Semantic Web 0 (2017) 1–10
IOS Press

Facilitating Filtering of Web Feature Services with their Automated Description

Jeremy Siao Him Fa, David A. McMeekin , Geoff West and Simon Moncrieff
Cooperative Research Centre for Spatial Information, Deparment of Spatial Sciences
Curtin University, Bentley, Western Australia

[b] *jeremy.siao@live.com*
[c] *d.mcmeekin@curtin.edu.au*
[d] *gawwwest@gmail.com*
[e] *s.moncrieff@live.com*

Abstract. An ontology is introduced to facilitate the filtering of Open Geospatial Consortium (OGC) web feature services (WFS) by automatically and semantically describing them. This paper furthers existing work by describing OGC Web Feature Services at the layer level and enabling a more complete representation of the web services' processes and capabilities by reusing existing ontologies. To evaluate the premises of the ontology, key questions are given and answered using SPARQL queries. Components used in the ontology are explained, and newly created ontology components are detailed out. The result obtained is an ontology linking the OWl-S, GeoSparql, vCard, and HTTP ontologies together to semantically describe WFS, and hence enabling a uniform querying platform for their capabilities and services.

Keywords: ontology, web services, ogc, automation, filtering

1. Introduction

Open Geospatial Consortium Web Services (OWS) are web services catering for geospatial data. They have specific spatial filters and operations that are unique to spatial data such as finding the intersections and union of two geometries. Though there are many OWS such as Web Feature Services (WFS) [25], Web Mapping Services (WMS) [8], Web Processing Services (WPS) [9], and Web Coverage Service (WCS) [3], finding a specific web service is a struggle as filtering gathered web services still needs to be done manually [6]. Systems exist where multiple OWS can be queried and compared, but registering the web services is still a manual process [15]. Other methods to discover OWS include searching through geoportal servers, catalogs such as Data.gov, ArcGIS, GEOSS, or from spatial queries made in search engines [10]. While there are resources available to gather and find OWS, the main issue is having to filter them to find a specific feature for the user's needs.

This manual filtering process is time consuming and strenuous due to the layered approach that OWS is based on as it requires the understanding of the XML description of the service. As an example, a WFS uses three layers to describe the web service. Firstly, the GetCapabilities of the service is called to find the capabilities of the service, and which features and filters it serves. Second is the DescribeFeatureType layer where the structure of a specific feature is provided in XML. Then the last layer is the GetFeature layer, it provides the actual data of a particular feature. The GetFeature layer can be queried based on the filters described in the GetCapabilities layer, and based on the feature attributes obtained from the DescribeFeatureType layer making these three layers co-dependent for a full understanding of the service. This method of serving data is methodical and appropriate for large volumes of data that is characteristic of spatial data but is time consum-

ing when done manually. As such, ways to automate the finding and querying of web services for particular feature characteristics would lead to less time filtering them and more time in using them.

Thus, the ontology proposed in this paper[1], called Web Service Ontology (WSO), aims at facilitating the automated filtering and description of OWS to improve their consumption for both human users and machines. This is achieved by extending W3C existing ontologies to cater for the specifics of OWS. The scope for this paper is WFS but can be extended to other OWS. The ontology extended is the Web Ontology Language for Services (OWL-S). By reusing OWL-S, the terms used in WSO can be queried as per OWL-S expectations. Additionally, WSO reuses the vCard [12], GeoSparql [21], and HTTP [13] ontologies where adequate to describe OWS.

This paper is organised as follows. The background of the technologies and terminologies used are described in the next section, alongside related work done in this area. The methodologies employed is explained after. Following up is a description of the proposed ontology outlining each ontology used and how they link together. Afterwards, a summary of the ontology as a whole is provided, followed by the application and evaluation of the ontology. This paper then concludes with a discussion of the ontology presented.

2. Background

Efforts to facilitate the semantic ability of web services have been made by the World Wide Web Consortium (W3C) who provides ontologies to describe web services - the Web Ontology Language for Services (OWL-S) [18], and the Semantic Web Services Ontology (SWSO) [2]. These ontologies are not specific to OWS, and therefore some adaptations specific to OWS must be made. Both of the ontologies recommended by W3C have vocabularies to describe the profile of Web Services (WS), alongside their model and grounding. The changes required lie in the grounding of the ontologies, where they rely on a Web Service Definition Language (WSDL) [7] representation of a web service. However, as OWS do not require to be WSDL compliant, it follows that they are not, and thus cannot be directly used with either OWL-S or SWSO.

Other efforts attempted at describing OWS semantically. Chen et al. [6] proposed a methodology to find OGC web services and describing them using ontologies but their methods, though based on OWL-S, do not consider the grounding component and only the service profile and model are used. Their implementation only considers the GetCapabilities call of a web service, while in this paper, the description of an OWS is described at the layer level, and a grounding for OWS is used.

Stock et al. [23] on the other hand, created their own OGC grounding to cater for OGC web services. They state that the grounding used is a simplified one reflecting the OWL-S grounding, and that a more standards-compliant solution using a RDF mapping of WSDL by Kopeck [14] was not achieved due to time constraint. The ontology described in this paper aims at describing a more complete grounding of OWS alongside other ontologies to cater for spatial filtering as well.

3. Methodology

There are various methodologies that have been suggested and used for ontology design [4,5,16,17,20,26, 27,24]. Nonetheless, not one methodology has been agreed upon to be used as the de facto in ontology design, and no formal methods have been widely adopted by the community. As such, the methodology used in this paper is a hybrid of common steps from various others.

The first step is defining the scope and the purpose of the ontology [20,5,4,17,26]. The purpose of the designed ontology is to enable automated discovery of OWS, as well as allowing machines to automatically parse and query the WS. The scope of this ontology is purely WFS version 1.0.0 as it is more widely used. An example of a WFS with only version 1.0.0 serviced is from the statutory authority in charge of property and land information in Western Australia (Landgate) [2].

The second step is to identify some queries that the ontology is meant to answer. This step helps in finding the motivation behind the ontology creation, and can also be used as an initial evaluation of the ontology [11]. Additionally, this step can be used in conjunction with step one to identify the scope [20]. Such queries will be expressed in SPARQL in the evaluation section.

These questions will be transformed into SPARQL and then run against a populated ontology to find out if

[1] available at `www.purl.org/net/wso#`

[2] `https://www2.landgate.wa.gov.au/ows/wfspublic_4283/wfs?SERVICE=WFS&REQUEST=getcapabilities`

it answers those questions, and if so the ontology services its core purpose. The third step is finding ontologies that can be reused [4,20,26]. This step includes analysing existing ontologies within the same domain to (1) find reusable ontologies, and (2) getting knowledge of the domain. Analysing existing ontologies is a way to gain partial knowledge about the domain to be represented [4]. Given that the purpose of the ontology is to enable auto discover-ability, it is important that it uses widely-accepted ontologies as foundation.

The fourth step requires a brainstorm of the terms, and concepts to be used in the ontology. This step is identified as the 'ontology capture' stage [**?**], but is also part of other methodologies proposed [20][4]. The terms should be directly related to the task and purpose of the ontology. At this stage the specific categories of the terms do not need to be determined. This step should be carried out alongside the other ontologies from step three, where terms already defined in another ontology should not be repeated but made aware of. In this step, the terms identified should be expanded by adding synonyms, descriptions, type, and sources so as to be reminded of the purpose of the terms [4].

The next step differs from different methodologies employed based on their granularity. In a methodology [20], the class definition and class hierarchy step are grouped together, while separating the property definition. In another methodology [4] the classes and properties of an ontology are identified at the same time as step four before the hierarchy and class relations. Other methods include grouping class definitions and property definitions alongside cardinality restraints and semantic relationship in one big step [5], but does not provide much granularity as to what they encompass. Both class identification and property identification can also be separated into two distinctive steps [26] but this does not consider the general brainstorming phase (step four).

Thus, for this paper, the methodology used is one where the granularity is as defined as possible. The next steps are separated into identifying the classes and properties first, then classifying the classes into a hierarchy.

After step four, the identification of key terms, the fifth step divides the terms into either classes or properties. Classes are terms that can act as single entities usually nouns, while properties are terms that describe a class, usually verbs [4]. The sixth step is grouping the classes into a hierarchy. It can be represented by visualising the most abstract/general class on top of a tree structure, with each branch subdividing into more specific leaves. For this methodology, there are three ways of deriving a class hierarchy [**?**]:

- Top-down approach is where the developer starts with the most abstract or general class, and then derives the leaves of that class into more specific terms.
- Bottom-Up approach is where the starting classes are the most specific ones, and then more general terms are built as their parent node.
- Middle-Level approach is a combination of (1) and (2). It means that a term that lies somewhere in between the tree structure is the starting node. Then both the leaves (more specific terms), and the parent (more abstract terms) are derived from it.

Though all of these methods can work [**?**], a middle-level approach is more intuitive for the characterisation of non-biological taxonomies [22]. Thus, that approach was used.

The seventh step includes defining the attributes required in a class. Some classes can be changed into attributes, and if a specific term can only be represented with a specific data type (string, integer, float), then it is an attribute [4]. The term 'slot' is also used when talking about attributes [20], and are identified as intrinsic such as the flavor of wine, extrinsic such as the name, a part of an object such as the course of a meal, or relationships between instances of a class. In reference to how ontologies are implemented though, class attributes are triples with either the predicate 'owl:ObjectProperty' when linking classes, or 'owl:DatatypeProperty' when linking classes to data types [26]. As such for the purpose of this paper, an attribute/property is considered to be any entity that is linked with either of these two predicates.

The eighth step is to decide upon the cardinalities, restrictions, and rules placed upon each property and class; the logic of the ontology. Cardinalities refer to the number of associations a class can have with another. For example, a person can only have one gender, while having up to a maximum of four limbs. Cardinalities are a subset of restrictions, but restrictions also include domain (the subject) and range (the object) of properties. They can define a set of values that are restrictive of a certain property, as well as limit the values used in datatypes. Rules on the other hand are used to infer new information. An inherent rule in reasoners is that if class B is a subclass of class A, and class C is a subclass of class B, then class C is also a subclass of class A.

The last step involves implementing or coding the ontology using any semantic language of preference. The implementation language used is OWL. This step also encompasses populating the ontology with instances, as well as evaluating the ontology based on the queries found in step two.

4. Pre-Requisites

In order to design an ontology, some pre-requisites have to be met. The purpose, scope, queries to be addressed, and existing ontologies related to the subject matter have to be identified.

4.1. Purpose

The ontology's purpose is to enable automated filtering of OWS, as well as allowing machines to automatically parse and query the web service.

4.2. Scope

The scope of this ontology is purely WFS version 1.0.0 as it is more widely available. An example of a WFS with only version 1.0.0 serviced is Landgate's WFS[3].

4.3. Queries

The queries that the ontology needs to address are as follows:

- What available WFS are there?
- What feature types do a particular WFS service?
- What are the filters a particular WFS has?
- How to make a request call to a service?
- What are the metadata assigned with specific feature types?
- What feature types lie within a certain bounding box?

These questions must be answerable by the ontology.

[3] `https://www2.landgate.wa.gov.au/ows/wfspublic_4283/wfs?SERVICE=WFS&REQUEST=getcapabilities`

4.4. Ontologies to be reused

For that purpose, different ontologies with different purposes have been scouted:

1. OWL-S [19] by W3C for the general description of the service;
2. vCard [12] for the description of the contact information of WFS;
3. HTTP [13] for the message passing and protocol employed by WFS; and
4. GeoSparql [21] for the geospatial content found in WFS.

4.5. Namespace

Namespaces are important as they are the reference point to any ontology's domain URI. As we are using multiple ontologies, the namespaces used for each ontology have to be clearly identified:

1. geo: refers to the GeoSparql ontology,
2. vcd: refers to the vCard ontology,
3. http: refers to the HTTP ontology,
4. cnt: refers to the Content ontology,
5. service: refers to OWL-S Service ontology,
6. profile: refers to OWL-S Profile ontology,
7. process: refers to OWL-S Process ontology,
8. grounding: refers to OWL-S Grounding ontology, and
9. wso: refers to the proposed ontology (Web Service Ontology).

5. Ontologies Reused

This section explains the rationale behind the chosen ontologies, and their purpose in WSO.

5.1. OWL-S

OWL-S is a W3C recommendation ontology. This ontology was chosen because of its comprehensive documentation, as well as its structural alignment to WFS structure. The purpose of OWL-S is to enable the discoverability, invocation, composition, and monitoring of web services with high degree of automation [19]. The ontology has one major class *service:Service* subdivided into three other classes: *service:ServiceModel*, *service:ServiceProfile*, and *service:ServiceGrounding* as depicted in figure 1.

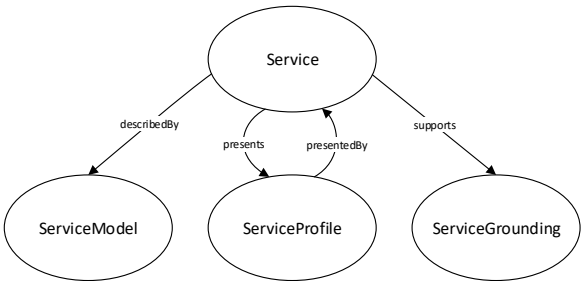

Fig. 1. Owl-S Core

1. *service:ServiceModel* dictates how the service works, by providing a schema of the service's processes, and indicating at an abstract level how each process is performed;
2. *service:ServiceProfile* is tasked with advertising the service, indicating any basic information relating to the service;
3. *service:ServiceGrounding* is a concrete realisation of *service:ServiceModel*. It provides the explicit details regarding a service's processes, such as protocols, message formats, serialization, transport, and addressing. This part of the ontology has to be linked to a respective Web Service Definition Language (WSDL) document.

5.1.1. Profile

The *service:ServiceProfile* has two predicates *service:presentedBy* and *service:presents*. Additional details regarding advertising the service are detailed out in its subclass *profile:Profile*. *profile:Profile* allows for specification of basic details regarding a service functionalities such as *profile:serviceCategory*, *profile:serviceName*, *profile:textDescription*, and *profile:contactInformation*. While the first three are datatype properties and thus can only link to literals, the *profile:contactInformation* is a predicate with unspecified range. That predicate is thus used to link the OWL-S ontology to the vCard ontology (section 5.2) while still respecting the ontology's axioms.

Furthermore, to advertise the filtering abilities of a WFS, a *wso:Filter* class which is a subclass of *process:Input* is created. By making it a subclass of *process:Input*, filters serviced by a particular WFS can be queried. Each filters available for OGC standards have their own class created and made a subclass of *wso:Filter*.

The profile has two predicates: *profile:hasInput* and *profile:hasOutput* that dictate which inputs and outputs a particular process has. In terms of a WFS, the inputs and outputs are feature types and geometries respectively. Hence, the GeoSparql ontology is used (section 5.3) to conceptualise feature layers alongside their geometries. The two ontologies are integrated by creating a *wso:FeatureType* class with a subclass relationship to both *geo:SpatialObject* and *process:Input*, and a *wso:Geometry* class which is a subclass of both *geo:Geometry* and *process:Output*.

Making subclasses of these classes allow the ontology to take advantage of the already defined classes *geo:SpatialObject* and *geo:Geometry*, without modifying their axioms. At the same time, these classes are inferred as *process:Input*, and *process:Output* allowing the OWL-S and the GeoSparql ontologies to be integrated together. The Turtle syntax of the created classes are provided below:

```
wso:Filter a owl:Class;
rdfs:subClassOf process:Input .

wso:Geometry a owl:Class;
rdfs:subClassOf process:Output , geo:Geometry
    .

wso:FeatureType a owl:Class;
rdfs:subClassOf process:Input , geo:
    SpatialObject .
```

The *wso:Geometry* class is changed into a subclass of *process:Output* as the output of a GetFeature call is a geometry. *wso:FeatureType* is included as a subclass of *process:Input* because a WFS call requires the user to specify which feature they wish to query.

5.1.2. ServiceModel

The *service:ServiceModel* class provides an abstract description of the service's processes. In this case, it is assumed that the only process required by a WFS is GetFeature, as information provided by both GetCapabilities, and DescribeFeatureType is stored in the ontology and therefore can be queried from the ontology itself. The subclass *process:AtomicProcess*, describes the GetFeature request of a WFS. *process:AtomicProcess* links to *process:Input* allowing the atomic process to dictate that the input of a GetFeature call can be *wso:FeatureType*, and *wso:Filter* as explained in the previous section. The output of the atomic process also follows this idea due to the subclass relationship. For this component, no major modifications were made except the inputs and outputs classes as described in the previous section.

5.1.3. ServiceGrounding

The *service:ServiceGrounding* class links the OWL-S ontology to a WSDL document, but as OWS do not

require to be WSDL compliant, a different approach is taken. Instead of OWL-S *grounding:WsdlGrounding*, a *wso:OgcHttpGrounding* class is created which is a subclass of *grounding:Grounding* and by association a subclass of *grounding:ServiceGrounding*. The *wso:OgcHttpGrounding* is then linked to a *wso:OgcHttpAtomicProcessGrounding* class mirroring *grounding:WsdlAtomicProcessGrounding*, which then links to the HTTP ontology (section 5.4) through the predicates *wso:ogcHttpOutputMessage*, *wso:ogcHttpInputMessage*, and *wso:ogcHttpConnection*. *wso:OgcHttpGrounding* uses the OWL-S predicate *grounding:hasAtomicProcessGrounding* to link to *wso:OgcHttpAtomicProcessGrounding*. This does not violate OWL-S axioms as both the subject and object of the triple are subclasses of the domain and range of the predicates. The *wso:OgcHttpAtomicProcessGrounding* can also be linked back to the OWL-S ontology via the predicate *grounding:owlsProcess* as specified in the OWL-S ontology. The description of these triples is provided below in Turtle syntax.

```
wso:OgcHttpGrounding
rdf:type owl:Class;
rdfs:subClassOf grounding:Grounding,
[
        rdf:type owl:Restriction;
        owl:onProperty grounding:
            hasAtomicProcessGrounding;
        owl:allValuesFrom wso:
            OgcHttpAtomicProcessGrounding
] .

wso:OgcHttpAtomicProcessGrounding
rdf:type owl:Class;
rdfs:subClassOf grounding:
    AtomicProcessGrounding .

wso:ogcHttpConnection
rdf:type owl:ObjectProperty;
rdfs:domain wso:OgcHttpAtomicProcessGrounding
    ;
rdfs:range http:Connection .

wso:ogcHttpInputMessage
rdf:type owl:ObjectProperty;
rdfs:domain wso:OgcHttpAtomicProcessGrounding
    ;
rdfs:range http:Request .

wso:ogcHttpOutputMessage
rdf:type owl:ObjectProperty;
rdfs:domain wso:OgcHttpAtomicProcessGrounding
    ;
rdfs:range http:Response .
```

5.2. vCard

The vCard [12] ontology is an RDF/OWL representation of the vCard specification which was developed by the Internet Engineering Task Force (IETF). Its purpose is to describe individuals and entities. It contains information such as addresses, emails, and contact information.

Figure 2 demonstrates how the vCard ontology integrates with OWL-S. The *service:ServiceProfile* from OWL-S is the linking point between the two ontologies. OWL-S predicate *profile:contactInformation* has an unspecified range, and from its documentation, it states that it can be restricted to another ontology with vCard and Friend of a Friend (FOAF) ontologies given as examples.

vCard is used over FOAF due to vCard being more focussed on an organization than FOAF. Contact information for organisations are not currently specified in FOAF as it is more catered towards individuals rather than groups. Thus, vCard was used in this instance. Additionally, it is a representation of the vCard specification which is in XML, and can be directly transformed into an ontology and integrated into WSO.

No change was made to the vCard ontology. It is simply linked to *service:ServiceProfile* using the predicate *profile:contactInformation* as described in section 5.1.1.

5.3. GeoSparql

GeoSparql is an OGC standard developed to allow representation and querying for spatial objects on the semantic web [21]. Though GeoSparql is intended to be primarily an extension of SPARQL, few implementations have been made and full scalable implementation has not yet been achieved [1]. Its vocabulary specifications are still useable alongside other ontologies and SPARQL, as it simply specifies certain characteristics of spatial information which is beneficial for our purposes.

The GeoSparql ontology has predicates that allow for representation of relations between *geo:SpatialObject* classes. Three relation family are specified, namely the *Simple Feature*, *Egenhofer*, and *RCC8* relation family. Only the *Simple Feature*

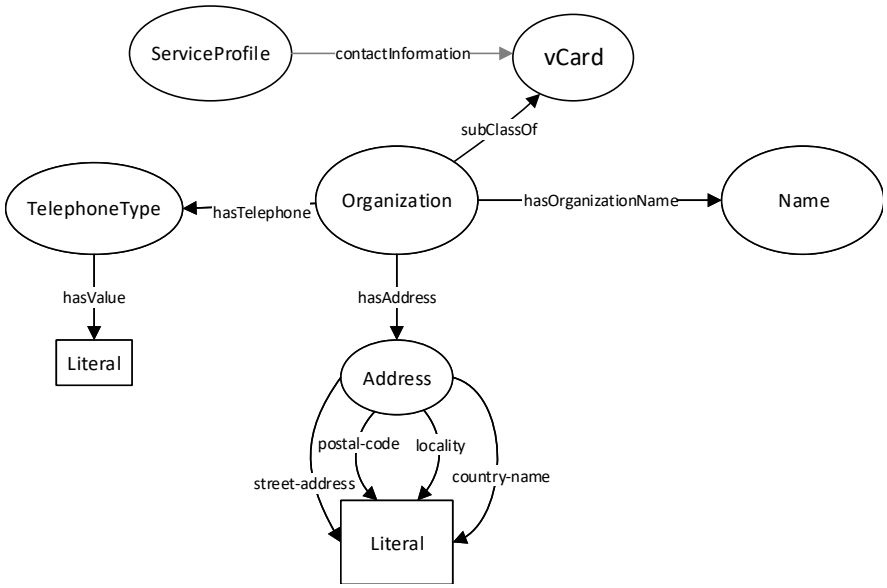

Fig. 2. vCard Ontology

Table 1
Simple Feature Predicates

Relation Name	Relation URI	Domain/Range
equals	geo:sfEquals	geo:SpatialObject
disjoint	geo:sfDisjoint	geo:SpatialObject
intersects	geo:sfIntersects	geo:SpatialObject
touches	geo:sfTouches	geo:SpatialObject
within	geo:sfWithin	geo:SpatialObject
contains	geo:sfContains	geo:SpatialObject
overlaps	geo:sfOverlaps	geo:SpatialObject
crosses	geo:sfCrosses	geo:SpatialObject

relation family predicates are shown in table 1 and only the *sfEquals* predicate is shown in figure 3 due to space constraints.

SPARQL examples provided by the GeoSparql specification [4] includes (1) finding features that contain a specific geometry, (2) features that are within a transient bounding box, (3) finding features that touches the union of two other features, (4) finding the closest features to a geometry, and (5) finding features that overlap.

Following the examples provided, it reasons that the GeoSparql ontology would be most beneficial to represent spatial objects within OWS, and to enable semantic spatial queries.

In conjunction with this ontology, another ontology that can be used alongside is the NeoGeo ontology[5]. It is a vocabulary that focuses on the description of geometries, and thus can be used to extend the Geometry class from GeoSparql. That ontology was not used though as the only additional information required from it is that of a bounding box, and the bounding box class used in NeoGeo is not detailed upon.

Modifications to the GeoSparql ontology first includes the addition of a *wso:BoundingBox* class which is a subclass of *geo:Geometry*. The bounding box literals can be represented in terms of GML or WKT as a *geo:Geometry* class is linked to *geo:asWKT* and *geo:asGML* predicates. Secondly, the classes *wso:FeatureAttribute* and *wso:FeatureProfile* have been added to store the metadata of a feature found in its WFS DescribeFeature and GetCapabilities requests respectively. These classes can be used when a user wishes to find a feature with a particular metadata associated it. To allow for a query requesting both feature profiles and feature attributes, they were made a subclass of *wso:Metadata*. *wso:Metadata* has datatype properties identifying its name and value, and predicate *wso:hasProfile*, and *wso:hasAttribute* are made sub-properties of *wso:hasMetadata*. This allows both predicates to also be considered as being *wso:hasMetadata*.

These classes are described below in Turtle syntax:

[4] `http://www.opengeospatial.org/standards/geosparql`

[5] `http://geovocab.org/geometry`

```
wso:Metadata rdf:type owl:Class;
    rdfs:subClassOf process:Input,
    [
        rdf:type owl:Restriction;
        owl:onProperty wso:metadataName;
        owl:allValuesFrom xsd:string
    ] ,
    [
        rdf:type owl:Restriction;
        owl:onProperty wso:metadataType;
        owl:allValuesFrom xsd:anyURI
    ] ,
    [
        rdf:type owl:Restriction;
        owl:onProperty wso:metadataValue;
        owl:allValuesFrom rdfs:Literal
    ] .

wso:metadataName
    rdf:type owl:DatatypeProperty;
    rdfs:domain wso:Metadata;
    rdfs:range xsd:string .

wso:metadataType
    rdf:type owl:DatatypeProperty;
    rdfs:domain wso:Metadata;
    rdfs:range xsd:anyURI .

wso:metadataValue
    rdf:type owl:DatatypeProperty;
    rdfs:domain wso:Metadata;
    rdfs:range rdfs:Literal .


wso:hasMetadata
    rdf:type owl:ObjectProperty;
    rdfs:domain wso:FeatureType;
    rdfs:range wso:Metadata .

wso:FeatureAttribute
    rdf:type owl:Class;
    rdfs:subClassOf wso:Metadata .


wso:hasAttribute
    rdf:type owl:ObjectProperty;
    rdfs:subPropertyOf wso:hasMetadata;
    rdfs:domain wso:FeatureType;
    rdfs:range wso:FeatureAttribute .

wso:FeatureProfile
    rdf:type owl:Class;
    rdfs:subClassOf wso:Metadata .

wso:hasProfile
    rdf:type owl:ObjectProperty;
    rdfs:subPropertyOf wso:hasMetadata;
    rdfs:domain wso:FeatureType ;
    rdfs:range wso:FeatureProfile .

wso:BoundingBox
    rdf:type owl:Class;
    rdfs:subClassOf wso:FeatureProfile, geo:
        Geometry .

wso:hasBoundingBox
    rdf:type owl:ObjectProperty;
    rdfs:subPropertyOf wso:hasMetadata;
    rdfs:domain wso:FeatureType;
    rdfs:range wso:BoundingBox .
```

5.4. HTTP

The HTTP ontology intends to represent the messages sent and received from a client to a server. It describes the messages and the protocols used in the request and response from each party. This ontology is used in the grounding of OWL-S by extending upon the *wso:OgcHttpAtomicProcessGrounding* class from WSO (see section 5.1.3).

The HTTP ontology [13] allows the description of message parsing at a concrete level which is required by OWL-S grounding. By using the HTTP ontology, the ontology can be adapted to any HTTP request which the web is based on. In this paper it represents an OGC Http grounding. Secondly, its documentation [13] states that it can be used to report test results and evaluation providing a way to control web service quality when used alongside quality assurance tools. It also enables the identification of required conformance by a web service - a precise constraint regarding the server's request. The only limitation identified are privacy issues, but given that its usage in this paper is to access publicly available web services, this limitation is of no concern in this instance.

The modifications made to the HTTP ontology include linkages between *wso:OgcHttpAtomicProcessGrounding* to the *http:Request*, *http:Connection*, and *http:Response* classes of the ontology as explained in section 5.1.3. The *cnt:Content* class is from the Content-in-RDF ontology[6]. Though in the specification of the HTTP ontology, *cnt:ContentAsBase64* is used as the range of the predicate *http:body*, the superclass *cnt:Content* was substituted instead, as it provides more flexibility, and given that geometries will be the results retrieved, it is appropriate to give the freedom to choose which *cnt:Content* class each of the response or request call have: *cnt:ContentAsBase64*, *cnt:ContentAsText* or *cnt:ContentAsXML*. The *cnt:ContentAsBase64*

[6]urlhttps://www.w3.org/TR/Content-in-RDF10/

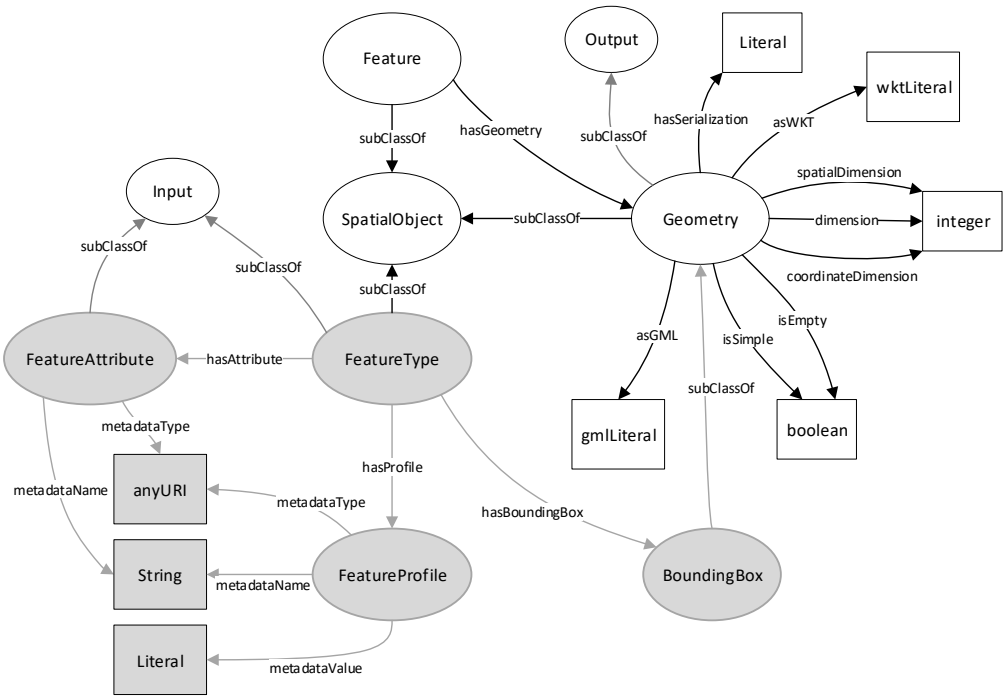

Fig. 3. GeoSparql Ontology

class is more appropriate for abstract utilisation of the HTTP ontology, but given the specifications of this proposed ontology, the request and response have different content types - *cnt:ContentAsText* and *cnt:ContentAsXML* respectively for JSON, GML, and XML.

The Content class can also be linked to *process:Input* and *process:Output* of the OWL-S ontology. Below are the classes that have not already been explained in previous sections.

```
wso:wfsRequestBody
    rdf:type owl:ObjectProperty;
    rdfs:subPropertyOf http:body;
    rdfs:domain wsolt:WfsRequest;
    rdfs:range cnt:ContentAsText .

wso:wfsResponseBody
    rdf:type owl:ObjectProperty;
    rdfs:subPropertyOf http:body;
    rdfs:domain wso:WfsResponse;
    rdfs:range cnt:ContentAsXML .
```

6. Ontology Components Linkages

This section describes a summary of the linkages between the various ontologies used. Figure 5 demonstrates the different ontologies used with each box separating them, the ontology the boxes represent is noted in bold.

The ontologies that are reused are the OWL-S ontology, the GeoSparql ontology, the HTTP ontology and the vCard ontology. The grey ellipses and arrows denote the classes and properties added to the reused ontologies.

Using the OWL-S ontology on the left as a starting point: *service:ServiceGrounding* is used as a superclass of *wso:OgcHttpGrounding*, and is the main link towards the HTTP ontology. *wso:OgcHttpGrounding* is made a subclass of *service:ServiceGrounding* as it follows the same idea as the OWL-S grounding specifications regarding WDSL. The *grounding:WsdlGrounding* class is a subclass of *service:ServiceGrounding*. In that manner, any *service:Service* instance follows the predicate *service:supports* towards a grounding. In the OWL-S ontology, the predicate *grounding:hasAtomicProcessGrounding* has a domain of *grounding:Grounding* and a range of *grounding:AtomicProcessGrounding*. As *wso:OgcHttpGrounding* is a subclass of *grounding:Grounding* and *wso:OgcHttpAtomicProcessGrounding* is a sub-

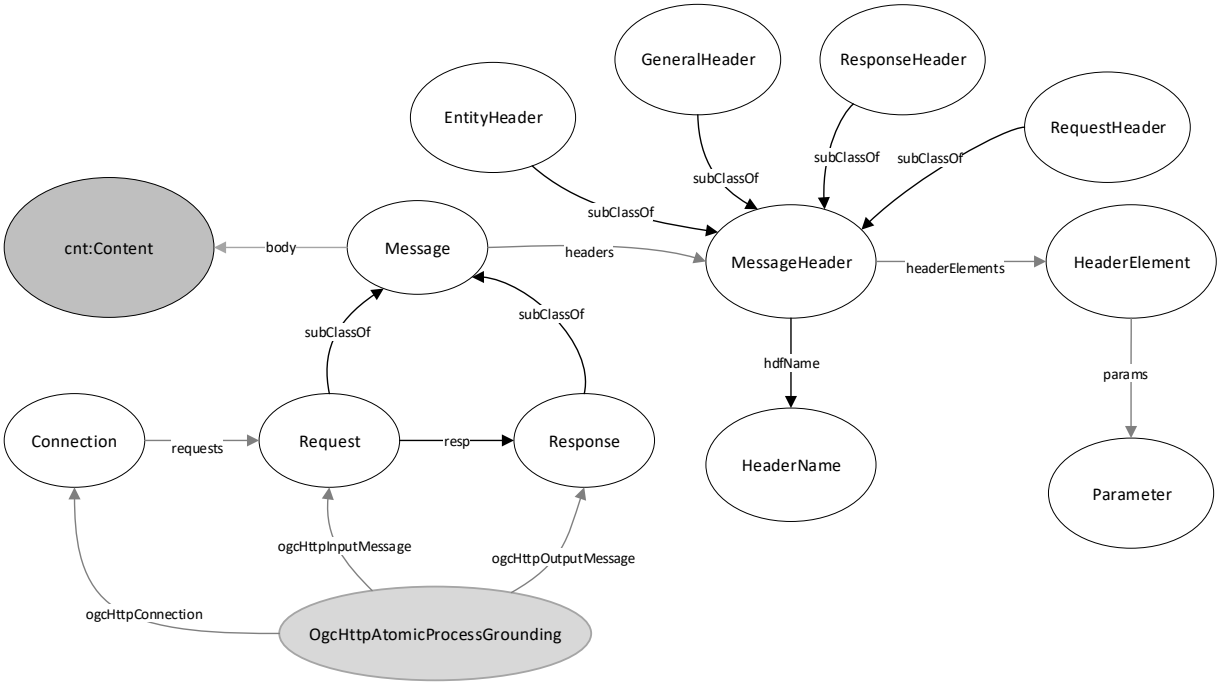

Fig. 4. Modified HTTP Ontology

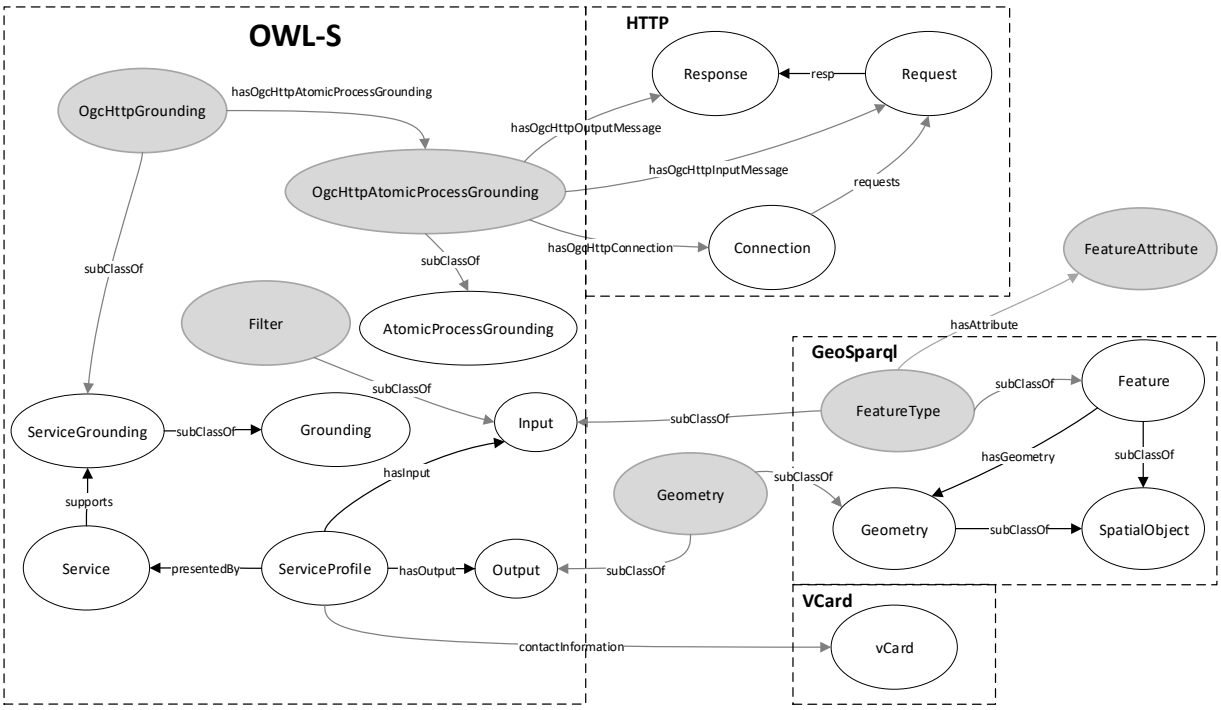

Fig. 5. Ontology Core

class of *grounding:AtomicProcessGrounding*, *service:supports* can therefore be used between these two classes.

The OWL-S grounding links to the HTTP ontology via the predicates *wso:ogcHttpOutputMessage*, *wso:OgcHttpInputMessage*, and *wso:OgcHttpConnection*. These naming conventions follow the predicates used in the OWL-S WSDL grounding. *wso:ogHttpInputMessage* is linked to the *http:Request* class as a request is the input required to make a WFS call. The result of that call is considered to be the output explaining the link to *wso:ogcHttpOutputMessage*, and the *http:Connection* class via the *wso:ogcHttpConnection* predicate.

To link the OWL-S ontology to the vCard ontology, the *profile:contactInformation* is used. It links the *service:ServiceProfile* to a vCard class from the vCard ontology. In the OWL-S specifications, it mentions that the predicate *profile:contactInformation* could be linked to other ontologies such as vCard, so therefore, that ontology was used.

The GeoSparql ontology is linked to OWL-S via a *wso:FeatureType* class which is a subclass of both *geo:SpatialObject* and *process:Input*. That subclass relationship is used to take advantage of the already defined class *geo:SpatialObject* without modifying its axioms. The *wso:FeatureType* can thus be considered both an input of the OWL-S ontology, and a feature of the GeoSparql ontology.

A similar idea is used on the *wso:Geometry* class, which is a subclass of *geo:Geometry* and *process:Output* to benefit from the advantages of both without modifying their axioms.

Having such relationships allow the ontology to determine that an Input from the OWL-S ontology has the same axioms as a FeatureType from the GeoSparql ontology. The same idea applies for the Output class and the Geometry class.

Furthermore, the *wso:FeatureAttribute* class has been added to the *wso:FeatureType* class to enable a feature to link to its attributes which is characteristic of spatial features. Another class that's been added is the *wso:Filter* class which is a subclass of *process:Input*. This subclass relationship's goal is to imply that a filter can be used as an input in a WFS call. This would allow for the discovery of filters a particular WFS services.

The diagram does not show the entirety of the ontologies. Only the core entities are shown to depict the main linkages among the ontologies.

7. Application

The ontology has been applied to three different WFS: (1) Landgate's WFS[7], DELWP's WFS[8], and LINZ's WFS[9]. The ontology was automatically populated from these three WFS by using Extensible Stylesheet Language Transformations (XSLT). This section shows how the ontology can be queried.

7.1. Querying the ontology

In reference to section 4.3, the minimum queries this ontology needs to be able are demonstrated. A sample SPARQL query for each query is provided alongside the results obtained after applying the ontology to the three web services.

Below are the prefixes used in the ontologies and their respective URIs:

```
wso: <http://www.purl.org/net/wso\#>
service: <http://www.daml.org/services/owl-s
    /1.2/Service.owl#>
profile: <http://www.daml.org/services/owl-s
    /1.2/Profile.owl#>
process: <http://www.daml.org/services/owl-s
    /1.2/Process.owl#>
grounding: <http://www.daml.org/services/owl-
    s/1.2/Grounding.owl#>
vcd: <http://www.w3.org/2006/vcard/ns#>
http: <http://www.w3.org/2011/http#>
geo: <http://www.opengis.net/ont/geosparql#>
cnt: <http://www.w3.org/2011/content#>
```

What available WFS are there?
This query allows the user to find the WFS that are linked within the ontology. The WFS can be discovered automatically using a web crawler and then added onto the ontology. By using this query, the user does not need to know the URL of any newly added WFS. A SPARQL query to achieve this is:

```
SELECT DISTINCT ?service_name
WHERE {
        ?wfs_service service:presents ?
            wfs_profile .
```

[7] https://www2.landgate.wa.gov.au/ows/wfspublic_4283/wfs?SERVICE=WFS&REQUEST=getcapabilities

[8] http://services.land.vic.gov.au/catalogue/publicproxy/guest/dv_geoserver/wfs?request=getCapabilities

[9] https://data.linz.govt.nz/services;key=41b3c3b90c0247b587f512e1a4741498/wfs/?request=getCapabilities

```
        ?wfs_profile profile:serviceCategory
            ?service_cat .
        ?service_cat profile:categoryName ?
            cat_name .
        ?wfs_profile profile:serviceName ?
            service_name .
        FILTER(?cat_name = "Web Feature
            Service")
}
```

This query finds all the feature services whose *category* predicate from their ServiceProfile equates to the string 'Web Feature Service'. If it matches that filter, then the service node and its name are returned.

The particular filter can be modified based on specific needs. A simple string matching is used here as the identification of what constitute a WFS is out of scope and can be expanded upon. The results of the query are shown in table 2.

service_name
DELWP Web Feature Service
LINZ Data Service
SLIP Public Web Feature Service - EPSG 4283

Table 2
Get WFS Results

What feature types do a particular WFS service?
This question is aimed at finding out the different features of a specific WFS. It enables the user to browse through the feature types of a WFS. The query is kept general, and more filters can be applied to more specific scenarios. The SPARQL query is:

```
SELECT DISTINCT ?input_value
WHERE {
        ?wfs_service service:presents ?
            wfs_profile .
        ?wfs_profile profile:serviceCategory
            ?service_cat .
        ?service_cat profile:categoryName ?
            cat_name .
        ?wfs_profile profile:serviceName ?
            service_name .
        ?wfs_profile profile:hasInput ?input
            .
        ?input a wso:FeatureType .
        ?input wso:hasProfile ?profile .
        ?profile wso:metadataName ?input_name
             .
        ?profile wso:metadataValue ?
            input_value .
        FILTER(
                ?input_name = "Name"
                && ?cat_name = "Web Feature
                    Service"
                && regex(str(?service_name),
                    "DELWP ")
        )
} LIMIT 5
```

This query matches a service whose category is 'WFS', and the service name containing the string 'DELWP'. It then retrieves all nodes linked by the *hasInput* predicate, and checks if they are from the type *wso:FeatureType*. The results are shown in table 3.

input_value
datavic:FLORAFAUNA1_NV2005_EVCBCS_1_2
datavic:MINERALS_RRREGO100_POLYGON
datavic:FLORAFAUNA1_NV2005_EXTENT
datavic:FLORAFAUNA1_NV2005_EVCBCS_16_2
datavic:FLOOD_STRUCTURE_ROAD_EMBANKMENT

Table 3
Get Feature from DELWP Results

What are the filters a particular WFS service?
Similar to the previous question, the aim of this question is to find out the different filters that a specific WFS offers. A reason for such a query is if a user requires a specific filter to be present in a WFS. This query can be combined with the above query to process both features and filters in one go, while at the same time adding more specific filters to the SPARQL query to provide a more specific WFS.

```
SELECT DISTINCT ?service_name ?filter_label
WHERE {
        ?wfs_profile a profile:Profile .
        ?wfs_profile profile:serviceName ?
            service_name .
        ?wfs_profile profile:hasInput ?filter
             .
        ?filter a wso:Filter .
        ?filter rdfs:label ?filter_label .
        FILTER (regex(str(?service_name), "
            LINZ ")
}
```

This query retrieves a service name includes the string 'LINZ'. It then retrieves all nodes linked by the *hasInput* predicate, and checks if they are of the type *wso:Filter*. Table 4 shows the results of the query.

How to make a request call to a service?
This query is important for more varied usage of the ontology. Even though this article covers only WFS, it can be used for other OWS such as WMS, and WCS. As such, this query would enable a user to find out the

service_name	filter_label
LINZ Data Service	Beyond
LINZ Data Service	Contains
LINZ Data Service	Crosses
LINZ Data Service	DWithin
LINZ Data Service	Disjoint
LINZ Data Service	Equals
LINZ Data Service	Intersect
LINZ Data Service	Overlaps
LINZ Data Service	Touches
LINZ Data Service	Within

Table 4
Get Filters from LINZ Results

exact details of how to make a request call to a web service.

```
SELECT ?service_name ?method_resource ?uri
WHERE {
        ?profile service:presentedBy ?
            wfs_service .
        ?profile profile:serviceName ?
            service_name .
        ?wfs_service service:supports ?
            grounding.
        ?grounding grounding:
            hasAtomicProcessGrounding ?
            atomic_process .
        ?atomic_process wso:
            ogcHttpInputMessage ?request .
        ?request http:requestURI ?uri .
        ?request http:mthd ?method .
        ?method rdf:resource ?method_resource
}
```

The query asks for the grounding of all services present in the ontology. It fetches the URI of the service endpoint as well as the methods available to query them. At this stage, only a URI defining the methods are presented. The results are seen in table 5.

What are the metadata assigned with specific feature types?
This query allows a user to find specific attributes with each feature type from one or more WFS. The filtering aspect can thus be more detailed and specific to the user's needs giving back only feature types with metadata that the user wants.

```
SELECT DISTINCT  ?name_value ?attribute_name
WHERE {
        ?feature_type a wso:FeatureType .
        ?feature_type wso:hasProfile ?profile
             .
        ?profile wso:metadataName ?name.
        ?profile wso:metadataValue ?
            name_value .
        ?feature_type wso:hasAttribute ?
            attribute .
        ?attribute wso:metadataName ?
            attribute_name .
        FILTER (?name = 'Title' && ?
            name_value = "Protected Areas")
}
```

The SPARQL query looks for a *wso:FeatureType* associated with a profile and attributes. It then looks for its attributes, and returns them to the user. Each of these metadata can be further process and filtered based on the user's needs. In the example, the feature type queried has to have the name 'Protected Areas'. The results of the query is shown in table 6.

name_value	attribute_name
Protected Areas	ctrl_ms_vst
Protected Areas	start_date
Protected Areas	napalist_id
Protected Areas	overlays
Protected Areas	name
Protected Areas	reserve_purpose
Protected Areas	type
Protected Areas	recorded_area
Protected Areas	legislation
Protected Areas	section

Table 6
Get Attribute Results

What feature types lie within a certain bounding box?
This query demonstrates a spatial operation applied on the ontology. Only feature types that match the spatial filter will be returned to the user. The query will thus show all feature types from multiple WFS that matches that particular spatial filter - in this instance a 'within' filter.

```
SELECT DISTINCT ?name_value
WHERE {
        ?feature_type a wso:FeatureType .
        ?feature_type wso:hasProfile ?profile
             .
        ?profile wso:metadataName ?name.
        ?profile wso:metadataValue ?
            name_value .
        ?feature_type wso:hasBoundingBox ?
            bbox .
        FILTER (
                ?name = 'Title'
                && regex(str(?name_value), "
                    water")
```

service_name	method_resource	uri
LINZ Data Service	http://www.w3.org/2011/http-methods#GET	https://data.linz.govt.nz/services/wfs?request=GetFeature
DELWP Web Feature Service	http://www.w3.org/2011/http-methods#GET	http://services.land.vic.gov.au/catalogue/publicproxy/guest/dv_geoserver/datavic/wfs?request=GetFeature
SLIP Public Web Feature Service - EPSG 4283	http://www.w3.org/2011/http-methods#GET	http://www2.landgate.wa.gov.au/ows/wfspublic_4283/wfs?request=GetFeature
LINZ Data Service	http://www.w3.org/2011/http-methods#POST	https://data.linz.govt.nz/services/wfs
DELWP Web Feature Service	http://www.w3.org/2011/http-methods#POST	http://services.land.vic.gov.au/catalogue/publicproxy/guest/dv_geoserver/datavic/wfs
SLIP Public Web Feature Service - EPSG 4283	http://www.w3.org/2011/http-methods#POST	http://www2.landgate.wa.gov.au/ows/wfspublic_4283/wfs?

Table 5
Get Request Call Results

```
                && geof:within(?bbox,  "
                    POLYGON((140.0 -74.0,
                    140.0  -33.0, 144.0
                    -33.0, 144.0 -74.0, 140.0
                     -74.0))"^^geo:wktLiteral
                    )
)
}
```

The query looks for subclasses of *wso:FeatureType* and finds their bounding box. It then uses the GeoSparql filter to find the bounding boxes that falls within the specified polygon. An additional filter is added to only return feature types with the string 'water' in them. The results are shown in table 7.

name_value
Potential Groundwater Dependent Ecosystem (GDE) Mapping for the Glenelg Hopkins CMA
Potential Groundwater Dependent Ecosystem (GDE) Mapping for the Wimmera CMA
Groundwater Management Area Subzones
Potential Groundwater Dependent Ecosystem (GDE) Mapping for the Mallee CMA

Table 7
Get Feature Type in Bounding Box Results

8. Conclusion

This paper described the Web Services Ontology (WSO) ontology whose aim is to facilitate the automatic description of OGC compliant web feature services while enabling filtering of those web services down to the layer level. It reuses four ontologies namely: (1) OWL-S, (2) GeoSparql, (3) vCard, and (4) HTTP. The methodology used has been detailed alongside an explanation of the ontologies used and how they are utilised with WSO. This paper concludes with the application of the ontology and some sample queries that have been identified as necessary in order for the ontology to fulfil its purpose.

Acknowledgements

This work has been supported by the Cooperative Research Centre for Spatial Information, whose activities are funded by the Business Cooperative Research Centres Programme, and by the contribution of the Australian Government Research Training Program Scholarship.

References

[1] Spiros Athanasiou, Lily Bezati, Giorgos Giannopoulos, Kostas Patroumpas, and Dimitris Skoutas. GeoKnow âĂŞ Making the Web an Exploratory for Geospatial Knowledge. Technical report, 2012.

[2] Steve Battle, Abraham Bernstein, Harold Boley, Benjamin Grosof, Michael Gruninger, Richard Hull, Michael Kifer, David Martin, Sheila Mcllraith, Deborah McGuin-

ness, Jianwen Su, and Said Tabet. Semantic Web Services ontology(SWSO). URL: http://www.w3.org/Submission/2005/SUBM-SWSF-SWSO-20050909/ [accessed: 2017-03-10].

[3] Peter Baumann. OGC® WCS 2.0 Interface Standard- Core: Corrigendum. *OGC® Standards*, pages 1–57, 2012. URL http://www.opengeospatial.org/standards/wcs%5Cnpapers2://publication/uuid/D303A640-AF41-432D-B72C-593E1BDCFF47.

[4] Julita Bermejo. A simplified guide to create an ontology. *Madrid University*, DRAFT:1–12, 2007. URL http://tierra.aslab.upm.es/documents/controlled/ASLAB-R-2007-004.pdf.

[5] G Brusa, Ml Caliusco, and Omar Chiotti. A process for building a domain ontology: an experience in developing a government budgetary ontology. *Proceedings of the second Australasian workshop on Advances in ontologies*, 72(c):7–15, 2006. ISSN 1-920-68253-8. . URL http://dl.acm.org/citation.cfm?id=1273661.

[6] Nengcheng Chen, Zeqiang Chen, Chuli Hu, and Liping Di. A capability matching and ontology reasoning method for high precision OGC web service discovery. *International Journal of Digital Earth*, 4(6):449–470, 2011. ISSN 1753-8947. .

[7] Roberto Chinnici, Moreau Jean-Jacques, Arthur Ryman, and Sanjiva Weerawarana. Web services description language (wsdl) version 2.0 part 1: Core language. URL: http://www.w3.org/TR/2001/NOTE-wsdl-20010315 [accessed: 2017-03-10].

[8] Jeff de la Beaujardiere. OpenGIS® Web Map Server Implementation Specification. *Organization Environment*, 2006. . URL http://www.opengeospatial.org/standards/wms.

[9] Theodor Förster. OpenGIS Web Processing Service. 2007.

[10] Percivall George. Finding OGC WMS, WFS, WCS services, 2014. URL: http://www.opengeospatial.org/blog/2034 [accessed: 2017-08-03].

[11] M Grüninger, Mark S Fox, and Michael Gruninger. Methodology for the Design and Evaluation of Ontologies. *International Joint Conference on Artificial Inteligence (IJCAI95), Workshop on Basic Ontological Issues in Knowledge Sharing*, pages 1–10, 1995. ISSN 0269-8889. . URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.44.8723.

[12] Renato Iannella and James McKinney. vcard ontology - for describing people and organizations. URL: http://www.w3.org/TR/2014/NOTE-vcard-rdf-20140522/ [accessed: 2017-03-10].

[13] Johannes Koch, Carlos A Velasco, and Philip Ackermann. