JSON documents into an RDF graph based on mappings between document elements and concepts and properties in our ontology.
- We enhance the generated RDF graph with links to resources in the LOD cloud through purpose-built functions.

The remainder of this paper is structured as follows: We start by describing our conceptual framework in Section 2. We present our ontology to model Sensor Cloud data in Section 3. In Section 4, we describe our mechanism for transforming a set of JSON documents into RDF. Implementation details are given in Section 5. Section 6 outlines how we augment the resulting RDF graph with external information. We discuss related work in Section 7. Section 8 concludes the paper and gives an outlook into future work.

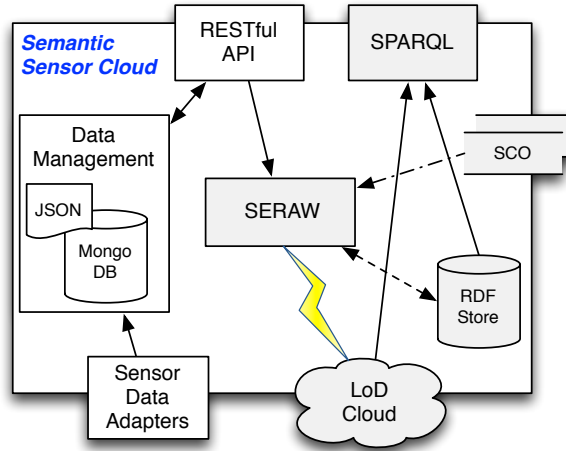
## 2 The Semantic Sensor Cloud

In this section, we describe the conceptual framework for the Semantic Sensor Cloud, which integrates semantics into the existing Sensor Cloud. Figure 1 shows the main components of the framework, where the components on the right side (shaded in gray colour) belong to the semantic enrichment part, while the components on the left belong to the Sensor Cloud. Following the layered architecture for web accessible sensor data in [4], the Semantic Sensor Cloud can be seen as a middleware at the sensor web layer between the application layer and the sensor network layer, managing the heterogeneity of sensor resources and making them usable at the application level. In particular, the components at the top (i. e., RESTful API and SPARQL) interact with applications, and the components at the bottom (i.e. sensor data adaptors and LOD cloud) interact with external sources of sensor data.

The Sensor Cloud is a sensor data infrastructure developed within CSIRO. The main goal is to provide a centralised point of access for environmental sensor data collected from various sensor networks in the organisation. Scalability and support for application development were the main drivers that influenced design decisions. The infrastructure consists of a number of components. A detailed description of the infrastructure is beyond the scope of this paper. Here, we only give a brief description of the main components. The *RESTful API* (SC-API) provides data access and manipulation. The *data management* component addresses how data is structured, identified, and partitioned into documents. *Sensor Data Adapters* transform data from external sources into the JSON format. JSON possesses several benefits for application development, especially when developing web applications using JavaScript. MongoDB <sup>4</sup> is used as the data storage backend, primarily for its flexibility regarding data structure.

Within the infrastructure, sensor data is virtually structured hierarchically, following *Network*  $\rightarrow$  *Platform*  $\rightarrow$  *Sensor*  $\rightarrow$  *Phenomenon*  $\rightarrow$  *Observation*. This hierarchical structure resembles the deployment of sensors in the physical

<sup>4</sup> <http://www.mongodb.org>



**Fig. 1.** The Semantic Sensor Cloud Conceptual Framework

world. Each sensor network consists of platforms, with each platform having one or more sensors attached; each sensor observes one or more phenomena. Besides observations, information such as sensor device characteristics and calibration history can also be accessed, which in the infrastructure is currently represented using a JSON profile of StarFL [9].

The hierarchical structure of the data is reflected by the URLs used to access data. Networks, platforms, sensors, and phenomena have names to identify them. These names form a relative key structure. That is, the name of a network uniquely identifies it in the infrastructure. The name of a platform is unique among other platforms in the same network. There may, however, be a platform with the same name allocated to a different network. URLs are currently the only way for external applications to access documents in the Sensor Cloud. An example for URL [http://www.sense-t.csiro.au/sensorcloud/v1/network/TIA/platform/Tamar\\_Ridge/sensor/RIMCO\\_7499/phenomenon/rainfall](http://www.sense-t.csiro.au/sensorcloud/v1/network/TIA/platform/Tamar_Ridge/sensor/RIMCO_7499/phenomenon/rainfall) is shown in Figure 2. The returned JSON document refers to the network “TIA”, platform “Tamar\_Ridge”, sensor “RIMCO\_7499”, and phenomenon “rainfall”. The content of the file is determined by the data management component. In this example it shows, among other data, the location of the platform in terms of its coordinates, date and number of observations, and links to the related observations (time series) and to the sensing procedure.

