EXPLORING THE USE OF THE SEMANTIC WEB FOR DISCOVERING AND PROCESSING DATA FROM SENSOR OBSERVATION SERVICES

A thesis submitted to the Delft University of Technology in partial fulfillment
of the requirements for the degree of

Master of Science in Geomatics

by

Ivo de Liefde

June 2016

Ivo de Liefde: *Exploring the Use of the Semantic Web for discovering and processing data from Sensor Observation Services* (2016)

The work in this thesis was made in the:

GDMC

Geo-Database Management Centre
Department of the OTB
Faculty of Architecture & the Built Environment
Delft University of Technology

Supervisors:	M. de Vries
	B.M. Meijers
Co-reader:	S. Zlatanova

ABSTRACT

This thesis aims to design a method that uses the semantic web to improve sensor data discovery as well as the integration and aggregation of sensor data from multiple sources.

Literature about the Sensor Web Enablement (SWE) standards, the Catalog Service for the Web (CSW), semantic web technologies and so-called sensor data middleware is analysed.

The outline of a method is presented to automatically create an online semantic knowledge base of sensor metadata. It is also presented how such a knowledge base can be used to automatically translate logical queries of users to technical queries for retrieving observation data from Sensor Observation Service (SOS).

Based on this method a proof of concept application has been created.

ACKNOWLEDGEMENTS

I would like to thank Florian Fichtner for taking the time to read my acknowledgements.

CONTENTS

ACRONYMS

1 INTRODUCTION

From 2020 onwards all member states of the European Union (EU) should provide sensor data to the Infrastructure for Spatial Information in Europe (INSPIRE) in order to comply with annex II and III of the INSPIRE directive [INSPIRE, 2015]. For this a number of SWE standards are required to be used [INSPIRE, 2014]. The sensor web is a relatively new development and there are still many questions on how to structure it. This thesis aims to design a method to publish and link sensor metadata on the semantic web to improve the discovery, integration and aggregation of sensor data using SWE standards.

1.1 BACKGROUND

In 2008 the Open Geospatial Consortium (OGC) introduced a new set of standards called Sensor Web Enablement (SWE). These standards make it possible to connect sensors to the internet and retrieve data in a uniform way. This allows users or applications to retrieve sensor data through standard protocols, regardless of the type of observations or the sensor's manufacturer [Botts et al., 2008]. Among other standards SWE includes the Observations and Measurements (O&M) which is a model for encoding sensor data, the Sensor Modelling Language (SensorML) which is a model for describing sensor metadata and the SOS which is a service for retrieving sensor data [Botts et al., 2007]. O&M has also been adopted by the International Organisation for Standardisation (ISO) under ISO 19156:2011 [ISO, 2011].

Recently OGC has defined the role which their standards could play in smart city developments [Percivall, 2015]. Smart cities can be defined as "enhanced city systems which use data and technology to achieve integrated management and interoperability" [Moir et al., 2014, p. 18]. Research on smart cities has shown a great potential for using sensor data in urban areas. Often this is presented in the context of the Internet of Things (IoT) [Zanella et al., 2014; Wang et al., 2015a]. The IoT can be described as "the pervasive presence around us of a variety of *things* or *objects* ... [which] are able to interact with each other and cooperate with their neighbors to reach common goals" [Atzori et al., 2010, p. 2787].

Parallel to the development of the sensor web other research has focused on the semantic web, as proposed by Berners-Lee et al. [2001]. This is a response to the traditional way of using the web, where information is only available for humans to read. The semantic web is an extension of the internet which contains meaningful data that machines can understand as well. Rather than publishing documents on the internet the semantic web contains linked data using the Resource Description Framework (RDF), also known as the *web of data* [Bizer et al., 2009]. Data in RDF can be queried using the SPARQL Protocol and RDF Query Language (SPARQL) at so-called SPARQL endpoints. The Web Ontology Language (OWL) is an extension of

RDF and was designed "to represent rich and complex knowledge about things, groups of things, and relations between things" [OWL working group, 2012]. Originally, the semantic web intended to add metadata to the internet [Lassila and Swick, 1999]. However, today it is being used for linking any kind of data from one source to another in a meaningful way [Cambridge Semantics, 2015].

Sheth et al. [2008] proposes to use semantic web technologies in the sensor web. This Semantic Sensor Web (SSW) builds on standards by OGC and the World Wide Web Consortium (W3C) "to provide enhanced descriptions and meaning to sensor data" [Sheth et al., 2008, p. 78]. W3C responded to this development by creating a standard ontology for sensor data on the semantic web [Compton et al., 2012].

1.2 PROBLEM STATEMENT

Finding sensor data that can be retrieved using open standards is not easy. The implementation of the sensor web is still in an early stage. At the moment there are only a limited number of SOS implementations available on the web and they contain a limited amount of data. In the Netherlands the SOS by the Dutch national institute for public health and the environment (RIVM) is one of the first ones to be developed. It has only recently been launched and contains data on air quality. A number of other organisations still use a custom Application Programming Interface (API) to retrieve data from sensors connected to the internet. The problem of these custom APIs is that it is very hard to create an application that automatically retrieves data from them, because they have not implemented standards regarding the content of their service, the metadata models behind it or the kind of requests that can be made. It forces the application to have knowledge built in on the specifics of the individual APIs that are being used.

It has been researched to what extent a catalogue service could be useful for discovering sensor data from a SOS using the web service interfaces Sensor Instance Registry (SIR) [Jirka and Nüst, 2010] and Sensor Observable Registry (SOR) [Jirka and Bröring, 2009]. Catalogue services have already been available for example for the Web Map Service (WMS), Web Feature Service (WFS) or Web Coverage Service (WCS) [Nebert et al., 2007]. However, for the sensor data sources used in this paper no register or catalogue service has been implemented. Atkinson et al. [2015] also argues that catalogue services have a number of major disadvantages. It places a very high burden on the client to not only know where to find the catalogue service, but also to have knowledge on all kinds of other aspects (e.g. its organisation, access protocol, response format and response content) [Atkinson et al., 2015, p. 128]. Atkinson et al. suggest that linked data is therefore a much better solution for discovering sensor data.

However, for sensor data to be discovered on the semantic web there have to be inward links, from other sources linking towards the sensor (meta)data. Current research on the SSW has focused on publishing sensor data on the semantic web with links that point outwards [Atkinson et al., 2015; Janowicz et al., 2013; Pschorr, 2013]. This gives meaning to the data and is useful in order to work with the data, but it has a very limited effect on the discovery of the sensor data by others.

One of the challenges of using sensor data is the difficulty of integrating it from different sources to perform data fusion [Corcho and Garcia-Castro, 2010; Ji et al., 2014; Wang et al., 2015b]. Data fusion is "a data processing technique that associates, combines, aggregates, and integrates data from different sources" [Wang et al., 2015a, p. 2]. Even if the sources comply with the SWE standards it is challenging, since the data can be of a different granularity, both in time and space. Spatio-temporal irregularities are a fundamental property of sensor data [Ganesan et al., 2004].

The question arises to what extent the semantic web could be a better solution for publishing sensor data than the current geoweb solutions like SOS. The geoweb has some very good qualities, such as very structured approaches through which (sensor) data can be retrieved using well defined services. These standardised services have been accepted by large organisations as OGC and ISO. Furthermore, they are often based on years of discussion. This is different from for example web pages where content can be completely unstructured. The response of a SOS also contains some semantics about sensor data. There can be x-links inside the Extensible Markup Language (XML) with Uniform Resource Identifier (URI)s that point to semantic definitions of objects.

Still, the semantic web could be beneficial for the geoweb. Since data on the web has a distributed nature it can be questioned whether centralised catalogue services are feasible to create. It places a burden on the owner of the SOS to register with a catalogue service. Also, there could be multiple of these services on the web creating issue regarding the discovery of relevant catalogues. The semantic web could solve this issue by getting rid of the information silos and storing data directly on the web instead. This allows the interlinking and reuse of data on the web, which makes it easier to find related data. For automatic integration and aggregation it could be useful that the semantic web is machine understandable.

In conclusion, the problem to be addressed is the lack of knowledge on how to exploit the full potential of the sensor web using the semantic web. Creating the right links could greatly enhance the discovery, integration and aggregation of sensor data. However, there is no method yet to establish this linked metadata for sensors, while the standardised nature of a SOS should allow for generating it in an automated process. This thesis will create a design for such an automated process, research how to establish inward links and explore the advantages and disadvantages of publishing sensor metadata on the semantic web with a proof-of-concept implementation.

1.3 MOTIVATION

Sensor data ties together many different fields of research. On the one hand there is research on how to create the most efficient sensor networks that uses the least amount of power to transfer the observed data over long distances [Korteweg et al., 2007; Xiang et al., 2013]. This involves academic fields such as mathematics, physics and electrical engineering. On the other hand there is research that uses sensor data to gain insights into real world phenomenon. This involves academic fields such as geography, environmental studies and urbanism. In order to connect these scientific fields, studies have focused on the use of computer science and standardisation for transferring sensor data over the internet.

In the future more sensor data is expected to be produced [Price Waterhouse Coopers, 2014]. Both experts and non-experts will be involved in this development. Experts will produce more data because of European legislation (INSPIRE). Non-experts will be involved more often via smart cities and IoT developments where users or consumer electronics produce sensor data as well. This vast amount of data could be very useful for academic research, provided researchers are able to find the data they need online and are able to integrate and aggregate data from heterogeneous sources. Publishing sensor metadata on the semantic web could make it easier to find what you need through related data on the internet. Having a automated process for this and being able to seamlessly integrate and aggregate data from different sources could be of great use for research such as van der Hoeven et al. [2014], Van der Hoeven and Wandl [2015] and Theunisse [2015]. They are examples of studies that try to understand phenomenon in the built environment using sensor data. Currently data collection and processing takes up a large part of the research, while with the implementation of SWE standards and the use of the semantic web this might be significantly reduced.

As sensor data is becoming more important the gap between the logical data requests of users and the technical requests defined by sensor web protocols should be bridged. A logical sensor data request contains at least a geographical area, a type of observation and a time range. An example of a simple logical request is: The average air quality per month in neighbourhoods of Delft over the last five years. To translate this to a technical request specific knowledge is required about things like URIs, encodings, data models, service models and data formats. Whether an automated process could perform this translation is therefore part of the research in this Thesis.

1.4 RESEARCH QUESTIONS

This thesis aims to design a method that uses the semantic web to improve sensor data discovery as well as the integration and aggregation of sensor data from multiple sources. The following question will be answered in this research:

> *To what extent can the semantic web improve the discovery, integration and aggregation of distributed sensor data?*

To answer this question it is broken down into four subquestions:

- To what extent can sensor metadata be automatically retrieved from any SOS?
- To what extent can sensor metadata from a SOS be automatically converted to linked data and published on the semantic web?
- What is an effective balance between the semantic web and the geo web in the chain of discovering, retrieving and processing sensor data?
- To what extent can already existing standards for retrieving data be (re)used for a service that supplies integrated and aggregated sensor data?

2 RELATED WORK

To answer the main question of this thesis it is important to get a good understanding of the current state of the art in sensor web research. A number of topics are relevant for this, such as existing SWE standards for publishing sensor data to the web, the development of linked data ontologies for observation data, extending catalog services to include sensor metadata and methods for adding semantics to SWE data services. This chapter discusses the recent relevant literature on these topics.

2.1 SENSOR WEB ENABLEMENT

Botts et al. [2007] present Sensor Web Enablement (SWE), which is a suite of standards developed by OGC. It contains two parts: the information model and the service model. The information model includes O&M, SensorML and SWE common. O&M defines the data model and encoding for observation data and SensorML defines the data model and encoding for sensor metadata. SWE common is a low-level data model for exchanging sensor related data. The service model contains the SOS, Sensor Planning Service (SPS), Sensor Alert Service (SAS) and Web Notification Service (WNS). A SOS can be used to retrieve observation data, a SPS to plan actions of a sensor, and a SAS to receive alerts about subscribed events. With a WNS users can have asynchronous dialogues (message interchanges) with one or more other services. Recently the SensorThings API has been added to the list of SWE services (see `http://ogc-iot.github.io/ogc-iot-api/index.html`). The SensorThings API is a service for retrieving observation data and sensor metadata for IoT applications. This thesis focusses on O&M, SensorML and SOS. The following paragraphs will therefore described these standards in more detail.

2.1.1 Observation and Measurements

In the Observations and Measurements (O&M) data model an observation is modelled using a number of concepts. First of all, there is a feature of which sensor data is wanted. This is the so-called Feature of Interest (FOI). This feature has a property that can be observed, also known as an observable property. For example, the air at a certain location (FOI) has a certain temperature (observable property). A FOI can also be a geographical feature such as a river or a forest area. O&M uses the concept of FOI rather than the sensor location, because for some observations the exact location may not be trivially available. Furthermore, specific specimens are in some cases removed from their sampling location and observed at another location afterwards. This would create problems when defining a location of observation, hence the use of Feature of Interests in the O&M data model. An observation is defined as the "act of measuring or otherwise determining the value of a property" [ISO, 2011, p. 3]. The observation uses a procedure, which is a

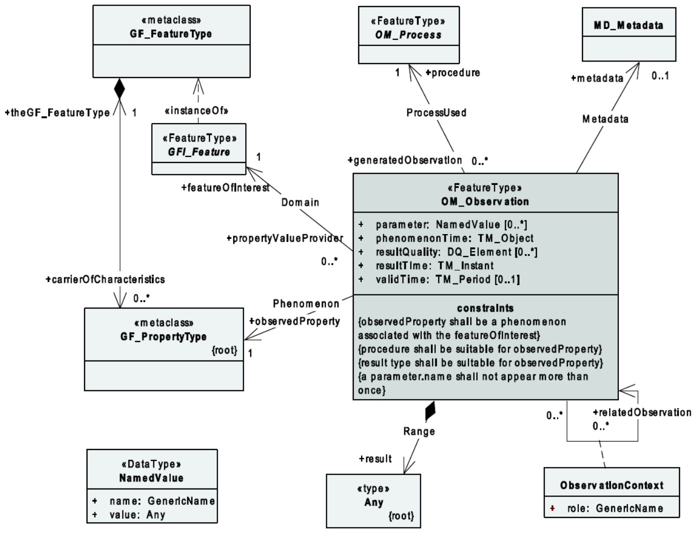

Figure 2.1: Basic observation in the O&M data model [ISO, 2011, p. 9]

method, algorithm or instrument, or system of these. A procedure can therefore be a sensor, an algorithm processing the raw observation data and/or a system of sensors observing a property of the FOI. The output of the procedure is a result. The result consists of a value and a corresponding Unit of Measurement (UOM). For example: 15 (value) degrees Celcius (UOM).

A procedure can produce many results about the same FOI over time. Therefore, three temporal concepts have been modelled: result time, phenomenon time and valid time. The result time is the timestamp of the observation result, or in other words: the moment the observation result became available. It is not necessarily the time that corresponds to the phenomenon, depending on when the observation procedure is performed (specimens can be taken from a sampling location and observed later) and the amount of time it takes to perform a procedure. Therefore, the phenomenon time describes the time that the observation applies to the property of the FOI. The valid time is a time range that indicates when an observation is a valid indication of the property of the FOI. When observing glacier motion at a speed of several meters per year an observation result might be valid for a day. On the other hand, observations with large fluctuations during the day might only be valid for a number of minutes up to an hour. This is the case for example with air quality observations in cities, that have large daily fluctuations with peaks during the rush hours.

Additional metadata about an observation can be added using the metadata class in the data model. To provide context about an observation the 'related observations' class has been added to the O&M model as well. Figure 2.1 shows the Unified Modeling Language (UML) diagram containing the different concepts explained in this paragraph and the relations between them.

2.1.2 SensorML

The Sensor Modelling Language (SensorML) aims to "provide a robust and semantically-tied means of defining processes and processing components associated with the measurement and post-measurement transformation of observations" [Open Geospatial Consortium, 2014, p. ix]. Creating interoperability at the syntactic and semantic level allows processes to be better

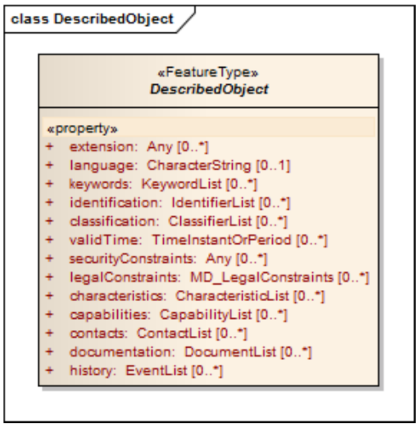

Figure 2.2: 'DescribedObject' class in SensorML [Open Geospatial Consortium, 2014, p. 39]

understood by machines, utilized automatically in complex workflows, and to be easily shared between intelligent sensor web nodes. SensorML is a framework that can be used to define the geometric, dynamic, and observational characteristics of sensors and sensor systems in order to achieve syntactic and semantic interoperability.

In SensorML there is a destinction between two types of processes: physical and non-physical processes. For a physical process information about the spatio-temporal position is important. This is the case with for example detectors, actuators, and sensor systems. Non-physical processes are mathematical operations or functions. Both types of processes are modelled as a specialisation of the 'DescribedObject' class. This class provides a set of metadata which is useful for all process classes in SensorML. Figure 2.2 shows the properties of a described object class instance. The 'AbstractProcess' class is derived from the described object and includes the properties: inputs, outputs, parameters, typeOf, featureOfInterest, configuration and modes. This forms the basis for defining both physical and non-physical processes. The 'AbstractPhysicalProcess' class is derived from the describe object class and adds spatial and temporal coordinates for the physical process device. The final step in defining a physical process is to specialise the abstract physical process with the 'PhysicalComponent' class. The physical component describes the device that provides a processing function. Figure 2.3 shows the UML class diagram of a physical process.

Although SensorML is not dependent on O&M the result of a process modelled by SensorML is typically considered as an observation result. This is the case if it is measuring a physical property or phenomenon. Therefore, the output values described in SensorML and resulting from a sensor or process may be encoded as an O&M Observation. Inversely, the procedure property of an O&M Observation instance can be described using SensorML [Open Geospatial Consortium, 2014].

2.1.3 Sensor Observation Service

The Sensor Observation Service (SOS) is the SWE web service for retrieving observation data. There are three core requests that can be made to retrieve sensor (meta)data from a SOS: `GetCapabilities`, `DescribeSensor` and `GetObservation`. `Get Capabilities` returns a complete overview of

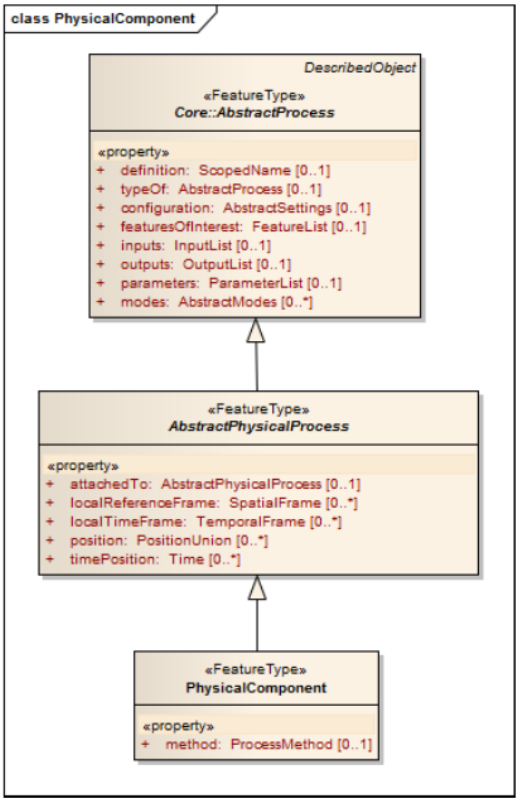

Figure 2.3: Definition of a physical process in SensorML [Open Geospatial Consortium, 2014, p. 57]

what the SOS has to offer. The `DescribeSensor` request returns detailed information about individual sensors and `GetObservation` returns observation data. These three core requests are mandatory in a SOS under the 2.0 specifications [Bröring et al., 2012]. There are also a number of optional extensions to a SOS: `GetFeatureOfInterest`, `GetObservationById`, `InsertCapabilities`, `InsertObservation`, `InsertSensor`, `DeleteSensor`, `InsertResult`, `InsertResultTemplate` and `GetResultTemplate`. Requests can be made as using the HyperText Transfer Protocol (HTTP) methods GET or POST. A GET request is a Uniform Resource Locator (URL) with input data as Key-Value Pair (KVP). Using POST the request is described in an XML document which is send to the SOS. There can be different response formats, but there is always at least the option to retrieve the response as an XML document. Based on the specification by Bröring et al. [2012] the following paragraphs describe both the core and optional requests of a SOS, as well as the structure of their responses.

Get capabilities

The `GetCapabilities` request is the first step in communicating with a SOS. The request is made by taking the HTTP address of the SOS and adding `service=SOS&request=GetCapabilities`. It returns a document including information on what the service has to offer. The document contains a number of sections: service identification, service provider, operations metadata, filter capabilities and contents.

In the service identification section there is general information about the service, such as the title and supported SOS versions, but also whether there are fees or access constraints. The service provider section contains details on which organisation provides the SOS and lists their contact information. The operations metadata section lists the supported request types. It also contains an overview of all features-of-interest, observed properties, procedures and offerings. Offerings are groupings of FOIs collected by one procedure.

The contents section describes the data that can be retrieved, listed per offerings. Each offering has an identifier, together with information on the procedure and observable properties. Which filters can be applied in a request is described in the filter capabilities section. The supported parameters for both spatial and temporal filters are listed here.

Describe sensor

The `DescribeSensor` request gives detailed information about a specific sensor. The request is built by taking the HTTP address of the SOS and adding `service=SOS&version=2.0.0&request=DescribeSensor&procedure=aprocedure&proceduredescriptionformat=aformat` where the SOS version, the procedure and the procedure description format have to contain values defined in the capabilities document. The response of a `DescribeSensor` request is a SensorML document as described in Paragraph **??**.

Get observation

Using `GetObservation` requests observation measurements can be retrieved. The request is made by taking the HTTP address of the SOS and adding

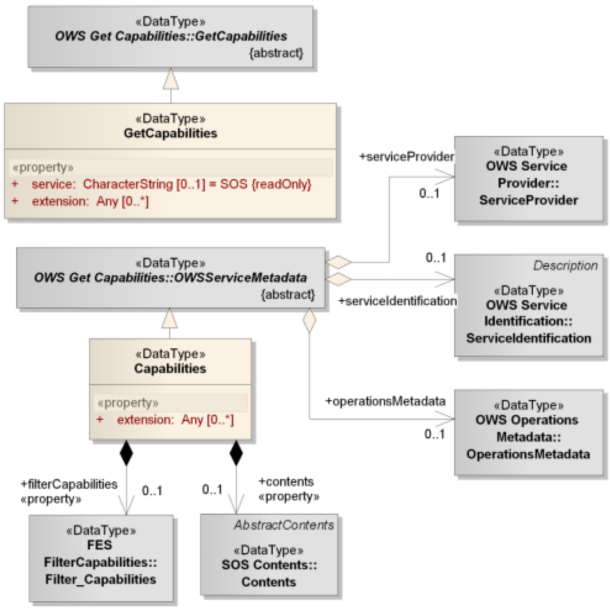

Figure 2.4: Data model behind the capabilities document of a SOS [Bröring et al., 2012]

service=SOS&version=2.0.0&request=GetObservation. This returns a response with the default parameters, which can differ from one SOS to another. To further specify the request, paramaters should be added such as: observed property, procedure, feature-of-interest, offering and outputformat. Spatial and temporal filters can be added if they are supported by the service. The `GetObservationByID` request is an extension of `GetObservation` and let's users retrieve sensor data using an identifier that points to a specific observation.

