Exploring the use of the semantic web for discovering, retrieving and processing data from Sensor Observation Services

I. de Liefde

TUDelft

EXPLORING THE USE OF THE SEMANTIC WEB FOR DISCOVERING, RETRIEVING 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, retrieving and processing data from Sensor Observation Services* (2016)

The work in this thesis was made in the:

Department of GIS Technology
OTB Research Institute for the Built Environment
Faculty of Architecture & the Built Environment
Delft University of Technology

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

ABSTRACT

Developments such as smart cities, the Internet of Things (IoT) and the Infrastructure for Spatial Information in Europe (INSPIRE) are producing a growing amount of observation data. The Open Geospatial Consortium (OGC) has developed Sensor Web Enablement (SWE) standards for modelling and publishing this data online. However, their use is currently limited to geo information specialists, who have knowledge about which data services are available and how to access them. With the use of the semantic web, online processes can automatically find and understand observation metadata. This opens up the SWE services to a large user audience. Therefore, this thesis has designed a conceptual system architecture that uses the semantic web to improve sensor data discovery as well as the integration and aggregation of sensor data from multiple sources.

The presented conceptual system architecture contains two web processes. The first process automatically creates an online semantic knowledge base of sensor metadata, by harvesting Sensor Observation Services. The metadata is based on the Observations and Measurements (O&M) data model and includes which sensors are deployed, what they observe, how they observe it and at which SWE service their data can be requested. The second process can be used to automatically translate logical queries of users into observation data requests. It also performs further processing before returning the observation data to the user. These two processes have been tested in a proof of concept implementation.

A Web Processing Service (WPS) has been created as a proof of concept, making the two processes available online. The proof of concept is able to harvest sensor metadata, convert it to linked data, and publish it on the semantic web with links to and from other metadata. The resulting semantic knowledge base improves the discovery of observation data, and enables it to be machine understandable. The Sensor Observation Services can also be used in combination with each other due to the harmonisation involved in creating this linked data. The WPS is therefore able to retrieve and process observation data from multiple Sensor Observation Services, using input parameters, such as: *What are the average particulate matter levels per month in neighbourhoods of Delft over the last five years?*

Creating a completely automated process for harvesting metadata, which works with every Sensor Observation Service is not yet feasible, as non-meaningful identifiers are allowed to be used according to the SWE standards. Linked data ontologies should also be extended to include all available metadata. Currently there are no ontologies able to define the concept of a geo web service. The final conclusion is that the SPARQL endpoints used in this thesis cannot cope with complex vector geometries. This is due to their inability to handle verbose queries. Based on these conclusions, five recommendations are presented.

ACKNOWLEDGEMENTS

CONTENTS

ACRONYMS

API Application Programming Interface 76
CSW Catalog Service for the Web 77
CORINE Coordination of Information on the Environment 44
CRS Coordinate Reference System 16
DE-9IM Dimensionally Extended Nine-Intersection Model 26
EEA European Environment Agency 44
FOI Feature of Interest 73
GML Geography Markup Language 10
GIS Geographical Information System 44
INSPIRE Infrastructure for Spatial Information in Europe 1
IoT Internet of Things 81
IRCEL-CELINE Belgian interregional environment agency 44
IRI International Resource Identifier 16
ISO International Organisation for Standardisation 1
JSON JavaScript Object Notation 41
KVP Key-Value Pair 9
OGC Open Geospatial Consortium 76
O&M Observations and Measurements 73
OWL Web Ontology Language 2
PURL Persistent Uniform Resource Locator 16
RDF Resource Description Framework 73
RIVM Dutch national institute for public health and the environment 2
SensorML Sensor Modelling Language 1
SIR Sensor Instance Registry 2
SOR Sensor Observable Registry 2
SOS Sensor Observation Service 73
SPARQL SPARQL Protocol and RDF Query Language 2
SSNO Semantic Sensor Network Ontology 18
SSW Semantic Sensor Web 2
SWE Sensor Web Enablement 73
UOM Unit of Measurement 6
URI Uniform Resource Identifier 73
URN Uniform Resource Name 14
URL Uniform Resource Locator 73
W3C World Wide Web Consortium 2
WKT Well-Known Text 75
WPS Web Processing Service 74
XML Extensible Markup Language 3

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 Sensor Web Enablement (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:

- Observations and Measurements (O&M) which is a data model and encoding specification for sensor data,
- the Sensor Modelling Language (SensorML) which is a model for describing sensor metadata, and
- the Sensor Observation Service (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 web [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 which can be retrieved using open standards is challenging. The implementation of the sensor web is still at 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 like 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 at a catalogue service. Also, there could be multiple of these services on the web creating issues 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 enable it to be generated 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 phenomena. 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 the European legislation (INSPIRE). Non-experts will be involved more often via smart cities and IoT developments, where users and their consumer electronics produce sensor data as well. This vast amount of data could be useful for academic research, provided researchers are able to find the data, and to integrate and aggregate it from heterogeneous sources. Publishing sensor metadata on the semantic web could make it easier to find what you need through related data. 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 which try to understand phenomena 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 queries of users and the technical requests defined by sensor web protocols should be bridged. A logical sensor data query contains at least a geographical area, a type of observation and a time range. For example: *What are the average particulate matter levels 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 conceptual system architecture 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 further specified into four subquestions:

- To what extent can sensor metadata be automatically retrieved from any Sensor Observation Service?
- To what extent can sensor metadata from a Sensor Observation Service be automatically converted to linked data and published on the semantic web?
- What is an effective balance between the semantic web and the geoweb 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 research questions of this thesis an understanding of the current state of the art in sensor web research is required. A number of topics are relevant for this, such as existing SWE standards for publishing sensor data to the web (Section 2.1), the development of the semantic web (Section 2.4) and linked data ontologies for observation data (Section 2.5), and also their relation to developments in the IoT (Section 2.6). Furthermore, extensions to the OGC catalog service are described for including sensor metadata (Section 2.3), as well as methods for adding semantics to SWE data services (Section 2.7). This chapter discusses the recent 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 of SWE 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 subsections 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 classes. Figure 2.1 shows the Unified Modeling Language (UML) diagram containing the different concepts that will be explained in this subsection and the relations between them. First of all, there is a feature of which a measurement is being taken. 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 available. For example, in some cases specimens are removed from their sampling location to be observed

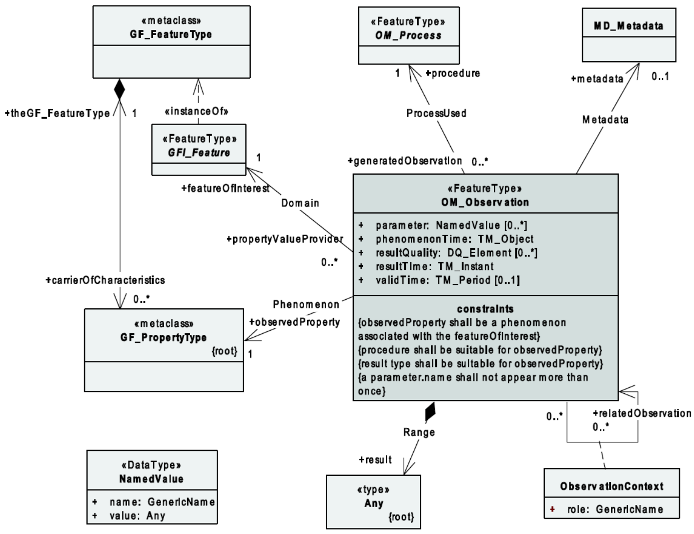

Figure 2.1: Basic observation in the O&M data model. Image courtesy by [ISO, 2011, p. 9]

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 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 execute a procedure. Therefore, the phenomenon time describes the time that the observation applies to the property of the FOI. The valid time is an extent 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 as well, to indicate that two observations have similar characteristics.

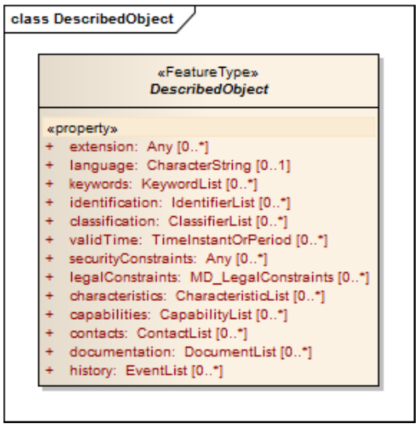

Figure 2.2: `DescribedObject` class in SensorML. Image courtesy by [OGC, 2014, p. 39]

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" [OGC, 2014, p. ix]. Creating interoperability at the syntactic and semantic level allows processes to be better understood by machines, utilised 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 distinction between two types of processes: physical and non-physical. 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. Figure 2.3 shows the UML class diagram of a physical process. 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.

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 of a real world feature. 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 [OGC, 2014].

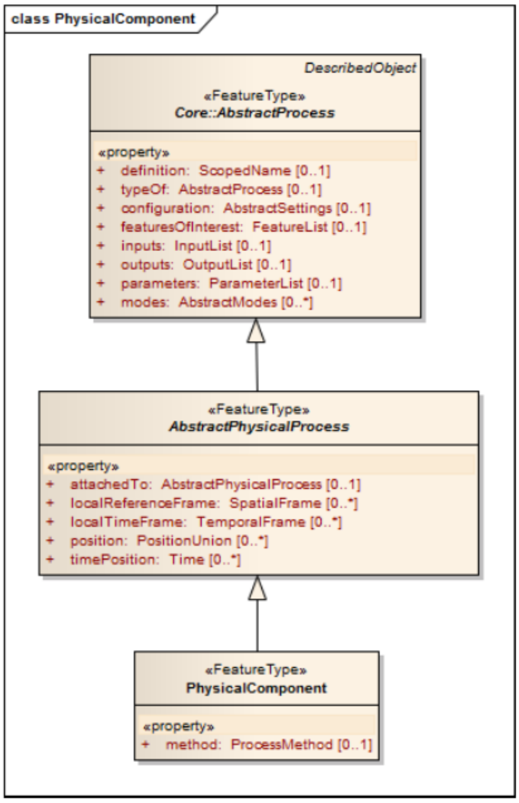

Figure 2.3: Definition of a physical process in SensorML. Image courtesy by [OGC, 2014, p. 57]

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`. `GetCapabilities` returns an overview of the metadata about what the SOS has to offer. The `DescribeSensor` request returns detailed information about individual sensors. `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 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 Pairs (KVPs). Using the POST method the request is described in an XML document which is send to the SOS. The SOS could return data to the client using different kinds of response formats, but there is always at least the option to retrieve it as an XML document. Based on the specification by Bröring et al. [2012] the following describes both the core and optional requests of a SOS, as well as the structure and content 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.

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 Subsection 2.1.2.

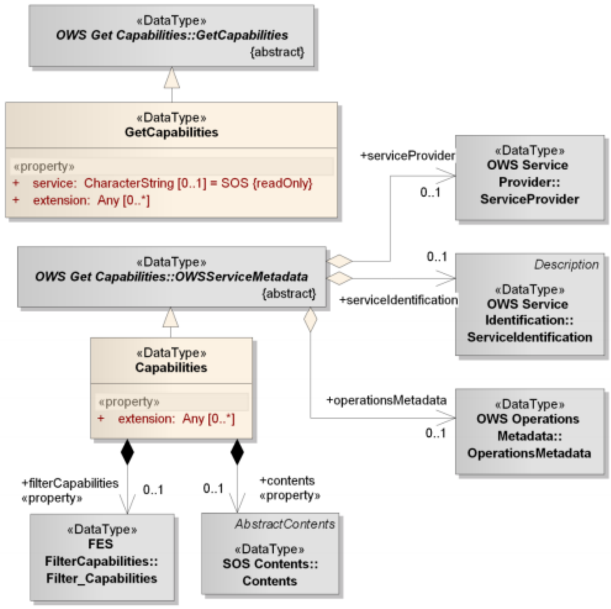

Figure 2.4: Data model behind the capabilities document of a SOS. Image courtesy by Bröring et al. [2012]

Get observation

Using `GetObservation` requests observation data can be retrieved. The request is made by taking the HTTP address of the SOS and adding `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, parameters 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 lets users retrieve sensor data using an identifier that points to a specific observation.

Get feature of interest

The `GetFeatureOfInterest` request returns information about the FOI of a certain observation. The response can contain all FOIs or only a subset which 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` as defined by ISO 19109 [ISO, 2005]. These features are implemented in the Geography Markup Language (GML) as defined by ISO 19136 [ISO, 2007]. The GML elements `gml:AbstractFeature` and type `gml:AbstractFeatureType` [ISO, 2011, p. 38] are used for `GFI_features`.

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 Subsection 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.

An 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. InsertResult is useful when there is limited communication bandwidth and processing power. It 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 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 FOI in space and/or time [INSPIRE, 2014]. Sampling is performed when a FOI 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 FOIs. 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 FOI 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*.

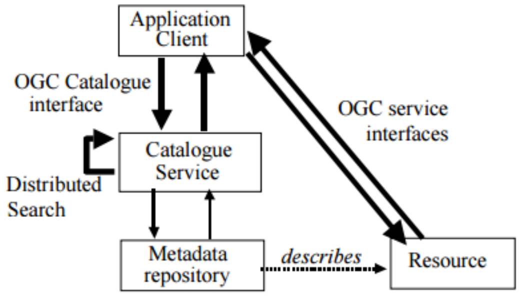

Figure 2.5: Architecture of Catalog Service for the Web. Image courtesy by [OGC, 2007, p. 26]

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, starting with the Catalog Service for the Web (CSW) (Subsection 2.3.1), after which the Sensor Observable Registry (SOR) (Subsection 2.3.2) and Sensor Instance Registry (SIR) (Subsection 2.3.3) 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" [OGC, 2007, p. xiv]. Figure 2.5 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.

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

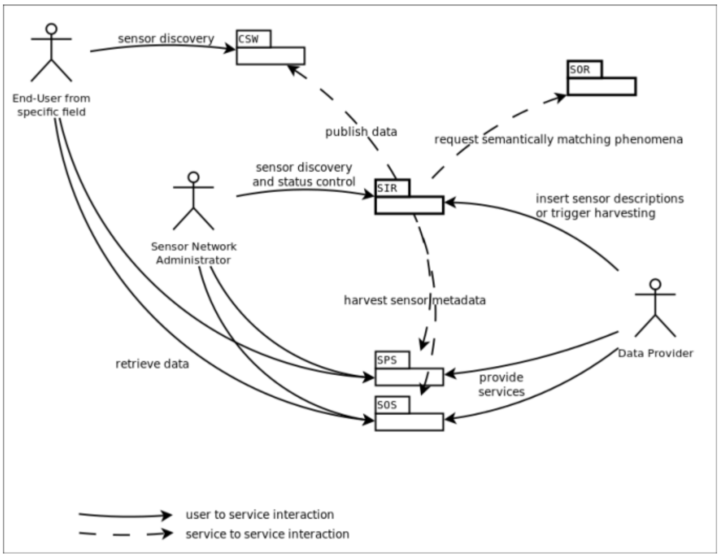

Figure 2.6: Overview of the interaction between CSW, SIR, SOR and potential users. Image courtesy by 52 North [2016]

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 Subsection 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 *generalisation*, *specialisation* 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 phenomenon.