The main goal of our work is to semantically augment the Sensor Cloud with knowledge management components that enable the automated generation of RDF from the sensor data. At the heart of our work is SERAW, a system that transforms documents accessible via the SC-API into RDF. SERAW makes use of an ontology developed as extension of existing ontologies particularly for the Sensor Cloud. By making use of a set of purpose-build functions, SERAW allows



```

{
  platform: "Tamar Ridge",
  sensor: "RIMCO_7499",
  id: "rainfall",
  name: "rainfall",
  - profile: {
    type: "*FL core",
    - platform: {
      - location: {
        longitude: "146.884223",
        latitude: "-41.192966",
        elevation: "0.0"
      },
      summary: "This platform is deployed in a vineyard."
    },
    sensor: { },
    phenomenon: { }
  },
  - observation: {
    oldestDate: "2009-11-21T10:10:51+1100",
    latestDate: "2010-04-26T23:00:51+1000",
    count: 22548
  },
  - observations: {
    href: http://www.sense-t.csiro.au:80/sensorcloud/v1/network/TIA/platform/Tamar_Ridge/sensor/RIMCO_7499/phenomenon/rainfall/observations
  },
  - sensing: {
    href: http://www.sense-t.csiro.au:80/sensorcloud/v1/network/TIA/platform/Tamar_Ridge/sensor/RIMCO_7499/phenomenon/rainfall/sensing
  },
  - parent: {
    href: http://www.sense-t.csiro.au:80/sensorcloud/v1/network/TIA/platform/Tamar_Ridge/sensor/RIMCO_7499/phenomenon
  }
}

```

Fig. 2. Sensor Cloud RESTful API Example

to include references to external resources and information. The generated data is maintained in an RDF triple store and accessible via a SPARQL endpoint. Figure 3 shows a screen shot of the query interface for SERAW. In the following sections, we describe components in the Semantic Sensor Cloud in more detail.

### 3 The Sensor Cloud Ontology

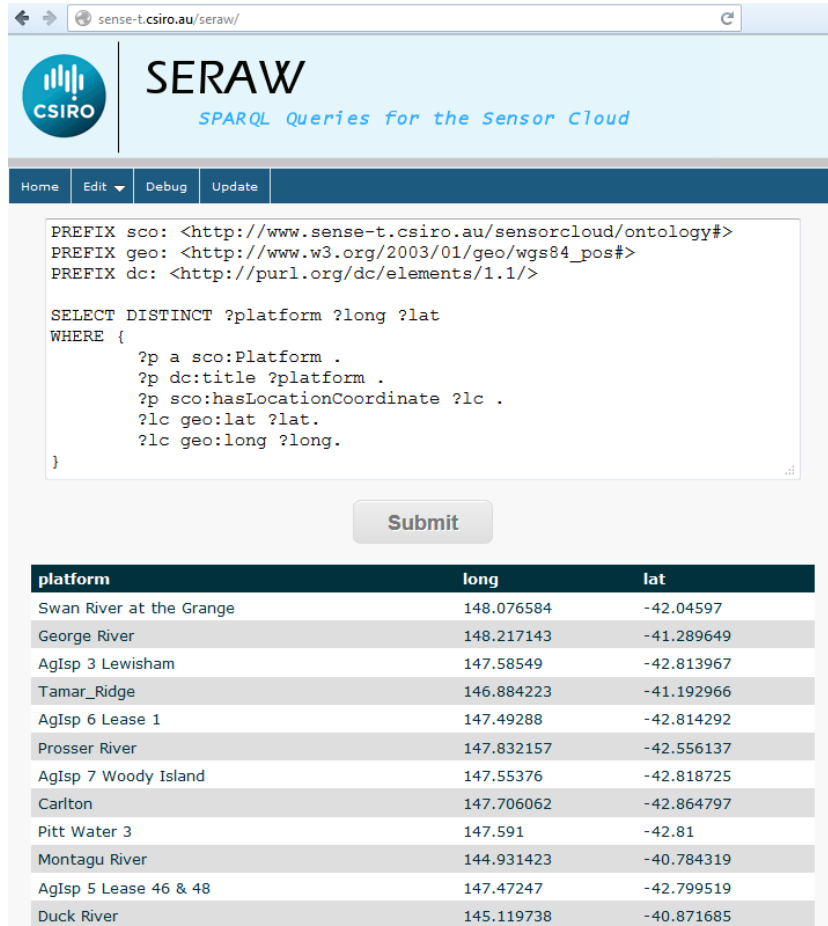
In order to semantically describe sensor data from the Sensor Cloud we created the *Sensor Cloud Ontology* (SCO)<sup>5</sup>. The principle behind the design of the ontology is to use and extend existing ontologies, meanwhile aligning with the Sensor Cloud terminologies. Accordingly, classes and properties were created and mapped to the ones in existing ontologies, as described below. The advantages are that sensor data can be queried according to the original terminologies while their consistency can be checked against SCO. We use the ontology to facilitate syntactic-to-semantic transformation in SERAW (Section 4).

Following best practices, we reused several ontologies, namely, the SSN ontology (SSNO), an ontology for the OGC's Observation and Measurements (OM)<sup>6</sup>, and the Basic Geo (WGS84) (GEO) vocabulary<sup>7</sup>. The main criterion to use these

<sup>5</sup> <http://www.sense-t.csiro.au/sensorcloud/ontology>

<sup>6</sup> <http://def.seegrid.csiro.au/isotc211/iso19156/2011/observation>

<sup>7</sup> [http://www.w3.org/2003/01/geo/wgs84\\_pos](http://www.w3.org/2003/01/geo/wgs84_pos)



PREFIX sco: <http://www.sense-t.csiro.au/sensorcloud/ontology#>  
 PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84\_pos#>  
 PREFIX dc: <http://purl.org/dc/elements/1.1/>

```

SELECT DISTINCT ?platform ?long ?lat
WHERE {
  ?p a sco:Platform .
  ?p dc:title ?platform .
  ?p sco:hasLocationCoordinate ?lc .
  ?lc geo:lat ?lat.
  ?lc geo:long ?long.
}
  
```

Submit

platform	long	lat
Swan River at the Grange	148.076584	-42.04597
George River	148.217143	-41.289649
AgIsp 3 Lewisham	147.58549	-42.813967
Tamar_Ridge	146.884223	-41.192966
AgIsp 6 Lease 1	147.49288	-42.814292
Prosser River	147.832157	-42.556137
AgIsp 7 Woody Island	147.55376	-42.818725
Carlton	147.706062	-42.864797
Pitt Water 3	147.591	-42.81
Montagu River	144.931423	-40.784319
AgIsp 5 Lease 46 & 48	147.47247	-42.799519
Duck River	145.119738	-40.871685

Fig. 3. Query Interface for SERAW

ontologies is that they are derived from standardisation efforts. Figure 4 shows the main classes and properties of SCO, where the prefix of a class, e. g., `ssn` of `ssn:Observation`, indicates the ontology the class comes from.

We extend these ontologies in several ways. In SCO we define `sco:Network`, `sco:Platform`, and `sco:Sensor` to make explicit the fact that networks, platforms, and sensors in the Sensor Cloud are `ssn:Systems` (contrary to the definition of platforms and sensors in SSNO). `sco:Sensor`, for example, is defined as subclass of `ssn:SensingDevice` which in turn is a subclass of `ssn:System`. To describe sensor observations, we introduce concepts `sco:ObservedPhenomenon`, `sco:ObservationResult`, and `sco:TimeSeriesObservedValue` as subclasses of `ssn:Observation`, `ssn:SensorOutput`, and `ssn:ObservationValue`, respectively. These concepts reflect the particular use of terms phenomenon and ob-

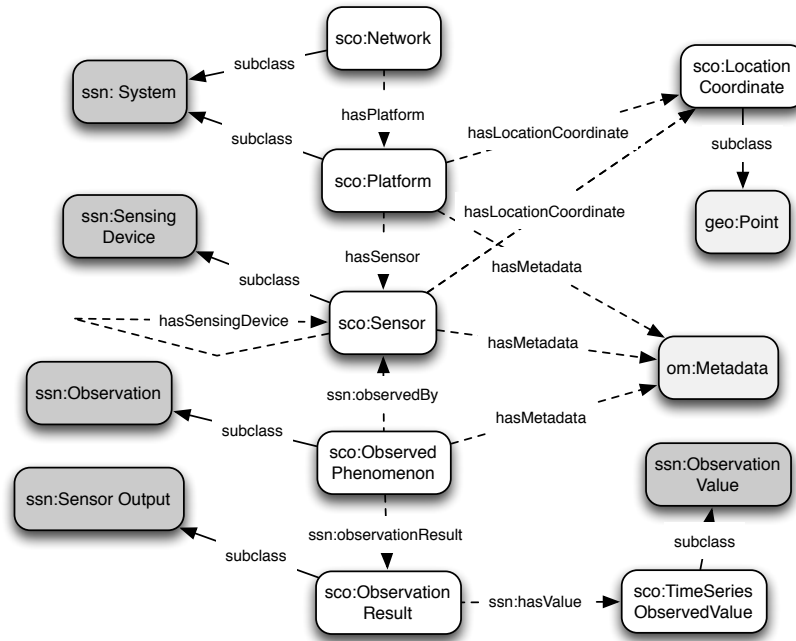


Fig. 4. Main Concepts in the Sensor Cloud Ontology

servation in the hierarchical document structure of the Sensor Cloud. To be compatible with GEO and OM, we introduced `sco:LocationCoordinate` as a type of `geo:Point`, and used `om:Metadata` to describe metadata of several classes (e.g. `sco:Sensor`). In doing so, we are then able to use standardised (ISO) vocabularies for coordinates, deployment, and quality that are left open in SSNO. Furthermore, we introduced some properties that are specific to the Sensor Cloud, e.g., those describing the number of time-value pairs of time series, and the first or last observation time.

Besides the above ontologies, we also used existing vocabularies such as `dcterms:source` (for the URL of the original JSON documents). In addition, we align with UCUM<sup>8</sup> for the units of measure, MUO<sup>9</sup> for the physical qualities, and Climate and Features<sup>10</sup> for the properties. This alignment is enabled by subclassing `sco:ObservedProperty` and `dul:UnitOfMeasure` accordingly.

The SCO data instances can be queried through the interface shown in Figure 3. The shown query returns longitude and latitude for all platforms in the Sensor Cloud. Another example is the following query that retrieves observation quality information represented as observation metadata.

<sup>8</sup> <http://purl.oclc.org/NET/muo/ucum/>

<sup>9</sup> <http://purl.oclc.org/NET/muo/muo#PhysicalQuality>

<sup>10</sup> <http://www.w3.org/2005/Incubator/ssn/ssnx/cf/cf-property>

```

SELECT DISTINCT ?sensingDevice ?observation ?obsMetadata
               ?qualityInfo ?methodType
WHERE {
  ?sensingDevice a ssn:Sensor .
  ?sensingDevice sco:hasObservedPhenomenon ?observation .
  ?observation sco:hasMetadata ?obsMetadata .
  ?obsMetadata md:dataQualityInfo ?qualityInfo .
  ?qualityInfo dq:report ?report .
  ?report dq:evaluationMethodType ?methodType .
}

```

## 4 Semantic RESTful API Wrapper (SERAW)

Within this section we describe how SERAW generates an RDF representation for sensor data accessible via the SC-API. Our main goal is to create a single RDF graph from a set of inter-linked JSON documents. Transformation is guided by user-provided URL expressions and transformation scripts. URL expressions associate documents of different type (e.g., Network, Platform, Sensor) with transformation scripts. Each script then describes how document elements are mapped to concepts and properties in a given ontology, i.e., SCO in our case. For SERAW we developed a simple mapping language to express transformation scripts. As with any formal language, there is a trade-off between functionality and complexity of use. Languages such as XSLT are powerful but difficult to use, especially for domain experts with limited computer programming experience. Furthermore, most approaches focus on transforming a single document without considering the problem of creating and maintaining relationships between resources in different input documents. We start by defining URL expressions (Section 4.1) before describing our mapping language (Section 4.2).

### 4.1 URL Expressions

URLs in SC-API follow patterns that reflect the structure of data in the Sensor Cloud. SERAW exploits these patterns by means of URL expressions to match URLs. The syntax of URL expressions is similar to that of URLs. For simplicity, we assume that a URL is composed of two parts: a *service identifier* (a string consisting of protocol, server name, domain name, and optional port information), and a (potentially empty) array of strings, called *path names*. A URL expression also has a service identifier, and a (potentially empty) array of strings which could be path variables or path names. A *path variable* is a regular expression. We reference variables from 0 to  $n$  based on the order of their occurrence. A URL matches a URL expression if the service identifiers are equal, the path arrays are of same size, and each path name in the URL matches the corresponding element in the URL expression. When a URL matches an expression we bind the variables in the expression to the path names in the URL. Figure 5 shows an example URL expressions. Regular expressions are enclosed in square brackets and follow Java's regular expression syntax (e.g., `'.*'` matches any character sequence). Both URLs `http://sense-t.csiro.au/sensorcloud/v1/network/TIA/platform/Tmar_Ridge` and `http://sense-t.csiro.au/sensorcloud/v1/`



`network/SouthEsk/platform/Ben_Lomond` match the pattern in Figure 5. In the first case, variable 0 is bound to path name *TIA* and variable 1 is bound to *Tmar\_Ridge*.



**Fig. 5.** URL expression matching any platform document in the Sensor Cloud

We use a set of URL expressions to identify relevant documents. Only documents having a URL matching one of the given URL expressions are considered by SERAW. The use of URL expressions gives us flexibility in that we can restrict the set of documents that are transformed (down to transforming only a single document). Starting from a set of URLs, SERAW traverses the documents accessible via the SC-API by following links between them. To detect cycles in the document graph, we keep track of all URLs visited. With each URL expression we optionally associate a transformation script. For each document, we execute all transformation scripts that are associated to any URL expressions matched by its URL. Associating transformation scripts with URL expressions allows us to handle each document type differently. By executing translation scripts, we add individuals to the ontology, generating the RDF representation for the documents accessible via the SC-API.

## 4.2 Transformation Scripts

Transformation scripts are programs in our mapping language. The language is inspired by formal mapping languages (e. g., [1]) that model ontology classes as unary predicates, and properties as binary predicates. Scripts are basically lists of statements that create ontology individuals for elements in JSON documents. We use *path expressions* to identify these elements. Given that there currently does not exist a standard path or query language for JSON, we implemented our own JSON path expression language (inspired by XPath and existing implementations like JSONPath<sup>11</sup>). We describe syntax and semantics of our mapping language using the script shown in Figure 6. We assume that the script is associated with the URL expression in Figure 5 and we use document `http://sense-t.csiro.au/sensorcloud/v1/network/TIA/platform/Tmar_Ridge` as our example.

Transformation scripts take the document (i. e., a JSON object) and a list of path names bound to variables in the matched URL expression as input. These path names are accessible within scripts by referencing the corresponding variables using double square brackets. In our example, `[[0]]` and `[[1]]` represent

<sup>11</sup> <http://goessner.net/articles/JsonPath/>

```

1   for / {
2       y1 = sco : Platform([[0]] + "_" + [[1]]) {
3           for platform/location {
4               y2 = sco : LocationCoordinate([[0]] + "_" + [[1]]) {
5                   geo : lat(y2, @latitude^^double);
6                   geo : long(y2, @longitude^^double);
7                   geo : alt(y2, @elevation^^double);
8               }
9               sco : hasLocationCoordinate(y1, y2);
10              sco : hasPlatform(sco : Network([[0]]), y1);
11          }
12      }
13  }

```

**Fig. 6.** Example Transformation Script

‘TIA’ and ‘Tmar\_Ridge’, respectively. Variables play an important role in the generation of unique ontology individual identifiers. For example, by executing line 2 (Figure 6), we create an instance of `sco:Platform` for the given JSON object (identified by path /), whose identifier is ‘Platform\_TIA\_Tmar\_Ridge’, constructed by concatenating the concept label `Platform` with path names ‘TIA’ (i. e., `[[0]]`) and ‘Tmar\_Ridge’ (i. e., `[[1]]`). This instance is uniquely identified among all individuals of the ontology. Here, we make use of the fact that path names in document URLs form relative keys. To avoid redundancies, we use a lookup table to maintain the individuals created, and an individual is created only if it is not in the table. In our case it is common that some information is repeated within different documents (e. g., location information is contained within platform and sensor documents) or accessible via different URLs (e. g., the same information about operational properties of a specific sensor type may occur in different documents due to the hierarchical structure that groups sensors under networks and platforms).

Lines 3-8 process the elements under path `platform/location`. Note that we use a path that is relative to the JSON element identified by the surrounding for-statement. There might be multiple elements in a document that match a given path expression. The for-statement is executed for each of them. The statement in line 3 creates an instance of `sco:LocationCoordinate` which has datatype properties `geo:lat`, `geo:long` and `geo:alt`. The statements in line 5-8 create instances of these datatype properties. The prefix ‘@’ indicates that the property value comes from the primitive JSON element identified by the respective relative path (e. g., `latitude`). An optional type information (e. g., `^^double`) indicates how the value is represented in RDF. The statements in lines 9 and 11 are examples for creating instances of object properties. It is important to note that we may reference an individual before it is created. In line 11, for example, it is possible that the referenced `sco:Network` with identifier ‘Network\_TIA’ has not already been created (i. e., it is not in the lookup table). If that is the case, we defer creation of references until referenced individuals have been created.

## 5 Implementation of SERAW

SERAW is implemented in Java and uses Jena API to generate RDF. We run SERAW as a web service using Apache Tomcat. The crawler is a separate process within the system. It crawls documents accessible through the SC-API and transforms them into RDF. Execution times of SERAW depend on the number and complexity of transformation scripts, as well as on network latency. Given the large number of documents accessible via SC-API, the overall time for transformation is several minutes. We conducted initial experiments to evaluate the impact of document access and transformation on overall execution time. Here, we focus on transforming documents representing sensing platforms using the script in Figure 6. All experiments were performed on a standard Dell Latitude laptop with a 2.53 GHz Intel Core i5 CPU and 4GB of RAM. All times are averaged over ten runs. There are a total of 74 platforms in SC-API with an average document size of 497 bytes. Our experimental results indicate that network latency (as expected) has the major influence on execution times. The average time for accessing a document was 37.55 ms. Transforming a document, on the other hand, only took 1.02 ms on average.

SERAW creates an RDF version of data accessible via the SC-API. Derived data raises the issue of timeliness. As data in the Sensor Cloud is updated, our RDF outdates over time. There is an obvious difference between update frequencies of different document types. While sensor observations get updated frequently, sensor metadata and deployment information is less likely to be updated. Depending on the type of information that is transformed into RDF and the application requirements we choose the frequency with which we re-run the transformation process. When doing so, we overwrite results from the previous run. The user interface also allows the process to be run on demand to generate an up-to-date RDF version of the data.

The SERAW user interface provides the ability to create and edit transformation scripts. It further provides the opportunity to test the results of created scripts on a restricted set of URLs. Figure 7 shows part of the interface that allows a user to specify a set of URLs and run the transformation scripts on the documents returned by these URLs. The interface displays the transformation result and any error messages generated during the transformation.

## 6 Linking to External Resources

The motivation for our work was not only to generate an RDF version of the data in the Sensor Cloud, but also to augment the resulting RDF graph with external information. As such we consider including links to external resources in the RDF graph. Several authors have recently described approaches for publishing semantic sensor data that includes links to resources in the LOD cloud [14,13,11,2,7]. In most of these approaches, links are hard-coded into the system. For SERAW, we implemented a library of purpose-built functions that retrieve resource URIs and other information from the web. Calls to these functions can be made from



**Fig. 7.** User Interface for Testing Transformation Scripts

within transformation scripts to include returned results into the generated RDF graph. The main benefit of having a library of functions for retrieving external information is reusability of code.

One example function is the generation of address information for a given location. The Sensor Cloud contains location information in form of latitude and longitude. We implemented a function, called *revGeoCode*, which uses Google’s Geocoding API<sup>12</sup> to transform location coordinates into a human-readable address. The API returns a list of results containing postal addresses and other geographical names for a location, with the most specific one returned first. Our implementation of *revGeoCode* returns the string value of the first `formatted_addresses` field in the result list (if any), e.g., *619 Auburn Road, Kayena TAS 7270, Australia* for the sensor platform at Tamar Ridge (Figure 7).

<sup>12</sup> <https://developers.google.com/maps/documentation/geocoding/>

Other examples are functions that query DBpedia <sup>13</sup>, Geonames <sup>14</sup>, or Freebase <sup>15</sup> to retrieve a URI for a given location name. The following SPARQL query basically represents the function that queries DBpedia for location URIs, called *DBpLocation*, where *var* is the location name provided as parameter with the function call:

```
SELECT ?city WHERE {
  ?city rdf:type <http://dbpedia.org/class/yago/TownsInTasmania> .
  FILTER (regex(?city, var )
}
```

We further combined functionalities into a function *DBpLatLonLocation* that takes latitude and longitude values as input, retrieves a place name for the location from Google’s Geocoding API (from the first address component of type *locality* in the result), and uses the place name (if any) to retrieve a URI from DBpedia. We implemented similar functions for Geonames and Freebase, called *GeoLatLonLocation* and *FBLatLonLocation* respectively. Figure 8 outlines the two-step process of using latitude and longitude information to generate a link to a location resource.

In transformation scripts, we make function calls when generating datatype properties. For example, adding the following line to the script in Figure 6 (anywhere in between lines 4-8) will take values of latitude and longitude in the JSON element and add location information to the individual identified by *y2*.

```
skos:relatedMatch(y2, DBpLatLonLocation(@latitude, @longitude));
```

We added location information for 56 of 74 platforms in the Sensor Cloud using *DBpLatLonLocation*. We also added information from Geonames and Freebase. DBpedia, Geonames, and Freebase give only general information about a particular place. This information, nevertheless, boosts up the knowledge that can be used by other applications (i.e., Agriculture Decision Support Systems). In the future, we will extend our library of functions, e.g., for retrieving soil types as provided by the European Environment Agency <sup>16</sup>. We also plan to add functionality to enable linking to geographical resources by querying services such as the Geographical Service <sup>17</sup> and LinkedGeoData <sup>18</sup>.

## 7 Discussion and Related Work

We designed SERAW to complement existing Web APIs that publish data in JSON format by providing semantic enhancements and query capabilities that are not available otherwise. While we use the Sensor Cloud as our main motivation,

<sup>13</sup> <http://dbpedia.org>

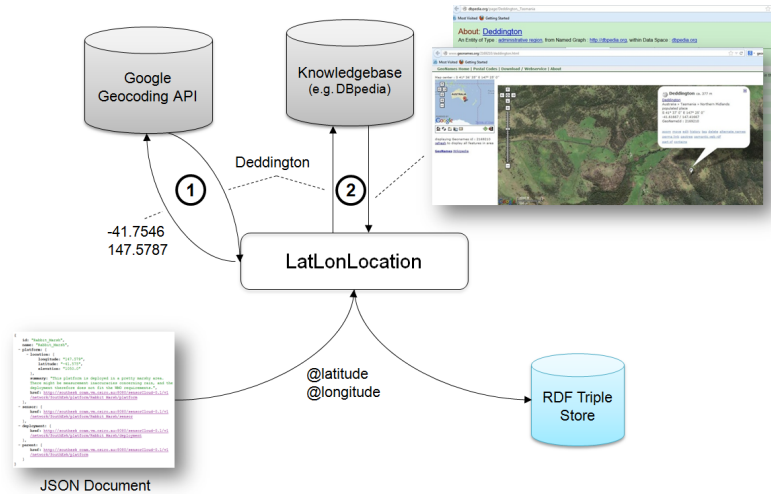
<sup>14</sup> <http://www.geonames.org>

<sup>15</sup> <http://www.freebase.com>

<sup>16</sup> <http://www.eea.europa.eu/data-and-maps/data/soil-type>

<sup>17</sup> <http://geoservice.psi.enacting.org>

<sup>18</sup> <http://linkedgeodata.org>



**Fig. 8.** Function to Retrieve Location URI based on Latitude and Longitude Information

the transformation mechanism described in this paper is applicable to other APIs. The semantic sensor web [14] was a major source of inspiration for the development of SERAW. In the semantic sensor web, sensor data is annotated using ontologies and published following linked data principles. Thus, SERAW is related to several efforts in publishing semantic sensor data [2,11,10,7,8].

In [2], *Barnaghi and Presser* present Sens2Web, a platform for publishing linked sensor data. Contrary to SERAW, sensor descriptions are entered into Sens2Web via a user interface. Sens2Web then transforms the data into RDF and runs a set of predefined queries to link to other resources. The resulting data is accessible through a SPARQL endpoint. In [11], the authors describe a workflow for transforming raw sensor observations from weather stations in the United States into RDF with links to other datasets in the LOD cloud. The described solution is specific to the given use case. In [10], the High-level API for Observations (HLAPIO) is presented. The primary focus of HLAPIO is on publishing sensor data using RESTful and linked data principles. HLAPIO uses a mapping language for data transformation that is intended for relational data and not for JSON documents. A linked data model and a RESTful proxy for OGCs Sensor Observation Service (SOS) is described in [7] to publish SOS data as linked data. The software is installed as a facade to an existing SOS. *Le-Phuoc et al.* describe their Linked Stream Middleware (LSM) [8]. LSM provides wrappers to access and integrate sensor stream data with other data sources. Users can annotate and visualise data using a web interface. LSM provides standard SPARQL queries and continuous queries using an adaptive query processor for linked stream data. Compared to SERAW, these existing approaches provide only limited possibilities for users to influence the transformation process.

SERAW comes with a flexible, light-weight mapping language. There exist a number of mapping languages and mechanisms for other data formats (e. g., Triplify<sup>19</sup>, and D2RQ<sup>20</sup> for relational databases) to produce RDF from data. To the best of our knowledge, none of them deals with the transformation of inter-linked JSON documents into RDF. D2RQ is a declarative language that maps elements in a relational database schema to classes and properties in ontologies. Our mapping language is similar to D2RQ, however, it maps JSON documents and elements to ontology classes and properties, and also, it includes specific features that allow us to deal with the hierarchical structure of JSON documents and links between these documents.

SERAW has an extendable set of purpose-built functions for creating links to the LOD cloud. Having a library of functions facilitates reuse of functionality. We see our approach towards link generation as a step in the right direction that hopefully will find adoption in other systems.

## 8 Conclusion and Future Work

In this paper we explored ways for transforming sensor data available via an existing data infrastructure as a semantically enriched RDF dataset. We provide *Knowledge Management* components that access components of our existing Sensor Cloud and generate RDF that can be queried via a SPARQL endpoint or re-used by applications for different purposes. The requirements for semantic web technologies within a sensor data infrastructure can vary according to the architecture level ranging from sensor devices to analytics to applications. We believe that SERAW is a generic tool that can be used across different levels in such infrastructure since it allows mappings rules and services to be combined.

One area of future work is the development of a more sophisticated solution to update our RDF copy of the Sensor Cloud data. Instead of transforming the whole set of documents, the goal is to transform only those that have been modified recently. We are able to identify documents that have been modified from the log files of the web server running the Sensor Cloud. We can then selectively transform only these documents. To do so requires to keep track of provenance in the RDF graph to identify those parts of the graph that were derived from a particular JSON document.

A similar problem arises with the generation of links to external resources. We generate links to external resources every time we run the transformation process. For functions that use information about the location of a platform, for example, the results are likely to be the same in every run (unless the platform is a mobile platform). One path of future work is to cache results from calls to external services. Provenance, again, plays an important role here, i. e., keeping track of parameter values (e. g., latitude and longitude) to determine whether a document has change in between runs or not.

<sup>19</sup> <http://triplify.org/Overview>

<sup>20</sup> <http://d2rq.org>

## Acknowledgements

The authors would like to thank the development team of the Sensor Cloud for their support. In particular, we would like to thank Chris Peters, Chris Sharman, and Peter Taylor for their assistance and feedback during the development of SERAW. We would also like to thank Michael Compton for his valuable feedback during preparation of the manuscript.

The Intelligent Sensing and Systems Laboratory and the Tasmanian node of the Australian Centre for Broadband Innovation is assisted by a grant from the Tasmanian Government which is administered by the Tasmanian Department of Economic Development, Tourism and the Arts.

## References

1. An, Y., Borgida, A., Mylopoulos, J.: Constructing complex semantic mappings between xml data and ontologies. In: International Semantic Web Conference. pp. 6–20 (2005)
2. Barnaghi, P., Presser, M.: Publishing linked sensor data. In: Proceedings of the 3rd International Workshop on Semantic Sensor Networks (SSN10) (2010)
3. Botts, M., Percivall, G., Reed, C., Davidson, J.: Ogc sensor web enablement: Overview and high level architecture. Tech. Rep. OGC 07-165, Open Geospatial Consortium (2007)
4. Broering, A., Echterhoff, J., Jirka, S., Simonis, I., Everding, T., Stasch, C., Liang, S., Lemmens, R.: New generation sensor web enablement. *Sensors* 11, 2652–2699 (2011)
5. Fielding, R.T.: Architectural styles and the design of network-based software architectures. Ph.D. thesis (2000)
6. Heath, T., Bizer, C.: *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web, Morgan & Claypool Publishers (2011)
7. Janowicz, K., Bröring, A., Stasch, C., Schade, S., Everding, T., Llaves, A.: A restful proxy and data model for linked sensor data. *Environment* 1(ii), 1–30 (2011)
8. Le-Phuoc, D., Nguyen-Mau, H.Q., Parreira, J.X., Hauswirth, M.: A middleware framework for scalable management of linked streams. *Web Semant.* 16, 42–51 (Nov 2012), <http://dx.doi.org/10.1016/j.websem.2012.06.003>
9. Malewski, C., Simonis, I., Terhorst, A., Bröring, A.: Starfl - a modularised metadata language for sensor descriptions. *International Journal of Digital Earth* 1, 1–20 (2012)
10. Page, K.R., Frazer, A.J., Nagel, B.J., Roure, D.C.D., Martinez, K.: Semantic access to sensor observations through web apis. vol. 0, pp. 336–343. IEEE Computer Society, Los Alamitos, CA, USA (2011)
11. Patni, H., Henson, C., Sheth, A.: Linked sensor data. In: Collaborative Technologies and Systems (CTS), 2010 International Symposium on. pp. 362–370 (may 2010)
12. Prud’hommeaux, E., Seaborne, A.: Sparql query language for rdf. W3c recommendation, W3C (Jan 2008), <http://www.w3.org/TR/rdf-sparql-query/>
13. Sequeda, J., Corcho, O.: Linked stream data: A position paper. In: Proceedings of the 2nd International Workshop on Semantic Sensor Networks (SSN09) (2009)
14. Sheth, A., Henson, C., Sahoo, S.S.: Semantic sensor web. *IEEE Internet Computing* 12(4), 78–83 (2008)
15. W3C Incubator Group: Semantic sensor network xg final report. Tech. rep., <http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/>