Get feature of interest

The `GetFeatureOfInterest` request allows to retrieve information about the FOI of a certain observation. The response can be all FOIs or only ones that are related to a specific observed property, procedure or spatial filter. This request also allows logical operators, for example: "GetFeatureOfInterest (observedProperty := temperature AND procedure := thermometerX OR anemometerY)" [Bröring et al., 2012, p. 40]. The response is a document with 'GFI_features' (ISO 19109) [International Organisation for Standardisation, 2005], which are implemented in the Geography Markup Language (GML) (ISO 19136) [International Organisation for Standardisation, 2007] by the element gml:AbstractFeature and type gml:AbstractFeatureType [ISO, 2011, p. 38].

Transactional extensions

It is possible to update the content of a SOS using transactional request. There are six of these requests that can be implemented: `InsertCapabilities`, `InsertObservation`, `InsertSensor`, DeleteSensor, `InsertResultTemplate` and `InsertResult`. `InsertCapabilities` allows a request to add data to the capabilities document described in Paragraph 2.1.3. The request contains three mandatory parameters and one optional parameter: it is required to have the `procedureDescriptionFormat`, `FeatureOfInterestType` and `ObservationType`. Optionally, `SupportedEncoding` could be added to the request. An `InsertCapabilities` should be made in combination with a `InsertObservation`, `InsertSensor` or `InsertResult` request.

A `InsertResultTemplate` request allows to upload a template for result values. It should contain data about the offering, the observation template, the result structure and result encoding. The actual results can be added later using an `InsertResult` request. This request is different from the `InsertObservation` request, as it only inserts the result value of an observation, assuming that the metadata is already present in the SOS. This is useful when there is limited communication bandwidth and processing power. This request has two mandatory parameters: a pointer to the template and the observation value to be inserted. An `InsertObservation` request allows observations to be added to a registered sensor system. It also has two mandatory parameters: a pointer to an offering and the observation to be inserted.

Individual sensors can be inserted or deleted using `InsertSensor` and `DeleteSensor` requests. For inserting a sensor the following parameters are required: the procedure description format, a procedure description, the observable property, a feature of interest type and an observation type. For deleting a sensor only an identifier pointing to a specific sensor needs to be passed as a parameter.

2.2 SEMANTIC WEB

The semantic web is an extension of the internet using standards by the W3C. It aims to provide a framework for defining parts of data which are machine understandable, instead of publishing documents on the web containing data that only humans are able to read and understand. Figure 2.5 shows the hierarchy of data on the semantic web, starting with URIs at the bottom which are unique identifiers for parts of data. Their actual content is made available in self describing documents. In these documents concepts are defined (re)using existing vocabularies and ontologies. The top layers of the pyramid (logic, proof and trust) have not yet been standardized, and are still a topic of research.

The semantic web is based on the open world assumption. This means that "the thruth of a statement is independent of whether it is known" [Hebeler et al., 2011, p. 103]. Traditional software application use the closed world assumption. Here a statement is assumed false if the answer is unknown. A simple example of this is a database table containing costumers of a store. If a person's name is not in that table the closed world assumption decides that this person is not a costumer of the store. The name is unknown and hence the statement 'he/she is a customer' is false. An open world assumption would decide that it is simply unknown whether this person is a costumer, since it hasn't been described anywhere. The closed world assumption only works in an environment where all information is complete. There should be no possibility in our example that people who are customers are not present in the database table. The case of the internet is that there is an enormous collection of data stored in different places and it can never be guaranteed that this data is complete. Therefore the semantic web is based on the open world assumption [Hebeler et al., 2011].

In the next paragraphs the framework of the semantic web – the Resource Description Framework (RDF) – will be described, followed by different RDF notations. The last part explains the concept of Persistent Uniform Resource Locators (PURLs).

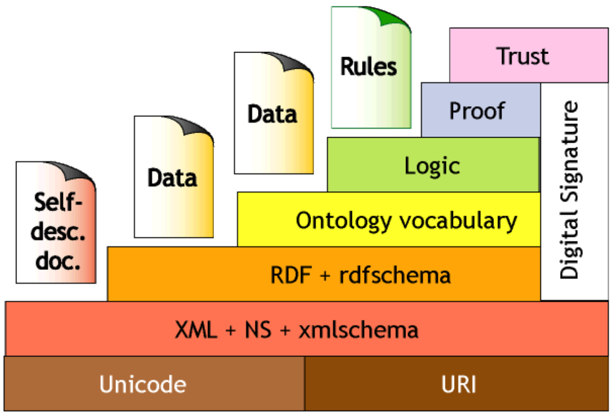

Figure 2.5: Hierarchy of the semantic web [1]

2.2.1 Resource Description Framework

In RDF data is stored as so-called 'triples'. These triples are structured as: subject, predicate and object [Berners-Lee et al., 2001]. The subject and the object are things and the predicate is the relation between these two things. For example, to define a geographic feature such as the municipality of Delft on the semantic web a number of triples can be made. Figure 2.6 shows how Delft can be defined as a municipality with a certain geometry using triples of subject, predicate and object.

Three types of data can make up these triples [Manola et al., 2014]. The first type is an International Resource Identifier (IRI). This is a reference to a resource and can be used for all positions of the triple. A URL is an example of an IRI, but IRIs can also refer to resources without stating a location or how it can be accessed. An IRI is a generalisation of an URI, allowing also non-ASCII characters. In the example of the municipality of Delft, IRIs can be used to define 'Delft' and 'Municipality', but also for the predicates 'is a' and 'has geometry'. The second type of data is a literal. A literal is a value which is not an IRI, such as strings, numbers or dates. These values can only be used as object in a triple. In the example of Delft, a literal could be used to store the geometry of the boundary: POLYGON((x_1, y_1 x_2, y_2 ... x_n, y_n, x_1, y_1)). A literal value can have a datatype specification [Cyganiak et al.,

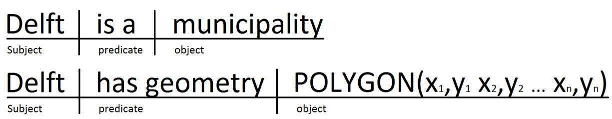

Figure 2.6: Triples of object, predicate and subject define Delft as a municipality with a geometry

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>

<http://example.com/Delft> a <http://dbpedia/resource/Municipality> ;
    geo:hasGeometry "<http://www.opengis.net/def/crs/EPSG/0/4258> POLYGON(( x1 y1, x2 y2, ... xn yn, x1 y1 ))"^^geo:wktLiteral
```

Figure 2.7: Triples of Figure 2.6 in the Turtle notation

2014]. This is added to the literal with the ^^ symbols, followed by the IRI of the datatype specification. In Figure 2.7 the datatype is 'geo:wktLiteral'.

Sometimes it is useful to refer to things without assigning them with a global identifier. The third type is the blank node and can be used as an subject or object without using an IRI or literal [Manola et al., 2014].

2.2.2 Notation

There are a number of different notations for writing down these triples (serialisation), such as XML [Gandon and Schreiber, 2014], N3 [Berners-Lee and Connolly, 2011] and Turtle [Beckett et al., 2014]. Turtle will be used in this thesis, because it is a commonly used notation which is also relatively easy to read for humans. Figure 2.7 shows how the triples of Figure 2.6 would be written using the Turtle notation. Turtle puts IRIs between brackets and ends a triple with a point.

In Figure 2.7 the DBPedia IRI is used for the object 'Municipality'. The 'is a' predicate is represented by a built-in RDF predicate which can be written simple as 'a'. The second predicate is 'hasGeometry' for which the GeoSPARQL IRI is used. The geometry is a literal in the Well-Known Text (WKT) format. Note that the subject is only written once when there are multiple triples with the same subject. This is because triples that share the same subject are divided by semicolons in the Turtle notation. A point marks the end of the last triple with a specific subject.

2.2.3 Persistent Uniform Resource Locators

URLs are an essential part of the web. They can point to websites, but also to semantic definitions of concepts. However, if an URL changes the existing links towards this URL are broken. To prevent this Persistent Uniform Resource Locators are being used. A Persistent Uniform Resource Locator (PURL) is a "naming and resolution service for general Internet resources" [Shafer et al., 2016]. This allows organisations to change the location of their data without changing the URL to which can be linked by other websites or linked data. A PURL server receives the URL and redirects the client to the current location of the resource. If the location of the resource changes, the server can be informed. It will then redirect clients to the new location (Figure 2.8).

2.3 SENSOR METADATA IN CATALOGUE SERVICES

The OGC has developed a way of including sensors into a catalogue service using sensor registries. The different services related to this are described in this paragraph, starting with the Catalog Service for the Web (CSW), after

1 from https://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html by Tim Berners-Lee

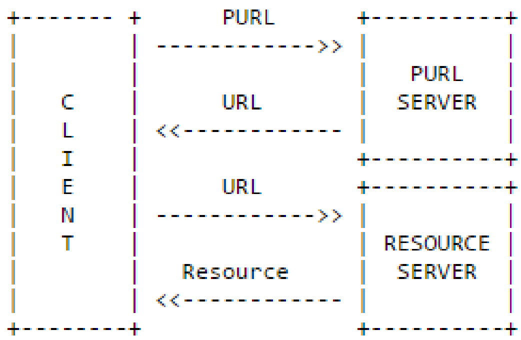

Figure 2.8: Persistent Uniform Resource Locator (PURL) resolves to the current resource location. By Shafer et al. [2016]

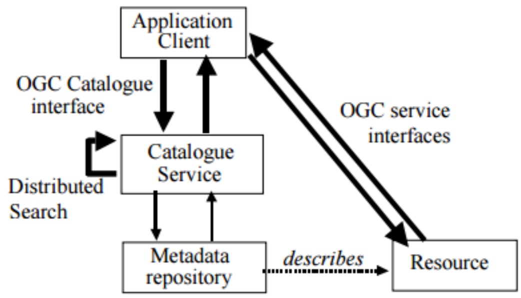

Figure 2.9: CSW model architecture [Open Geospatial Consortium, 2007, p. 26]

which the Sensor Observable Registry (SOR) and Sensor Instance Registry (SIR) services will be presented.

2.3.1 Catalog Service for the Web

The CSW is an OGC standard for a geoweb service that contains "collections of descriptive information (metadata) for data, services, and related information objects" [Open Geospatial Consortium, 2007, p. xiv]. Figure 2.9 shows the intermediary role of this service between client and data sources. It also shows that the CSW can use data from three different sources to return to a client: a local metadata repository, a resource service, or another CSW. The resource service can use an OGC interface, but this is not required.

The CSW has the following requests for retrieving metadata: `GetCapabilities`, `DescribeRecord`, `GetDomain`, `GetRecords` and `GetRecordByID`. Metadata producers can use the `Transaction` and `Harvest` requests to respectively change or add content of a CSW. The `GetCapabilities` request returns a capabilities document to the client showing what the service has to offer. This request and response is required for all OGC geoweb services and contains sections such as: service identification, service provider, operations metadata and filter capabilities. The service identification section lists general information about the service, such as the title and supported CSW versions, but also whether there are fees or access constraints. The service provider section contains details on which organisation provides the SOS and lists their contact information. The operations metadata section lists the supported request types. The filter capabilities show the different filters that are supported by the CSW instance.

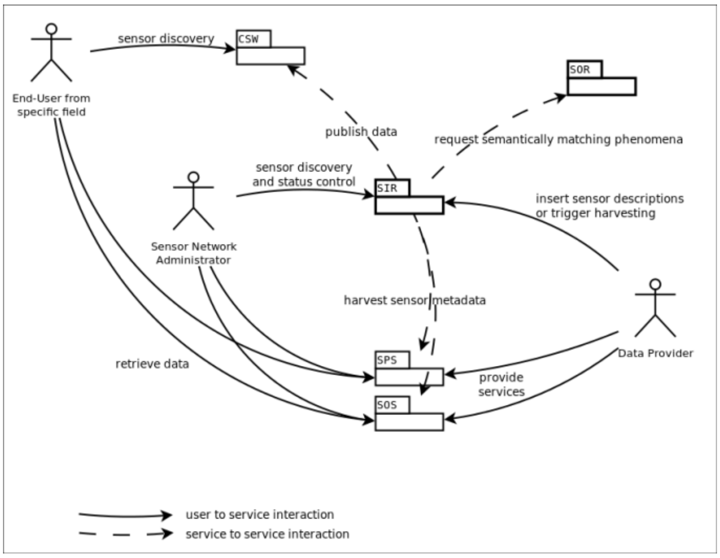

Figure 2.10: Overview of the interaction between CSW, SIR, SOR and potential users [52 North, 2016]

The `DescribeRecord` request allows the client to recieve a description of (a part of) the information model inside the catalog service. Parameters for namespaces or type names can be added to retrieve a part of the information model. The optional `GetDomain` request can be used to retrieve information about the range of values for a metadata record element or request parameter. A `GetRecords` request can be made to search for and retrieve catalogue records or to see if a they are present. The query element is the encoding for the search part of the `GetRecords`. The constraint parameter contains the query element and the constrain language parameter set the language that is being used to make the query. To see whether a record is present the outputSchema parameter and the 'ElementName' or 'ElementSetName' parameter(s) should be used. The `GetRecordsByID` request returns records using their identifier.

The `Transaction` request can be used to create, modify and delete catalogue records. HTTP GET requests using KVP are not supported for this operation, because it is not a convenient way to encode the transaction payloads. Instead POST requests should be used exclusively for this. The transaction element in the request contains the type of action: insert, update and/or delete.

2.3.2 Sensor Observable Registry

The SOR is "a web service interface for managing the definitions of phenomena measured by sensors as well as exploring semantic relationships between these phenomena" [Jirka and Bröring, 2009, p. vi]. This is a web service developed by OGC to enable semantic reasoning on sensor networks, especially concerning phenomenon definitions. This should make it easier to discover sensors that observe a certain phenomenon and to interpret sensor data.

The SOR has four different requests: `GetCapabilities`, `GetDefinitionURNs`, `GetDefinition` and `GetMatchingDefinitions`. The `GetCapabilities` request provides an overview of what the SOR has to offer. The capabilities document that is returned to the client contains four sections. These sections are required for every OGC geoweb service and the first three are the same as described in Paragraph 2.3.1. The fourth section is the content section. This part of the capabilities document contains the following information: the number of entries inside the SOR instance, keywords describing the content of the SOR instance, the application domain for which a specific SOR can be applied and an 'ontologyRepositoryURL'. This URL points to a repository that contains an ontology used by this SOR.

`GetDefinitionURNs` returns a list of Uniform Resource Names (URNs) identifying the definitions that are present in the SOR. Optionally a client can add a 'SearchSensor' parameter to filter URNs. This parameter takes a substring that shall occur within the definition URNs to be returned. Also, a maximum limit for the amount of returned URNs can be added using the 'maxNumberOfResults' parameter. The optional parameter 'startResultElement' can be used to input the number of the first returned result element. For retrieving the definition of a specific URN a `GetDefinition` request can be made. This request takes an URN of the phenomenon for which a definition should be retrieved as input. The definition is returned as a GML dictionary entry.

`GetMatchingDefinitions` allows clients to retrieve definitions of observables which in some way are related to another given phenomenon. The relations 'generalization', 'specialization' and 'equivalency' are currently supported for finding matching definitions [Jirka and Bröring, 2009] and have to be specified in the request using the 'matchingtype' parameter. This parameter can take one of three values: SUPER_CLASS, EQUIVALENT_CLASS or SUB_CLASS. Additionally, the request can have the 'searchDepth' parameter, which represents the maximum amount of steps that are allowed in case of a transitively related phenomena.

2.3.3 Sensor Instance Registry

Another web service interface specification by OGC is SIR. SIR is aimed at "managing the metadata and status information of sensors" [Jirka and Nüst, 2010, p. xii]. The goal of this web service is to close the gap between metadata models based on SensorML, which is used in SWE, and the metadata model used in OGC catalogue services. Furthermore, it provides functionalities to discover sensors, to harvest sensor metadata from a SOS, to handle status information about sensors and to link SIR instances to OGC catalogue services.

There are 14 different requests that can be made at a SIR. First of all, there is the `GetCapabilities` request, which provides an overview of what the service has to offer. For searching and retrieving sensor metadata there are the `SearchSensor` and `DescribeSensor` requests. The search sensor request can take a identifier of a specific sensor that is being searched or a search criteria. The criteria can be a value of a metadata property or a spatial filter. The describe sensor request is similar to the one that is supported by SOS (see Paragraph 2.1.3). It returns the SensorML description of a specific sensor.

The `HavestService` request starts a process to retrieve all available sensor metadata from a specific SWE service. It takes the 'ServiceURL' as input, combined with a service type such as SOS, SPS or SAS. The response document of a harvest request contains a summary of the changes that were performed in the SIR database.

Transactions for individual sensors can be made using `InsertSensorInfo`, `DeleteSensorInfo` and `UpdateSensorDescription`. The insert sensor info request has two mandatory input parameters: 'SensorIDInSIR', which should contain a unique identifier for the inserted sensor and 'SensorDescription', which should contain its metadata. Optionally, a reference to an SWE service can be provided that contains a description of the sensor. The delete sensor info request requires the identifier of the sensor metadata to be deleted, together with a boolean value for whether all data should be deleted or only certain references. In case of the latter the references to be deleted should be provided as well. The update sensor description request contains the identifier of the sensor for which metadata should be updated together with the new sensor description.

There are four requests for managing sensor status information, which are optional for the implementation of a SIR. The `GetSensorStatus` request is similar to the `SearchSensor` request, but returns the status of the sensor. To retrieve specific status information a property filter can be added. The `SubscribeSensorStatus` request allows users to automatically retrieve status information of sensors. It contains the 'SubscriptionTarget' which defines where the status information should be send to. The other parameters specify what kind of status information should be received from which sensors, like a regular `GetSensorStatus` request. The response includes a subscription identifier and an expiration date. With the `RenewSensorStatusSubscription` request the expiration date can be extended. It takes the subscription identifier as input. The `CancelSensorStatusSubscription` request allows users to cancel their subscription before the expiration data. Users can also add status information themselves with the `InsertSensorStatus` request.

Managing the connection between a SIR and a OGC catalog can be done using `ConnectToCatalog` and `DisconnectFromCatalog` requests [Jirka and Nüst, 2010]. For connecting to a catalog the URL of the catalog should be provided, combined with time interval. This interval will be used to set the update time: all metadata changes that occurred in the SIR since the previous interval are pushed to the catalog service simultaneously. This connection can be cancelled with a `DisconnectFromCatalog` request.

2.4 SEMANTIC SENSOR DATA MIDDLEWARE

Henson et al. [2009] and Pschorr [2013] suggest adding semantic annotations to a SOS which they call Semantically Enabled SOS (Sem-SOS). In Sem-SOS the raw sensor data goes through a process of semantic annotating before it can be requested with a SOS service. The retrieved data is still an XML document, but with embedded semantic terminology as defined in an ontology. The data retrieved from Sem-SOS is therefore semantically enriched.

Pschorr et al. [2010] has created a prototype that is able to find sensors from a SOS using linked data. The user can input a location and find sensors that are located nearby. They acknowledge the advantages of linked data over a catalogue service. However, the method presented by Pschorr et al. [2010] is still limited to retrieving sensors from a single source in a buffer around a point location.

Janowicz et al. [2013] has specified a method that uses a Representational State Transfer (REST)ful proxy as a façade for SOS. When a specific URI is requested the so-called Semantic Enablement Layer (SEL) translates this to a

SOS request, fetches the data and translates the results back to RDF. In this method the sensor data is converted to RDF on-the-fly. This allows the data to be interpreted by both humans and machines.

Atkinson et al. [2015] have identified that "distributed heterogeneous data sources are a necessary reality in the case of widespread phenomena with multiple stakeholder perspectives" [Atkinson et al., 2015, p.129]. Therefore, they propose that methods should be developed to move away from the traditional dataset centric approaches and towards using linked data for cataloguing. This has the potential to bring together data and knowledge from different areas of research about the same (or similar) features-of-interest. It is also argued that using both linked data services and data-specific services could ease the transition into the linked data world.

2.5 SEMANTIC WEB AND THE INTERNET OF THINGS

Barnaghi et al. [2012] describe how the semantic web could be of great importance for the IoT. Jazayeri et al. [2015] describe the SOS as one of the protocols that IoT devices could use.

2.6 SENSOR DATA ONTOLOGIES

Ontologies are necessary to provide meaning to data on the semantic web and to create semantic interoperability. Three recent efforts for developing a standard ontology for sensor data based on SWE standards will be discussed here.

2.6.1 Semantic sensor network ontology

W3C has developed an ontology for sensors and observations called the Semantic Sensor Network Ontology (SSNO). This ontology aims to address semantic interoperability on top of the syntactic operability that the SWE standards provide. To accommodate different definitions of the same concepts the broadest definitions have been used. Depending on the interpretation these can be further defined with subconcepts. The SSNO is based on the stimulus-sensor-observation pattern, describing the relations between a sensor, a stimulus and observations (Figure 2.11). Sensors are defined as "physical objects ... that observe, transforming incoming stimuli ... into another, often digital, representation", stimuli are defined as "changes or states ... in an environment that a sensor can detect and use to measure a property" and observations are defined as "contexts for interpreting incoming stimuli and fixing parameters such as time and location" [Compton et al., 2012, p. 28]. The ontology can be used to model sensor networks from four different perspectives (sensor, observation, system, and feature & property), which they discuss together with additional relevant concepts.

2.6.2 Observation capability metadata model

Hu et al. [2014] have reviewed a number of metadata models (including SensorML and SSNO) for the use of earth observation (including remote sensing). They argue that all of the current metadata models are not sufficient

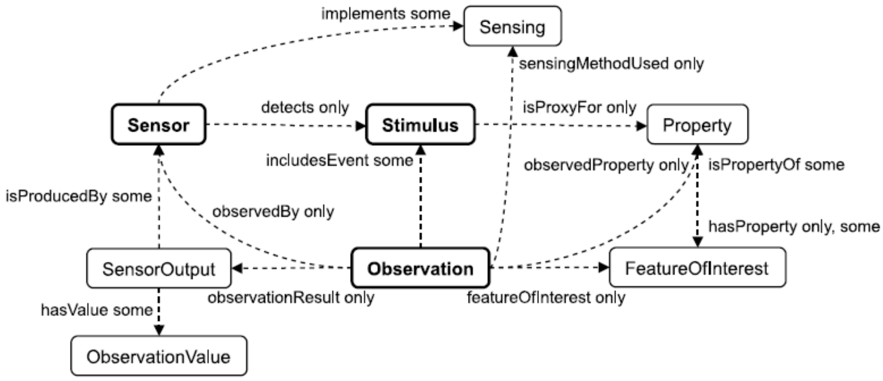

Figure 2.11: The stimulus–sensor–observation pattern [Compton et al., 2012, p. 28]

for sensor data discovery. This conclusion is based on an evaluation of six criteria. Three steps were identified in the process of obtaining relevant sensor data for earth observation, which have been used to derive criteria for their evaluation framework. These steps are sensor filtration, sensor optimisation and sensor dispatch. The filtration of sensors should result in a set of sensors that meets the requirements of the application: It should measure the right phenomenon, be active, be inside the spatial and temporal range, and have a certain sample interval. In sensor optimisation the selected sensors should be combined to complement or enhance each other. To do this, the observation quality, coverage and application is relevant. In the last step – sensor dispatch – the data should be retrieved, stored and transmitted. In every evaluated model the same sensors can be described in different ways or only partially, which affects the outcome of the sensor dispatch.

Therefore, a metadata model is proposed that "reuses and extends the existing sensor observation-related metadata standards" [Hu et al., 2014, p. 10546]. It is composed of five modules: observation breadth, observation depth, observation frequency, observation quality and observation data. They should be derived from metadata elements described using the Dublin Core metadata element set. These five modules can then be formalised following the SensorML schema which can be queried by users via a 'Unified Sensor Capability Description Model-based Engine'.

2.6.3 Om-lite & sam-lite ontologies

Cox [2015b] has been working on new semantic ontologies based on O&M. Previous efforts, such as the SSNO have been using pre-existing ontologies and frameworks. However, there are already many linked data ontologies that could be useful for describing observation metadata, such as space and time concepts. Also, the SSNO does not take sampling features into account. Therefore, Cox [2015b] proposes two new ontologies: OWL for observations or om-lite [Cox, 2015a], which defines the concepts from O&M regarding observations and OWL for sampling features or sam-lite, which defines the sampling feature concepts [Cox, 2015d]. A mapping of the SSNO to om-lite is also provided.

Cox [2015b] describes how the PROV ontology [Lebo et al., 2013] can be directly used inside om-lite. The PROV ontology is "concerned with the production and transformation of Entities through time-bounded Activities, under the influence or control of Agents" [Cox, 2015b, p. 12]. This is a very convenient ontology for modelling real world entities, such as sensors, observation processes and sampling processes. Many other ontologies could be implemented in combination with om-lite and sam-lite, depending on the kind of observations that are being modelled and the data publisher's preference.

2.7 SENSOR DATA AGGREGATION

Sensor data aggregation can be performed for two purposes: To reduce the energy constraint of sensor networks [Korteweg et al., 2007] or to sample a feature-of-interest in space and/or time [INSPIRE, 2014]. Sampling is performed when a feature-of-interest is not accessible, in which case "observations are made on a subset of the complete feature, with the intention that the sample represents the whole" [Cox, 2015a]. Stasch et al. [2011a] proposes a Web Processing Service (WPS) that retrieves sensor data from a SOS service in order to aggregate it based on features-of-interest. The approach by Stasch et al. [2011b] is similar, but takes sensor data as input that is already published on the semantic web.

Ganesan et al. [2004] stresses that spatio-temporal irregularities are fundamental to sensor networks. Irregular sampling can have a potentially large influence on the accuracy of the aggregated outcome. For example, averaging sensor data from a feature-of-interest that is being sampled densely in some parts and more sparsely in other parts could lead to inaccurate results. To counter this the values of the densely sampled area should have a lower weight than the values from the sparsely sampled area. The same holds true for temporal irregularities [Ganesan et al., 2004]. Also, Stasch et al. [2014] argue that in order for automatic aggregation to work there needs to be semantics on which kind of aggregation methods are appropriate for a specific kind of sensor data. Not all kinds of aggregation are meaningful (e.g. taking the sum of temperature values). This requires a formalisation of expert knowledge which they call semantic reference systems.

3 METHODOLOGY

A number of studies related to this thesis have been reviewed in Chapter 2. This chapter discusses why the semantic web will be used for linking sensor metadata and which methods will be used to achieve this. The SWE standards, the om-lite and sam-lite ontologies, and RDF will be described. Also a brief overview of the data use for this thesis is presented.

3.1 SENSOR METADATA ON THE SEMANTIC WEB

Sem-SOS [Henson et al., 2009; Pschorr, 2013] as well as SEL [Janowicz et al., 2013] focus on combining the sensor web with the semantic web, but do not address the integration and aggregation of sensor data. Similarly, Atkinson et al. [2015] proposes to expose sensor data to the semantic web in order to find other kinds of related data about the same feature-of-interest. Data that can be collected for another area of research. However, Atkinson et al. [2015] do not mention the integration of complementary sensor data from heterogeneous sources either. Stasch et al. [2011b] and Stasch et al. [2011a] suggest interesting methods for aggregating sensor data based on features-of-interest. However, also these studies use sensor data from only a single source into account. Moreover, Corcho and Garcia-Castro [2010] and Ji et al. [2014] argue that methods for integration and fusion of sensor data on the semantic web is still an area for future research. Data fusion is "a data processing technique that associates, combines, aggregates, and integrates data from different sources" [Wang et al., 2015a, p. 2].

Jirka and Nüst [2010] and Jirka and Bröring [2009] present methods for including SOS services in an OGC catalogue service using SOR and SIR. Making sensor metadata available in a catalogue service will improve the discovery. However, discovery through the semantic web is likely to be more effective, since links can be created towards the sensor data from many different sources of related information. Another advantage is that links can be created by everybody that publishes linked data on the web, allowing sensor data to be used for implementations that were not identified beforehand by the publisher. Also, the semantic web will be easier to access, while the catalogue service can only be requested at a certain URL which has to be known to potential users.

Since data on the web has a distributed nature it can be questioned whether centralised catalogue services are feasible to create. It places a burden on the owner of the SOS to register with a catalogue service. Also, there could be multiple of these services on the web creating issue regarding the discovery of relevant catalogues. The semantic web could solve this issue by getting rid of the 'dataset-centric' approach and adding metadata directly to the web instead.

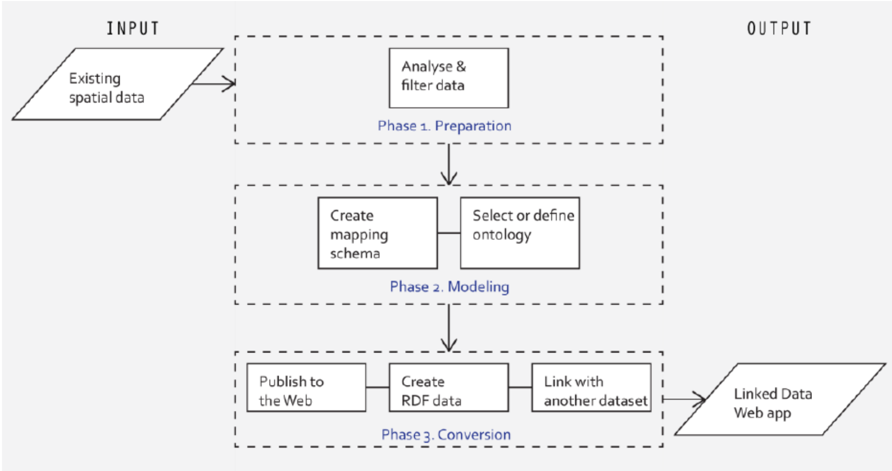

Figure 3.1: Workflow diagram for publishing linked open data from existing spatial data [Missier, 2015, p. 28]

3.2 CREATING LINKED DATA

For publishing sensor data on the semantic web a conversion of this data to RDF is required. For this the method by Missier [2015] will be used. She developed a workflow for publishing linked open data, using existing spatial data as input (Figure 3.1). This workflow consists of three phases: preparation, modelling and conversion. The next paragraphs will describe these three phases in more detail.

3.2.1 Preparation

The preparation phase deals with data acquisition, analysis and filtering. After an acquired dataset has been selected to convert to linked data it should be carefully examined. When creating linked data it is important to know the content and format of the input data. Missier [2015] explains that this understanding of the data is crucial for selecting the right ontologies and using the right software tools to process the data. Data filtering should be performed to select the parts of the dataset that need to be mapped to linked data. In this step the data quality could also be improved if necessary. The result of this phase should be a clean dataset, with semantics about the content, which has been filtered to only contain the parts of data that are required as linked data.

3.2.2 Modelling

The first step in modelling data to linked data is to select an ontology. An ontology is part of a data model, that contains semantics of how objects in the real world are mapped inside a dataset. To improve interoperability is preferred to (re)use an already existing ontology. However, if there is no suitable ontology available a new one should be created. To select an ontology all ontologies that describe the type of data inside the dataset should be listed. The one that fits the dataset and the final application best should be selected from these ontologies. Missier [2015] stresses that it is also important to reuse common predicates associated with an ontology. This makes the data more understandable and it is easier to merge with other datasets using the same ontology and predicate.

3.2.3 Conversion

In the conversion of data to linked data Missier [2015] describes two approaches: the RDF storage approach and the 'on-the-fly' conversion. In the first approach data is converted to linked data and stored in a triple store. This triple store can be queried by a SPARQL endpoint. Alternatively, data can be stored in a spatial database and converted to RDF on the fly when it is requested. There are advantages and disadvantages to both methods. In general creating a triple store creates duplicate data, which should be very well maintained to prevent the duplicate data being out of sync. On the other hand creating a triple store is a one time operation after which the querying with SPARQL is quite efficient. The on-the-fly conversion does not create duplicate data, but has to convert its data to RDF for every SPARQL query.

3.3 SPATIAL QUERIES WITH SPARQL

The SPARQL query language can perform many different kinds of queries on RDF triples. However, in SPARQL no spatial queries have been implemented. For this purpose GeoSPARQL and stSPARQL have been created. The next paragraphs will describe these two extensions of SPARQL in more detail.

3.3.1 GeoSPARQL

GeoSPARQL "defines a vocabulary for representing geospatial data in RDF, and it defines an extension to the SPARQL query language for processing geospatial data" [Perry and Herring, 2012, p. xvi]. It allows for defining geometric data in RDF and performing spatial queries. The Dimensionally Extended Nine-Intersection Model (DE-9IM) [Strobl, 2008] has been implemented to find topological relations between two geometries. GeoSPARQL has been implemented in the 'Parliament' SPARQL endpoint [Battle and Kolas, 2012].

Listing 3.1: A GeoSPARQL query to find the names of features that contain a point geometry

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT
?name
WHERE {
?feature geo:hasGeometry ?geom .
?feature foaf:name ?name.
FILTER
    (geof:sfContains(?geom,"<http://www.opengis.net/def/crs/EPSG/0/4258>
    POINT(4.289244 52.027337)"^^geo:wktLiteral))
}
```

3.3.2 stSPARQL

The Strabon endpoint uses stRDF, which is "a constraint data model that extends RDF with the ability to represent spatial and temporal data" [Koubarakis and Kyzirakos, 2010, p. 425]. The stRDF model can be queried using stSPARQL, which syntax is similar to GeoSPARQL (listing 3.1 & 3.2). Both extensions of SPARQL use filter expressions to perform spatial operations on WKT or GML geometries. The definition of geometries and the syntax of the filter expression differ slightly.

Listing 3.2: A stSPARQL query to find the names of features that contain a point geometry

```
PREFIX strdf: <ttp://strdf.di.uoa.gr/ontology#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT
?name
WHERE {
?feature strdf:hasGeometry ?geom .
?feature foaf:name ?name.
FILTER (?geom contains "POINT(4.289244
    52.027337);<http://www.opengis.net/def/crs/EPSG/0/4258>"^^strdf:WKT)
}
```

3.4 ONTOLOGIES

To publish data on the semantic web ontologies are required to specify the different classes and their relations. An ontology for static geographic data has to be connected to an ontology for sensor metadata. From the UML diagram in Figure **??** the classes Process, ObservedProperty and FeatureOfInterest can be mapped to classes belonging to OWL for observations [Cox, 2015c]. SamplingFeature and Sampling point can be mapped to classes from OWL for sampling features [Cox, 2015d]. The PROV ontology can be used for sensor and sensor observation service classes [W3C Semantic Sensor Network Incubator Group, 2011]. These ontologies will be described in more detail in the next paragraphs.

3.4.1 Observation metadata

In Paragraph 2.6 a number ontologies are described that define observations. The evaluation of observation metadata ontologies by Hu et al. [2014] is interesting, since it exposes what the relevant aspects are in the process of observation discovery. However, their proposed model focusses mainly on including remote sensing and imagery data in metadata models that were not originally created for this kind of data. The SSNO is an ontology that clearly describes the process between sensor, stimulus and observation. However, Cox [2015b] points out that an important aspect of describing a sensor network is missing in this ontology: the sampling. Also, the om-lite and sam-lite ontologies by Cox [2015b] are lightweight ontologies that can be complemented by already existing linked data ontologies. They do not rely on the (heavy) ISO specifications that date from before the semantic web,

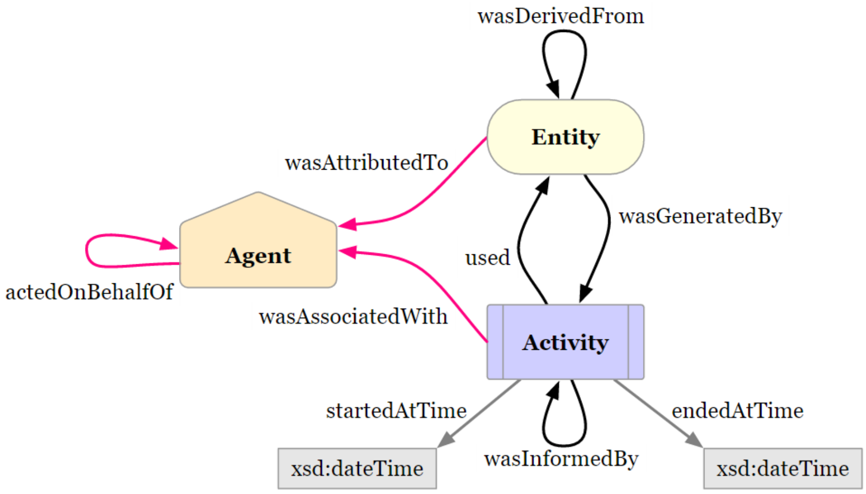

Figure 3.2: Basic classes and relations in the PROV-O by [Lebo et al., 2013]

unlike the SSNO. The om-lite and sam-lite ontologies will therefore be used in this thesis.

3.4.2 Provenance

Cox [2015b] describes that the PROV ontology can be used in combination with om-lite to keep track of changes in observation metadata. This 'PROV-O' aims to semantically define the concepts behind provenance in linked data. Provenance has to do with changes that occur over time. This ontology allows the definition of Entities, Activities and Agents. An entity is physical, digital, conceptual, or other kind of (real or imaginary) thing. An Activity is something that takes place over a period of time and acts with entities. An Agent is something that is in some way responsible for an activity taking place, an entity existing, or for another agent's activity. Figure 3.2 shows the relations between Entities, Activities and Agents.

3.5 WEB PROCESSING SERVICE

The OGC Web Processing Service is a standard interface for making simple or complex computational processing services accessible as web services. Originally, it has been created with spatial processes in mind, but it can also be used to insert non-spatial processing into a web services environment [Open Geospatial Consortium, 2015, p. 8]. Via the WPS jobs can be controlled and monitored, which run certain processes (Figure 3.3). Similar to WFS, WPS and SOS, the WPS has requests for retrieving metadata: `GetCapabilities` and `DescribeProcess`. On top of that there are is the `Execute` request to execute a process, and a number of other requests for various purposes: `GetStatus`, `GetResult` and `Dismiss`. All requests will be briefly described in this paragraph.

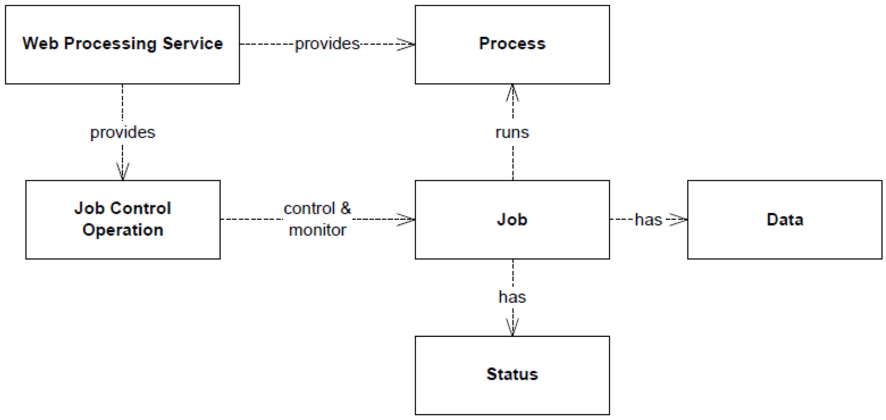

Figure 3.3: Artifacts of the WPS service model [Open Geospatial Consortium, 2015, p. 15]

3.5.1 Get Capabilities

All OGC web services give an overview of what they have to offer using the so-called `GetCapabilities` request. The request can be made by taking the HTTP address of the WPS and adding: `service=wps&request=getcapabilities`. This returns a document with information about the service metadata, the basic process offerings, and available processes. Figure shows the model of this capabilities document. The service identification section of the document gives a description of the service, including the versions that it supports and potential fees or access constraints. The service provider section gives information about the organisation or people who maintain the WPS and also includes their contact information. The operations metadata lists the different requests that are implemented in this particular WPS and the HTTP addresses to which the GET and POST requests can be send to. The content section of the capabilities document provides an overview of the available process offerings. For every process an identifier, title and description are listed. Optionally an extensions section can be added that describes additional service capabilities. At the end of the document the language section lists the languages that are supported and the default language that is being used. An example of an capabilities document can be found in Appendix B.1

3.5.2 Describe Process

A more detailed description of a process listed in the capabilities document of the WPS can be retrieved using the `DescribeProcess` request. This request requires the identifier of the process to be passed as a parameter. Optionally, a specific language can be requested for the response document (from the list of available languages in the capabilities document). The process description includes information about input parameters and output data, such as their identifiers, data type, mime types or default values. A describe sensor request can be made by putting the base address of the WPS and adding: `service=wps&version=1.0.0&request=describeprocess&identifier=an_identifier` where the version parameter should be in accordance with the supported version(s) from the capabilities document and the identifier should be taken from the process offerings

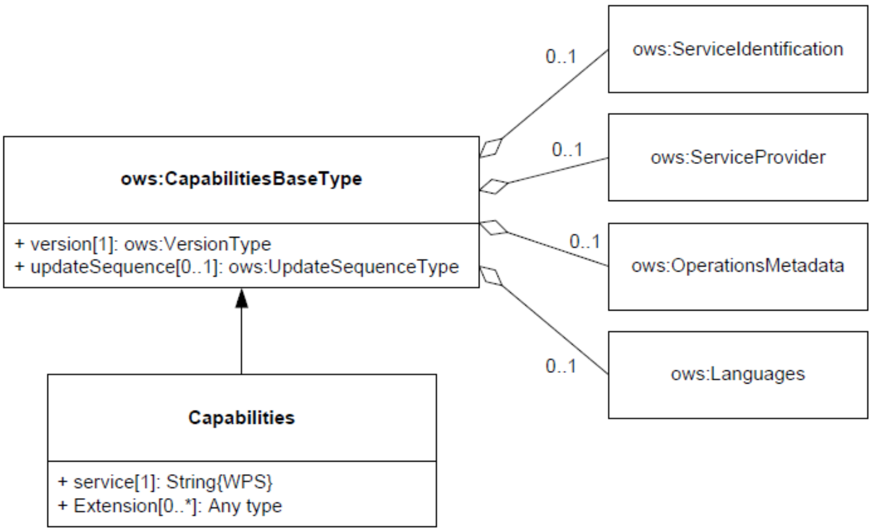

Figure 3.4: UML model of WPS capabilities document [Open Geospatial Consortium, 2015, p. 70]

section of the capabilities document. An example of an describe sensor response document can be found in Appendix B.2

3.5.3 Execute

A WPS process can be started using the `Execute` request. To make this request the HTTP address of the WPS is extended with: `service=wps&version=1.0.0&request=execute&identifier=GetSensors` where the version parameter should be in accordance with the supported version(s) from the capabilities document and the identifier should be taken from the process offerings section of the capabilities document. Additionally, the desired execution mode can be added to the request (synchronous, asynchronous or auto) as well as the desired output format (response document or raw data). By default the execution mode is set to 'document'.

Depending on the individual requirements of the process, input parameters can be added using `&DataInputs=[parameterName1=value1;parameterName2=value2]`. The parameter names are defined in the describe process response document, as well as the allowed values for each parameter. There are two kinds of input parameters that can be put in a `Execute` request: literal and complex data inputs. Literal data inputs can be a string of characters in consisting of different data types (e.g. Double, Integer, String), a given value range, and associated units (e.g. meters, degrees Celsius) [Open Geospatial Consortium, 2015, p. 36].

Complex data inputs are made for inputting complex vector-, raster- or other kind of data. This data can be inserted directly in the request or indirectly by referencing to a file. The process will then first retrieve this file from a remote server before running. A complex data input defines the allowed mime types that the process accepts.

When the process finishes an execute response document is retrieved. This document has a process section with the identifier, title and abstract of the finished process. It also contains a status section with the time the process ended and whether it finished successfully. If any output data has been produced an 'ProcessOutputs' section is created that contains the identifiers of the outputs and the corresponding data. An example of an execute response document can be found in Appendix B.3

3.5.4 Other requests

WPS processes can run synchronously or asynchronously. With a synchronous execution the connection with the client stays open until the process has finished. However, for processes that take longer to execute the asynchronous mode is better suited. The process will continue running after the connection has been closed. With a `GetStatus` request the client can check whether the process is still running. This request is structured the same as the execute request, but with the mode set to 'status'. Once it has finished the `GetResult` request allows the client to retrieve the output data. The `Dismiss` request can be made to communicate to the server that the client is no longer interested in the results of a job. This job will then be cancelled and its output deleted. A job identifier is a required parameter for all three of these requests.

3.6 DATA

A number of datasets have been used in this thesis. The following paragraphs will go over the different datasets, describing their sources and contents.

3.6.1 Vector data

Topography

The datasets of Dutch provinces (provincies, Figure A.2) and municipalities (gemeenten, Figure A.1) have been downloaded from `https://www.pdok.nl/nl/producten/pdok-downloads/basis-registratie-kadaster/bestuurlijke-grenzen-actueel`. For the Netherlands there are 12 features in the provinces and 393 in the municipalities dataset.

It has been challenging to obtain data of administrative boundaries of Belgium (even from the INSPIRE data portal). Therefore, all data for Belgium was retrieved from `http://www.gadm.org/`. There are also 12 features in the provinces (including the capitol region of Brussels) and there are 589 features in the municipalities dataset.

The country datasets have also been downloaded from `http://www.gadm.org/` (Figure A.3). The administrative unit data contains the names of the administrative units and their (polygon) geometry.

Land cover

Data on land cover will be used to complement the data of administrative units. A section of the 2012 dataset from the Coordination of Information on the Environment (CORINE) programme will has been selected for this (Figure A.4). The entire CORINE dataset was retrieved from `http://land.copernicus.eu/pan-european/corine-land-cover/clc-2012`. The features overlapping the Netherlands and Belgium have been retrieved from this dataset using the open source QGIS software and stored in a separate database in Postgres.

The database contains polygon geometries (Figure A.5) with a unique identifier and a code that refers to the type of landcover. These codes can be looked up in the accompanied spreadsheet file containing the legend table of CORINE 2012.

3.6.2 Raster data

Data is often used in a raster representation for computations in a Geographical Information System (GIS). For natural phenomenon a raster representation is especially well suited. The European Environment Agency (EEA) reference grid is a standard grid which covers Europe. It is available with a resolution of 100km², 10km² and 1km². In this thesis the EEA grid cells with a resolution of 100km² (Figure A.6) and 10km² (Figure A.7) have been used that overlap the Netherlands and Belgium. 15 grid cells of 100km² and 843 grid cells of 10km² have been selected from the original dataset.

3.6.3 Sensor data

Air quality sensor data will be used from the RIVM (`http://inspire.rivm.nl/sos/`) and from the Belgian interregional environment agency (IRCEL-CELINE) (`http://sos.irceline.be/`). Both of these organisations have a SOS where data can be retrieved according to the SWE standards. The one of the RIVM has been online since the 21st of August, 2015. IRCEL-CELINE already made the SOS available on the first of January, 2011. Figure A.8 and Figure A.9 show the sensor networks of both organisations. They provide different kinds of sensor data, such as particulate matter (PM_{10}), nitrogen dioxide (NO^2) and ozone (O^3). Figure A.10 shows one of the sensor locations in the city center of Amsterdam.

3.7 PREPARING LINKED DATA

Linked data has been prepared that is used to retrieve and process sensor data on the semantic web (Figure 3.5). This is done for vector data sets of administrative units and land cover features, and for raster data sets of EEA grids with a resolution of 10km² and 100km².

Three types of administrative units have been converted to linked data: countries, provinces and municipalities. Every administrative unit has a name, 'type' and (multi)polygon geometry assigned to it (Figure 3.5). The administrative unit type is defined by DBPedia URIs of country, province and municipality.

The CORINE 2012 land cover dataset contains features with an identifier, a land cover type and a (multi)polygon geometry (Figure 3.5). The identifier has the form of: 'EU-' plus a unique seven digit number. The land cover type is defined by a three digit number, which can looked up in the provided spreadsheet containing the legend.

The EEA reference grid with resolutions of 10km² and 100km². Every feature is defined by an identifier, a resolution and a point geometry of the origin (Figure 3.5). The identifier is a code given to a feature by the EEA and has the form of: resolution + 'E' + x coordinate + 'N' + y coordinate.

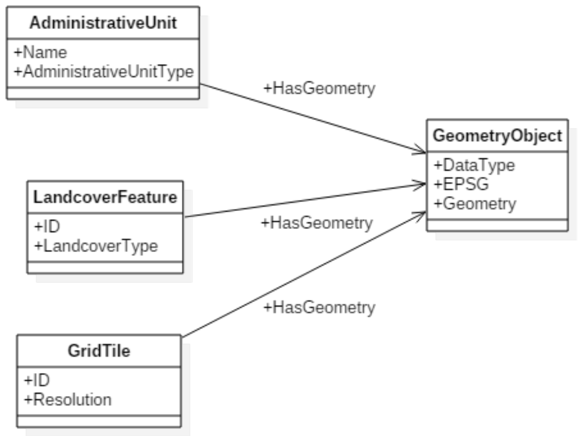

Figure 3.5: Model of vector and raster features

4 DESIGN

This thesis aims to design a method that uses the semantic web to improve sensor data discovery as well as the integration and aggregation of sensor data from multiple sources. The outline of this method will be described in this chapter using the existing methods from Chapter 3.

4.1 CREATING LINKED DATA FROM SENSOR METADATA

The process of automatically creating linked data from sensor metadata is shown in Figure 4.1. A WPS contain processes for retrieving metadata, converting it to linked data and outputting it to a triple store. Data from a SOS is retrieved by a WPS process. This process converts it to linked data. The output is an RDF document containing the metadata as triples. These documents are posted to a SPARQL endpoint, where they can be queried.

The workflow of this WPS process is shown in Figure 4.2. It is an adaption of the workflow by Missier [2015], which was originally intended for creating linked data about vector parcel data. The input of the process should be the HTTP address of a SOS. Since both the requests and the data model are standardized in a SOS the process should be able to automatically perform the tasks for creating linked data. The first step is to make requests to the SOS to retrieve it's metadata. This results in a number of XML documents that need to be filtered. The second step is to map the data inside these XML documents to linked data ontologies. In the final step RDF documents are created of the mapped metadata and published on the web.

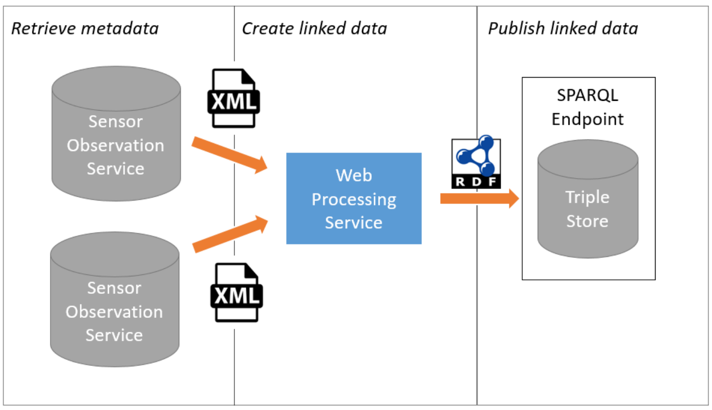

Figure 4.1: General overview of creating linked data of metadata from Sensor Observation Services

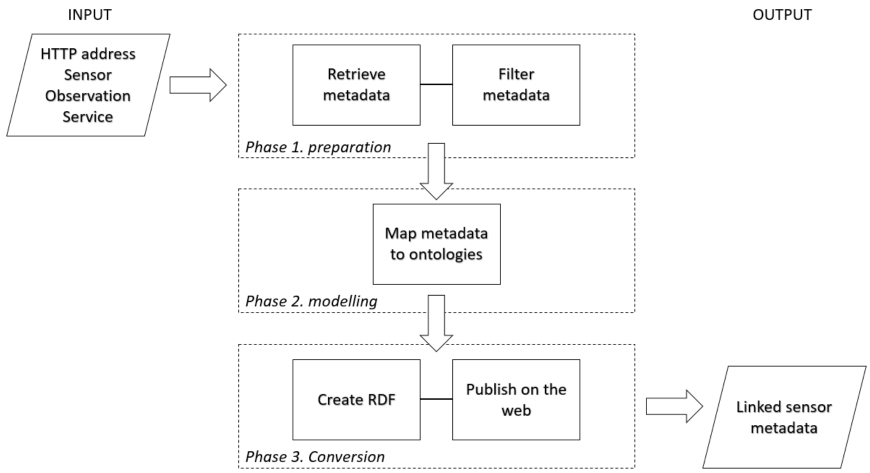

Figure 4.2: Workflow diagram of web process for creating linked sensor metadata (adapted from Missier [2015])

4.1.1 Retrieving metadata from the Sensor Observation Service

The first step of creating an online knowledge base with sensor metadata is to retrieve the metadata from the different Sensor Observation Services. This data has to be understood in order to map it to an ontology and it should be filtered to only contain the required parts of data. The next paragraphs will describe the way sensor metadata is modelled in a SOS, with which requests it can be retrieved and how it should be filtered.

Sensor metadata model

A sensor observation service describes a number of its properties that are required to know in order for clients to request data from it. It identifies the organisation that maintains it, with at least the organisation's name and its contact information. Optionally, the organisations website, keywords and an abstract about the SOS can be supplied. The SOS also describes its identifier and HTTP addresses (the address for sending requests can differ for POST or GET requests). It also lists the SOS versions and response formats it supports. The access constraints and fees are also mentioned. In most cases the use is free of charge and without access constraints. However, it is possible for an organisation to restrict the use of the SOS in these ways.

In the SWE standards a sensor is modelled using two entities: a procedure and a Feature of Interest (FOI). The procedure is the method of sensing and the FOI is the feature of which the sensor is sensing a certain property. Therefore, the observable property ties together the procedure and feature of interest. It should be noted that the geometry of a FOI is not necessarily always a point geometry. It can also be generalized into larger features (e.g. multiple sensors observing different parts of one lake).

In version 2.0 of the Sensor Observation Service Interface Standard [Bröring et al., 2012] an offering is defined as a grouping of FOIs, which have a common procedure. The constraint of sharing the same procedure has been added in version 2 of this document to solve the ambiguity of offerings in SOS 1.0. The purpose of offerings is to allow users to query the observation data more efficiently. FOIs that are often queried together are therefore grouped into the same offering for efficient retrieval.

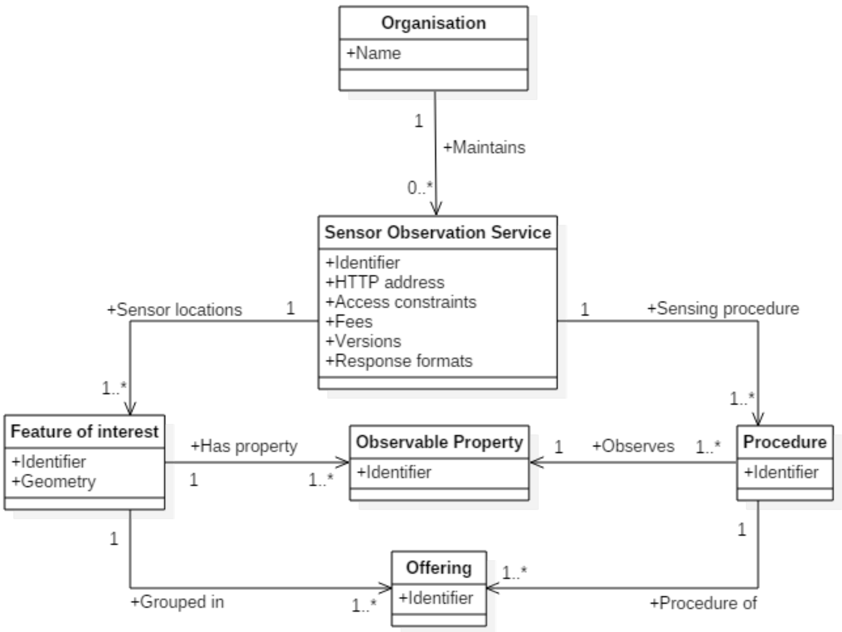

Figure 4.3: Sensor metadata derived from a SOS

Metadata Requests

To retrieve metadata from a SOS a `GetObservation` request is made first. This is a request with a very generic structure. The GET request is created by adding `service=SOS&request=GetCapabilities` to the HTTP address of the SOS. For example, the RIVM has its SOS at the address: `http://inspire.rivm.nl/sos/eaq/service?`. Therefore, the capabilities document can be retrieved using the following URL: `http://inspire.rivm.nl/sos/eaq/service?service=SOS&request=GetCapabilities`.

This requests returns the capabilities document of the SOS (see Paragraph 2.1.3). This lists the identifier of each FOI, each procedure and each observed property. It also has a section where the offerings that it contains are being described. This description of an offering includes a unique identifier, a procedure and the corresponding observed property. Additionally, descriptions can be added such as a bounding box, temporal range, FOI type and response format.

Unfortunately, the capabilities document is not able to provide information about which procedure is being applied for which feature of interest. This is crucial information for knowing which deployed sensors can be queried using a particular SOS. Also, the features' geometries cannot be retrieved from the capabilities document. Based on that document it is not yet clear which sensor locations are being used and what is measured at a specific location. Therefore, a `GetFeatureOfInterest` request can be made to retrieve the location of each FOI. Such a request can be made by adding `service=SOS&version=2.0.0&request=GetFeatureOfInterest`. The version KVP should correspond to the version declared in the capabilities document. A pointer to a specific FOI is optional and usually all FOIs are returned by default. Using the example of the SOS by the RIVM the `GetFeatureOfInterest` request looks like this: http://inspire.rivm.nl/sos/eaq/service?service=SOS&version=2.0.0&request=GetFeatureOfInterest.

On the one hand, a `GetFeatureOfInterest` document does not necessarily provide information about the procedures that are related to a certain feature of interest. On the other hand, a `DescribeSensor` request does not always relate the process to a FOI either. However, a `GetObservation` requests return observation data grouped per feature of interest. Therefore, small amounts of data can be retrieved from each offering using `GetObservation` requests to link the FOIs to procedures and observed properties. When possible a temporal filter should be used to limit the data traffic. Using this method every procedure and offering can be related to a set of FOIs with their corresponding geometry. This represents the collection of sensor devices of which data can be retrieved by sending requests to the SOS.

Filter Metadata

The documents that are returned by the SOS contain a lot of information. In some cases the returned information can be limited by adding filters to the requests. However, not all SOS have supported all filters and not all unnecessary data can be filtered out. Therefore, the XML documents that are returned should often be filtered on the client side. In a XML document every element should be defined using a namespace. Often these prefixes are defined in the xmlns tag at the top of the document to refer to these namespaces. These namespaces and corresponding tags can be used to filter the response documents for the content that is required. It should be noted that there are multiple namespaces that could be used to define the same concept. However, the potential namespaces that can be encountered are restricted by the schema describing the content of a response document. This schema is usually referenced to in the start tag of the response document.

4.1.2 Modelling with the om-lite and sam-lite ontologies

After the metadata has been retrieved from the SOS and filtered (Figure 4.3) it has to be mapped to linked data ontologies. For this the om-lite and sam-lite ontologies are being used in combination with the PROV and GeoSPARQL ontologies. Figure 4.4 shows that the om-lite and sam-lite can be used to describe classes of Figure 4.3 from an observation perspective. Figure 4.5 shows that classes of Figure 4.3 can be described from a provenance perspective as entities, agents and processes.

A SOS is modelled as an agent with a specific name, that acts on behalf of a certain organisation. The organisation, access constraints, fees, versions and response formats are properties of the SOS. Every sensor is described by a procedure and a certain feature of interest. The sensor class was not present in the model of Figure 4.3, because the SOS does not define sensors. However, the sensor class has been added to the semantic model to make the relation between procedure and sampling point explicit.

In Figure 4.4 the collection of sampling features is added. These are collections of all FOIs from different Sensor Observation Services of which the same property is being observed. The sampling collections are currently modelled to only contain sampling points. This is because the FOI of an air quality sensor is equal to the bubble of air directly around the sampling point. Other types of sampling features can be added when the application requires this. The offering class is modelled as a specialization of the collec-

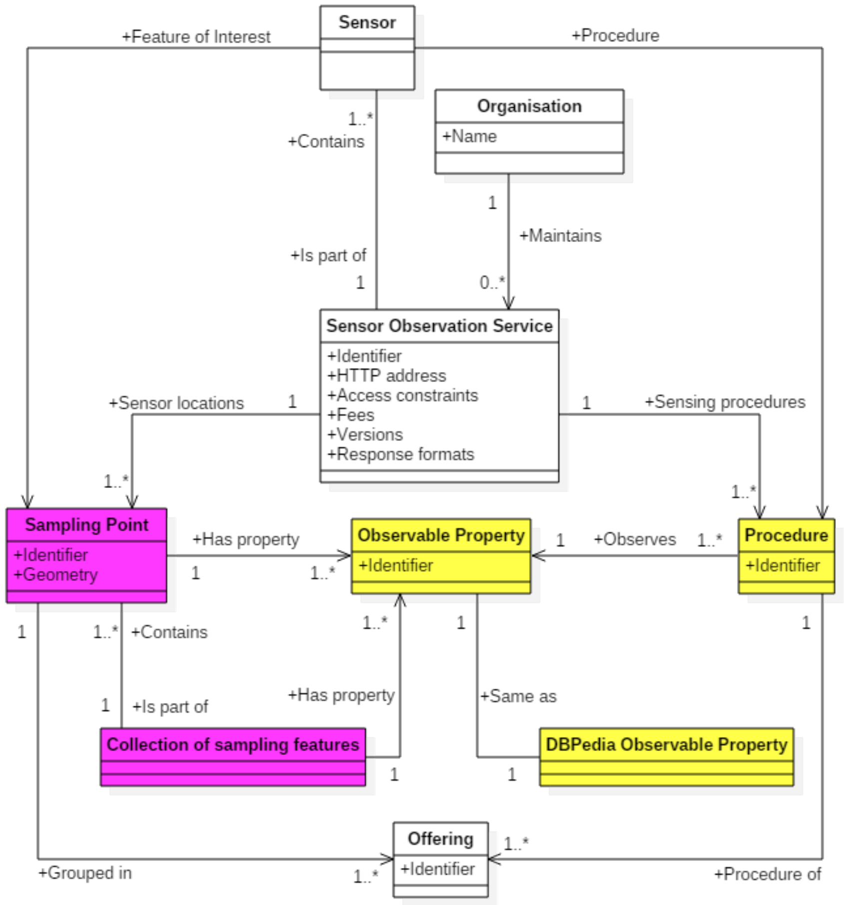

Figure 4.4: Sensor metadata modelled with the om-lite (classes in yellow) and sam-lite (classes in purple) ontologies

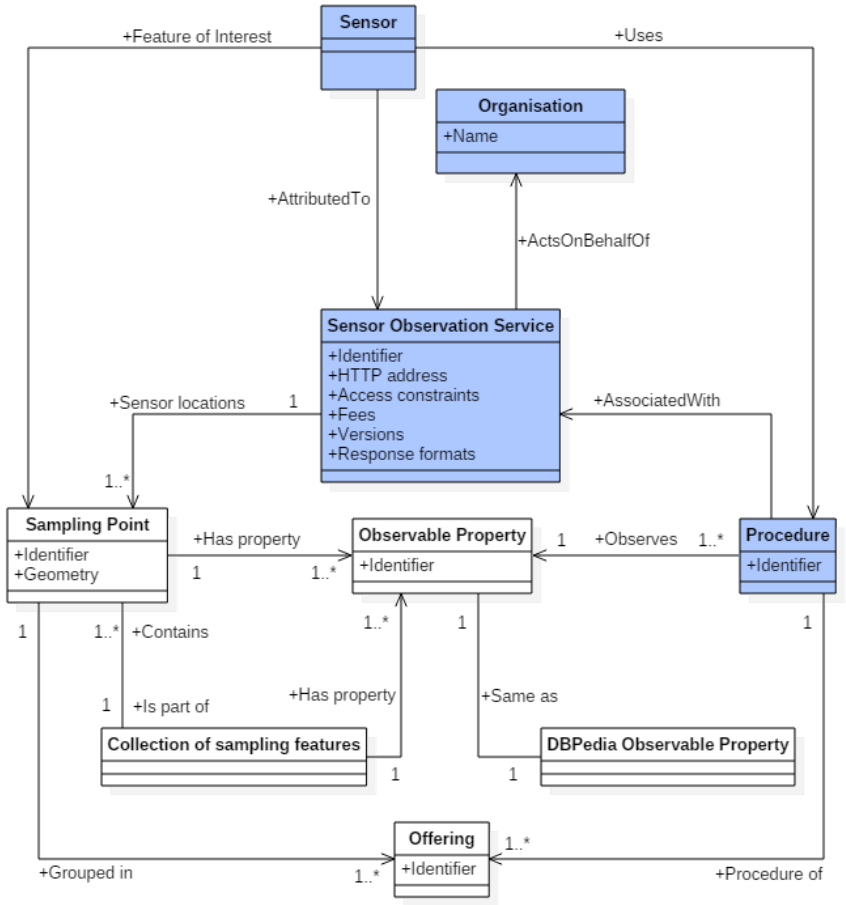

Figure 4.5: Sensor metadata modelled with the PROV ontology (PROV classes in blue)

Table 4.1: Types of PURLs [Shafer et al., 2016]

PURL Type	Meaning	HTTP Shorthand
301	Moved permanently to a target URL	Moved Permanently
302	Simple redirection to a target URL	Found
303	See other URLs (use for Semantic Web resources)	See Other
307	Temporary redirect to a target URL	Temporary Redirect
404	Temporarily gone	Not Found
410	Permanently gone	Gone

tion of sampling features. It contains a subset of the sampling points that are part of the same offering at a particular SOS.

The last class that has been added to the model as shown in Figure 4.4 with respect to the model from Figure 4.3 is the DBPedia observed property class. Every observed property that is defined in a SOS relates to a certain observed property as defined by DBPedia. Since SOS requests require their own identifiers as input the observed property class exists twice in the model: one as defined by the SOS and one as defined by DBPedia. For the same reason all sampling points, processes and offerings have a 'name' attribute in addition to their URI. These store the original (non-semantic) identifier that they were given by the SOS.

4.1.3 Output linked sensor metadata

Once the data has been retrieved and mapped to their corresponding classes in the ontologies RDF triples can be created to link the data together. These triples should be stored in files and posted to a SPARQL endpoint. The following two paragraphs will describe these steps in more detail.

Create RDF

For every mapped part of metadata from the SOS one or more triples are created. These triples consist of at least of URIs. However, preferably they are URLs that can be resolved to an RDF document that contains semantic information about what it represents. To do so every part of metadata is automatically assigned a Persistent Uniform Resource Locator (PURL). For this kind of URL to resolve a PURL server should be set up. This server performs one of the six tasks shown in Table 4.1 whenever a PURL is being retrieved. The structure of a PURL consists of the HTTP address of the PURL server, with a unique identifier attached to it. For example, `http://www.examplePURLServer.com/unique_identifier`.

Once the metadata is assigned PURLs the triples can be created. Since the metadata of the sensors is being returned by the SOS in a structured way links can be created between URLs of FOIs, observable properties, procedures and the other classes shown in Figure 4.4. After these triples have been created the linked data should be serialized to an RDF document. For this a specific notation has to be selected.

Publish RDF on the web

To store the linked data created under the previous paragraph a SPARQL endpoint has to be set up. This endpoint is connected to a triple store. This means that the RDF documents containing linked data can be send to it using POST requests. After it has been stored in the triple store users can query the triples using the SPARQL query language.

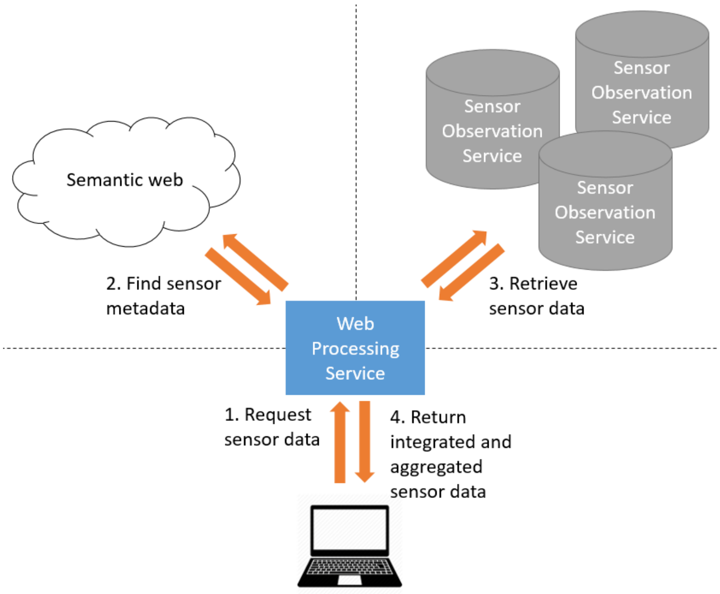

Figure 4.6: General overview of creating linked data of metadata from Sensor Observation Services

Once the linked data is inside the triple store the PURLs can be redirected to SPARQL `Describe` queries. A describe query takes a URI as input and returns all the triples that contain this URI as either an subject, predicate or object. This means that all information that the endpoint has about a particular URI is being returned. If the PURL server receives a request for a particular PURL it makes a `Describe` query to the endpoint and returns to the client all the linked data that it retrieved about this PURL.

4.2 USING LOGICAL QUERIES TO RETRIEVE SENSOR DATA

The second web process looks on the semantic web for sensors that observe a certain property in a specific area. It collects the data for these sensors at their corresponding Sensor Observation Service. When multiple data sources are found the data is integrated into a single dataset. The sensor data is temporally aggregated before it is returned to the user. Optionally, spatial aggregation can be performed as well.

Figure 4.6 shows the process of retrieving sensor data. The workflow of this web process is described in Figure 4.7.

4.2.1 Discovering sensors

For discovering sensors there are a number of input parameters for the second process: An observed property, a set of names of spatial features, a temporal range and granularity. Additionally, a type of spatial aggregation can be added. The input data is inserted in a number of SPARQL queries. First the geometry is retrieved for all features inside the input list of features. This is done using the name or identifier attribute of spatial features at the endpoint.

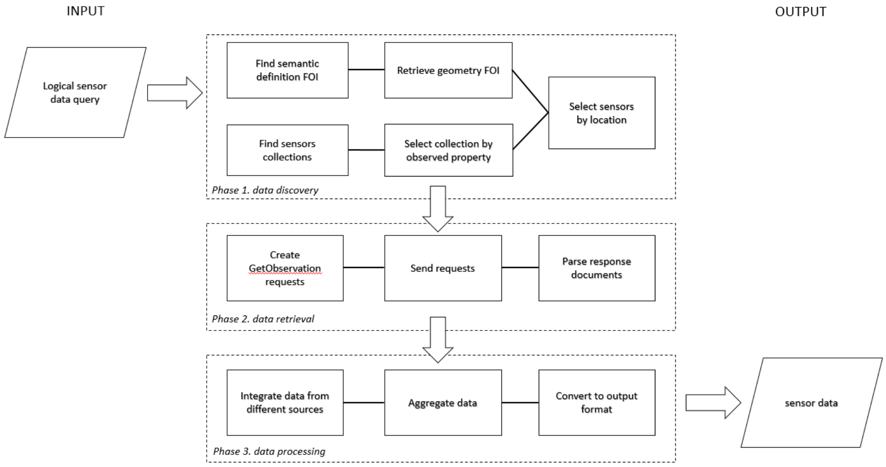

Figure 4.7: Workflow diagram of web process for retrieving sensor data

Then a second SPARQL query selects the sensors that are part of a collection that observe a certain property of the FOIs. This query also contains a spatial filter with the retrieved geometries. This makes sure only sensors in the requested areas are returned. For all discovered sensors the corresponding SOS URL, observed property, offering and process identifiers are retrieved. These original SOS identifiers are required for creating `GetObservation` requests.

4.2.2 Retrieving sensor data

The SPARQL queries return a table with all necessary information per sensor on each row. The columns of this table are: sensor URI, FOI geometry, FOI identifier, procedure identifier, observed property identifier, offering identifier, and the last column contains the URL of the SOS. The data that is returned by the SPARQL queries described in Paragraph 4.2.1 correspond to the `GetObservation` input parameters as described in Paragraph 2.1.3. This request is send out using all the values per row in the returned table of sensor metadata and is structured by taking the URL of the SOS and extending it with:

```
service=SOS&version=2.0.0&request=GetObservation&procedure=
the_procedure&offering=the_offering&observedproperty=the_
observed_property&responseformat=http://www.opengis.net/
om/2.0&featureOfInterest=the_feature_of_interest.
```

If the SOS supports temporal filters these should be added to only retrieve observation data from the temporal range that the user is interested in. To include a temporal filter the following parameters can be added to the above GET request:

```
&temporalFilter=om:resultTime,start_time/end_time
```

4.2.3 Processing sensor data

The requests described in Paragraph 4.2.2 result in a XML document with observation data for every sensor. In each document the observations have to be retrieved, with their corresponding result time, phenomenon time and

UOM. All of these observation have to be stored in a uniform way in another file or file-like object. If the temporal filter was not supported by the SOS the observation should only be stored if it is inside the requested temporal range. Also the FOI geometry should be stored with the observation for data visualisation or spatial aggregation later on. This process repeats itself until the observations have been retrieved from all XML documents and they are all stored in a single dataset.

The next step is to temporally aggregate the observation data. For every sensor location the result time is compared to the temporal range and the value for temporal granularity. It should then be appended to a list that corresponds to a certain subset of the temporal range. After this temporal sorting has been performed all observation values in a list should be aggregated according to a user defined method.

If a spatial aggregation method is part of the user's logical query then this is performed after the temporal aggregation. For each of the spatial features for which observation data has been retrieved a single list of aggregated data is produced. This list contains the observation value for each time interval (with a length equal to the temporal granularity) inside the temporal range. The spatial aggregation is performed in a similar fashion as the temporal aggregation. First all observation data is order in a list per time interval per spatial feature. After all observations have been ordered the required aggregation method is performed to produce a single value for each of the lists.

Examples of aggregation methods that can be used for spatial or temporal aggregation are the average, median, minimum, maximum or sum. Which method is being used is dependent on the information need of a user. Therefore, these methods can be extended with more complex aggregation techniques as described by [Ganesan et al., 2004]. Also, as Stasch et al. [2014] argue a 'check' could be implemented to make sure users do not (accidentally) request a meaningless type of aggregation, such as the sum of temperature values over a certain area.

The last step is to convert the integrated and aggregated data to a desired output format. There are many different formats that could be used for returning sensor data. However, there are two output formats for which an O&M schema has been defined: it can be returned as an XML document [ISO, 2011] or as a JavaScript Object Notation (JSON) object [Cox and Taylor, 2015].

4.3 CREATE INWARD LINKS FROM DBPEDIA

5 PROOF OF CONCEPT

This chapter takes the designed method from Chapter 4 and applies it using existing software tools, and python libraries and modules to create a proof of concept. First, the implementation of the process for automatically creating linked sensor metadata will be described. After that the implementation of the process for retrieving sensor data using logical queries is described. The third part of this chapter describes how the two processes have been made available online. The final part shows the web application that has been created as an interface for the second web process.

5.1 CREATING LINKED DATA FROM SENSOR METADATA

The first step of the method described in Chapter 4 is to automatically harvest sensor metadata from a SOS using various requests. This data should then be mapped to ontologies and serialized to an RDF document. The next paragraphs will describe how these steps have been implemented using the Python programming language.

5.1.1 Making sensor metadata requests

A Python class object is created for the SOS based on Figure 4.3. This class contains the different variables and has built in functions to automatically retrieve the metadata. When a SOSclass instance is created the URL of the SOS has to be entered as input value. The initialisation of the SOSclass instance creates empty variables are created for storing information about the SOS' organisation, supported SOS versions, response formats, FOIs, offerings, observable properties and procedures. The FOIs are stored in a Python dictionary to allow the geometry to be stored, as well as information about the Coordinate Reference System (CRS), and the procedures and observed properties it is related to. Similarly, the procedures are stored in a dictionary to be able to easily access which observable property, which FOIs and which offerings are related to a procedure.

After the initialisation is finished the method `SOSclass.request()` is automatically triggered. This function starts by sending a `GetCapabilities` request to the SOS. For making HTTP GET and POST requests the Requests library for Python is used (see `http://docs.python-requests.org`). Listing 5.1 shows how to create a `GetCapabilities` request with the HTTP GET function from this library.

Based on the content that is being retrieved from the capabilities request further requests are being made. First, the geometries of the FOIs are collected using `GetFeatureOfInterest` requests. Similar to the `GetCapabilities` there are no specific parameters required except for: `service=SOS&version=2.0.0&request=GetFeatureOfInterest`. However, Listing 5.1: Creating a HTTP Get request using Python's Request library

```
import requests

sosURL = "http://example.com/SOS?"

# Create the GetCapabilities request string
GetCapabilities =
    "{0}service=SOS&request=GetCapabilities".format(sosURL)

# Send the request
r = requests.get(GetCapabilities)

# Print the response document
print r.content
```

one of the Sensor Observation Services used in this thesis had implemented their `GetFeatureOfInterest` request to always require a feature id parameter. Therefore, the following exception had to be built in case this error is returned: `service=SOS&version=2.0.0&request=GetFeatureOfInterest&featureOfInterest=allFeatures`.

After the geometries of the FOIs have been retrieved they should be linked to procedures and observable properties. As described in Paragraph 4.1.1 it is not mandatory to define these relations in the `GetCapabilities`, `GetFeaturesOfInterest` or `DescribeSensor` response documents. Therefore, small amounts of observation data are requested from the sensors using `GetObservation` requests.

5.1.2 Map metadata to ontologies

After each request is send an XML document is returned. To retrieve data from these XML documents the LXML library for Python is used (see `http://lxml.de`). With this library the XML document can be loaded into an Python object, which allows for easy XML processing, such as looping through the elements and searching the whole document for elements with specific tags. Listing 5.2 shows a snippet of code that takes the response document retrieved from Listing 5.1 and that uses the LXML library to find all offerings presented in this document. All offerings are returned as a list and stored inside the variable 'SOSclass.offerings'. With this principle all metadata from the SOS is retrieved from the XML response documents and stored for further processing.

Once all the relevant metadata has been retrieved from the SOS and stored inside the class object it should be mapped to linked data ontologies. For the Python package RDFlib is used (see `https://rdflib.readthedocs.org/`). RDFlib defines an RDF graph to which triples can be added. Listing 5.4 shows a snippet of code that defines an RDF graph and adds all procedures of a SOS to it with the type 'http://def.seegrid.csiro.au/ontology/om/om-lite#process'. A semantic URL is defined for each procedure. It is created using a combination of the name of the organisation and a unique number for each procedure. The domain of the URL points to the PURL server. The same principle of creating triples from sensor metadata is applied to all classes and relations described in Figure 4.4 and 4.5.

Listing 5.2: Creating an Etree object from an XML response document using Python's LXML library

```
import lxml

# Store the retrieved document as an Etree object
tree = etree.fromstring(r.content)

# Retrieve the namespaces from the XML document
nsm = tree.nsmap

# Find all subsets of the XML document that are inside a
# ‘sos:ObservationOffering’ element
SOSclass.offerings = tree.findall(".//sos:ObservationOffering", nsm)
```

Listing 5.3: Creating an RDF graph object with the Python package RDFlib

```
import rdflib

# The domain of the PURL server
PURLZ = "http://example.com/PURLZ"

# Create the OM-lite namespace
oml =
    rdflib.Namespace("http://def.seegrid.csiro.au/ontology/om/om-lite#")

# Initialize a graph object
g = Graph()

for i, procedure in enumerate(SOSclass.procedures):
   # Define a URIs for the procedures
   procedureURI = URIRef("{0}/{1}_PROC_{2}".format(PURLZ,
       SOSclass.organisation, i))

   # Add all procedures to the graph and define them
   # as om-lite processes
   g.add( ( procedureURI, RDF.type, oml.Process) )
```

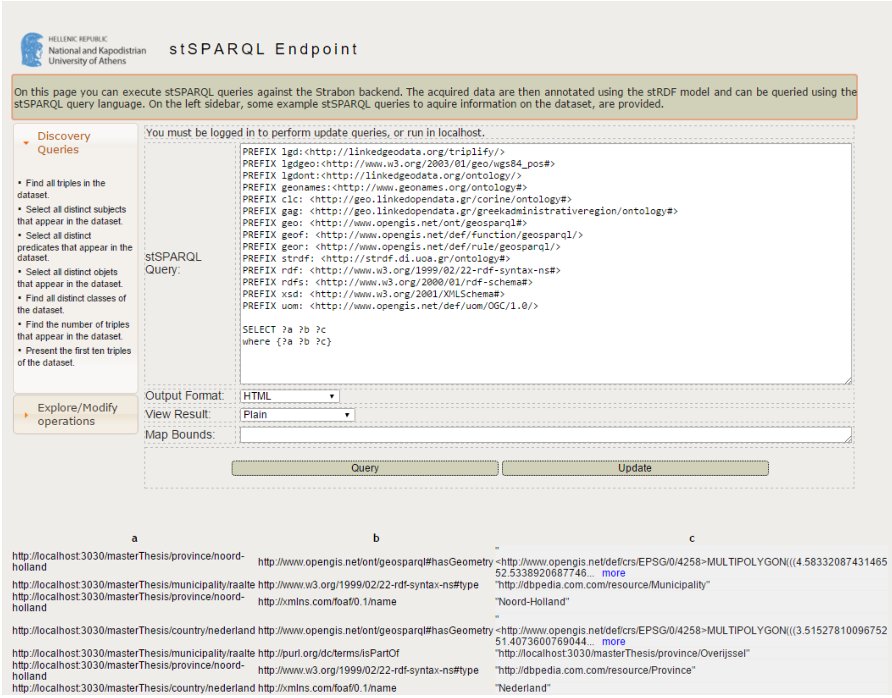

Figure 5.1: User interface of the Strabon endpoint

5.1.3 Publish linked data

For publishing linked data the Strabon endpoint (Figure 5.1) is used in combination with an Apache Tomcat server. The Strabon endpoint is a semantic spatiotemporal RDF store, originally developed for the European 'Semsor-Grid4Env' project (see `http://strabon.di.uoa.gr/`). It uses a Postgres database with the Postgis extension to store RDF triples and it allows spatial SPARQL queries using the stSPARQL extension (Paragraph 3.3.2).

The first step in publishing the graph that is created using RDFlib is to store it in an RDF document. This process is called the serialization of an RDFlib graph. To perform this process function 'serialize' of the RDFlib package is used. This function is a method of the graph object and requires two input parameters: the name of the output file and the notation of the triples. In the proof of concept the Turtle notation is being used.

After the RDF documents have been created they should be posted to the SPARQL endpoint. To post a document to Strabon the client should first login to the endpoint. To do this a Session object is created using the Requests library. This session object posts it login credentials to the endpoint, after which it can be used for posting RDF data. Listing 5.4 shows how to log in to the endpoint with a session object. After that the graph is serialized to the document 'sensors.ttl', which can then be posted to the Strabon endpoint.

For creating Persistent Uniform Resource Locators the Purlz software has been used (see `http://www.purlz.org/`). All URIs that are created get a PURL assigned to it. The PURL resolves the URI to a `DESCRIBE` query at the endpoint. This query is structured as a get request: `http://localhost/strabon-endpoint-3.3.2-SNAPSHOT/Describe?submit=describe&view=HTML&handle=download&format=turtle&query=DESCRIBE<an_URI>`. The request has '`/Describe?submit=describe`' to call the script that deals with describe queries and to tell it that the request is also submitting this type

Listing 5.4: Serializing the RDFlib graph object and posting it to the Strabon endpoint

```
import os
import requests
import rdflib

# Define the login parameters in a dictionary
login = {"dbname":"endpoint", "username":"your_username",
    "password":"your_password", "port":"5432",
    "hostname":"localhost", "dbengine":'postgis"}

# Create a new session
session = requests.Session()

# Log in with the session to the Strabon endpoint using the login
    parameters
r = session.post("http://example.com/strabon-endpoint/DBConnect",
    data=login)

# Serialize the graph to the file sensors.xml
g.serialize("sensors.ttl",format="turtle")

# Post the sensors.xml file to the endpoint
r = session.post("http://example.com/strabon-endpoint/Store",
    data={"view":"HTML", "format":"Turtle",
    "url":"file:///{0}/sensors.ttl".format(os.getcwd()),
    "fromurl":"Store from URI" })
```

Listing 5.5: Example of a PURL batch file (containing one PURL)

```
<purls>
  <purl id="/masterThesis_tudelft/responseFormat_0" type="302">
    <maintainers>
    <uid>admin</uid>
    </maintainers>
    <target url="http://example.com/strabon-endpoint/Describe?
      view=HTML&amp;handle=download&amp;
      format=turtle&amp;submit=describe&amp;query=DESCRIBE
      &lt;http://localhost:8099/masterThesis_tudelft/RIVM_PROC_0&gt;"/>
  </purl>
</purls>
```

of query. The parameters '`view=HTML&handle=download`' indicate that the endpoint's website is requested, but the returned data should be a download file instead of an HTML page. The parameter '`&format=turtle`' sets the RDF notation of the download file to Turtle and '`&query=DESCRIBE <an_URI>`' is the SPARQL query that contains the URI between brackets.

Every URI that is assigned to a part of sensor metadata (like the 'procedureURI' in Listing 5.4) is written to an XML file with the parameters: ID, PURL type, and target address. Optionally, information about the person or organisation maintaining the PURL can be added. The ID is the original URI that is being resolved to the target address. The PURL type is set to 302, which means that it redirects the client to the target address. Alternative types can be found in Table 4.1. After all URIs have been added to the so called XML 'batch' file [PURL, 2016], the file can be posted to the Purlz server. Posting XML documents to the PURL server is similar to posting RDF documents to the Strabon endpoint as shown in Listing 5.4. An example of a 'batch' file can be seen in Listing 5.5. These XML files are created using the Python package LXML.

5.2 USING LOGICAL QUERIES TO RETRIEVE SENSOR DATA

The first process described how the proof of concept implementation harvests sensor metadata from a SOS and publishes it as linked data in a SPARQL endpoint. The second process uses this data and its semantics to automatically retrieve data based on a user's logical sensor data query. This process is created around 'request' class which has been made to store all logical input parameters and automatically convert them to SOS requests and return the data in an integrated and aggregated way. The following paragraphs describe how the different parts of this process have been implemented.

5.2.1 Input parameters

The proof of concept implementation takes a number of input parameters. First of all, the observable property, which will be related to a DBPedia definition. The second parameter is the category of input features. This can be set to administrative units (country, province or municipality), land cover or raster. The third parameter is a list of input feature. This is a list of names or identifiers that correspond to the category of input features. For example, 'Delft municipality' can be used as input since it includes the feature name and it's category. The next parameter is the temporal range. This has to be a list of two ISO datetime strings representing the start and end time. The fifth parameter is the temporal granularity, represented by an integer and a datetime unit like hour(s), day(s) or week(s). For example a temporal range of '1 hour' will aggregate the data temporally to a single value per hour. The sixth input parameter is the temporal aggregation method. This is the method for aggregating data between start and end time using the provided temporal granularity. The last input parameter is the method of spatial aggregation. This method will be applied to aggregate the data based on the input features.

5.2.2 Discovering sensors

The input category is a starting point for the process to find the geometries of the input features. It creates a SPARQL query that retrieves the geometries of the features by selecting all features of the input category and filtering them by name. Listing 5.6 shows that the filter expression is defined first. This is then added to the a predefined template of a SPARQL query together with the feature category. For the currently implemented categories (landcover, raster, municipalities and provinces) the DBPedia URL is simple found by adding it to the standard method for defining URIs: `http://dbpedia.org/resource/`.

A SPARQL query is then made to find a sensor collection that is related the requested observed property. From this collection sensors are selected that overlap with the previously found geometries. Unfortunately SPARQL queries are not allowed to exceed a certain number of characters. This creates problems when querying larger vector geometries (provinces and countries). For these queries two alternatives have been implemented: using the EEA reference grid as a spatial index for vector geometries and using bounding box queries at the SOS.

For the first alternative the EEA raster cells are retrieved instead of the vector data. Only the cells are requested that overlap with the vector geometry. For these cells all sensor locations are requested. However, the result of this is that too many sensor locations are retrieved, also ones that are outside the original vector feature. Therefore the WPS performs the spatial filter and removes all locations that are outside of the requested feature.

This type of query is shown in Listing 5.7. The raster cells are added in the spatial filter expression and a DBPedia URI of an observed property is added instead of the `<<insert_observed_Property>>` placeholder. Multiple spatial features can be put in the filter expression using the logical OR operator represented by two vertical lines: || . With these two parameters the query can find all relevant sensors from different sources stored at the endpoint using the model shown in Figure 4.4. It also returns all necessary data to make `GetObservation` requests at the sensors' corresponding Sensor Observation Services.

The second alternative checks which Sensor Observation Services have sensor locations within the bounding box of the vector geometry that observe a certain property. This returns a list of Sensor Observation Services. For all of these `GetObservation` requests are made with a spatial filter.

Listing 5.6: Example script that sends a SPARQL query for retreiving geometries of input features

```
import requests

# Define the location of the endpoint
myEndpoint = 'example.com/strabon-endpoint/Query'

for i, feature in enumerate(self.featureNames):

   # Create a list with a filter expression for every name
   featureNamesList = '?name = "{0}"'.format(feature)

   # Join them together and place them inside the SPARQL filter
       function
   featureFilter = "FILTER( {0} )".format( " ||
       ".join(featureNamesList) )


# Define the SPARQL query
query = """
   SELECT
      ?feature ?geom ?name
   WHERE {{
      ?feature <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
           <http://dbpedia.org/resource/{0}> .
      ?feature <http://strdf.di.uoa.gr/ontology#hasGeometry> ?geom .
      ?feature <http://xmlns.com/foaf/0.1/name> ?name .
   {1}
   }}""".format(self.featureCategory.title(), featureFilter)

# POST the SPARQL query to the endpoint
r = requests.post(myEndpoint, data={'view':'HTML', 'query': query,
     'format':'SPARQL/XML', 'handle':'download', 'submit':'Query' })

# Print the results
print r.content
```

Listing 5.7: A spatial SPARQL query for discovering sensors and their SOS related metadata

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX strdf: <http://strdf.di.uoa.gr/ontology#>
PREFIX oml: <http://def.seegrid.csiro.au/ontology/om/om-lite#>
PREFIX saml: <http://def.seegrid.csiro.au/ontology/om/sam-lite#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dbp: <http://dbpedia.org/resource/>
PREFIX dc: <http://purl.org/dc/terms/>

SELECT
   ?sensor ?geom ?FOIname ?procName ?obsPropertyName ?offeringName
   ?sosAddress
WHERE {
   ?collection rdf:type saml:SamplingCollection .
   ?collection omlite:observedProperty <<insert_observed_Property>> .
   <<insert_observed_Property>> owl:sameAs ?obsProperty .
   ?obsProperty foaf:name ?obsPropertyName .
   ?collection saml:#member ?FOI .

   ?offering saml:member ?FOI .
   ?offering prov:specializationOf ?collection .
   ?offering foaf:name ?offeringName .

   ?FOI strdf:hasGeometry ?geom .
   ?FOI foaf:name ?FOIname .

   ?sensor oml:featureOfInterest ?FOI .
   ?sensor oml:procedure ?procedure .
   ?sensor dc:isPartOf ?sos .
   ?sos owl:sameAs ?sosAddress .

   ?procedure omlite:observedProperty ?obsProperty .
   ?procedure foaf:name ?procName .
   ?offering oml:procedure ?procName .

   FILTER(
      strdf:contains("POLGYON(( <<insert_coordinates>>
          ))"^^strdf:#WKT, ?geom)
   )
}
```

Listing 5.8: Script that creates an LXML graph object called spatialFilter to add to a `GetObservaton` POST request

```
spatialFilter = etree.SubElement(getObservation,
    "{http://www.opengis.net/sos/2.0}spatialFilter")

bbox = etree.SubElement(spatialFilter,
    "{http://www.opengis.net/fes/2.0}BBOX" )

valueReference = etree.SubElement(bbox,
    "{http://www.opengis.net/fes/2.0}ValueReference")
valueReference.text =
    "om:featureOfInterest/sams:SF_SpatialSamplingFeature/sams:shape"

envelope = etree.SubElement(bbox,
    "{http://www.opengis.net/gml/3.2}Envelope")
envelope.attrib["srsName"] =
    "http://www.opengis.net/def/crs/EPSG/0/4258"

LLcorner = etree.SubElement(envelope,
    "{http://www.opengis.net/gml/3.2}lowerCorner")
LLcorner.text = "{1} {0}".format(Xmin, Ymin)
URcorner = etree.SubElement(envelope,
    "{http://www.opengis.net/gml/3.2}upperCorner")
URcorner.text = "{1} {0}".format(Xmax, Ymax)
```

5.2.3 Retrieving sensor data

After all sensor locations have been retrieved from the SPARQL endpoint `GetObservation` requests are made. These require the identifiers that were given to the observed properties and procedures by their SOS, instead of the semantic URLs that were assigned to them. The requests are structured as explained in Paragraph 2.1.3. When possible, the request is extended with a temporal filter to only retrieve data inside the required temporal range. The output format is set to XML using the `http://www.opengis.net/om/` schema.

Listing 5.8 shows how to use the LXML python package to create a spatial filter in XML which can be added to a `GetObservation` POST request. This has been implemented in the second approach described in Paragraph 5.2.2. For all retrieved sensor data using this approach the client still has to check whether the returned FOIs are really inside the vector geometry requested by the user.

The XML documents received using the `GetObservation` are loaded into Python using the LXML package. The first step is to find all 'sos:observationData' elements inside it. These elements contain observations according to the O&M schema. There are however some implementation differences in the response documents. Some Sensor Observation Services return sensor data as an O&M measurement. Others use the 'SWE:dataArray' type. A response document with the O&M measurements contains separately nested elements for each individual observation. Every observation result has its own element defining the result value, result time, UOM, procedure, feature of interest and observed property.

The SWE data array is an array of observations that share the same metadata. For all observations that have the same feature of interest, procedure, observed property and UOM the result data is joined together into an array of results values combined with the result or phenomenon time. The result value is separated from the result or phenomenon time using a predefined 'tokenseparator'. Each individual observation in the data array is separated using a predefined 'blockseparator'. The data from the received XML documents is directly added to an individual comma separated value string per combination of observed property and UOM. When the temporal filter cannot be used all data is looped over first to remove observations outside the temporal range.

5.2.4 Data aggregation

After all observation data is retrieved it is first aggregated temporally. An empty dictionary is created to store each temporal granularity range that is inside the requested temporal range for all observations. A loop goes over the comma separated values and sorts them based on their result time per sensor location. The start time is subtracted from the result time, which results the time range from the start of the temporal range to the time of the observation. From this time range a modulo operation calculates how many times the temporal granularity fits in the time range between the start of the temporal range and the time of the observation. The start time is added to the temporal granularity times the outcome of the modulo operation to calculate the dictionary key to sort the observation by.

As soon as all the observations have been sorted the data is aggregated. For all values per key the average, minimum, maximum, median or sum is calculated. The resulting value replaces the values in the dictionary. Listing 5.9 shows how the observation data stored as comma separated value strings can be quickly aggregated to a single value. If spatial aggregation is part of the sensor data request this is performed after the temporal aggregation. Using Shapely's 9-intersection model functions the sensor locations are ordered per spatial feature. Finally, all values are aggregated per feature per temporal range. For both temporal and spatial aggregation the basic methods have been implemented such as average, minimum, maximum, sum and median. These methods can be further extending to give more reliable results [Ganesan et al., 2004] or to give more semantic meaning to the aggregation methods [Stasch et al., 2014].

5.3 SETTING UP THE WEB PROCESSING SERVICES

The proof of concept web processes described in Paragraph 5.2 and 5.1 have been added to a WPS using the PyWPS software. PyWPS is an implementation of the Web Processing Service standard from the Open Geospatial Consortium using the Python programming language. It is an open source project and aims to enabling the integration, publishing and execution of Python processes via the WPS standard (see `http://pywps.org/`).

Listing 5.10 shows how a WPS process is defined. A PyWPS process is created using the 'Process' class. Instances of this class contain all the functionality and metadata of the WPS processes. The Process class consists of two parts: the `__init__` method and the `execute` method. The `__init__` method initializes a WPS process by giving it an identifier, title, abstract, version number, together with other kinds of (optional) metadata. It also defines the inputs and outputs. The `execute` method is where the functionality of

Listing 5.9: Script that performs basic temporal aggregation methods on a comma separated value string

```
import numpy

# Example comma separated value string
csvString = "2016-06-25T09:00:00,15.2;2016-06-25T09:02:00,15.5"

# split the string into a list with individual
# observations using the block separator
observations = csvString.split(";")

# Aggregate the observations with a method selected by the user
if self.tempAggregation == "average":
   aggregatedData = (sum([float(x.split(",")[1]) for x in
       observations]) ) / float(len(observations))
elif self.tempAggregation == "minimum":
   aggregatedData = min([float(x.split(",")[1]) for x in observations])
elif self.tempAggregation == "maximum":
   aggregatedData = max([float(x.split(",")[1]) for x in observations])
elif self.tempAggregation == "sum":
   aggregatedData = sum([float(x.split(",")[1]) for x in observations])
elif self.tempAggregation == "median":
   aggregatedData = numpy.median(numpy.array([float(x.split(",")[1])
       for x in observations]))
```

the process is defined. The process in Listing 5.10 imports a class called 'SOS' with methods for retrieving metadata from a Sensor Observation Service and a function called 'linkedDataCapabilities' to convert this data to RDF.

PyWPS has been installed with the method described by Deltares [2016]. The WPS is hosted using the XAMPP software. XAMPP is an open source Apache distribution, that includes a number of useful features such as the Tomcat server (used for hosting the Strabon endpoint). A pywps.cgi file is defined that points to the location of the PyWPS installation and placed in Apache's cgi-bin folder. The last step in hosting processes such as in Listing 5.10 is to store the process definition in the 'pywps_processes' folder of the PyWPS installation and adding its identifier to the `__init__` file in that same folder.

5.4 CREATING A WEB APPLICATION FOR RETRIEVING SENSOR DATA

A web application has been created as an interface for the WPS from Paragraph 5.2. This web application has been created using. Flask is a microframework for creating web applications using Python (see http://flask.pocoo.org/). The first interface that users encounter is shown in Figure 5.2. It shows a map on which features can be selected and unselected by clicking on or by manually writing their names in the form. Also an observed property can be selected in this form.

The WPS as described in Paragraph 5.2 then looks for sensors that observe this property of their FOI and that are located in the selected area. Once

Listing 5.10: Script that defines a web proces using PyWPS

```
from pywps.Process import WPSProcess
from sosRequests import *
from linkedDataCapabilities import *

class Process(WPSProcess):

def __init__(self):

WPSProcess.__init__(self,
identifier = "LinkedDataFromSOS",
title="Creates Linked Data of SOS metadata",
abstract="""This process takes an HTTP address of a Sensor Observation
    Service (SOS) as input and converts the metadata to linked
    data.""",
version = "1.0",
storeSupported = True,
statusSupported = True)

# Adding process input
self.urlIn = self.addLiteralInput(
identifier = "input_url",
title = "Input a string containing an HTTP address of a Sensor
    Observation Service (SOS). For example:
    'http://someaddress.com/sos?'",
default = "http://inspire.rivm.nl/sos/eaq/service?",
type = "StringType"
)

def execute(self):
url = self.urlIn.getValue()

# Create SOS instance with the URL as input and
# retrieve its metadata.
sos = SOS(url)

# Create and publish linked data from the above
# retrieved metadata
linkedDataCapabilities(sos)

if (__name__ == "__main__"):
Process = Process()
Process.execute()
```

Figure 5.2: Request sensor data interface (step 1) with parameters for observed property and spatial features

the sensors have been found they are visualized on the map by orange dots which can be clicked to see their URI, the observed property URI and the HTTP address of the corresponding SOS (Figure 5.3). The form below the map is now automatically replaced by a form with parameters for spatial and temporal aggregation and selecting a temporal range (Figure 5.4). After the user has entered the required values for these fields the request can be submitted by clicking the bottom right button. The observation data is now retrieved by the WPS from the different Sensor Observation Services. After the WPS has finished a graph is returned to the user providing a visual impression of the data (Figure 5.5). The graph is created using Vega-Lite, a high-level visualization grammar developed by Interactive Data Lab of the University Washington (see `https://vega.github.io/vega-lite/`). This graph can be exported from the application as a .PNG image or the data can be downloaded using the download button.

Figure 5.3: Selecting a sensor on the map show semantic URLs of the sensor, its observed property and the address of its SOS

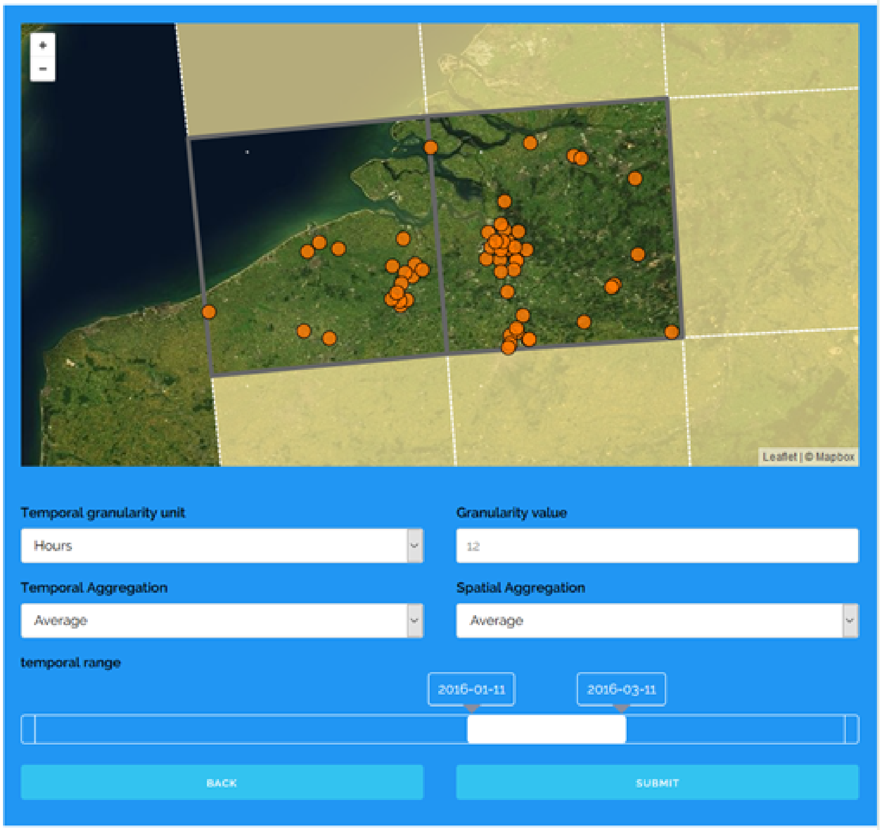

Figure 5.4: Request sensor data interface (step 2) with parameters for temporal granularity, spatial and temporal aggregation methods and temporal range

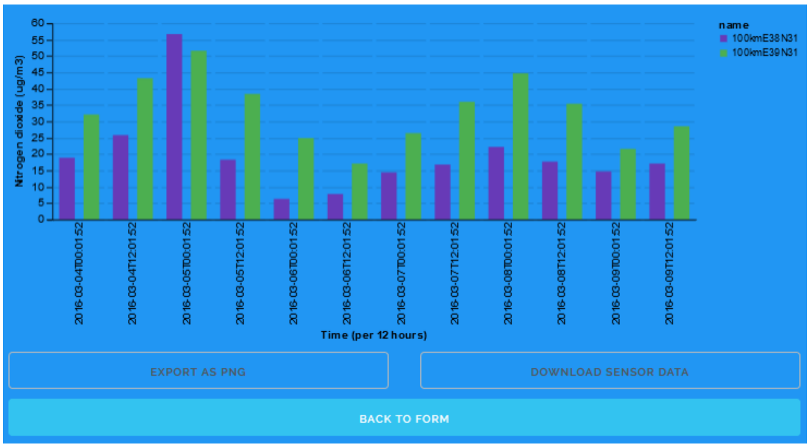

Figure 5.5: Returned observation data is visualised as a vega-lite graph, with the option to export the graph or download the data

6 RESULTS

In this thesis a method has been developed that uses the semantic web to improve sensor data discovery as well as the integration and aggregation of sensor data from multiple sources. Chapter 4 provides the outline of such a method and Chapter 5 presents a proof of concept. In the following paragraphs the outcomes and results of this method are discussed. Afterwards, a comparison is made with the Catalog Service for the Web (CSW) and with semantic sensor data middle ware.

6.1 IMPLEMENTATION DIFFERENCES BETWEEN SENSOR OBSERVATION SERVICES

The sources of sensor data used in this thesis were two Sensor Observation Services. The first SOS is maintained by the Dutch national institute for public health and the environment (RIVM). This service contains air quality sensor data for the Netherlands. The second SOS is maintained by the IRCEL-CELINE and contains air quality sensor data for Belgium. In the process of making an automated way for approaching these Sensor Observation Services a couple of implementation differences surfaced. Even though both use the SWE standards it turned out they are not exactly the same.

6.1.1 URI definitions

First of all, they have different approaches for making identifiers. The RIVM has URIs for acpfoi like `NL.RIVM.AQ/SPO_F-NL00002_00008_101_101`. Offerings are named like `NL.RIVM.AQ/STA-NL00002/38` and procedures like `NL.RIVM.AQ/SPP-NL_A_5090150901`. Their observed properties URIs are reusing the 'Eionet' vocabulary by the EEA and look like: `http://dd.eionet.europa.eu/vocabulary/aq/pollutant/1`. The IRCEL-CELINE has a different approach to these identifiers. They describe FOIs with URIs such as `BELAB01`. Procedures are simply assigned a five or six digit integer like `10607`. Observed properties have recieved identifiers which are a combination of letters and integers, such as `44201 - 03`. Offerings are named as a combination of observed property and procedures: `44201 - 03_-_6711`. Looking at both methods for creating identifiers it is clear that there is no resemblance between the two Sensor Observation Services. There is not an easy way to match the identifiers of both Sensor Observation Services. This has to do with the fact that the SWE standards typically allow 'any URI' to be provided, without further specification of its structure.

6.1.2 Content of response document

Another difference between the two Sensor Observation Services is that they supply different kinds of content in their response documents. For example, offering definitions in the capabilities document from IRCEL-CELINE contain bounding boxes to provide an indication of their geographical coverage. On the other hand, the capabilities document by the RIVM does not provide any data related to the physical locations at all. The `DescribeSensor` responses by the IRCEL-CELINE include a point geometry as part of the sensor's metadata, while the RIVM provides the FOI identifier for which the geometry can be retrieved using a a `GetFeatureOfInterest` request. This `GetFeatureOfInterest` request by IRCEL-CELINE returns the geometries of FOIs with CRSs in the `urn:ogc:def:nil:OGC:unknown` format. The SOS by the RIVM returns geometries of FOIs with CRSs in the `http://www.opengis.net/def/nil/OGC/0/unknown` format.

Another implementation difference is visible in the `GetObservation` response documents. When retrieving sensor data from the SOS using `GetObservation` requests IRCEL-CELINE provides this data as an array of comma separated values, using the SWE Array Observation class. With this method all metadata that is shared by observations from the same sensor (observed property, procedure, FOI, UOM) are only defined once in the response document. The RIVM on the other hand provides the observation data embedded in XML tags using the O&M Measurement class. With this approach all observations are self describing, defining the metadata of a sensor for each observation it makes.

All these minor differences have caused the implementation of the method described in Chapter 4 to be more complex than initially expected. Especially providing a describe sensor document with as much information as possible (containing related features of interest, observed property and offerings) is very important for making sense of the metadata. It is much better to work with the response document if it uses the appropriate SensorML classes to describe sensor metadata, instead of adding it to the more general 'swe:keywords', or abstract section for example. Especially with optional metadata it is not wrong to include it in one of these sections, but this way it is not machine understandable.

6.2 SEMANTICS IN SENSOR OBSERVATION SERVICES

The metadata in Sensor Observation Services do not necessarily contain semantic URIs as identifiers. Therefore the method of Chapter 4 includes a process for creating these semantic URIs based on the SWE XML schema's and corresponding classes in linked data ontologies. However, two issues presented themselves during this process: some identifiers used in a SOS are completely meaningless, especially from a machine readable point of view. The lack of a defined observable property is the main issue here. The second issue is the complexity of automatically creating URIs for metadata, where each URI is unique, compact and valid.

6.2.1 Mapping of observable properties

In Paragraph 6.1 the differences between identifiers in the Sensor Observation Services by IRCEL-CELINE and the RIVM have been described. They are compliant with the SWE standards, which require any URI regardless of semantics. In the method of Chapter 4 the sensor location, temporal range and observed property are modelled as the main elements of the logical query input. Therefore, ill-defined observed properties are problematic when automatically retrieving (meta)data from different services. Both Sensor Observation Services used in this thesis had non-machine understandable identifiers for observed properties. The vocabulary used by the RIVM is a step in the right direction, but the identifiers do not resolve to an RDF document, but to the EEA website. This is mainly a human understandable solution. The SOS by IRCEL-CELINE contains identifiers which are a string of seemingly random characters, ending with an abbreviation of the observable property involved. This is partly human understandable.

Identifying which observable properties in one SOS are the same as observable properties found in another SOS was not automatically possible in this case. It could be achieved with a minimum of semantics, such as an RDF document containing just a triple with the 'owl:sameAs' predicate to a linked data ontology or the 'foaf:name' predicate to store at least the name of the observable property. By assigning just a single triple like one of these to a URI it also already possible for an automatic process to find out which URIs represent the same observable property.

A manual mapping had to be implemented, because the method was currently not able to automatically create linked data from the metadata of any SOS using just the HTTP address as input. This manual mapping is a small RDF document containing one triple for every observable property URI, using the 'owl:sameAs' predicate to link it to the corresponding DBPedia definition. The proof of concept implementation takes both the HTTP address of the SOS and this manual mapping as input.

6.2.2 Automatically creating URIs

The method presented by this thesis includes a process to automatically assign URIs to non-semantic data and to define URIs for concepts that are implicitly stored in a SOS. For example, a SOS could contain one procedure and ten FOIs. The actually deployed sensors are implicitly represented by combining the procedure with a FOI, although they are not part of the SOS XML schema's. If an URI is automatically created for data from an unknown source there is an amount of uncertainty about what the resulting URIs will look like. Either a random identifier can be used, or the identifier that is already provided by the data source. This identifier will most likely contain some reference to the nature of the real world thing the URI represents. Therefore this could be preferred to a completely random identifier. However, as shown in Paragraph 6.1 identifiers can also be random or non-semantic URLs. Since the O&M schema allows simply 'any URI', very long and strange URIs can be created when adding the given identifier to the URI.

Therefore, the proof of concept implementation used a combination of random and existing identifiers. It uses the name of the SOS organisation ('RIVM', 'Ircel-Celine'), the type of metadata ('procedure', 'FOI') and a random integer ranging from zero to the amount of instances of this type. However, this is not a watertight solution either, because the name of the organisation could create an invalid URL. In the proof of concept invalid characters and spaces had to be escaped to prevent this. The name of the organisation was selected over the title of the SOS, since those are often very verbose. However, using the name of the organisation the problem could arise that multiple Sensor Observation Services are maintained by the same organisation. This could lead to non-unique identifiers. The need for more research into methods for automatically creating robust URLs for data from different sources is further described in Chapter 9.

6.3 SPATIAL QUERIES WITH SPARQL

Spatial queries are an essential part of the method presented in this thesis. They are used in the process of selecting which sensors are relevant to a user's data request. However, the proof of concept implementation showed that the performance of vector geometries in SPARQL is not very good. Also, the order of latitude and longitude in point coordinates turned out to be a problem. In the next two paragraphs these issues will be discuss, together with the implemented solutions.

6.3.1 Vector queries with SPARQL

The method described in Paragraph 4.2 starts with discovering metadata of sensors that are inside a spatial feature. This feature can be a raster cell or vector geometry. Spatial SPARQL queries with raster cells are relatively fast. However, with vector geometries the performance is less good when large amounts of data is being retrieved. Therefore, three methods for spatial querying have been implemented for retrieving metadata of sensors inside a vector feature. First of all, SPARQL queries in which the complete geometry of a vector feature is added to a spatial filter expression. Second of all, SPARQL queries in which EEA raster cells overlapping the vector geometry are added to a spatial filter expression. Third of all, an approach in which the spatial querying is performed on the SOS side using spatial filters in `GetObservation` requests. Considering all advantages and disadvantages of the three methods the proof of concept implementation selects the second method by default. The other methods can be selected using an optional input parameter. The three methods are described in more detail below.

With the first method the WPS does not have to perform any further spatial queries. A single query retrieves the metadata of sensors inside the geometry. This is a fast procedure, since it uses the Postgis extension in combination with a Postgres database. However, when more complicated vector geometries are used the WKT definition get more verbose. This can lead to a rejection of the query by the endpoint, as it exceeds a maximum amount of characters defined by the server. The method has been tested with Strabon and Parliament endpoints, since they both handle spatial SPARQL queries. The Strabon endpoint has been used in the final proof of concept, because it handled long queries better. However, still a query gets can easily be rejected if complicated vector geometries are added to the filter expression.

The second method performs a rough spatial filtering at the side of the endpoint and the detailed spatial filtering at the WPS side. To do this raster cells overlapping the vector geometries are added to the SPARQL filter. The returned sensor metadata is then filtered by the WPS using the Shapely Python package. The advantage of this method is that it always works, regardless of the geometries that are used as input. However, the downside is that it introduces an extra step in the process of retrieving metadata. This results in a lower performance of the entire process of retrieving sensor metadata.

The third method retrieves addresses of Sensor Observation Services that have metadata in a certain area from the endpoint. To these returned addresses a `GetObservation` request is send with a spatial filter. The problem that was encountered with this method is the difference in filtering capabilities between Sensor Observation Services. Not all services had spatial filters implemented. If these filters would have been implemented by all of them it could have been a viable alternative. However, another disadvantage of this method is that since the detailed spatial filtering happens on the SOS side only a very rough spatial filtering is applied on the endpoint side using a bounding box. This could lead to useless requests to Sensor Observation Services that have sensors inside the bounding box of the vector geometry, but not inside the vector geometry itself.

6.3.2 Latitude and longitude order in point coordinates

During the process of creating the proof of concept implementation a problem has come up regarding the order in which latitude and longitude are being presented. The SOS of the RIVM provides point geometries in WGS84 as longitude, latitude and height. However, the Strabon endpoint expects the order to be latitude, longitude and height. Mixing up the order of a point coordinate results in wrong outcomes of spatial queries. The biggest issue with the coordinate order is that there is no description of it in the CRS. The WGS84 coordinates by IRCEL-CELINE do not cause this problem, while the WGS84 coordinates by the RIVM do. The order of longitude and latitude should be decided upon by the geomatics and geospatial community or explicitly defined to prevent confusion.

However, to make the current proof of concept work an ad hoc solution has been implemented based on gdal.org [2016]: coordinates defining their CRS in the format of `urn:ogc:def:crs:EPSG:unknown` are generally using the longitude, latitude order. Coordinates defining their CRS in the format of `http://www.opengis.net/def/nil/OGC/0/unknown` are generally using the latitude, longitude order. Therefore, the order is swapped for geometries with the former CRS format. The coordinates that use the latter format for their CRS are left as they are. This approach works for the Sensor Observation Services used in this thesis, but is not the preferred solution.

6.4 OUTPUT DATA

When sensor data is retrieved from the Sensor Observation Services and processed it is returned to the user as a JSON or XML file. The content of these documents are structured according to the O&M schema. However, this schema cannot be (re)used when data different procedures has been used to create a single aggregated value. Since different Sensor Observation Services use different procedures this is often the case when sensor data is integrated from multiple sources. A procedure is defined in O&M as a method, algorithm or instrument, or system of these. A procedure can therefore be a sensor, an algorithm processing the raw observation data and/or a system of sensors observing a property of the FOI. The first question that comes to mind is to what extent it is problematic for different procedures that observe the same property of a FOI to be aggregated into a single value. The answer to this question could differ from one procedure to the other and should be answered by people with expert knowledge on sensing procedures. It could also be semantically stored as suggested by Stasch et al. [2014] to allow this knowledge to be machine understandable. The method designed in this thesis can then be extended to make these decisions based on semantics about sensing procedures.

In the current proof of concept implementation all sensor data that observes the same property of their FOIs is allowed to be aggregated together. This decision makes it possible to integrate and aggregate sensor data retrieved from multiple sources before it is returned to the user. However, the problem remains that the procedure element of the O&M observation schema cannot be filled out. According to the XML specification the procedure is nillable, but only "where the information, though strictly required, is not available" [Open Geospatial Consortium, 2011, p. 42]. In the proof of concept the procedure element is left out of the response documents, but this is not in line with the O&M schema. Therefore, the schema should allow the procedure element to contain multiple observation procedures or another kind of value to describe the origin of the observation data.

As a workaround the aggregation could be seen as part of a larger observation system, starting at the sensors, and via the SOS going into an (aggregation) algorithm to produce the observation result. This system would then be handed a unique URI that is placed in the response document. This 'system' would be imagined ad hoc, only when a user's query is received. When only a any URI is returned – which suffices for the O&M observation schema – this is a relatively easy workaround, but the semantics that might have been present are then lost. When semantics are required for procedure definitions, RDF documents containing descriptions of the origin of the data should be created on-the-fly for every request. Whether this is a desirable and viable solution for including aggregated data in the O&M observation schema could be further researched.

6.5 COMPARING A SEMANTIC KNOWLEDGE BASE WITH A CATALOG SERVICE FOR THE WEB

In Paragraph 2.3.1 the Catalog Service for the Web has been described. This is a web service standard for discovering other geo web services such as WFS and WCS. For including Sensor Observation Services in these catalogs Jirka and Nüst [2010] and Jirka and Bröring [2009] have developed the SIR and SOR extensions. When the method of this thesis is compared with SIR and SOR it is visible that there are a lot of similar functionalities. First of all, in both methods there are harvest functions present, requiring an HTTP address of a SOS as input to retrieve sensor metadata. Also, with both methods the metadata is stored in a central location, providing users with an overview of available services. A third similarity is that requests can be for retrieving metadata about a subset of the sensors, which share certain properties.

However, there are also some considerable differences between the two methods. For example, the CSW does not necessarily support the same level of semantics that are required in a semantic knowledge base. The SIR harvesting procedure retrieves metadata from Sensor Observation Services according to the SWE XML schema's and this data is just transferred to another web service. The method presented in this thesis maps these XML schema's for SOS metadata to linked data ontologies. This adds meaning and allows it to be machine understandable.

A second difference has to do with the platform for retrieving sensor metadata. The CSW is an OGC web service, from which data can be retrieved by sending CSW requests, such as `GetCapabilities`, `DescribeRecord` or `GetRecord`. This web service has in common with the SOS that the client should know the exact HTTP address of the CSW as well as how to communicate with it. The method in this thesis creates linked data which is stored in a endpoint on the semantic web. Data from an endpoint can be queried using the SPARQL query language. It can also linked to and from by other related data which allows it to be discovered more easily.

The data model behind the SWE and CSW services are based on the closed world assumption. This means that the application is supposed to have complete information at its disposal with respect to the observation data. However, the semantic web is based on an open world assumption. In this case it is accepted that there might be information that is currently unknown, but might become available at a later moment in time. The semantic knowledge base allows other data that is directly or indirectly related to (the context of) an observation to be part of the data model. This allows semantic reasoning based on a combination of spatial and non-spatial data.

In their current implementation the SIR and SOR allow any SWE web service to be included in a CSW. Also Sensor Planning Services, Sensor Alert Services and Web Notification Services can be looked up using a CSW. These standards have not been taken into consideration for this thesis. Because of time limitations the method presented here focusses only on the SOS. However, The principles described here can be used to extend it to include any other SWE web services in the future as well (see Chapter 9).

6.6 COMPARING A SEMANTIC KNOWLEDGE BASE WITH SEMANTIC SENSOR MIDDLEWARE

The method presented in this thesis could also be compared to the so-called semantic sensor middleware as described in Paragraph 2.4. These middlewares create a layer on top of a SOS, allowing them to output their contents in RDF. Examples of this are the Semantically Enabled SOS (Sem-SOS) service by Henson et al. [2009] and Pschorr [2013], and the Semantic Enablement Layer (SEL) by Janowicz et al. [2013]. These middlewares offer similar functionality to the method presented in this thesis as they add semantics to sensor metadata. Under the hood they use a linked data model that allows the SOS to return RDF data based on the O&M data model.

A key difference between the semantic sensor middleware approaches and the method presented in this thesis is that the the Sem-SOS and SEL are based on a situation in which data from a single SOS is being used for an application. This thesis has a different scope, in which it has been acknowledged that multiple Sensor Observation Services could be relevant to a user's need for information. Therefore, the semantic web is used to automatically discover and retrieve data from different Sensor Observation Services. The WPS performs all of these tasks, placing a lower burden on the user. A semantic sensor middleware is not concerned with automating the process of discovering and retrieving observation data, but rather with the data format. In case of Sem-SOS and SEL one or more request will still have to be made to a SOS to decide whether the service is relevant to a user's logical data request. By explicitly storing the metadata in a semantic knowledge base instead of a semantically enriched SOS service this question can be answered with SPARQL queries created by a WPS.

The advantage of the middleware approaches is that data is not duplicated, but only returned in different formats. Duplicated data can become out of sync with their source, creating a situation where multiple versions of the same data are available online. To prevent this issue the data either has to be frequently updated or retrieved directly from the source instead of creating a duplicate. The Sem-SOS and SEL provide a method for retrieving RDF data about observations directly from the SOS source. This topic will be further discussed in Chapter 8.

Observation results are returned by the middleware implementations as RDF triples. The current proof of concept implementation presented in Chapter 5 outputs observation data using the O&M schema in either XML or JSON. These two formats have been selected because they are common for data transferral in web applications. However, it could be extended to output observation data in RDF as well. After all, the metadata is essentially converted back from linked data using the om-lite and sam-lite ontologies to the O&M schema by the WPS.

7 CONCLUSIONS

This thesis aimed to design a method that uses the semantic web to improve sensor data discovery as well as the integration and aggregation of sensor data from multiple sources. Each of the four subquestions posed in the introduction will be answered in this chapter before addressing the main research question.

SUBQUESTION 1. TO WHAT EXTENT CAN SENSOR METADATA BE AUTOMATICALLY RETRIEVED FROM ANY SOS?

Data inside a Sensor Observation Service can be automatically retrieved from a SOS, using the methods described in Paragraph 4.1.1. However, there are two things in the SWE standards that should be improved to make the proposed design work better.

First of all, the capabilities document is currently only required to contain a list of all features of interest as a parameter for the `GetObservation` request. Optionally the features of interest can also be mentioned as metadata per offering. In the case of air quality these features represent the sensor locations. However, it is merely required to list the URIs of the features of interest. To retrieve their geometries either `GetFeatureOfInterest` or `GetObservation` requests have to be made. This is especially cumbersome since the `GetFeatureOfInterest` response does not link features of interest to procedures or observed properties. This relation is therefore only visible by either combining the observed properties and procedures per offering to the `GetFeatureOfInterest` response or by requesting observations from each sensor location. Therefore, I propose that the capabilities document should not only list the URIs of the features of interest, but also describe the geometry and observed properties of each of them.

Secondly, in the current implementation of the XML schemas for SOS (`http://schemas.opengis.net/sos/2.0`) and O&M (`http://schemas.opengis.net/om/2.0/`) identifiers for features of interest, observed properties and procedures can be 'any URI'. This means that an URL with semantics can be provided, but a non-semantic URI would also be valid. I propose to make the standard more strict and require a semantic URL that can be resolved to an RDF document. Without these semantics it is hard to use a SOS for both humans and automatic processes, especially when two or more are used in combination with each other. Paragraph 4.1.3 describes how to use a PURL server for creating persistent semantic URLs that resolve to an RDF document.

Besides the two changes that should be made to the SWE standards there is another issue with automatically retrieving sensor metadata from a SOS, namely the order of coordinates for a feature of interest. Both Sensor Observation Services used in this thesis had a different order for the latitude and the longitude of their point coordinates. This is a standardisation issue already identified by many authors, which should be decided upon by the geomatics and geoscience community. For the method described in this thesis it is irrelevant which order is being used, as long as its clearly described in individual cases or prescribed using an international standard.

Subquestion 2. To what extent can sensor metadata from a SOS be automatically converted to linked data and published on the semantic web?

In a SOS there are XML schemas that contain general semantics about the metadata. They identify what the different URIs represent (e.g. observed properties, procedures, features of interest). The om-lite and sam-lite ontologies have been used to make linked data from this metadata. However, a number of classes should be added to these ontologies to make it suit this process better. First of all, a class is required that distinguishes the *process* of creating an observation from the physical *device* that uses this process. This 'sensor' class could be modelled as a device that uses a procedure at a certain sampling point. Adding this class takes away some of the ambiguity between defined processes and actually deployed sensors. Therefore, it will be easier to perform (spatial) SPARQL queries that return deployed sensors of which data can be retrieved.

Another class that should be represented in an ontology is the Sensor Observation Service. The current prototype design (Chapter 5) used a single endpoint for storing all metadata from Sensor Observation Services. Therefore, it was known that the source of data is a SOS, but it has not been properly defined in a linked data ontology. If programs crawling the semantic web can identify a data source such as a SOS and understand its allowed queries (e.g. `GetCapabilities`, `DescribeSensor`, `GetObservation`), they can retrieve data from it without requiring prior knowledge. This way sensor data using other platforms such as the SensorThings API could be discovered and retrieved in the same way and used in combination with each other. Therefore the extensions of current ontologies is further discussed as future research in Chapter 9.

Subquestion 3. What is an effective balance between the semantic web and the geo web in the chain of discovering, retrieving and processing sensor data?

A number of authors have shown that triple stores do not perform as good as databases when their data is requested via a web application. On the other hand, linked data is very well suited for discovering data as it is literally 'linked' to related data. Therefore, this thesis aimed to design a method for using the semantic web in combination with sensor web applications, where the semantic web contains metadata and the geoweb observation data.

However, there is a grey area of functions that could be implemented using either on of these two parts of the web.

For example, the semantic web could have a bounding box per SOS containing all features of interest it offers in combination with a list of all observed properties. In this case spatial and temporal filters would have to be applied at the SOS side when retrieving observation data. On the other hand, the semantic web could also contain detailed information about individual sensors. This way the Sensor Observation Services are only used to retrieve observation data of already selected sensors.

Both options have been considered. The second option has been used in this thesis for a number of reasons. First of all, it was found that not all Sensor Observation Services offer the same filter capabilities. In order for the first option to be viable every SOS should have a minimum amount of filter capabilities implemented by default. Second of all, if a user is interested in sensors located in a specific area the bounding box of all sensors might be misleading. If there is an outlying sensor or if all sensors are inside a curved or diagonal vector geometry a SOS might seem relevant while it actually is not. The result of this is that many unnecessary requests will be send to a SOS keeping the user waiting and the SOS server busy for no reason.

Third of all, semantic information about specific sensors can be linked to by other related linked data. This can be done by the organisation maintaining the sensor or by other organisations. For example links to the manufacturer, the quality of observations achieved by a certain model of sensors, or the conditions under which the sensor is placed could be useful for anyone interested in the observation data.

However, the down side of the second approach is that the performance is lower. Both the endpoint and the Sensor Observation Services perform well with spatial filters. But using a semantic knowledge base a larger amount of data needs to be send over the internet in the process of discovering sensors. Since GeoSPARQL and stSPARQL uses the verbose WKT or GML encodings the queries can be rejected for exceeding the maximum amount of characters. For this reason it is found that the performance of the endpoint should be improved, to better cope with vector geometries in SPARQL queries.

Still, discovering sensors is only a matter of seconds using the second approach. Automatically retrieving sensor data can take up to a couple minutes depending on the amount of sensors for which data is requested and the temporal range. However, it should be noted that performance optimization has not been a part of this thesis. It is likely that this can still be improved significantly (see Chapter 9).

SUBQUESTION 4. TO WHAT EXTENT CAN ALREADY EXISTING STANDARDS FOR RETRIEVING DATA BE (RE)USED FOR A SERVICE THAT SUPPLIES INTEGRATED AND AGGREGATED SENSOR DATA?

All data models and services in this thesis have been used because they are based on open standards. Designs for two processes have been explored: an automated process for creating linked data from metadata in a SOS and a process for discovering, retrieving and processing sensor data. These processes were created using OGC Web Processing Services, which is a standard

API for spatial data processes on the web. The WPS is well suited for these two applications.

The O&M standard could be reused on the semantic web using the om-lite and sam-lite ontologies. These are lightweight linked data ontologies based on O&M. Performing spatial queries on this linked data is possible using OGC's GeoSPARQL as well as using Strabon's stRDF (Paragraph 3.3).

The O&M observation schema is being reused for the spatially aggregated sensor data that is retrieved from the discovered Sensor Observation Services. However, the schema only allows sensor data from the same procedure. The result of this is that only observations from the same SOS fit in the O&M observation schema, as different procedures were implemented by the different organisations maintaining a SOS. Therefore, I propose to either allow the procedure element to contain an array of multiple procedures from which the data originates.

MAIN QUESTION: TO WHAT EXTENT CAN THE SEMANTIC WEB IMPROVE THE DISCOVERY, INTEGRATION AND AGGREGATION OF DISTRIBUTED SENSOR DATA?

In this thesis a method has been designed to create an online knowledge base with linked metadata extracted from Sensor Observation Services. This helps discovering, integrating and aggregation sensor data, while for efficient data retrieval the SOS is still used. The results show that such a knowledge base makes it easy to discover sensors and their corresponding sources. It allows for online processes to automatically retrieve and process sensor data. It can be generated from any SOS, if there are a minimum of semantics provided. Most SWE standards can be (re)used, although some need to be changed for the method presented here to work for every implementation of a SOS. Observed properties and procedures need to be semantically defined by the organisation maintaining a SOS and linked data ontologies should be extended to define the Sensor Observation Service and its offerings.

A number of design decisions have been made which will be further discussed in Chapter 8. Areas for improving and extending the presented methods are described in Chapter 9.

8 DISCUSSION

A method has been presented in this thesis and tested in a proof of concept implementation. In the process of designing this method and testing a number of design decisions had to be made. This chapter provides a brief reflection on each of these topics of discussion.

METADATA DUPLICATION

The method presented in this thesis takes metadata from a SOS, converts it to linked data and publishes it on the semantic web. In essence the data is being stored twice in two different locations with this method. Having duplicated data may not be desirable, because the duplicate can become out of sync with the corresponding data original at the source. Also more available storage space is required for the same amount of data.

However, extra functionality is achieved in return, primarily with respect to data discovery. This why the CSW also creates duplicate data of metadata from OGC geo web services. The SIR extension for the CSW has a fixed time interval that can be set, for performing regular updates. The current method presented in this thesis has not yet implemented an update mechanism. A solution for this could be to integrate the method into common SOS software packages to allow the SOS to update its own metadata on the semantic web. In this scenario an update will only take place when metadata is updated at the source.

The semantic sensor middleware solutions provide a method for outputting RDF documents directly from a SOS. This does not require any data duplication, but essentially adds an extra output format to the SOS based on a linked data model. The trade of between duplication and functionality is also visible here: these approaches do not offer more functionality for discovering sensor metadata than the SOS already had.

METADATA QUALITY

The method of Chapter 4 is aiming to automate as many steps in the chain of discovering and processing sensor data as possible. This includes a harvesting process which should be executed to automatically creating linked data from SOS metadata. The philosophy behind this is that an automated process is more convenient for users, because it saves them time and effort, and that it does not make any mistakes when performing the same task multiple times in a row. However, the metadata is not machine understandable if there is no meaning added at all to instances like observed properties inside the SOS. For example, a procedure represented by a five digit integer cannot be semantically interpreted. In this case, manual work has to be done to make it machine understandable. Only after this manual process it can be converted to linked data and published on the semantic web.

Nevertheless, even if a SOS has provided the minimum of semantics for the WPS to understand its content, an automated process is still error prone. The quality of the metadata in a SOS influences the quality of the metadata at the SPARQL endpoint. Since the method is designed to work with any SOS as input there lies a responsibility on the side of the organisations maintaining these services to provide sufficient, understandable and correct metadata.

EXPLICIT TOPOLOGICAL RELATIONS

Spatial features have topological relations with other spatial features. These relations can be made explicit on the semantic web using ontologies such as GeoSPARQL. However, in this thesis they have not been made explicit and are calculated on-the-fly with spatial queries using stSPARQL filter expressions. Making topological relations explicit in a subject-predicate-object structure could improve query speed, as they are likely less expensive than spatial queries. However, this is a trade-off with the required storage space. Furthermore, the chances of incorrect or broken links increase as both features and subsequently their topological relations can change over time.

THE USE OF A CATALOG SERVICE

The methods presented in Chapter 4 automatically retrieves metadata from a SOS. However, it could also have been designed on top of a CSW: the metadata from a SOS would be harvested by the CSW and this CSW would be connected to a semantic knowledge base. The advantage of this method would be that the CSW standard with SOR and SIR extensions could be reused as well. The method would then perform the same linked data conversions, but without the SOS harvesting. This approach has not been selected in this thesis because it also has a number of disadvantages. First of all, there would be even more data duplication than the current method. Data would be stored at the source, duplicated by the CSW and then duplicated again on the semantic web. Second of all, it places a greater burden on the organisation maintaining the SOS, as it would have to create a CSW next to it or make sure it is connected to a third parties CSW.

9 FUTURE RESEARCH

The method presented in this thesis has been created with a limited amount of time. Therefore, there are still a number of areas in which the method could be improved, and questions that should still be answered. This chapter will briefly go over each of these future research topics.

INCLUDING MORE WEB SERVICES

The current implementation has been created using the SOS. However, there are a number of other sensor web services based on SWE standards that could be included. First of all, there are the services that provide other kinds of functionalities besides data retrieval such as the Sensor Alert Service (SAS), Sensor Planning Service (SPS) and Web Notification Service (WNS). The inclusion of these services would broaden the range of tasks a WPS could automatically perform.

On the other hand, other services for observation data retrieval could be added as well, such as the tinySOS and the SensorThings API. These are services for retrieving observation data from a IoT perspective, with a focus on low energy consumption by the 'thing' that senses and more compact queries and responses than the verbose XML documents that the SOS uses. Nevertheless, all of these services are based on the O&M data model and therefore the principles applied to the SOS can also be applied to the other sensor data services.

Next to sensor related web services the method could also be extended to include other OGC geo web standards, such as the Web Feature Service (WFS) and Web Coverage Service (WCS). Similar to the SOS it's content could be made available in a machine understandable manner to allow them to be discovered and used in automated web processes.

IMPROVING PERFORMANCE

The main focus of developing the current method was to define its functionality. The proof of concept implementation showed that the performance of discovering sensor metadata was good. However, the process that automatically queries the Sensor Observation Services could still be made faster. Currently for every sensor a request is send. This approach could be improved by requesting all sensor data from the same SOS using fewer or perhaps even using only a single request. The available offerings of a SOS could play a role in this improvement.

EXTENDING LINKED DATA ONTOLOGIES

The ontologies that have been used in this thesis do not yet cover all concepts of sensor metadata. There are for example no semantic definitions of what OGC geo web services are, such as SOS. They are now defined as merely a URL, but this does not at all describe the service, its allowed requests and responses. For this and other missing classes as described in Chapter 6 and 7 linked data ontologies should be extended to properly define all sensor metadata.

EXTENDING THE METHODS OF AGGREGATION

Basic aggregation methods have currently be implemented. However, as Ganesan et al. [2004] points out spatial or temporal aggregation can give a distorted result if no weight are added to compensate for differences in density. This extension could therefore create more reliable outcomes produced by the WPS.

Another useful way to extend the aggregation methods of this thesis, is to describe each of these methods semantically. Stasch et al. [2014] proposes the use of semantic definitions of spatial aggregation. This could make the purpose of each these methods machine understandable, as well as the effect that it has on the input data. This could prevent an automated process of making a logical mistake that leads to meaningless output.

A DATA VISUALISATIONS

Figure A.1: Dataset of municipalities in the Netherlands and Belgium in 2015 (from Dutch cadaster and GADM.org)

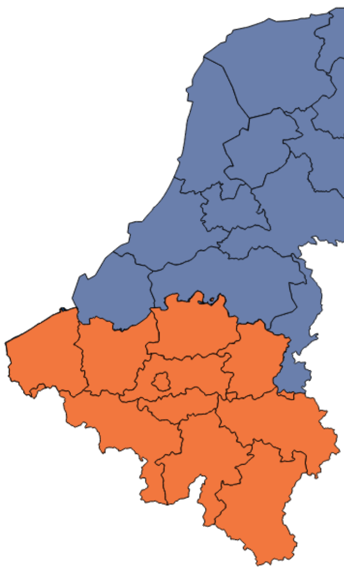

Figure A.2: Dataset of provinces in the Netherlands and Belgium in 2015 (from Dutch cadaster and GADM.org)

Figure A.3: Dataset of the Netherlands and Belgium in 2015 (from GADM.org)

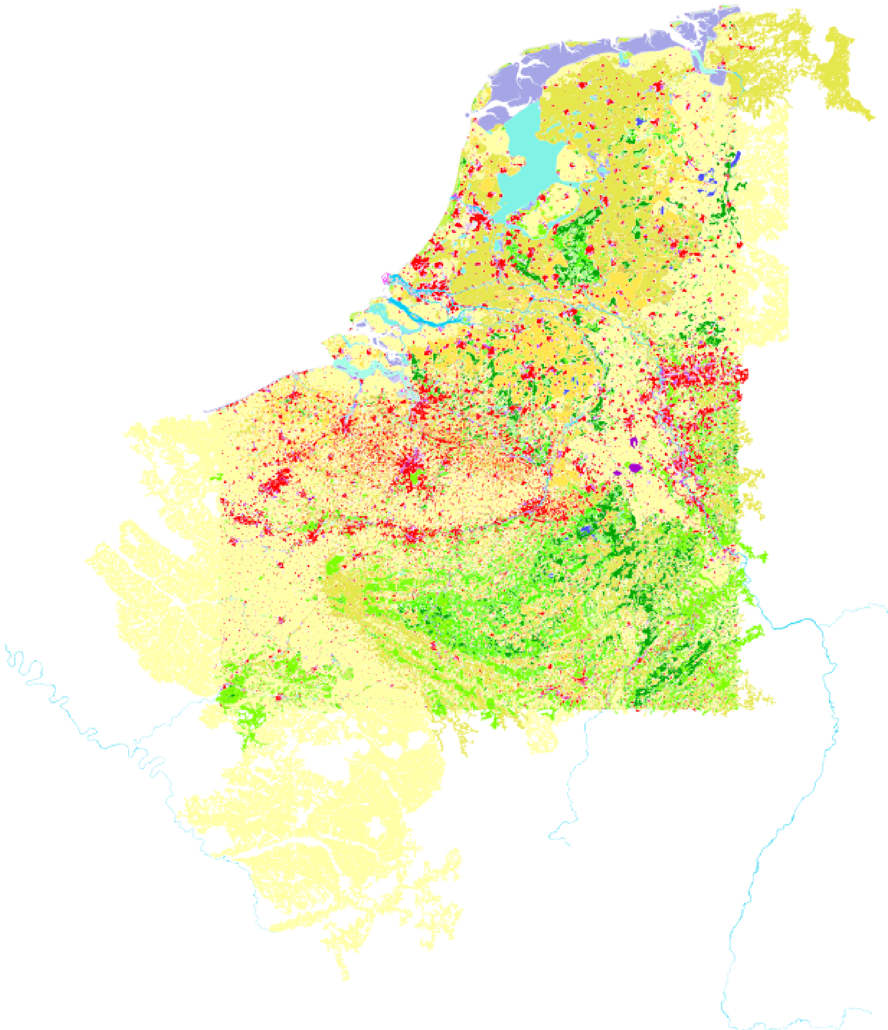

Figure A.4: Dataset of landcover in the Netherlands and Belgium in 2012 (from Copernicus The European Earth Observation Programme)

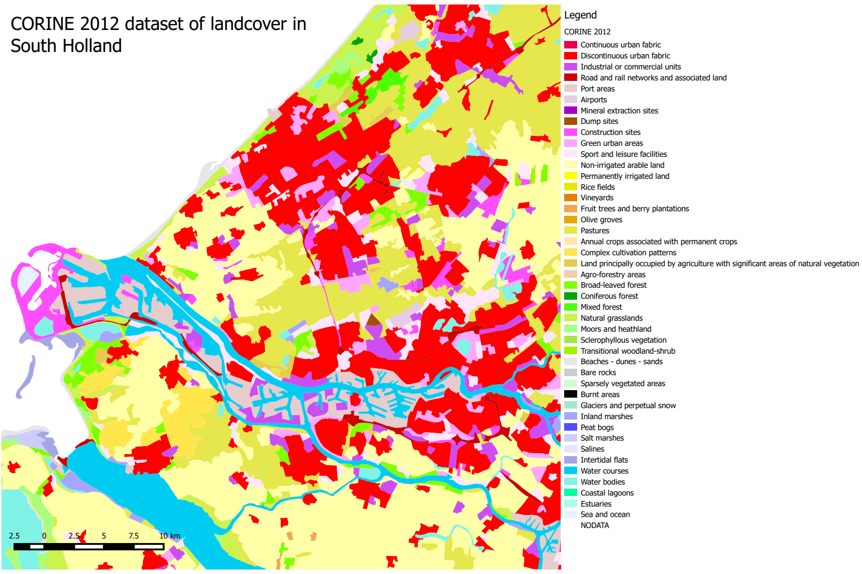

Figure A.5: Landcover of the province of South Holland (subsection of the dataset from Figure A.4)

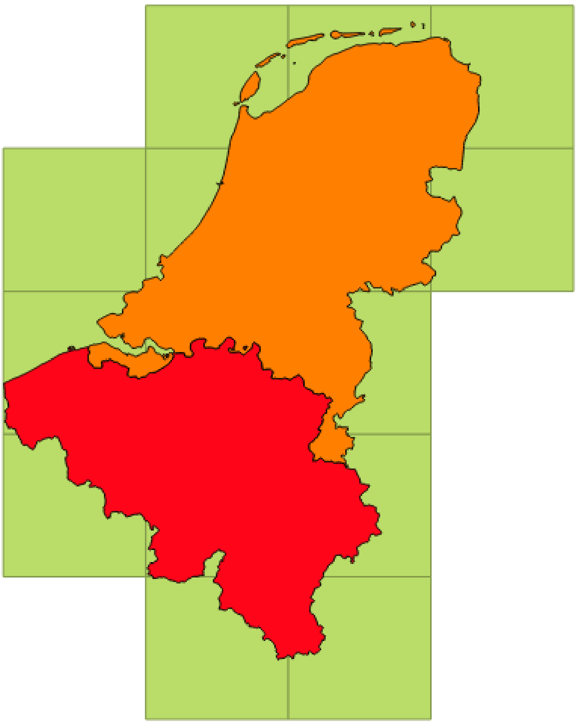

Figure A.6: EEA reference grid cells with a resolution of 100km² overlapping the Netherlands and Belgium

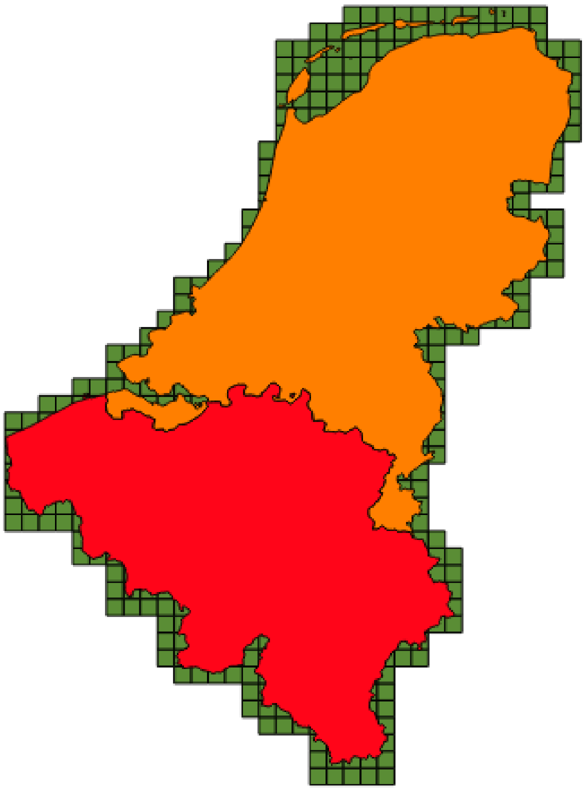

Figure A.7: EEA reference grid cells with a resolution of 10km² overlapping the Netherlands and Belgium

Figure A.8: Webmap by the RIVM showing their air quality sensor network (`http://www.lml.rivm.nl/meetnet`)

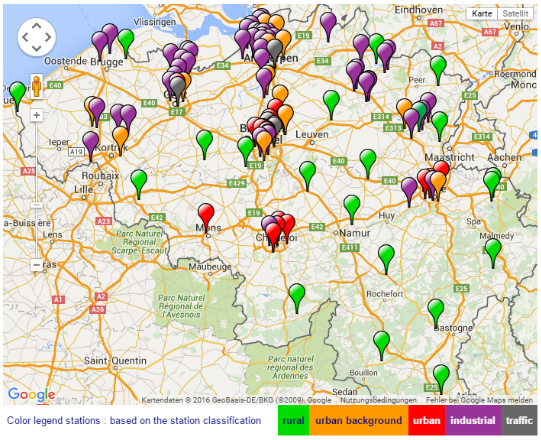

Figure A.9: Webmap by IRCEL-CELINE showing their air quality sensor network (http://www.irceline.be/en/air-quality/measurements/monitoring-stations/)

Figure A.10: Google Streetview image of RIVM sensor location in Amsterdam in 2015

B WEB PROCESSING SERVICE RESPONSE DOCUMENTS

B.1 EXAMPLE CAPABILITIES DOCUMENT

```xml
<wps:Capabilities xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:wps="http://www.opengis.net/wps/1.0.0"
    xmlns:ows="http://www.opengis.net/ows/1.1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    service="WPS" version="1.0.0" xml:lang="en-CA"
    xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
    http://schemas.opengis.net/wps/1.0.0/wpsGetCapabilities_response.xsd"
    updateSequence="1">
  <ows:ServiceIdentification>
    <ows:Title>PyWPS Server</ows:Title>
    <ows:Abstract>
      See http://pywps.wald.intevation.org and
          http://www.opengeospatial.org/standards/wps
    </ows:Abstract>
    <ows:Keywords>
      <ows:Keyword>WPS</ows:Keyword>
      <ows:Keyword>SWE</ows:Keyword>
      <ows:Keyword>SOS</ows:Keyword>
    </ows:Keywords>
    <ows:ServiceType>WPS</ows:ServiceType>
    <ows:ServiceTypeVersion>1.0.0</ows:ServiceTypeVersion>
    <ows:Fees>None</ows:Fees>
    <ows:AccessConstraints>none</ows:AccessConstraints>
  </ows:ServiceIdentification>
  <ows:ServiceProvider>
    <ows:ProviderName>Delft University of
        Technology</ows:ProviderName>
    <ows:ProviderSite
        xlink:href="http://masterthesistudelft.herokuapp.com/"/>
    <ows:ServiceContact>
    <ows:IndividualName>Ivo de Liefde</ows:IndividualName>
    <ows:PositionName>MSc. student Geomatics for the Built
        Environment</ows:PositionName>
    <ows:ContactInfo>
      <ows:Address>
      <ows:DeliveryPoint>Julianalaan 134</ows:DeliveryPoint>
      <ows:City>Delft</ows:City>
      <ows:PostalCode>2628 BL</ows:PostalCode>
      <ows:Country>the Netherlands</ows:Country>
      <ows:ElectronicMailAddress>i.deliefde@student.tudelft.nl</ows:ElectronicMailAddress>
      </ows:Address>
```

```xml
          <ows:OnlineResource
              xlink:href="http://masterthesistudelft.herokuapp.com/"/>
          <ows:HoursOfService>0:00-24:00</ows:HoursOfService>
          <ows:ContactInstructions>none</ows:ContactInstructions>
        </ows:ContactInfo>
        <ows:Role>
          Created WPS
        </ows:Role>
        </ows:ServiceContact>
      </ows:ServiceProvider>
      <ows:OperationsMetadata>
        <ows:Operation name="GetCapabilities">
          <ows:DCP>
            <ows:HTTP>
              <ows:Get
                  xlink:href="http://localhost/cgi-bin/wps?"/>
              <ows:Post
                  xlink:href="http://localhost/cgi-bin/wps"/>
            </ows:HTTP>
          </ows:DCP>
        </ows:Operation>
        <ows:Operation name="DescribeProcess">
          <ows:DCP>
            <ows:HTTP>
              <ows:Get
                  xlink:href="http://localhost/cgi-bin/wps?"/>
              <ows:Post
                  xlink:href="http://localhost/cgi-bin/wps"/>
            </ows:HTTP>
          </ows:DCP>
        </ows:Operation>
        <ows:Operation name="Execute">
          <ows:DCP>
            <ows:HTTP>
              <ows:Get
                  xlink:href="http://localhost/cgi-bin/wps?"/>
              <ows:Post
                  xlink:href="http://localhost/cgi-bin/wps"/>
            </ows:HTTP>
          </ows:DCP>
        </ows:Operation>
      </ows:OperationsMetadata>
      <wps:ProcessOfferings>
        <wps:Process wps:processVersion="1.0">
          <ows:Identifier>LinkedDataFromSOS</ows:Identifier>
          <ows:Title>Creates Linked Data of SOS
              metadata</ows:Title>
          <ows:Abstract>
            This process takes an HTTP address of a Sensor
                Observation Service (SOS) as input and converts
                the metadata to linked data.
          </ows:Abstract>
        </wps:Process>
```

```
    <wps:Process wps:processVersion="1.0">
      <ows:Identifier>GetSensors</ows:Identifier>
      <ows:Title>
        Automatically retrieves sensors from heterogenous
            sources using the semantic web
      </ows:Title>
      <ows:Abstract>
        This process takes a sensor data request with
            parameters for spatial features of interest,
            observed property, temporal range and
            granularity, and finds all relevant sensor data
            sources on the semantic web.
      </ows:Abstract>
    </wps:Process>
    <wps:Process wps:processVersion="1.0">
      <ows:Identifier>GetSensorData</ows:Identifier>
      <ows:Title>
        Automatically retrieves, integrates and aggregates
            heterogenous sensor data using the semantic web
      </ows:Title>
      <ows:Abstract>
      This process takes sensors found by the WPS
          'GetSensors' and automatically integrates and
          aggregates the data from different sources on the
          web.
      </ows:Abstract>
    </wps:Process>
  </wps:ProcessOfferings>
  <wps:Languages>
    <wps:Default>
      <ows:Language>en-CA</ows:Language>
    </wps:Default>
    <wps:Supported>
      <ows:Language>en-CA</ows:Language>
    </wps:Supported>
  </wps:Languages>
  <wps:WSDL xlink:href="http://localhost/cgi-bin/wps?WSDL"/>
</wps:Capabilities>
```

B.2 EXAMPLE DESCRIBE PROCESS DOCUMENT

```
  <wps:ProcessDescriptions
     xmlns:wps="http://www.opengis.net/wps/1.0.0"
     xmlns:ows="http://www.opengis.net/ows/1.1"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
     http://schemas.opengis.net/wps/1.0.0/wpsDescribeProcess_response.xsd"
     service="WPS" version="1.0.0" xml:lang="en-CA">
  <ProcessDescription wps:processVersion="1.0"
     storeSupported="true" statusSupported="false">
    <ows:Identifier>LinkedDataFromSOS</ows:Identifier>
    <ows:Title>Creates Linked Data of SOS metadata</ows:Title>
    <ows:Abstract>
      This process takes an HTTP address of a Sensor
          Observation Service (SOS) as input and converts the
          metadata to linked data.
    </ows:Abstract>
    <DataInputs>
      <Input minOccurs="0" maxOccurs="1">
        <ows:Identifier>observed_properties</ows:Identifier>
        <ows:Title>
          Input link to turtle file with mappings of
              observed property identifiers to DBPedia URIs
        </ows:Title>
        <LiteralData>
          <ows:DataType
              ows:reference="http://www.w3.org/TR/xmlschema-2/#string">string</
          <ows:AnyValue/>
          <DefaultValue>http://inspire.rivm.nl/sos/eaq/service?</DefaultValue>
        </LiteralData>
      </Input>
      <Input minOccurs="0" maxOccurs="1">
        <ows:Identifier>input_url</ows:Identifier>
        <ows:Title>
          Input a string containing an HTTP address of a
              Sensor Observation Service (SOS). For
              example: 'http://someaddress.com/sos?'
        </ows:Title>
        <LiteralData>
          <ows:DataType
              ows:reference="http://www.w3.org/TR/xmlschema-2/#string">string</
          <ows:AnyValue/>
          <DefaultValue>http://inspire.rivm.nl/sos/eaq/service?</DefaultValue>
        </LiteralData>
      </Input>
    </DataInputs>
  </ProcessDescription>
</wps:ProcessDescriptions>
```

B.3 EXAMPLE EXECUTE DOCUMENT

```
<wps:ExecuteResponse
    xmlns:wps="http://www.opengis.net/wps/1.0.0"
    xmlns:ows="http://www.opengis.net/ows/1.1"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
    http://schemas.opengis.net/wps/1.0.0/wpsExecute_response.xsd"
    service="WPS" version="1.0.0" xml:lang="en-CA"
    serviceInstance="http://localhost/cgi-bin/wps?service=WPS&request=GetCapabilities&version=1.
    statusLocation="http://localhost/wps/wpsoutputs/pywps-146072575066.xml">
   <wps:Process wps:processVersion="1.0">
     <ows:Identifier>GetSensors</ows:Identifier>
     <ows:Title>
        Automatically retrieves sensors from heterogenous
            sources using the semantic web
     </ows:Title>
     <ows:Abstract>
        This process takes a sensor data request with
            parameters for spatial features of interest,
            observed property, temporal range and granularity,
            and finds all relevant sensor data sources on the
            semantic web.
     </ows:Abstract>
   </wps:Process>
   <wps:Status creationTime="2016-04-15T15:09:52Z">
     <wps:ProcessSucceeded>PyWPS Process GetSensors
         successfully calculated</wps:ProcessSucceeded>
   </wps:Status>
   <wps:ProcessOutputs>
     <wps:Output>
        <ows:Identifier>output</ows:Identifier>
        <ows:Title>Output sensor data</ows:Title>
        <wps:Data>
           <wps:ComplexData mimeType="text/JSON">
              OUTPUT JSON DATA
           </wps:ComplexData>
        </wps:Data>
     </wps:Output>
   </wps:ProcessOutputs>
</wps:ExecuteResponse>
```

BIBLIOGRAPHY

52 North (2016). Sensor discovery. [online] http://52north.org/communities/sensorweb/discovery [accessed on April, 20th, 2016].

Atkinson, R. A., Taylor, P., Squire, G., Car, N. J., Smith, D., and Menzel, M. (2015). Joining the Dots: Using Linked Data to Navigate between Features and Observational Data. In *Environmental Software Systems. Infrastructures, Services and Applications*, pages 121–130. Springer.

Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer networks*, 54(15):2787–2805.

Barnaghi, P., Wang, W., Henson, C., and Taylor, K. (2012). Semantics for the Internet of Things: early progress and back to the future. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 8(1):1–21.

Battle, R. and Kolas, D. (2012). Enabling the geospatial semantic web with parliament and geosparql. *Semantic Web*, 3(4):355–370.

Beckett, D., Berners-Lee, T., Prud'hommeaux, E., and Carothers, G. (2014). W3C RDF 1.1 Turtle. [online] http://www.w3.org/TR/turtle/ [accessed on December 9th, 2015].

Berners-Lee, T. and Connolly, D. (2011). W3C Notation3 (N3): A readable RDF syntax. [online] http://www.w3.org/TeamSubmission/n3/ [accessed on December 9th, 2015].

Berners-Lee, T., Hendler, J., Lassila, O., et al. (2001). The semantic web. *Scientific american*, 284(5):28–37.

Bizer, C., Heath, T., and Berners-Lee, T. (2009). Linked data-the story so far. *Semantic Services, Interoperability and Web Applications: Emerging Concepts*, pages 205–227.

Botts, M., Percivall, G., Reed, C., and Davidson, J. (2007). OGC Sensor Web Enablement: Overview And High Level Architecture. OGC document 06-021r1.

Botts, M., Percivall, G., Reed, C., and Davidson, J. (2008). OGC sensor web enablement: Overview and high level architecture. In *GeoSensor networks*, pages 175–190. Springer.

Bröring, A., Stasch, C., and Echterhoff, J. (2012). OGC Sensor observation service interface standard.

Cambridge Semantics (2015). Introduction to the Semantic Web. [online] https://www.cambridgesemantics.com/semantic-university/introduction-semantic-web [accessed on December 8th, 2015].

Compton, M., Barnaghi, P., Bermudez, L., GarcíA-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., et al. (2012). The SSN ontology of the W3C semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17:25–32.

Corcho, O. and Garcia-Castro, R. (2010). Five challenges for the Semantic Sensor Web. *Semantic Web-Interoperability, Usability, Applicability*, 1.1(2):121–125.

Cox, S. J. D. (2015a). Observations and Sampling. [online] https://www.seegrid.csiro.au/wiki/AppSchemas/ObservationsAndSampling [accessed on December 1st, 2015].

Cox, S. J. D. (2015b). Ontology for observations and sampling features, with alignments to existing models.

Cox, S. J. D. (2015c). OWL for Observations. [online] http://def.seegrid.csiro.au/ontology/om/om-lite [accessed on November 24th, 2015].

Cox, S. J. D. (2015d). OWL for Sampling Features. [online] http://def.seegrid.csiro.au/ontology/om/sam-lite [accessed on November 24th, 2015].

Cox, S. J. D. and Taylor, P. (2015). OGC Observations and Measurements — JSON implementation.

Cyganiak, R., Wood, D., and Lanthaler, M. (2014). RDF 1.1 Concepts and Abstract Syntax. [online] https://www.w3.org/TR/rdf11-concepts [accessed on February 2nd, 2016].

Deltares (2016). Setting up pywps in a windows environment. [online] https://publicwiki.deltares.nl/display/OET/Setting+up+pyWPS+in+a+Windows+environment [accessed on May 1st, 2016].

Gandon, F. and Schreiber, G. (2014). W3C RDF 1.1 XML Syntax. [online] http://www.w3.org/TR/rdf-syntax-grammar/ [accessed on December 9th, 2015].

Ganesan, D., Ratnasamy, S., Wang, H., and Estrin, D. (2004). Coping with irregular spatio-temporal sampling in sensor networks. *ACM SIGCOMM Computer Communication Review*, 34(1):125–130.

gdal.org (2016). Gml – geography markup language; crs support. [online] http://www.gdal.org/drv_gml.html [accessed on May 5th, 2016].

Hebeler, J., Fisher, M., Blace, R., and Perez-Lopez, A. (2011). *Semantic web programming*. John Wiley & Sons.

Henson, C., Pschorr, J. K., Sheth, A. P., Thirunarayan, K., et al. (2009). SemSOS: Semantic sensor observation service. In *Collaborative Technologies and Systems, 2009. CTS'09. International Symposium on*, pages 44–53. IEEE.

Hu, C., Guan, Q., Chen, N., Li, J., Zhong, X., and Han, Y. (2014). An Observation Capability Metadata Model for EO Sensor Discovery in Sensor Web Enablement Environments. *Remote Sensing*, 6(11):10546–10570.

INSPIRE (2014). Guidelines for the use of Observations & Measurements and Sensor Web Enablement-related standards in INSPIRE Annex II and III data specification development.

INSPIRE (2015). INSPIRE Roadmap. [online] http://inspire.ec.europa.eu/index.cfm/pageid/44 [accessed on December 2nd, 2015].

International Organisation for Standardisation (2005). ISO 19109:2005; Geographic information — Rules for application schema.

International Organisation for Standardisation (2007). ISO 19136:2007; Geographic information — Geography Markup Language (GML).

ISO (2011). ISO 19156:2011; Geographic information – Observations and measurements. [online] http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=32574 [accessed on December 2 nd, 2015].

Janowicz, K., Broring, A., Stasch, C., Schad, S., Everding, T., and Llaves, A. (2013). A RESTful Proxy and Data Model for Linked Sensor Data. *International Journal of Digital Earth*, 6(3):233–254.

Jazayeri, M. A., Liang, S. H., and Huang, C.-Y. (2015). Implementation and Evaluation of Four Interoperable Open Standards for the Internet of Things. *Sensors*, 15(9):24343–24373.

Ji, C., Liu, J., and Wang, X. (2014). A Review for Semantic Sensor Web Research and Applications. *Advanced Science and Technology Letters*, 48:31–36.

Jirka, S. and Bröring, A. (2009). OGC Sensor Observable Registry Discussion Paper. Reference number: OGC 09-112.

Jirka, S. and Nüst, D. (2010). OGC Sensor Instance Registry Discussion Paper. Reference number: OGC 10-171.

Korteweg, P., Marchetti-Spaccamela, A., Stougie, L., and Vitaletti, A. (2007). *Data aggregation in sensor networks: Balancing communication and delay costs*. Springer.

Koubarakis, M. and Kyzirakos, K. (2010). Modeling and querying metadata in the semantic sensor web: The model stRDF and the query language stSPARQL. In *The semantic web: research and applications*, pages 425–439. Springer.

Lassila, O. and Swick, R. R. (1999). Resource Description Framework (RDF) Model and Syntax Specification. [online] http://www.w3.org/TR/PR-rdf-syntax/ [accessed on December 8th, 2015].

Lebo, T., Sahoo, S., and McGuinness, D. (2013). PROV-O: The PROV Ontology. [online] http://www.w3.org/TR/prov-o/ [accessed on December 11th, 2015].

Manola, F., Miller, E., and McBride, B. (2014). W3C RDF Primer. [online] http://www.w3.org/TR/rdf11-primer/ [accessed on December 9th, 2015].

Missier, G. A. (2015). Towards a Web application for viewing Spatial Linked Open Data of Rotterdam. Master's thesis, Delft University of Technology.

Moir, E., Moonen, T., and Clark, G. (2014). What are Future Cities: Origins, Meanings and Uses.

Nebert, D., Whiteside, A., and Vretanos, P. (2007). Opengis catalogue services specification.

Open Geospatial Consortium (2007). OGC Catalogue Services Specification.

Open Geospatial Consortium (2011). Observations and measurements – xml implementation.

Open Geospatial Consortium (2014). OGC SensorML: Model and XML Encoding Standard.

Open Geospatial Consortium (2015). OGC WPS 2.0 Interface Standard.

OWL working group (2012). Web Ontology Language (OWL). [online] http://www.w3.org/2001/sw/wiki/OWL [accessed on December 18th, 2015].

Percivall, G. (2015). OGC Smart Cities Spatial Information Framework. OGC Internal reference number: 14-115.

Perry, M. and Herring, J. (2012). GeoSPARQL - A Geographic Query Language for RDF Data.

Price Waterhouse Coopers (2014). Sensing the future of the Internet of Things. [online] https://www.pwc.com/us/en/increasing-it-effectiveness/assets/future-of-the-internet-of-things.pdf [accessed on December 18th, 2015].

Pschorr, J., Henson, C. A., Patni, H. K., and Sheth, A. P. (2010). Sensor discovery on linked data.

Pschorr, J. K. (2013). SemSOS: an Architecture for Query, Insertion, and Discovery for Semantic Sensor Networks. Master's thesis, Wright State University.

PURL (2016). Batch Uploading to a PURL Server v1.0-1.6.x. [online] https://code.google.com/archive/p/persistenturls/wikis/PURLBatchUploadingVersionOne.wiki [accessed on February 19th, 2016].

Shafer, K., Weibel, S., Jul, E., and Fausey, J. (2016). Introduction to Persistent Uniform Resource Locators. [online] https://purl.oclc.org/docs/long_intro.html [accessed on February 18th, 2016].

Sheth, A., Henson, C., and Sahoo, S. S. (2008). Semantic Sensor Web. *IEEE Internet Computing*, 12(4):78–83.

Stasch, C., Autermann, C., Foerster, T., and Pebesma, E. (2011a). Towards a spatiotemporal aggregation service in the sensor web. Poster presentation. In *The 14th AGILE International Conference on Geographic Information Science*.

Stasch, C., Schade, S., Llaves, A., Janowicz, K., and Bröring, A. (2011b). Aggregating linked sensor data. In Taylor, K., Ayyagari, A., and de Roure, D., editors, *Proceedings of the 4th International Workshop on Semantic Sensor Networks*, page 46.

Stasch, C., Scheider, S., Pebesma, E., and Kuhn, W. (2014). Meaningful spatial prediction and aggregation. *Environmental Modelling & Software*, 51:149–165.

Strobl, C. (2008). *Dimensionally Extended Nine-Intersection Model (DE-9IM)*. Springer.

Theunisse, I. A. H. (2015). The Visualization of Urban Heat Island Indoor Temperatures. Master's thesis, TU Delft, Delft University of Technology.

van der Hoeven, F., Wandl, A., Demir, B., Dikmans, S., Hagoort, J., Moretto, M., Sefkatli, P., Snijder, F., Songsri, S., Stijger, P., et al. (2014). Sensing Hotterdam: Crowd sensing the Rotterdam urban heat island. *SPOOL*, 1(2):43–58.

Van der Hoeven, F. D. and Wandl, A. (2015). Hotterdam: How space is making Rotterdam warmer, how this affects the health of its inhabitants, and what can be done about it. Technical report, TU Delft, Faculty of Architecture and the Built Environment.

W3C Semantic Sensor Network Incubator Group (2011). Semantic Sensor Network Ontology. [online] http://www.w3.org/2005/Incubator/ssn/ssnx/ssn [accessed on December 9th, 2015].

Wang, M., Perera, C., Jayaraman, P. P., Zhang, M., Strazdins, P., and Ranjan, R. (2015a). City Data Fusion: Sensor Data Fusion in the Internet of Things.

Wang, X., Zhang, X., and Li, M. (2015b). A Review of Studies on Semantic Sensor Web. *Advanced Science and Technology Letters*, 83:94–97.

Xiang, L., Luo, J., and Rosenberg, C. (2013). Compressed data aggregation: Energy-efficient and high-fidelity data collection. *Networking, IEEE/ACM Transactions on*, 21(6):1722–1735.

Zanella, A., Bui, N., Castellani, A., Vangelista, L., and Zorzi, M. (2014). Internet of things for smart cities. *Internet of Things Journal, IEEE*, 1(1):22–32.