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 Subsection 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 URL of the service 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 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 2.7 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 standardised, 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 to be 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 has not 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 internet 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 subsections the framework of the semantic web – the Resource Description Framework (RDF) – will be described (Subsection 2.4.1), followed by different RDF notations (Subsection 2.4.2). The last part explains the concept of Persistent Uniform Resource Locators (PURLs) (Subsection 2.4.3).

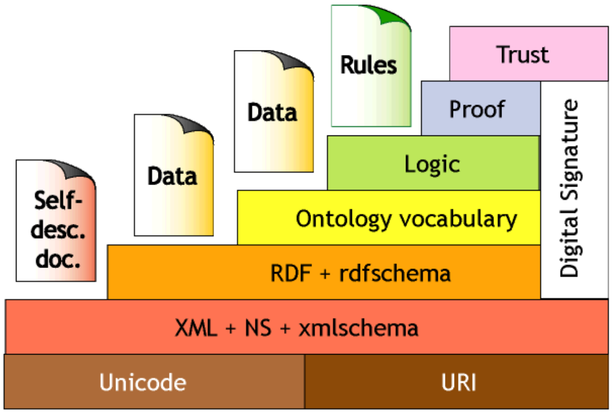

Figure 2.7: Hierarchy of the semantic web. Image courtesy by Berners-Lee [2000]

2.4.1 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 2.8 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 Uniform Resource Locator (URL) is an example of an IRI, but IRIs can also refer to resources without stating a location or how it can be accessed. These are the so-called URNs. An example of a URN is the following name for the WGS84 Coordinate Reference System (CRS): `urn:ogc:def:crs:EPSG::4326`. An IRI is a generalisation of an URI (Figure 2.9), but allows 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,

Subject	Predicate	Object
Delft	Is a	Municipality
Delft	Has geometry	POLYGON((X_1 Y_1, X_2 Y_2, ... X_n Y_n, X_1 Y_1))

Figure 2.8: Triples of object, predicate and subject define Delft as a municipality with a geometry

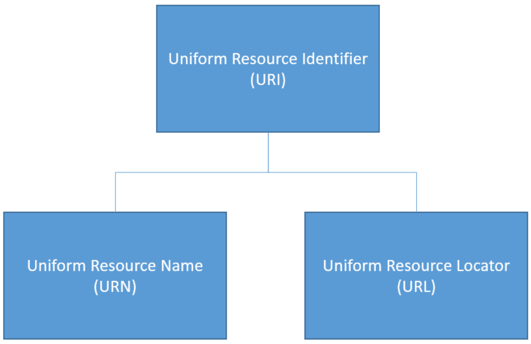

Figure 2.9: Structure of URI, URN and URL

```
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 2.10: Triples of Figure 2.8 in the Turtle notation

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., 2014]. This is added to the literal with the ^^ symbols, followed by the IRI of the datatype specification. In Figure 2.10 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].

2.4.2 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 2.10 shows how the triples of Figure 2.8 would be written using the Turtle notation. Turtle puts IRIs between brackets and ends a triple with a point.

In Figure 2.10 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. The N-triples notation is a simplification of Turtle. In this notation each line should contain a complete triple of subject, predicate and object, even if multiple triples share the same subject.

2.4.3 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 exist-

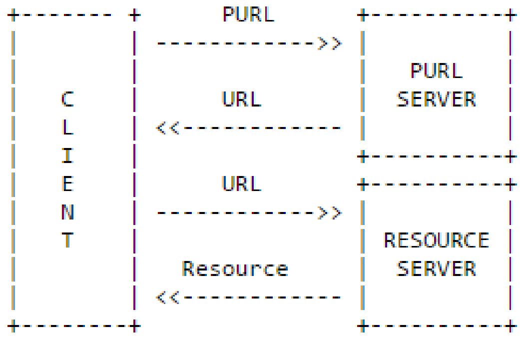

Figure 2.11: Persistent Uniform Resource Locator (PURL) resolves to the current resource location. Image courtesy by Shafer et al. [2016]

ing 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 2.11).

2.5 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.

2.5.1 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 2.12). 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.

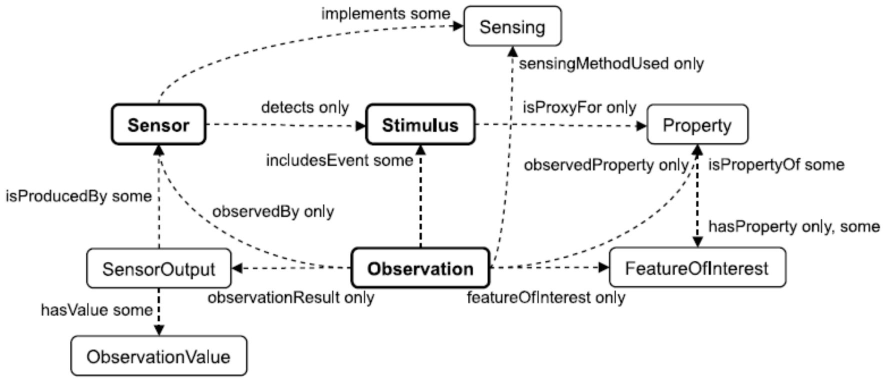

Figure 2.12: The stimulus–sensor–observation pattern [Compton et al., 2012, p. 28]

2.5.2 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 for sensor data discovery. 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'.

2.5.3 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 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.

2.6 SENSOR WEB AND THE INTERNET OF THINGS

A growing field of research is evolving around the Internet of Things (IoT). Sensor and network technology are becoming cheaper and the required hardware increasingly smaller. This allows them to be used for things that would traditionally not produce any sensor data at all, such as buildings, street lights and consumer electronics. Companies operating in a variety of different markets are investing in this development, accelerating the creation of an IoT [Price Waterhouse Coopers, 2014].

Barnaghi et al. [2012] point out that the heterogeneity of *things* makes interoperability among IoT devices a challenging problem. They explain that the semantic web could be of great importance for the description of IoT devices and their observations. However, "providing automated or semi-automated methods and tools to annotate, publish and access the semantic descriptions" is still described as a topic for future research [Barnaghi et al., 2012, p. 19].

Another study on the crossroads between the IoT and SWE has been performed by Jazayeri et al. [2015]. They have analysed and evaluated four interoperable open standards for the IoT: PUCK, TinySOS, SOS, and the SensorThings API. They found that all four open standards are useful for IoT, but that they all come with their own negative and positive attributes. This makes the decision for one of these standard dependent on the application for which they going to being used for. For memory constraint applications PUCK and TinySOS are the preferred candidates, while SOS (in combination with the the coap protocol) and the sensorThings API perform better in terms of bandwidth efficiency [Jazayeri et al., 2015].

2.7 SEMANTIC SENSOR DATA MIDDLEWARE

A number of studies have investigated how to add semantics to SOS. Often a layer of middleware is introduced, which connects the SWE service to the semantic web. These layers are placed on top of a SOS and process the requests it receives. They then return response documents using RDF, instead of the O&M schema.

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] have 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 presented method is still limited to retrieving sensors from a single source in a buffer around a point location.

Janowicz et al. [2013] have 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) FOIs. It is also argued that using both linked data services and data-specific services could ease the transition into the linked data world.

2.8 CONCLUDING REMARKS

In Section 2.7 the Sem-SOS [Henson et al., 2009; Pschorr, 2013] and SEL [Janowicz et al., 2013] approaches have been described. Both of these middlewares focus on combining the sensor web with the semantic web. However, they 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, they do not mention the integration of complementary sensor data from heterogeneous sources either. Stasch et al. [2011a,b] suggest interesting methods for aggregating sensor data based on Feature of Interests (FOIs). 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 Bröring [2009] and Jirka and Nüst [2010] 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.

3 METHODS

Chapter 2 discussed why the semantic web will be used for creating a semantic knowledge base of sensor metadata based on a review of recent literature and standards. This chapter presents a number of methods and standards which are required for creating a semantic knowledge base, such as a method for creating linked data (Section 3.1), observation data ontologies based on O&M (Section 3.2), a method for creating inward links using DBPedia (Section 3.3) and two implementations of spatial query functions in SPARQL (Section 3.4). The last part of this chapter describes the OGC standard for web services containing online processes: the Web Processing Service (WPS) (Section 3.5). Chapter 4 presents a conceptual system architecture based on these methods.

3.1 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 reused. 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 subsections will describe these three phases in more detail.

3.1.1 Phase 1: Preparation

The preparation phase is concerned 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. This content has been filtered to only contain the parts of data which are required as linked data.

3.1.2 Phase 2: Modelling

In modelling data to linked data the first step 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 it is preferred to (re)use an already existing ontology. Ontologies that have been implemented by others can be found in the Linked Data Cloud (`http://lod-cloud.net/`), which is an overview of linked data sets. Newly developed

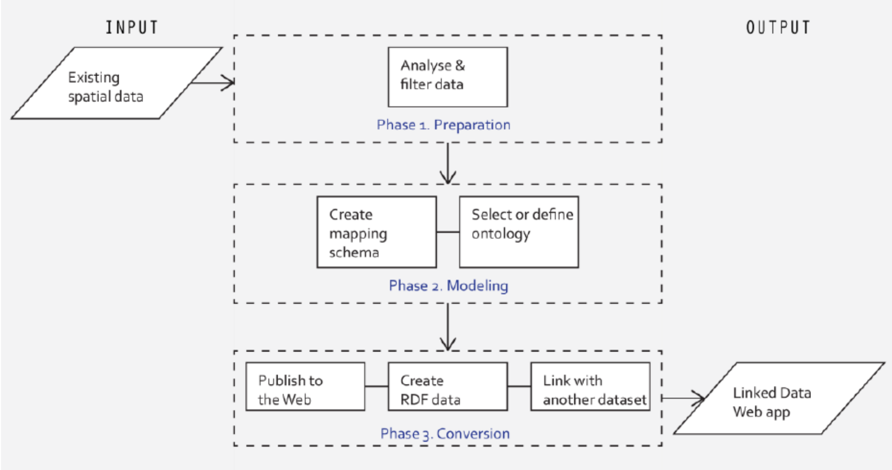

Figure 3.1: Workflow diagram for publishing linked open data from existing spatial data [Missier, 2015, p. 28]

ontologies can be found via an analysis of academic literature. 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. A mapping schema can be created to link parts of the data to their corresponding classes in the ontology. 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.

3.1.3 Phase 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. On the one hand, 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.

3.2 ONTOLOGIES

To publish data on the semantic web ontologies are required to specify the different classes and their relations. From the UML diagram in Figure 2.1 the classes `Process`, `ObservedProperty` and `FeatureOfInterest` can be mapped to classes belonging to OWL for observations [Cox, 2015c]. Instances of FOIs can be a `SamplingFeature` and a `SamplingPoint`. These can be mapped to classes from OWL for sampling features [Cox, 2015d]. The

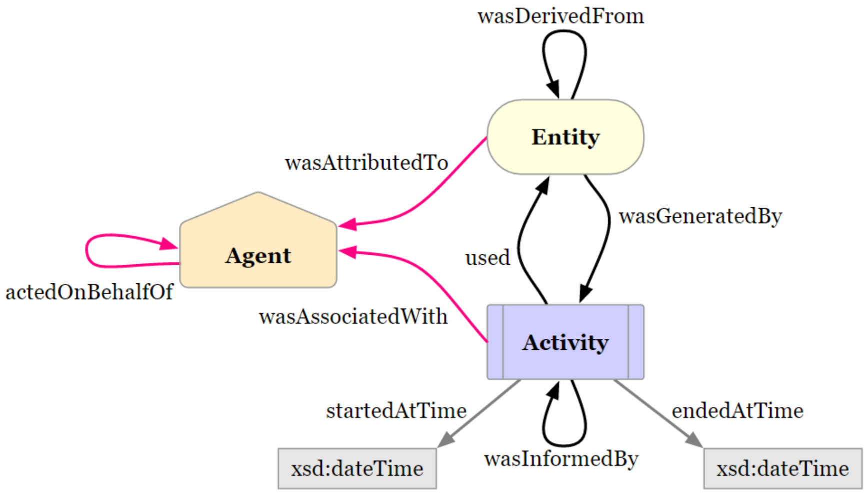

Figure 3.2: Basic classes and relations in the PROV-O by [Lebo et al., 2013]

PROV ontology can be used to describe changes in metadata for the `sensor` and `SensorObservationService` classes [W3C Semantic Sensor Network Incubator Group, 2011]. These ontologies will be described in more detail in the next subsections.

3.2.1 Observation metadata

In Section 2.5 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, unlike the SSNO. The om-lite and sam-lite ontologies will therefore be used in this thesis.

3.2.2 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. Figure 3.2 shows the relations between Entities, Activities and Agents. An Entity is a 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.

3.3 ADDING LINKS TO DBPEDIA

For adding outgoing links to their semantic knowledge base, DBPedia has created a GitHub repository called 'dbpedia-links' (`https://github.com/dbpedia/links`). The steps that should be followed to add links are described here, together with examples. The first step in establishing DBPedia links is to create an RDF file containing the links to be added. The subjects of these triples should be a DBPedia IRI, using either the main namespace `http://dbpedia.org/resource` or one of its `http://xxx.dbpedia.org/resource` subdomains, such as `http://nl.dbpedia.org/resource`. The predicates in this file can be: `owl:sameAs`, `skos:{exact | close | ...}Match`, domain-specific properties or `rdf:type` instances. The objects of the triples should contain the target URL of the DBPedia link.

The resulting file should contain one triple per line. The N-triples format is the required notation. After the file has been created it should be uploaded to GitHub using so-called pull requests. A pull request is a function implemented by GitHub in which a creator of (changes to) a document asks the administrators of a repository to include the new (changes to) a document to be accepted. The administrator will review the request and if no conflicts or mistakes are found he/she will accept it. Upon acceptance, the new additions are part of the repository.

3.4 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 subsections will describe these two extensions of SPARQL in more detail.

3.4.1 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 to define 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].

3.4.2 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 (Code examples 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.

Code example 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))
}
```

Code example 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)
}
```

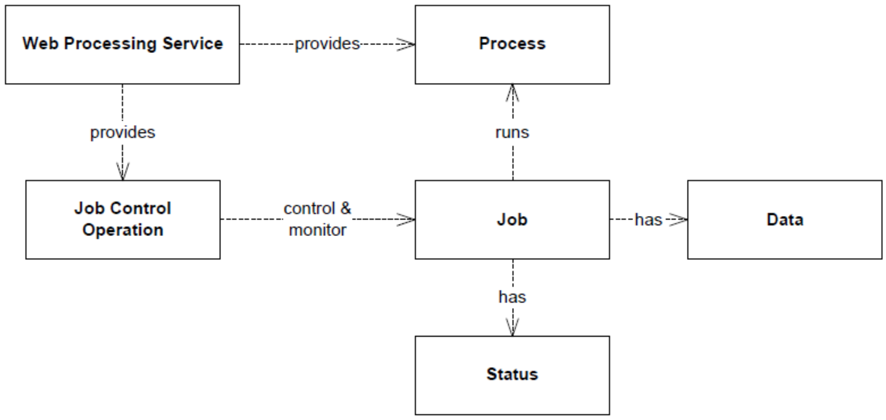

Figure 3.3: Artifacts of the WPS service model [OGC, 2015, p. 15]

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 [OGC, 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 section.

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 3.4 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 C.1.

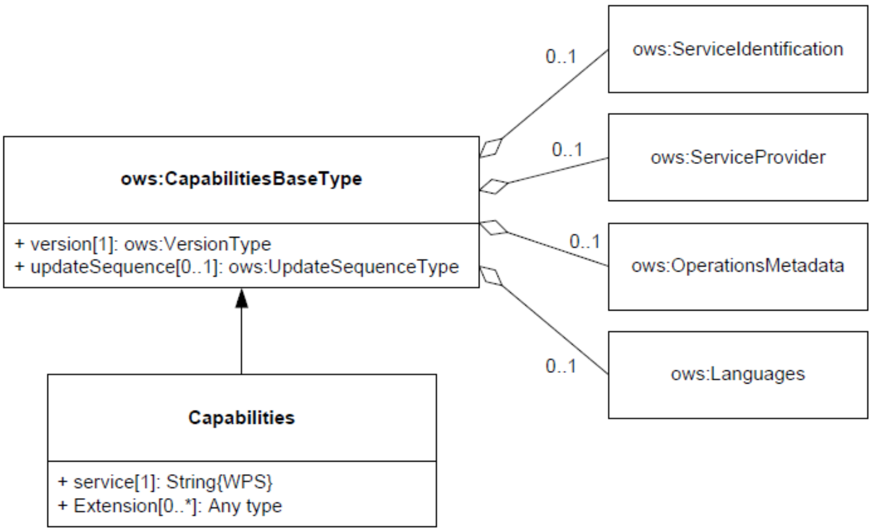

Figure 3.4: UML model of WPS capabilities document [OGC, 2015, p. 70]

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 section of the capabilities document. An example of a describe sensor response document can be found in Appendix C.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) [OGC, 2015, p. 36].

Complex data inputs are made for inputting complex vector-, raster- or other (non-spatial) 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 a *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 C.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 way 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.

4 CONCEPTUAL SYSTEM ARCHITECTURE

This thesis aims to design a conceptual system architecture 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 architecture will be described here using the methods of Chapter 3.

4.1 CREATING LINKED DATA FROM SENSOR METADATA

The proposed process of automatically creating linked data from sensor metadata is shown in Figure 4.1. A Web Processing Service (WPS) should be created containing processes for retrieving metadata, converting it to linked data and outputting it to a triple store. The data of a SOS is retrieved by a WPS process and automatically converted to linked data. The output are RDF documents containing the metadata as triples. These documents are posted to a SPARQL endpoint, where they can be queried using SPARQL.

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 of vector parcel data. The input of the process is the HTTP address of a SOS. Since both the requests and the data model are standardised 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 its 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.

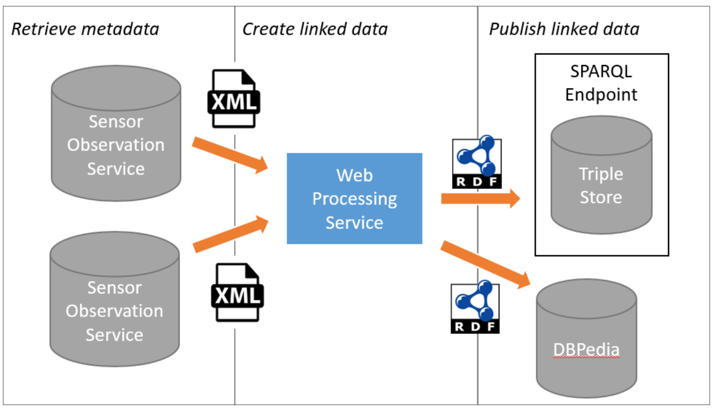

Figure 4.1: General overview of creating linked data of metadata from Sensor Observation Services

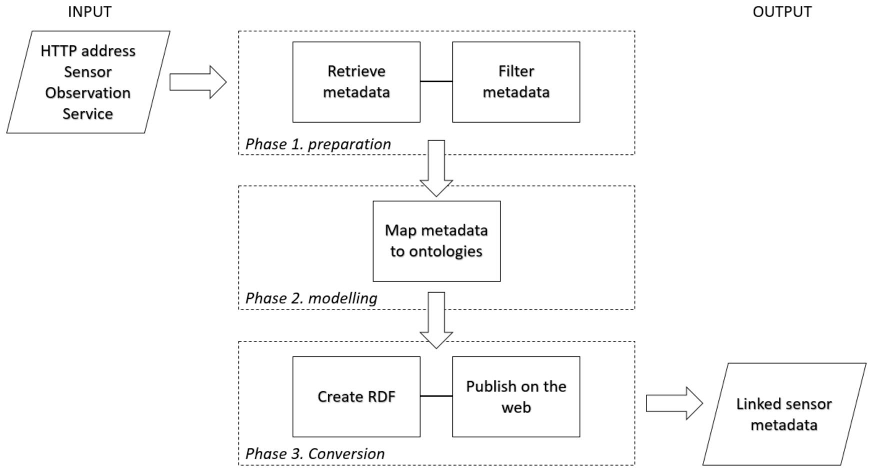

Figure 4.2: Workflow diagram of web process for creating linked sensor metadata (adapted from Missier [2015])

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 following will describe the way sensor metadata is modelled in a SOS (Figure 4.3), with which requests it can be retrieved and how it should be filtered.

Sensor metadata model

In the capabilities document a SOS describes a number of its properties, providing an overview of its metadata for clients looking for observation data. It identifies the organisation that maintains it, with at least the organisation's name and its contact information. Optionally, the organisation's website, keywords and an abstract about the SOS can be supplied. The SOS also describes its identifier and its HTTP addresses for sending 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 (sampling point) of which a certain property is being sensed. Roughly speaking, the FOI holds the sensor's location when it made an observation and the procedure describes the type of sensor. Therefore, the observable property ties together the procedure and feature of interest (Figure 4.3). It should be noted that on an abstract level the FOI does not necessarily have to be represented by a point geometry. It can also be generalized into larger features (e.g. multiple sensors observing different parts of a lake, where the lake is the ultimate FOI).

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

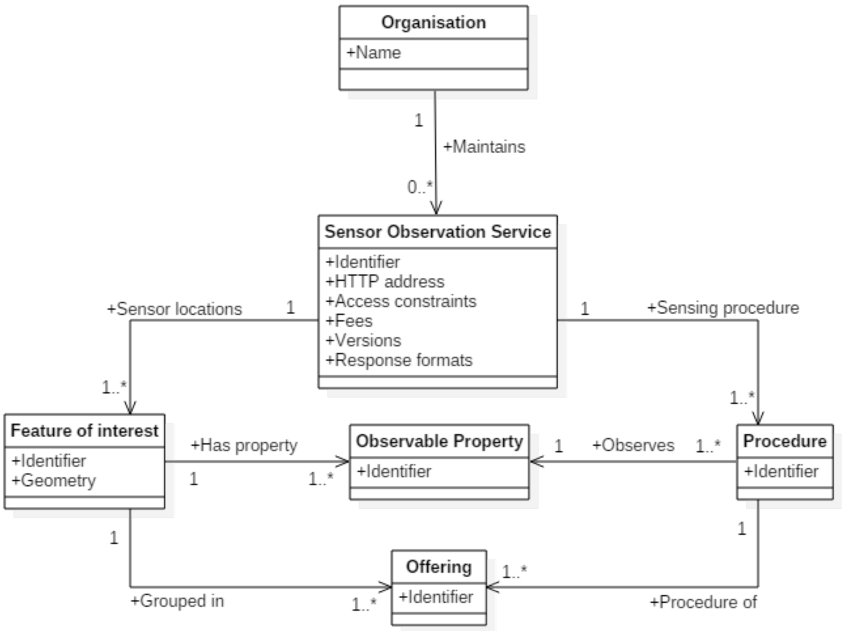

Figure 4.3: Sensor metadata retrieved from a SOS

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.

Metadata Requests

To retrieve metadata from a SOS a `GetCapabilities` 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 Subsection 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.

The capabilities document is unfortunately 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 clear yet 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` request always returns 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 Sensor Observation Services support 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 an XML document every element is defined using a namespace and a corresponding prefix. Often these prefixes are defined in the `xmlns` tag at the top of the document, to refer to these namespaces. The namespaces and corresponding prefixes 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 specification, describing the content of a response document. This schema is usually referenced to in the opening 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, 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 how 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. 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 (type of sensor) 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

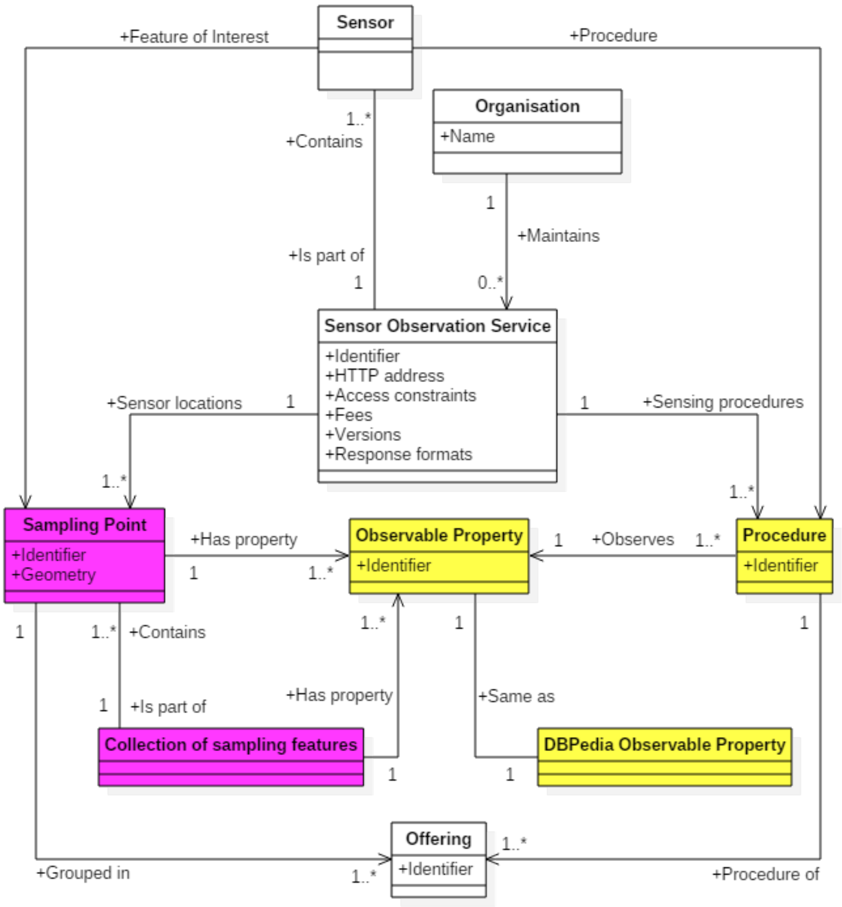

Figure 4.4: Sensor metadata modelled with the om-lite (classes in yellow) and sam-lite (classes in purple) ontologies

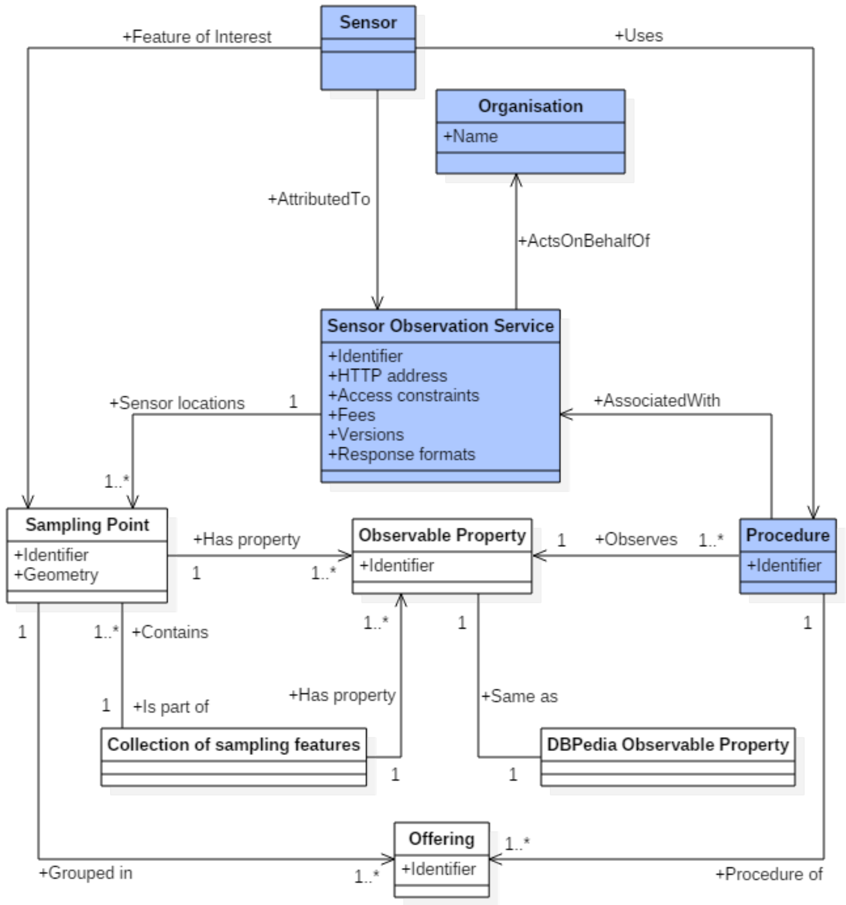

Figure 4.5: Sensor metadata modelled with the PROV ontology (PROV classes in blue)

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 location. Other types of sampling features can be added when the application requires this. The offering class is modelled as a specialisation of the collection 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 `observedProperty` 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 `observedProperty` 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) URI, used by the SOS.

4.1.3 Modelling with the DBPedia ontology

Apart from creating a descriptive mapping of classes – which are used to add semantic meaning to data – there should also be a mapping of inward links. This creates a bridge between related classes on the semantic web, from which the links trace back to the published semantic sensor metadata. Figure 4.6 shows the mapping between DBPedia classes and SOS classes. DBPedia contains a very broad description of a SOS. For discovering specific Sensor Observation Services it would be useful if all available SOS instances are linked to, by this DBPedia definition. For defining class instances the RDF schema provides a predicate `type` which could be used for this purpose. Every procedure or observable property instance represents a real world property or procedure. Some properties and procedures are already available as DBPedia definitions. These definitions should be linked by using the OWL `sameAs` predicate. The observable property could also be linked to its corresponding collection of sampling feautures. Since all the features a collection contains share the same observable property, the om-lite predicate `observedProperty` could be used. More relations could have been added to the model in Figure 4.6, such as the relation between sensor instances and the DBPedia definition of a sensor. However, links like these are left out, because they would add an overwhelming amount of triples on the side of DBPedia. Furthermore, all sensors can already be traced back via their FOIs, procedures, observed property and SOS.

4.1.4 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. This subsection will describe these steps in more detail.

Create Resource Description Framework data

For every mapped part of metadata from the SOS, one or more triples are created. These triples consist of at least Uniform Resource Names. However, preferably they are defined by Uniform Resource Locators which can be resolved to an RDF document, with semantic information about what it

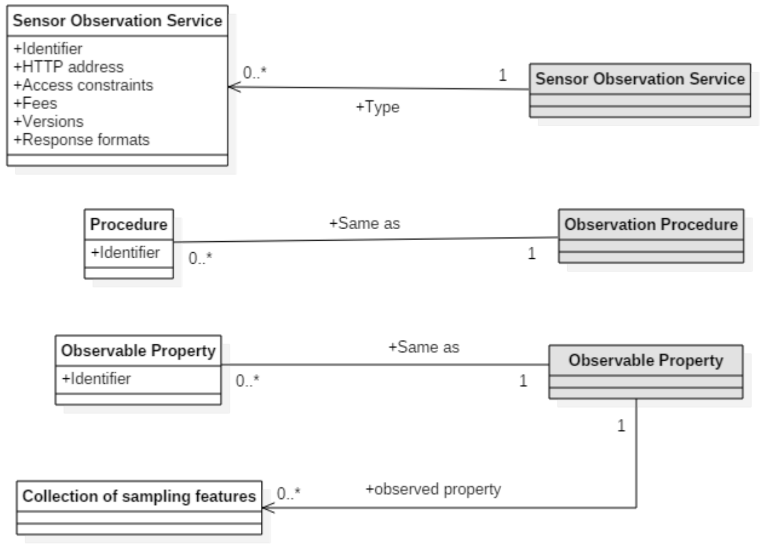

Figure 4.6: Mapping between DBPedia classes (grey) and SOS classes of Figure 4.4 (white)

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

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. Whenever a PURL is being retrieved, this server performs one of the six tasks shown in Table 4.1. 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 and 4.5. After these triples have been created the linked data should be serialised to an RDF document. The inward links mapping of Figure 4.6 has to be serialised into a separate document. For creating these two documents a notation has to be selected. As described in Section 3.3 the RDF document with DBPedia links should be stored using the N-triples notation.

Publish RDF on the web

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

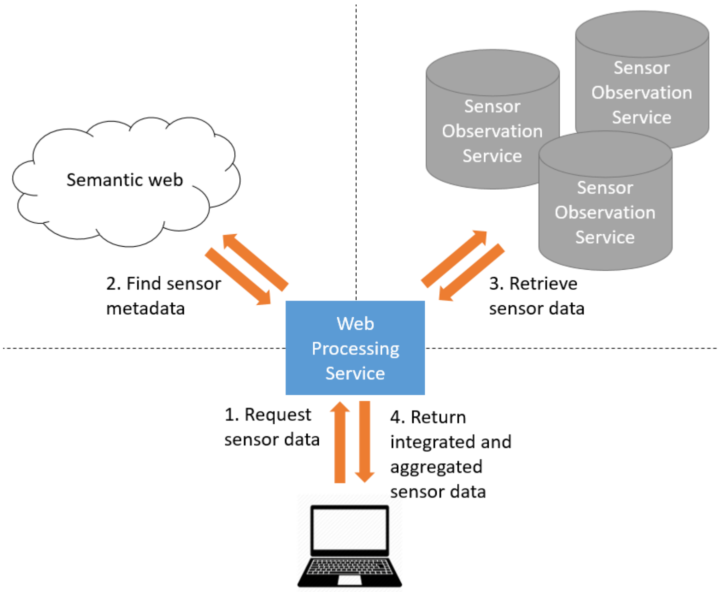

Figure 4.7: General overview of a Web Processing Service retrieving observation data using linked metadata of Sensor Observation Services

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 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.

The final step in publishing linked data is to make a request to the DBPedia administrators to accept the new links pointing to the observation metadata. This is done by sending a pull request to the DBPedia links GitHub repository, at the following address: `https://github.com/dbpedia/links`.

4.2 USING LOGICAL QUERIES TO RETRIEVE SENSOR DATA

The second web process retrieves a logical query and 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 4.7 shows the process of retrieving sensor data. The workflow of this web process is described in Figure 4.8.

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

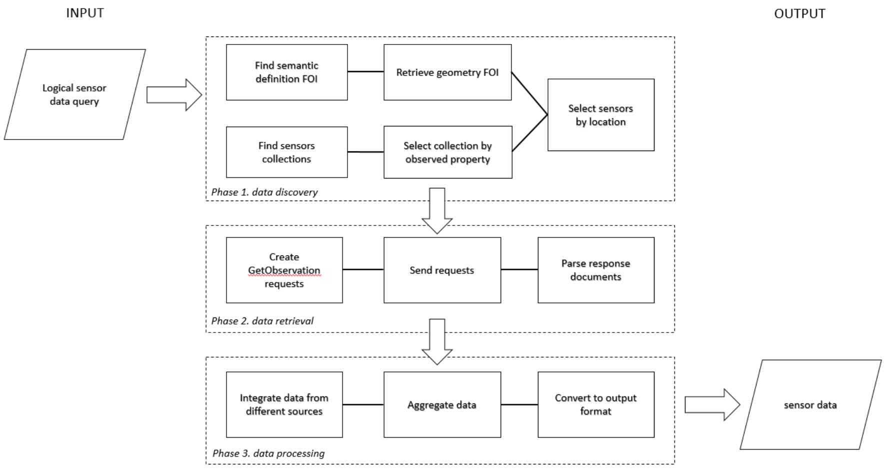

Figure 4.8: Workflow diagram of web process for retrieving sensor data

can be added. The input data, received by the WPS, is inserted in a number of SPARQL queries. First, the geometry is retrieved for all features inside the input list of FOIs. This is done using the name or identifier attribute of spatial features at the endpoint.

Then a second SPARQL query selects the sensors that are part of a collection, which observe a certain property of their FOIs. This query also contains a spatial filter with the retrieved geometries, which 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 Subsection 4.2.1 correspond to the `GetObservation` input parameters as described in Subsection 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, they should be added to only retrieve observation data from the temporal range in which the user is interested. 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 Subsection 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. Based on the temporal granularity subsets of the temporal range should be selected. After this *temporal sorting* has been performed, all observation values in each of these subsets should be aggregated according to a user defined method.

If a spatial aggregation method is part of the user's logical query, then it is performed after the temporal aggregation. For each of the spatial features for which observation data has been retrieved, a number of spatially aggregated data is produced. These are the observation results 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 are ordered in subset 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 subsets.

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 kind of information a user needs. 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 semantic reasoning 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]. The user has the option to retrieve the ouput data in one of these two formats.

5 PROOF OF CONCEPT

This chapter takes the designed system architecture of Chapter 4 and applies it using existing software tools, and Python libraries and modules to create a proof of concept. First, the data is described which has been used as input for the proof of concept (Section 5.1). Following, a detailed description of the implementation of the process for automatically creating linked sensor metadata is provided (Section 5.2). After that the implementation of the process for retrieving sensor data using logical queries is shown in Section 5.3. This section starts by showing the web application that has been created as an interface. The fourth part of this chapter describes how the two processes have been made available online in a Web Processing Service (WPS) (Section 5.4).

5.1 PREPARING GEOGRAPHIC DATA

A number of datasets have been used in this thesis, ranging from observation data inside Sensor Observation Services to vector and raster data, which can be used to request data about a certain area or inside certain features. The following subsections will go over the different datasets, describing their sources and contents (Subsection 5.1.1 to Subsection 5.1.3). The proof of concept requires the vector and raster datasets as linked data. However, since the spatial features were not available in RDF, they had to be converted first. The process of creating linked data of the vector and raster datasets is described (Subsection 5.1.4).

5.1.1 Vector data

For retrieving data inside a spatial feature, vector datasets have been converted to linked data. Data about administrative units are used to allow users to retrieve data about a municipality, province or country. Also, land cover is added to allow user queries to be focused on physical features, such as urban, agricultural or forest areas.

Administrative units

From the Dutch 'PDOK' website the datasets of Dutch provinces (*provincies*, Appendix A.2) and municipalities (*gemeenten*, Appendix A.1) have been downloaded. This data has been created by the Dutch Cadaster [Kadaster, 2015]. For the Netherlands there are 12 features in the provinces and 393 in the municipalities dataset.

It was very challenging to obtain data of administrative boundaries of Belgium (even from the INSPIRE data portal). Data is offered per municipality, province or region, To create a single data set of Belgium a large manual effort is required to the different local and regional data sets together. Therefore, all data for Belgium was retrieved from gadm.org [2015], a website which offers administrative units data sets of almost all countries in the world. This data set of Belgium contains 12 features representing the provinces (including the capitol region of Brussels) and 589 features representing the municipalities.

The country datasets have also been downloaded from gadm.org [2015] (Appendix A.3). The administrative unit data has been filtered to contain the names of the administrative units and their corresponding (multi)polygon geometry.

Land cover

Data on land cover is used to complement the data of administrative units. A section of the 2012 dataset from the Coordination of Information on the Environment (CORINE) programme, has been selected for this (Figure A.4). The entire CORINE dataset was retrieved from Copernicus Land Monitoring Services [2012]. The features overlapping the Netherlands and Belgium have been retrieved, by performing an intersection operation in the open source QGIS software. The resulting data is stored in a separate database in Postgres.

The database contains polygon geometries (Appendix 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.

5.1.2 Raster data

Data is often used in a raster representation for computations in a Geographical Information System (GIS). For natural phenomena a raster representation is especially well suited. The reference grid by EEA [2007] is a standard grid which covers Europe. It is available with a resolution of 100km², 10km² and 1km². In this thesis the European Environment Agency (EEA) grid cells with a resolution of 100km² (Appendix A.6) and 10km² (Appendix A.7) have been used which overlap the Netherlands and Belgium. 15 grid cells of 100km² and 843 grid cells of 10km² have been selected from the original dataset.

5.1.3 Sensor data

Air quality sensor data has been used from the Dutch national institute for public health and the environment (RIVM) and from the Belgian interregional environment agency (IRCEL-CELINE). Both of these organisations have a SOS where data can be retrieved according to the SWE standards. The SOS of the RIVM (`http://inspire.rivm.nl/sos/`) has been online since the 21st of August, 2015. IRCEL-CELINE has launched theirs on the first of January, 2011 (`http://sos.irceline.be/`). Appendix A.8 and Appendix A.9 show the sensor networks of both organisations. Their Sensor Observation Services provide different kinds of sensor data, such as particulate matter (PM_{10}), nitrogen dioxide (NO_2) and ozone (O_3). Appendix A.10 shows an example of a sensor location in the city centre of Amsterdam.

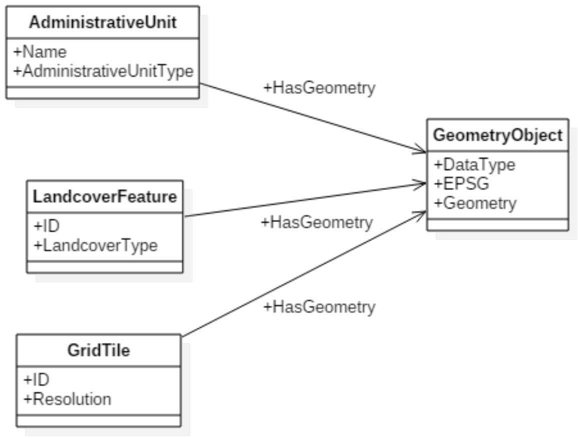

Figure 5.1: Model of vector and raster features

5.1.4 Creating linked data of spatial features

Linked data has been created in preparation for the proof of concept (Figure 5.1). The data is being used to retrieve and process sensor data on the semantic web. 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 $10km^2$ and $100km^2$. The method of Missier [2015] has been used for this (see Section 3.1).

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. The type of administrative units is defined using the DBPedia URIs for country, province and municipality.

The CORINE 2012 land cover dataset contains features with an identifier, a land cover type and a (multi)polygon geometry. 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 $10km^2$ and $100km^2$. Every feature is defined by an identifier, a resolution and a point geometry of the origin. The identifier is a code given to a feature by the EEA and has the form of: resolution + `E` + x coordinate + `N` + y coordinate.

5.2 CREATING LINKED DATA FROM SENSOR METADATA

After the linked data of spatial features has been prepared, this chapter continues with creating the proof of concept. The first part of the system architecture described in Chapter 4 is a process which automatically harvests sensor metadata from a SOS using various requests. The retrieved data should then be mapped to ontologies and serialised to an RDF document (Figure 5.2). The next subsections describe how these steps have been implemented using the Python programming language.

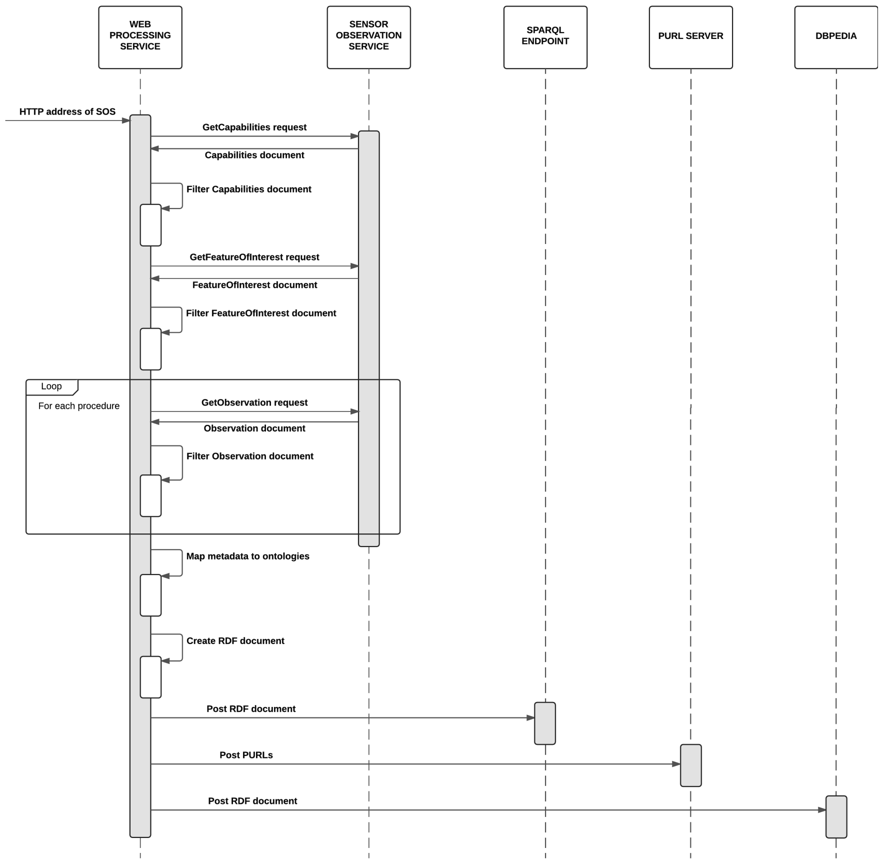

Figure 5.2: UML sequence diagram of the web process for creating linked sensor metadata

Code example 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
```

5.2.1 Making sensor metadata requests

A Python class object has been 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. To create a SOSclass instance, the URL of the SOS has to be entered as input value. The initialisation of the SOSclass instance creates empty variables 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. This allows information about the CRS, procedures and observed properties to be added to each FOI. 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 makes all requests to the SOS shown in Figure 5.2. It starts by sending a `GetCapabilities` request to the SOS. For making HTTP GET and POST requests the Requests[1] library for Python is used. Code example 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, 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 Subsection 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.

1 http://docs.python-requests.org

Code example 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)
```

5.2.2 Map metadata to ontologies

After each request is send an XML document is returned. To retrieve data from these XML documents the LXML[2] library for Python is used. With this library the XML document can be loaded into an Python object, which allows XML processing, such as looping through the elements and searching the whole document for elements with specific tags. Code example 5.2 shows a snippet of code that takes the response document retrieved from Code example 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 (see Figure 5.2). For handling RDF the Python package RDFlib[3] is used. RDFlib defines an RDF graph to which triples can be added. Code example 5.3 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.

5.2.3 Publish linked data

For publishing linked data the Strabon endpoint (Figure 5.3) is used in combination with an Apache Tomcat server. The Strabon[4] endpoint is a semantic spatio-temporal RDF store, originally developed for the European *Semsor-Grid4Env* project. It uses a Postgres database with the PostGIS extension to store RDF triples. It therefore allows spatial SPARQL queries to be made using the stSPARQL extension (Subsection 3.4.2).

2 http://lxml.de
3 https://rdflib.readthedocs.org/
4 http://strabon.di.uoa.gr/

Code example 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#")

# Initialise 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) )
```

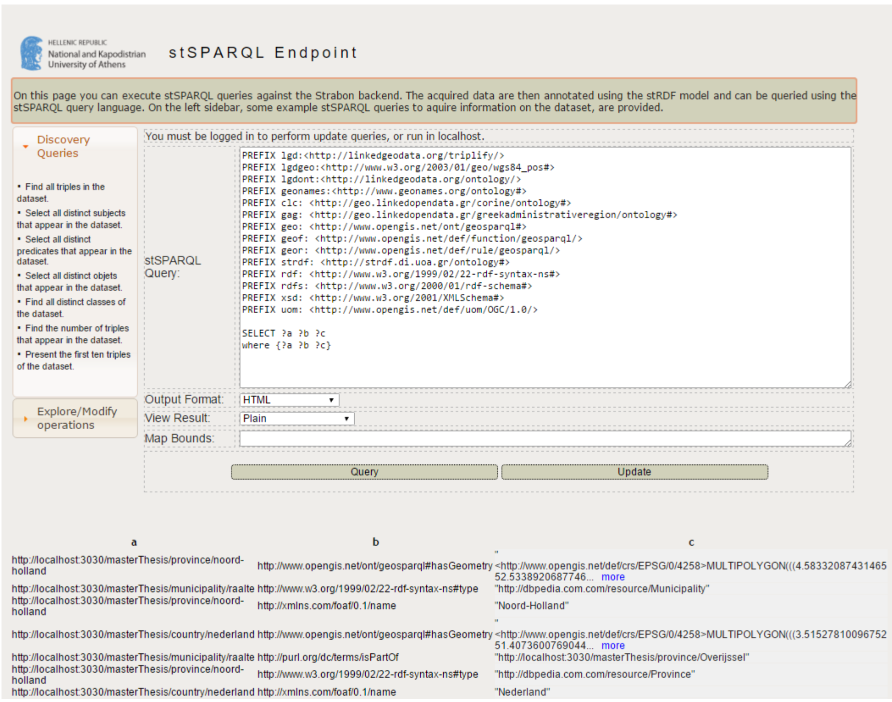

Figure 5.3: User interface of the Strabon endpoint

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

```
import os
import requests
import rdflib

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

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

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

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

# Post the sensors.ttl 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" })
```

The first step of publishing the graph which has been created using RDFlib, is to store it in an RDF document. This process is called the *serialisation* of an RDFlib graph. To do this the 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 at the endpoint. To do this, a Session object is created using the Requests library. This object posts it login credentials to the endpoint, after which it can be used for posting RDF data. Code example 5.3 shows how to log in to the endpoint with a session object. After that, the graph is serialised to the document sensors.ttl, which can then be posted to the Strabon endpoint.

For creating Persistent Uniform Resource Locators the Purlz[5] software has been used. 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 an HTTP GET request: http://localhost/strabon-endpoint-3.3.2-SNAPSHOT/Describe?submit=describe&view=HTML&handle=download&format=turtle&query=DESCRIBE<a_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 of query. The parameters 'view=HTML&handle=download' indicate that the endpoint's website is requested, but the returned data should be a downloadable file

5 http://www.purlz.org/

Code example 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>
```

instead of an HTML page. The parameter '`&format=turtle`' sets the RDF notation of the downloadable file to Turtle and '`&query=DESCRIBE <a_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 Code example 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 as described by 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 Code example 5.4. An example of a batch file can be seen in Code example 5.5. These XML files are created using the Python package LXML.

Chapter 4 also describes how to establish inward links from DBPedia, which improve the discovery of sensor metadata. However, this part of the method has not been implemented in the proof of concept, because the proof of concept has been tested locally. The DBPedia knowledge base is available online for a large user audience and can therefore only accept links to reliable resources. Once the method presented in this thesis has been further developed into a stable web process this functionality can be added as well. However, at the moment it could not be part of the proof of concept implementation.

5.3 USING LOGICAL QUERIES TO RETRIEVE SENSOR DATA

Section 5.2 described how the proof of concept implementation harvests sensor metadata from a SOS and publishes it as linked data in a SPARQL endpoint. The process presented in this section uses that data and its semantics, to automatically retrieve observations based on a user's logical query (Figure 5.4). For handling this process the `request` class has been created in Python, based on the architecture of Section 4.2. It has been implemented to store all logical input parameters, automatically convert them to SOS requests using the semantic web and to return the data in an integrated and aggregated way. The next subsection will first introduce the user interface,

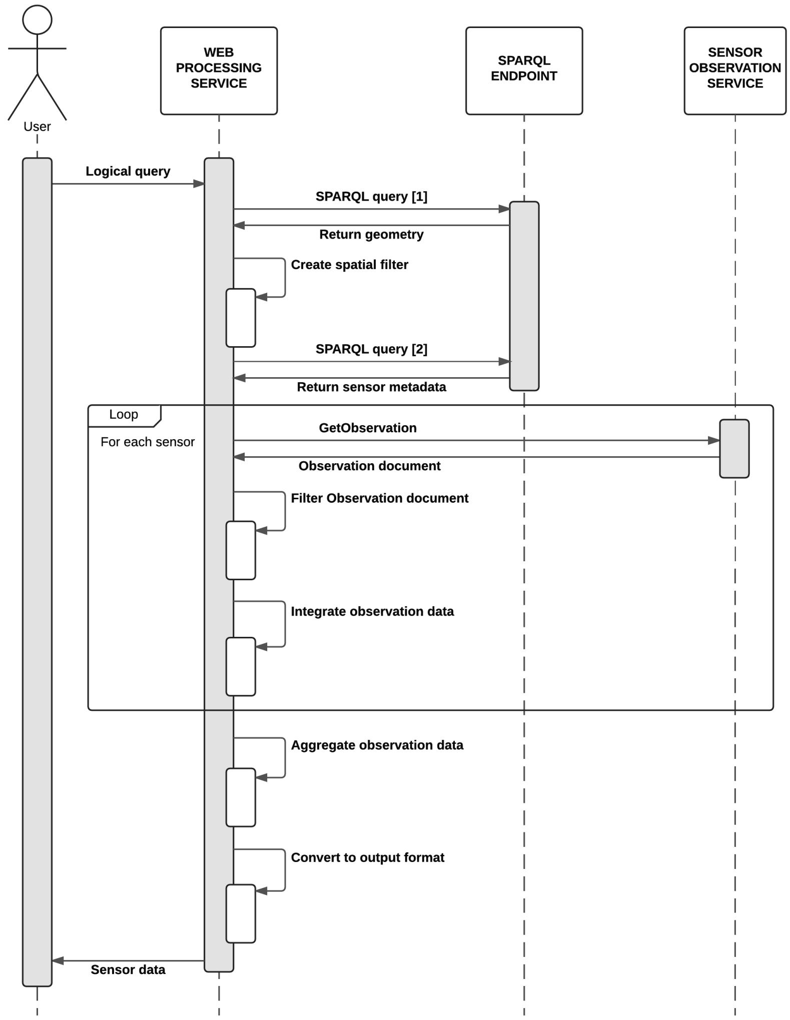

Figure 5.4: UML sequence diagram of the web process for retrieving sensor data

after which the following subsections describe how the different parts of the process have been implemented.

5.3.1 Web application for retrieving sensor data

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 features. 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 its 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 observation data per input feature.

A web application has been created as an interface for the proof-of-concept implementation. It has been created using Flask[6], which is a microframework for creating web applications with Python. The first interface that users see is a map on which features can be selected (Figure 5.5). This can be done by either clicking on them or by manually writing their names in the form below. Also an observed property can be selected in this form.

When the user submits this form the WPS starts looking for sensors that observe this property and that are located in the selected area (with the SPARQL queries shown in Figure 5.4). Once the sensors have been found they are visualised on the map as orange dots, which can be clicked on to see their URI, the observed property URI and the HTTP address of the corresponding SOS (Figure 5.6). Below the map a new form is loaded, which allows users to select spatial and temporal aggregation methods and a temporal range (Figure 5.7). As the user submits these preferences the WPS starts retrieving observation data, from the Sensor Observation Services that contain the selected sensors. After the process of Figure 5.4 has finished, the web application returns a graph to the user, providing a visual impression of the data (Figure 5.8). The graph is created using Vega-Lite[7], a high-level visualisation grammar, developed by Interactive Data Lab of the University of Washington. This graph can be exported as a .PNG image. Alternatively, the data itself can also be downloaded.

5.3.2 Discovering sensors

The input category is the 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. Code example 5.6 shows that the filter expression is defined first. This expression is 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 URL is found by adding it to DBPedia's standard method for defining URIs: `http://dbpedia.org/resource/`.

A SPARQL query is then made to find a sensor collection that is related to 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 as well: 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 some that are outside the original vector feature. Therefore, the WPS performs the spatial

6 `http://flask.pocoo.org/`
7 `https://vega.github.io/vega-lite/`

Figure 5.5: Request sensor data interface (step 1) with parameters for observed property and spatial features

Figure 5.6: Selecting a sensor on the map shows the semantic URL of the sensor, its observed property and the address of its SOS

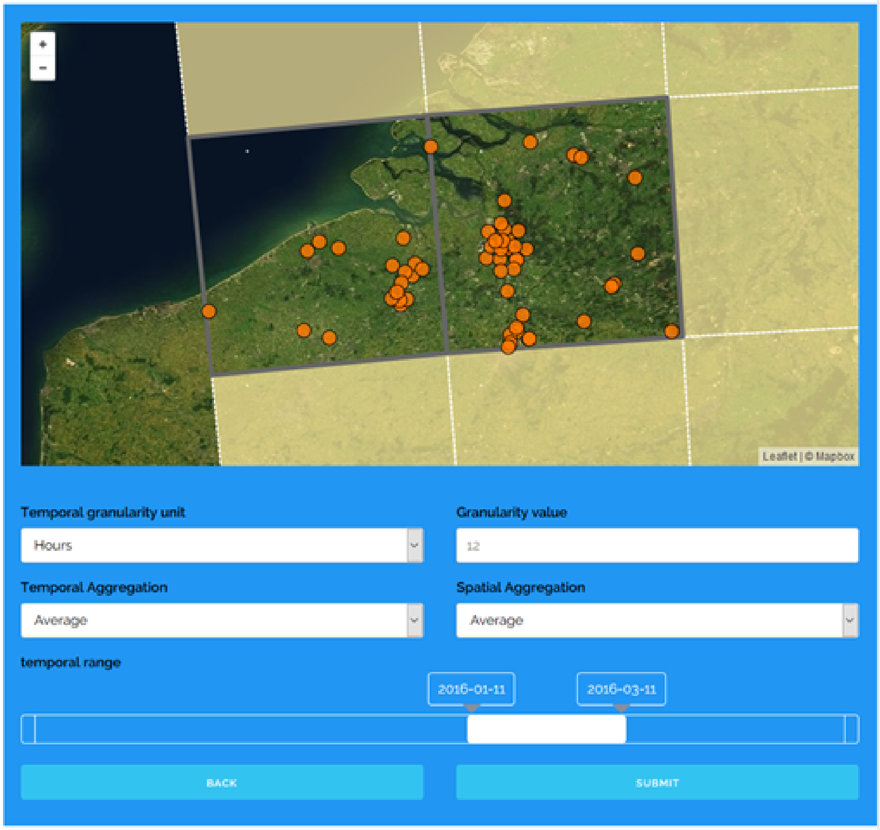

Figure 5.7: Request sensor data interface (step 2) with parameters for temporal granularity, spatial and temporal aggregation methods and temporal range

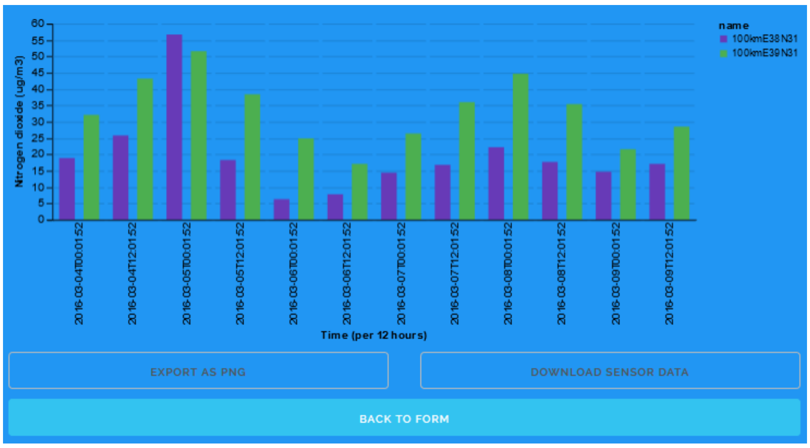

Figure 5.8: Returned observation data is visualised as a vega-lite graph, with the option to export the graph or download the data

Code example 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
```

filter afterwards and removes all locations that are outside of the requested feature.

This type of query is shown in Code example 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. 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 om-lite and sam-lite ontologies are used in this query to find the FOIs with their sensors and corresponding procedures. The stRDF ontology is used for the retrieval of FOI geometries, using a spatial filter.

The second alternative checks which Sensor Observation Services have sensors observing a certain property, within the bounding box of the requested vector geometry. The response of the SPARQL query is a set of Sensor Observation Services. For all of these services `GetObservation` requests are made containing a spatial filter. This performs the detailed spatial querying on the SOS side, instead of at the endpoint.

5.3.3 Retrieving sensor data

After all sensor locations have been retrieved from the SPARQL endpoint `GetObservation` requests are made. These require the identifiers of observed properties and procedures which were given by their SOS, instead of the semantic URLs which were assigned to them later. The requests are structured as explained in Subsection 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.

Code example 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. 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 from the `GetObservation` requests, 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, FOI and observed property.

The SWE data array is an array of observations that share the same metadata. For all observations which have the same feature of interest, procedure, observed property and UOM, the result data is joined together into an array of result 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 com-

Code example 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)
   )
}
```

Code example 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)
```

bination 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.

5.3.4 Data aggregation

After all observation data is retrieved, it is first aggregated temporally. An empty dictionary is created to store each temporal granularity range, which 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 outputs the time range from the start of the temporal range to the time of the observation. From this time range a modulo operation calculates the ratio between the temporal granularity and the time range between the start of the temporal range and the time of the observation. A dictionary key is created to sort the observations. To calculate the key the temporal granularity is multiplied by the outcome of the modulo operation, after which the start time is added to it.

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. Code example 5.9 shows how the observation data stored as strings containing comma separated values, 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 for each feature per temporal range. For both temporal and spa-

Code example 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]))
```

tial aggregation the basic methods have been implemented such as average, minimum, maximum, sum and median. These methods can be further extended to give more reliable results [Ganesan et al., 2004] or to give more semantic meaning to the aggregation methods [Stasch et al., 2014].

5.4 SETTING UP THE WEB PROCESSING SERVICES

The proof of concept web processes described in Section 5.2 and 5.3 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 enable the integration, publishing and execution of Python processes via the WPS standard (see `http://pywps.org/`).

Code example 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 the process is defined. The process in Code example 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

Code example 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()
```

Apache distribution, that includes a number of useful features such as the Tomcat server (used for hosting of the Strabon endpoint). A `pywps.cgi` file is defined, which 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 Code example 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.

6 RESULTS

Chapter 4 has presented a conceptual system architecture which uses the semantic web to improve sensor data discovery as well as the integration and aggregation of sensor data from multiple sources. This has been implemented in a proof of concept implementation (Chapter 5). In the following sections the outcomes and results of this architecture are discussed (Section 6.1 to 6.5). Afterwards, a comparison is made with the Catalog Service for the Web (CSW) (Section 6.6) and with semantic sensor data middle ware (Section 6.7).

6.1 IMPLEMENTATION DIFFERENCES OF SENSOR OBSERVATION SERVICES

The sources of sensor data used for the proof of concept 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 method for approaching these Sensor Observation Services a couple of differences came to light in their implementation. 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 FOIs, 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. It is not possible to automatically 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 differences have caused the proof of concept described in Chapter 5 to become more complex than initially expected. Providing a `describeSensor` document with as much information as possible (containing related features of interest, observed property and offerings) is especially important for making sense of the metadata. Working with response documents is much easier if the appropriate SensorML classes are being used to describe sensor metadata, instead of adding it to the more general `swe:keywords`, or abstract section for example. It is not inconsistent with SWE standards to include metadata in one of these ways, but it does not create machine understandable content.

6.2 SEMANTICS IN SENSOR OBSERVATION SERVICES

The metadata in Sensor Observation Services do not necessarily contain semantic URIs as identifiers. Therefore the workflow of Chapter 4 includes a process for creating these semantic URIs based on the SWE XML schemas. However, two issues occurred. First of all, the use of identifiers in a SOS which are completely meaningless, especially from a machine readable point of view. The lack of defined observable properties and procedures are the main problem, as they combined with a FOI represent a deployed sensor. Secondly, there is a lot of complexity involved in automatically creating URIs for metadata, where each URI is unique, compact and valid. These two issues will be further discussed in the following subsections.

6.2.1 Mapping of observable properties

In Section 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. The workflow of Chapter 4 uses the sensor location, temporal range and observed property as the main elements of the logical query input. Therefore, ill-defined procedures and 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. The vocabulary used by the RIVM for observed properties is a step in the right direction, as it provides a textual description. However, these identifiers do not resolve to an RDF document, but to the EEA website. This is therefore 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. For example, the `owl:sameAs` predicate can be used to link to a semantic ontology, or the `foaf:name` predicate can be used to store at least the full name of the observable property. As such, by assigning just a single triple 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 implementation was currently not able to automatically create linked data from the metadata of a 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 in 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 deployed sensors are implicitly represented by combining the procedure with a FOI, although they are not part of the SOS XML schemas. 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. 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 object the URI represents. Therefore, this could be preferred to a completely random identifier. However, as shown in Section 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. Nevertheless, 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.

6.3 MISSING CLASSES IN LINKED DATA ONTOLOGIES

A sensor has a large amount of metadata, which can be retrieved from a SOS. First of all, there is metadata according to the SWE schemas. This defines aspects, such as observed property, FOI and procedure, which can be described using om-lite and sam-lite ontologies (see Subsection 4.1.2, Appendix B.2 and Appendix B.3). Secondly, the SOS instance and its functionality are also part of a sensors metadata. However, a number of these metadata classes are missing in the linked data ontologies. There is for example, not a description of what a SOS is, which request and response formats they support, and what offerings are.

To overcome this issue in the proof of concept, the PROV ontology has been used to describe these classes on a more abstract level. As shown in Section 3.2.2 Sensor Observation Services, sensors, FOIs and procedures can be described in terms of `Entities`, `Agents` and `Activities`. Appendix B.1 provides an overview of how these instances are mapped to the PROV ontology. To link output formats and sensors to a SOS, some basic Dublin Core[1] classes have been used as well, such as *hasFormat* and *isPartOf*. These ontological classes have helped providing a minimal definition. It enabled a SPARQL query to retrieve the required information from the endpoint. However, the proof of concept knows that it only contained SOS metadata. The metadata itself is not sufficiently defined. If a web process does not know in advance that SOS metadata is described by the semantic knowledge base, it would not be able to find this out on its own.

6.4 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 spatially relevant to the user's data request. However, the proof of concept implementation showed that the performance of vector geometries in SPARQL is not optimal. Also, the order of latitude and longitude in point coordinates turned out to be a problem. In the next two subsections these issues will be discussed, together with the implemented solutions.

1 http://purl.org/dc/terms/

6.4.1 Vector queries with SPARQL

The workflow described in Section 4.2 starts with discovering metadata of sensors which 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 lacking when large amounts of data are being retrieved. The endpoint has limited its ability to receive queries to a maximum amount of characters per query. Vector geometries can easily exceed this length. Therefore, three methods for spatial querying have been implemented to retrieve metadata of sensors inside a vector feature. First of all, a method has been implemented which creates SPARQL queries with a spatial filter expression containing the original vector geometries. Secondly, an alternative has been implemented which creates SPARQL queries in which EEA raster cells overlapping the vector geometry are added to a spatial filter expression. Thirdly, an approach is tested in which the spatial querying is performed on the SOS side using spatial filters in `GetObservation` requests. The three methods are described in more detail below.

The WPS does not have to perform any further spatial queries with the first method. A single query retrieves the metadata of sensors inside the geometry. This is a fast procedure, since it uses a Postgres database in combination with the Postgis extension. Responding to a SPARQL query with a spatial filter is a matter of seconds in the proof of concept implementation. However, when more complicated vector geometries are used, the WKT definition gets 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 both of them handle spatial SPARQL queries. The Strabon endpoint has been used in the final proof of concept, because it handled long queries better. However, queries still get 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 which 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 sent with a spatial filter. The problem 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 Sensor Observation Services it could have been a viable alternative. However, another disadvantage of this method is that only a very rough spatial filtering is applied on the endpoint side using a bounding box, since the detailed spatial filtering happens on the SOS side. Therefore, Sensor Observation Services might have sensors inside the bounding box of the vector geometry, but not inside the vector geometry itself. This leads to unnecessary requests to those services Sensor Observation Services, lowering the performance of the entire process of retrieving sensor data.

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.

6.4.2 Latitude and longitude order in point coordinates

The proof of concept implementation had to be adjusted to cope with inconsistencies regarding the order in which latitude and longitude are being presented. The SOS of the RIVM provides point geometries as longitude, latitude. However, the Strabon endpoint expects the order to be latitude, longitude. The coordinates by IRCEL-CELINE do use the latitude, longitude order. 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 specification. 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 a remark by 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 which use the latter format for their CRS are left as they are. Geotools.org [2015] describes that in theory the format of a CRS URI has nothing to do with the latitude and longitude order. Therefore, there is no guarantee that this solution will continue to work over time as it is not part of any specification. Still, this approach works for the Sensor Observation Services used in this thesis, but it is not the preferred solution.

6.5 REUSING THE OM OBSERVATION SCHEMA FOR OUTPUT DATA

After sensor data is retrieved from the Sensor Observation Services and further 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 from different procedures has been used to create a single aggregated value. The Sensor Observation Services by the RIVM and IRCEL-CELINE both use different procedures for their observations. A procedure is defined in O&M as a method, algorithm or instrument, or a 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. It could be discusses to what extent it is problematic for different procedures which 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 the exact functioning of 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 sensors observing the same property of their FOIs can 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 added with different procedures. According to the XML specification, the procedure is nillable, but only "where the information, though strictly required, is not available" [OGC, 2011, p. 42]. This is not the case for the observations used in this thesis, they provide URIs for their procedures. 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 for dealing with procedures of aggregated data, 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 given a unique URI which is placed in the response document. This 'system' would be imagined ad hoc, only when a user's query is received. When only a URN is returned – which suffices for the O&M observation schema – this is a relatively simple workaround, but the semantics which 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 should be further researched.

6.6 COMPARISON WITH A CATALOG SERVICE FOR THE WEB

In Subsection 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 Bröring [2009] and Jirka and Nüst [2010] have developed the SIR and SOR extensions. When the method of this thesis is compared with SIR and SOR, a lot of similar functionalities can be observed. First of all, there are harvesting functions present, requiring an HTTP address of a SOS as input to retrieve sensor metadata. Also, the metadata is stored in a central location, providing users with an overview of available data and services. Furthermore, requests can be made for retrieving metadata about a subset of the sensors, which share certain properties. These are all characteristics of the CSW in combination with SIR and SOR, as well as of the conceptual system architecture described in Chapter 4.

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

A second difference is the platform used 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`. 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 be linked to and from 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 the open world assumption. In this case, currently unknown or unavailable information is not viewed as non-existing. The semantic knowledge base allows other data which 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 SWE related data and non-SWE related 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 8).

6.7 COMPARISON WITH SEMANTIC SENSOR MIDDLEWARE

The method presented in this thesis could also be compared to the so-called semantic sensor middleware as described in Section 2.7. These middlewares create a layer on top of a SOS, allowing them to output their contents semantically enriched. 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. In the backend they use a linked data model that allows the SOS to return semantic 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 Sem-SOS and SEL are enriching a single SOS with semantics. This thesis has a different scope, were metadata of multiple Sensor Observation Services are combined, providing an overview which allows for discovering and understanding sensor data (Figure 6.1). The WPS performs the tasks of discovering, retrieving, integrating and aggregating data, lowering the burden on users. A semantic sensor middleware has not been created to automatically discover and retrieve observation data, but focuses on making the SOS data accessible with added semantics or in the RDF format. In the 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, unless it is connected to a CSW. By explicitly storing the metadata in a semantic knowledge base instead of a semantically enriched SOS service, this question can be answered with

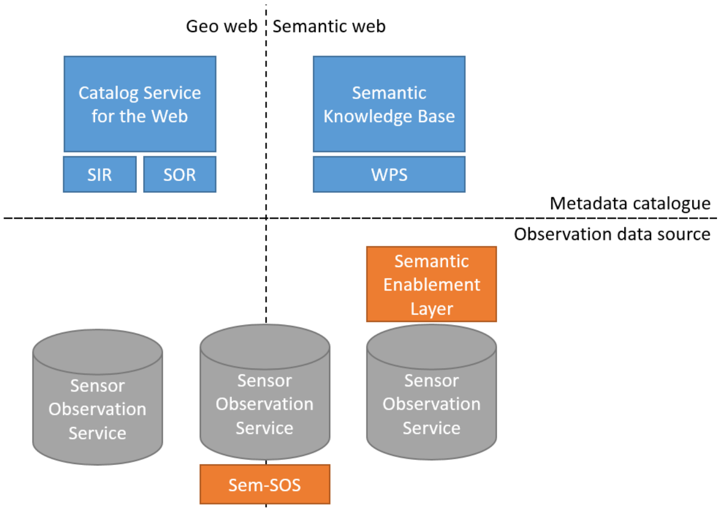

Figure 6.1: Distinction between solutions for metadata catalogues and observation data sources

SPARQL queries to the endpoint. Furthermore, the method described in this thesis does not only describe data semantically, but also makes it discoverable by adding links from another semantic knowledge base (DBPedia). This increases the chances for observation metadata to be discovered.

The advantage of the middleware approaches is that data is not duplicated, but only returned in a different format. 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 has to be frequently updated. Alternatively, it could also be retrieved directly from the source instead of duplicating it. The Sem-SOS and SEL provide a method for retrieving RDF data about observations directly from the SOS source.

6.8 CONCLUDING REMARKS

In Chapter 4 a conceptual system architecture has been presented. This architecture has been tested using a proof of concept implementation. The results of this proof of concept show that the conceptual system architecture can be implemented, creating a semantic knowledge base with sensor metadata combined from different Sensor Observation Services. However, a number of issues have also surfaced with respect to the SWE standards. There are differences in the use of SWE standards between Sensor Observation Services and there is not always a sufficient description of the metadata. This makes it difficult for other users (both human users and automated processes) to understand the metadata. Another issue is the complexity which is involved in creating completely automated processes for outputting linked data of SOS metadata. The data is retrieved from multiple sources and there can be differences in the way these sources have structured it.

The functionality of a semantic knowledge base has also been compared with the CSW and (through middleware) semantically enriched Sensor Observation Services. As Figure 6.1 shows, the CSW and a semantic knowledge base are both in the domain of metadata cataloguing. They harvest metadata from sources to provide a broad overview of all available data. Semantic sensor middleware layers can be used to add semantics to an individual data source. The use of semantic sensor middleware and a semantic knowledge base are therefore not mutually exclusive.

Performance optimisation has been beyond the scope of this thesis: The presented conceptual system architecture focused solely on the functionality. However, the proof of concept does provide an indication of the order of magnitude and potential performance bottlenecks. Currently, the retrieval of metadata using SPARQL queries is a matter of seconds, using a Strabon endpoint, metadata from two Sensor Observation Services, and the queries described in Subsection 5.3.2. The retrieval of observation data can go up to a number of minutes, depending largely on the amount of sensors and sources, and the requested temporal range.

7 CONCLUSIONS

This thesis aimed to design a conceptual system architecture 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, before answering the main research question (Section 7.1). The second section discusses the contributions of this thesis (Section 7.2), followed by a brief discussion (Section 7.3). At the end of the chapter, five recommendations are presented, based on the conclusions (Section 7.4).

7.1 ANSWERING THE RESEARCH QUESTION

First the four subquestions will be answered, followed by the main research question: To what extent can the semantic web improve the discovery, integration and aggregation of distributed sensor data?

1. *To what extent can sensor metadata be automatically retrieved from any SOS?*

   Data inside a Sensor Observation Service (SOS) can be automatically retrieved, using the methods described in Subsection 4.1.1. However, there are two things in the Sensor Web Enablement (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 Feature of Interests (FOIs) as a parameter for the `GetObservation` request. In the case of air quality these features represent the sensor locations. To retrieve their geometries either `GetFeatureOfInterest` or `GetObservation` requests have to be made. This is especially cumbersome since the `GetFeatureOfInterest` response is not required to link FOIs to procedures or observed properties. This fundamental relation between a sensor location, procedure and observed property is therefore only visible by either combining `GetCapabilities`, `GetFeatureOfInterest` and `DescribeSensor` responses, or by requesting observations from sensor locations. Therefore, I propose that the capabilities document should not only list the Uniform Resource Identifiers (URIs) of the FOIs, but also describe the geometry and observed properties of each of them.

   Secondly, in the current specification of the XML schemas for SOS and Observations and Measurements (O&M), identifiers for FOIs, observed properties and procedures can be *any URI*. This means, that a Uniform Resource Locator (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 Resource Description Framework (RDF) document. Without these semantics it is hard to use a SOS for both humans and automatic processes, especially when two or more services are used in combination with each other.

Besides the two changes which should be made to the SWE standards, there is another issue with automatically retrieving sensor metadata from a SOS: the order of coordinates for a FOI. For example, both Sensor Observation Services used in this thesis had a different order for the latitude and the longitude 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.

Concluding, sensor metadata can be automatically retrieved from any SOS. In the proof of concept implementation this process has also been successfully created. However, it could be further improved by implementing the above mentioned changes to the SWE standards.

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 which contain general semantics about the metadata. They identify what the different URIs represent (e.g. observed properties, procedures, features of interest). Using a workflow adapted from Missier [2015], this metadata can be automatically converted to linked data. The different elements in the XML schemas have been mapped to the corresponding classes in linked data ontologies. This allows a Web Processing Service (WPS) to create RDF documents, and publish them in a SPARQL endpoint. The om-lite, sam-lite and PROV ontologies have been used to describe the linked 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, better semantic reasoning can be implemented to return metadata about deployed sensors, of which data can be retrieved.

   Another class that should be represented in an ontology is the Sensor Observation Service. The proof of concept (Chapter 5) used a single endpoint for storing all metadata from Sensor Observation Services. Therefore, it was known that the data source 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 supported queries (e.g. `GetCapabilities`, `DescribeSensor`, `GetObservation`), they can retrieve data without requiring prior knowledge on SWE standards. Similarly, sensor data using other platforms such as the SensorThings API could be discovered and retrieved, and used in combination with each other.

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

   Triple stores do not perform as good as non-RDF relational databases when large amounts of data are being queried [Bizer and Schultz, 2009]. 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 functionality that could be implemented using either one of these two parts of the web.

On the one hand, 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 approaches have been considered.

In the proof of concept a more detailed semantic knowledge base is created for a number of reasons. First of all, it was found that not all Sensor Observation Services offer the same filter capabilities. For the first option to be viable every SOS should have a minimum amount of filter capabilities implemented by default. Secondly, if a user is interested in sensors overlapping spatial features, using the bounding box might not be very efficient. The bounding box is a rough generalisation of the FOIs and could include sensors that are not overlapping the intended features. The result of this is that many unnecessary requests will be send to a SOS. This lowers the overall performance, as the client has to wait for more requests to be executed, and the SOS server has to handle requests from more clients.

Thirdly, semantic information about specific sensors can be linked by other related linked data. This can be done by the organisation maintaining the sensor or by other organisations. For example, links to the sensor 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. Information like this could all be added to a semantic knowledge base and does not have to be provided by (the organisation maintaining) a SOS. When there are no detailed descriptions of individual sensors this kind of information could not be included in the semantic knowledge base.

However, the down side of the detailed semantic knowledge base is that more data has to be transferred over the internet. First a SPARQL query has to be made with a spatial filter, which includes verbose Well-Known Text (WKT) geometries. Then an RDF document containing detailed information about each sensor is returned, based on which SOS requests are performed. If only the addresses of Sensor Observation Services are returned, which have sensors inside a certain bounding box, both the SPARQL requests and responses are smaller. The detailed spatial query is then performed by the Sensor Observation Services and does not require as much data to be transferred over the internet, which makes it a more efficient procedure.

Still, discovering sensors is only a matter of seconds in the proof of concept. On the other hand, automatically retrieving observation data from Sensor Observation Services 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 optimisation is beyond the scope of this thesis. It is likely that this can still be improved significantly in the future (see Chapter 8).

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 WPS, which is a standard Application Programming Interface (API) for data processes on the web by the Open Geospatial Consortium (OGC). The WPS is well suited for these two applications.

   The O&M data model 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 (Section 3.4). These are both open standards for spatial query functions in SPARQL.

   The O&M Observation schema is being reused for the spatially aggregated sensor data, which is created from the data of discovered Sensor Observation Services. However, the schema only allows sensor data with a single procedure description. As different procedures were implemented by the different organisations maintaining a SOS, only aggregated observations from the same SOS fit in the O&M observation schema. The proof of concept integrates and aggregates data from different sources. Therefore, its output is not in accordance with the schema. I propose to allow the procedure element to contain a description of multiple procedures from which the data originates, to facilitate the integration of different sensor data sources.

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

In this thesis a conceptual system architecture 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 easier to discover sensors and their corresponding sources. Different Sensor Observation Services can be integrated, because their content is being semantically harmonised in the process of creating linked data, with the use of a WPS. Having linked data of SOS metadata also allows web processes to perform tasks automatically, such as data aggregation.

Only open standards have been used in the conceptual system architecture. They have been preferred over commercial ones, as they are based on consensus in the geosciences community and they are free to be (re)used by anyone. Any SOS can be harmonised and added to the semantic knowledge base, if there are a minimum of semantics provided. The data sources used in the proof of concept did not meet this criteria, because the SOS specifications are to some extent open for interpretation. A manual step had to be added to the workflow to cope with this. Therefore, the SWE standards should be extended to allow the presented method to be completely automated.

Primarily experts in the field of geo-information have knowledge on SWE standards and data services. The semantic knowledge base enables online processes to translate a logical queries for sensor data into technical queries. This makes sensor data accessible to a larger audience, who might not be familiar with URIs, encodings, data models, service models or specific data formats. With sensor metadata as linked data, a user only needs to enter parameters, such as: the type of sensor data, a spatial feature and a temporal range. For example: *What are the average particulate matter levels per month in neighbourhoods of Delft over the last five years?* A WPS can automatically translate this into `GetObservation` requests using the semantic knowledge base. The presented proof of concept has implemented this and automatically sends the requests to the discovered Sensor Observation Services, integrates their responses and performs further processing such as data aggregation. The user receives a single data set from the WPS, according to the logical data query.

7.2 CONTRIBUTIONS

The contribution of this thesis is a conceptual system architecture, including:

1. A process for harvesting and harmonising SOS metadata, which are stored as linked data in a semantic knowledge base.
2. A process for discovering, retrieving and processing observation data by translating a logical query to SOS requests, using a semantic knowledge base with linked sensor metadata.

The semantic knowledge base improves the following sensor web functionalities:

Discovery: Metadata is linked to and by related data, making it easier to find online.
Integration: Through a process of metadata harmonisation, data of different Sensor Observation Services can be used in combination with each other.
Aggregation: A semantic knowledge base with linked sensor metadata allows web processes to automatically perform tasks, such as spatial and temporal aggregation.

7.3 DISCUSSION

SOS metadata is stored twice in two different locations using the methods of this thesis. On the one hand, having duplicated data may not be desirable, because the duplicates can become out of sync with the corresponding original data at the source. Also, more storage space is required for the same amount of data. On the other hand, extra functionality is achieved when duplicating metadata to a semantic knowledge base (e.g. discovery, integration and aggregation). The Catalog Service for the Web (CSW) as well 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. It could be integrated into common SOS software packages, to (only) update when metadata changes at the source.

The conceptual system architecture of Chapter 4 aims to automate as many steps in the chain of discovering, retrieving and processing sensor data as possible. This includes a harvesting process which should be executed to automatically create linked data from SOS metadata. It is more convenient for users, because it requires less knowledge, it appeals to a broader audience and it makes less mistakes in repetitive tasks. However, with an automated process, 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.

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 performance, as they are less expensive than spatial queries. However, there 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 presented conceptual system architecture automatically retrieves metadata from a SOS. However, it could also have been designed on top of a CSW: the metadata from a SOS could be harvested by the CSW and this CSW could be connected to a semantic knowledge base. The advantage of this method is that the CSW standard with SOR and SIR extensions could be reused. 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 more data duplication: Data would be stored at the source, duplicated by the CSW and afterwards again on the semantic web. Secondly, it places a greater burden on the organisation maintaining the SOS. The organisation would have to create a CSW next to their SOS, or make sure it is connected to a third party's CSW.

7.4 RECOMMENDATIONS

Based on the conclusions of this thesis five recommendations have been formulated:

- *The SOS standard should require a more descriptive capabilities document.*

  Currently a number of requests have to be made to find out which deployed sensors can be accessed via a SOS. For making sense of SOS metadata and to identify deployed sensors, it is crucial that the relations between sensor locations, procedures and observed properties are described. Therefore, a capabilities document should not only list FOIs, procedures and observed properties, but it should also state their interrelationship.

- *The SWE standards specifications should contain stricter rules regarding the structure of URIs.*

Observed properties and procedures need to be semantically defined by the organisation maintaining a SOS, for the presented processes to be completed automated. Semantic descriptions are not mandatory using the current SWE standards specifications. Therefore, a more strict SWE specification should be created. It should require the use of semantic URIs, which resolve to RDF documents.

- *The O&M Observation schema should be able to include aggregated sensor data with the same observed property, created by different sensing procedures.*

  The output data of the proof of concept implementation is not in accordance with the O&M XML schema, because it outputs spatially aggregated observation data. This data can be retrieved from different sources, each using different procedures. This does not fit in the current O&M Observation schema. Therefore, the value of a procedure element should be allowed to include multiple procedures with the same observed property.

- *Linked data ontologies should be extended with classes for geo web services, supported requests and SOS content.*

  The om-lite and sam-lite ontologies cover a large part of the sensor metadata. However, these linked data ontologies should be extended to also define the SOS service, it's supported requests and additional metadata, such as observation offerings. These could not yet be semantically described in the proof of concept.

- *The performance of endpoints should be improved, to better cope with vector geometries in SPARQL queries*

  Since GeoSPARQL and stSPARQL require the verbose WKT or GML encodings for spatial filters, the queries can be rejected for exceeding the maximum amount of characters. The performance of endpoints should be improved, to allow more complex vector geometries to be used in SPARQL queries.

8 FUTURE WORK

There are still a number of areas in which the methodology presented in this thesis could be improved, and there are open questions which should still be answered. This chapter will briefly describe three areas of future work: including more web services in the current method (Section 8.1), improving the performance of querying observation data (Section 8.2) and extending the methods for spatial and temporal aggregation (Section 8.3).

8.1 INCLUDE MORE WEB SERVICES

The current proof of concept has been created using the SOS. However, there are a number of other sensor web services based on SWE standards that could be included as well. First of all, there are 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, such as the tinySOS and the sensorThings API. These are services for retrieving observation data from an Internet of Things (IoT) perspective. These observation data services are created with a focus on low energy consumption by the *thing* that senses, and more compact queries and responses than the verbose XML documents used by SOS. 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 these 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, its content could be made available in a machine understandable manner to allow them to be discovered and used in automated web processes.

8.2 IMPROVE PERFORMANCE

The main focus of the current conceptual system architecture was on the definition of 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, a request is send for every sensor. 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 might be used for this improvement.

8.3 EXTEND 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 weights 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 aggregation methods semantically. Stasch et al. [2014] proposes the use of semantic definitions of spatial aggregation. This could make the purpose of each of these methods machine understandable, as well as the effect that it has on the output 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. Data by Kadaster [2015] and gadm.org [2015].

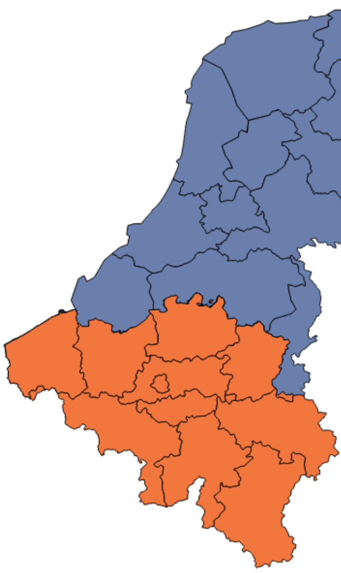

Figure A.2: Dataset of provinces in the Netherlands and Belgium in 2015 Data by Kadaster [2015] and gadm.org [2015].

Figure A.3: Dataset of the Netherlands and Belgium in 2015. Data by gadm.org [2015].

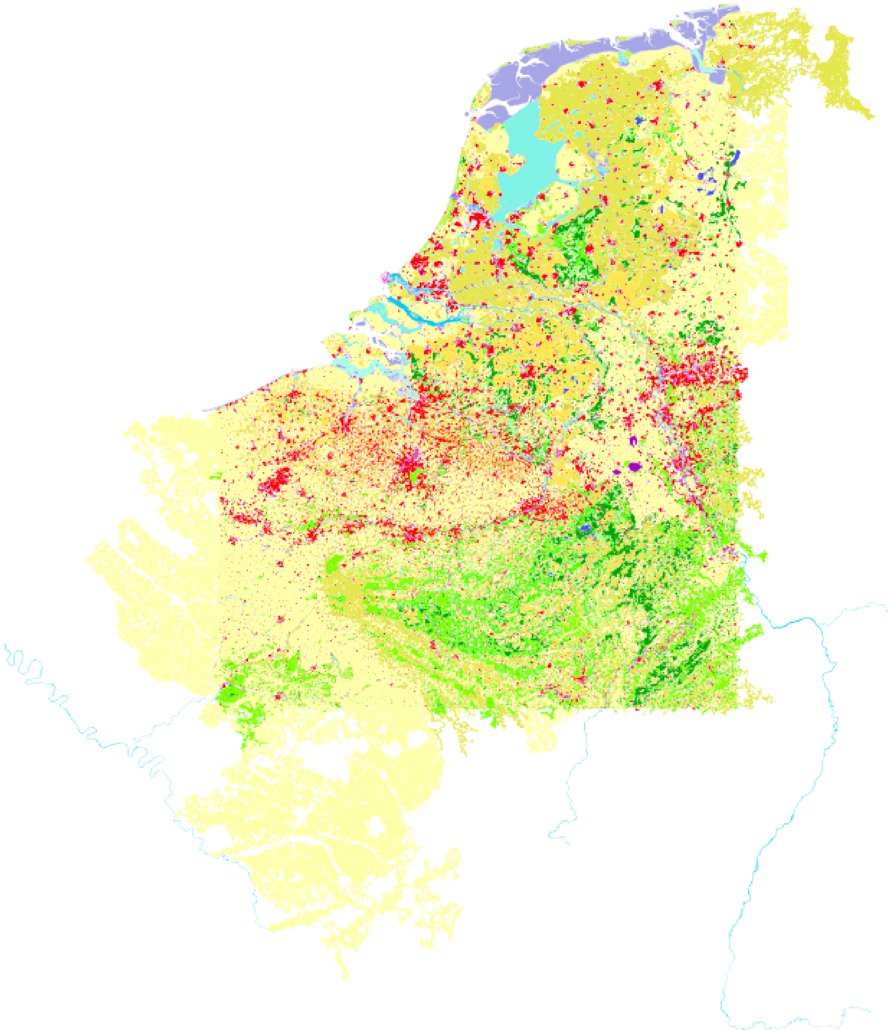

Figure A.4: Dataset of landcover in the Netherlands and Belgium in 2012. Data by Copernicus Land Monitoring Services [2012].

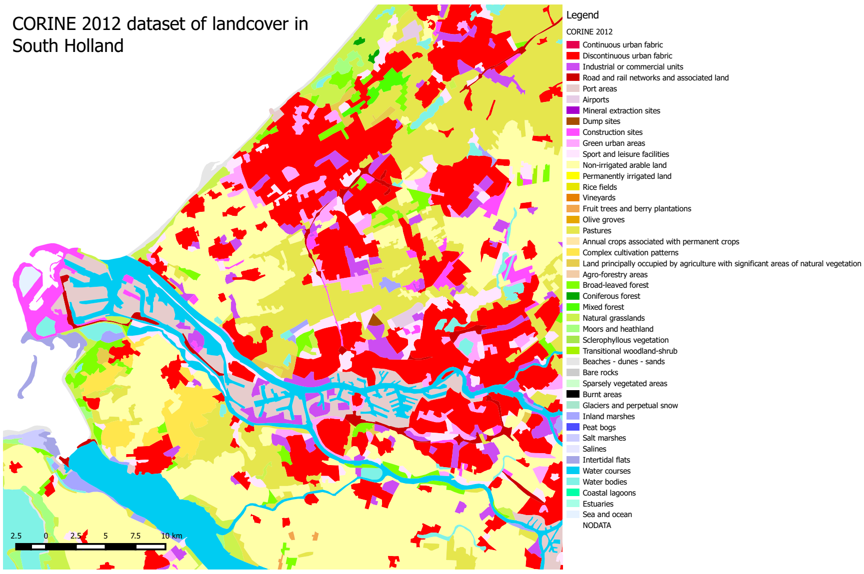

Figure A.5: Landcover of the province of South Holland. Zoom of the dataset from Figure A.4.

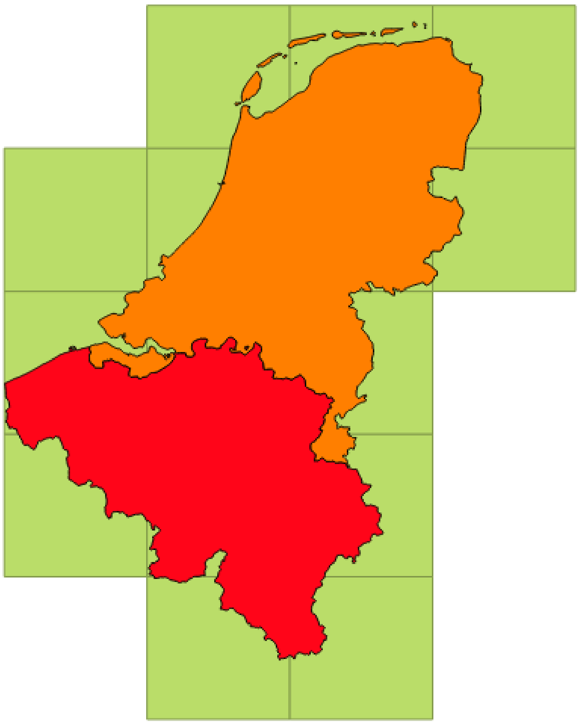

Figure A.6: EEA reference grid cells with a resolution of 100km² overlapping the Netherlands and Belgium. Data by EEA [2007].

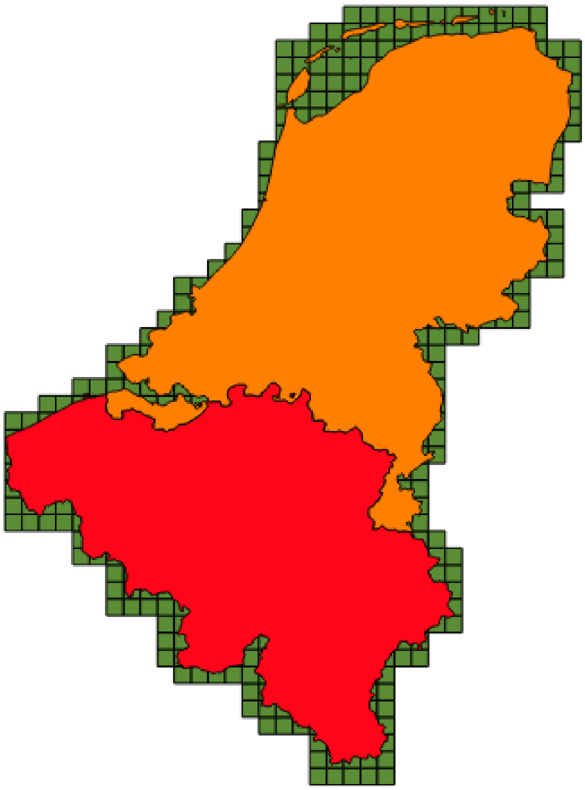

Figure A.7: EEA reference grid cells with a resolution of 10km² overlapping the Netherlands and Belgium. Data by EEA [2007].

Figure A.8: Webmap by the Dutch national institute for public health and the environment (RIVM) showing their air quality sensor network. Image courtesy by RIVM [2015].

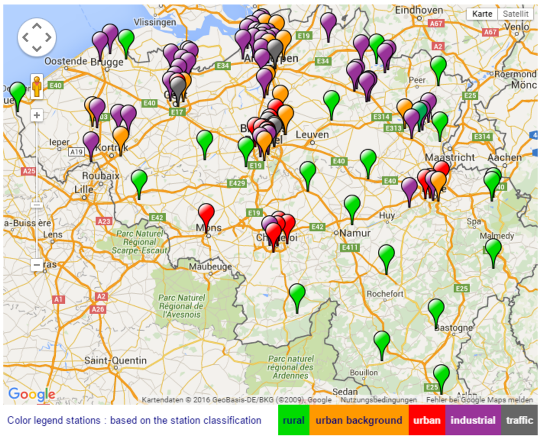

Figure A.9: Webmap by the Belgian interregional environment agency (IRCEL-CELINE) showing their air quality sensor network. Image courtesy by Ircel–Celine [2015].

Figure A.10: Google Streetview image of RIVM sensor location in Amsterdam in 2015. Image courtesy by Google (https://www.google.com/maps).

B MAPPINGS

Table B.1: Mapping of SOS instances using PROV-O

Instance	Mapped to
SOS	http://www.w3.org/ns/prov#Agent
Sensor	http://www.w3.org/ns/prov#Agent
Procedure	http://www.w3.org/ns/prov#Activity
FOI	http://www.w3.org/ns/prov#Entity

Table B.2: GetFeatureOfInterest mapping using om-lite and sam-lite

GetFeatureOfInterest content	Represents	Mapped to
sams:SF_SpatialSamplingFeature	Sampling feature	http://def.seegrid.csiro.au/ontology/om/sam-lite#SamplingFeature
sams:shape	Sampling point	http://def.seegrid.csiro.au/ontology/om/sam-lite#SamplingPoint
ns:pos	Point geometry	http://strdf.di.uoa.gr/ontology#hasGeometry
ns:pos[srsName]	CRS	

Table B.3: Mappings of capabilities document using om-lite and sam-lite

Capabilities content	Represents	Mapped to
ows:Title	Name of SOS	http://xmlns.com/foaf/0.1/name
ows:ProviderName	Name of organisation maintaining SOS	http://xmlns.com/foaf/0.1/name
ows:Parameter[@name='featureOfInterest']/ows:AllowedValues	Identifiers of all FOIs	http://def.seegrid.csiro.au/ontology/om/sam-lite#SamplingPoint
ows:Parameter[@name='procedure']/ows:AllowedValues	Identifiers of all procedures	http://def.seegrid.csiro.au/ontology/om/om-lite#Process
sos:ObservationOffering	Offerings	http://def.seegrid.csiro.au/ontology/om/sam-lite#SamplingCollection
swes:observableProperty	Observable property of offering	http://def.seegrid.csiro.au/ontology/om/om-lite#observedProperty
swes:procedure	Procedure of offering	http://def.seegrid.csiro.au/ontology/om/om-lite#Process
ows:Parameter[@name='responseFormat']/ows:AllowedValues	Identifiers of response formats	http://purl.org/dc/terms/hasFormat
ows:Parameter[@name="procedureDescriptionFormat"]	Identifiers of procedure description formats	http://purl.org/dc/terms/hasFormat

C WEB PROCESSING SERVICE RESPONSE DOCUMENTS

C.1 EXAMPLE CAPABILITIES DOCUMENT

```
<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>
```

```
        <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>
```

C.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>
```

C.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. (2000). Semantic web – xml2000. [online] https://www.w3.org/2000/Talks/1206-xml2k-tbl/Overview.html [accessed on May 7th, 2016].

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.

Bizer, C. and Schultz, A. (2009). The berlin sparql benchmark.

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.

Copernicus Land Monitoring Services (2012). Corine Land Cover; CLC 2012. [online] http://land.copernicus.eu/pan-european/corine-land-cover/clc-2012 [accessed on December 15th, 2016].

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].

EEA (2007). EEA Reference Grids. [online] http://www.eea.europa.eu/data-and-maps/data/eea-reference-grids [accessed on January 21st, 2016].

gadm.org (2015). Gadm database of global administrative areas. [online] http://gadm.org/ [accessed on December 15th, 2015].

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].

Geotools.org (2015). Axis order. [online] http://docs.geotools.org/latest/userguide/library/referencing/order.html [accessed on May 10th, 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 *International Symposium on Collaborative Technologies and Systems, 2009. CTS'09.*, 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].

Ircel–Celine (2015). Monitoring Stations: Locations of Stations. [online] http://www.irceline.be/en/air-quality/measurements/monitoring-stations/ [accessed on December 16th, 2015].

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

ISO (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.

Kadaster (2015). Bestuurlijke grenzen actueel. [online] https://www.pdok.nl/nl/producten/pdok-downloads/basis-registratie-kadaster/bestuurlijke-grenzen-actueel [accessed on December 16th, 2015].

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.

OGC (2007). OGC Catalogue Services Specification.

OGC (2011). Observations and measurements – xml implementation.

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

OGC (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].

RIVM (2015). Landelijk Meetnet Luchtkwaliteit: Meetnetoverzicht. [online] http://www.lml.rivm.nl/meetnet/ [accessed on December 16th, 2015].

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. *IEEE/ACM Transactions on Networking*, 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.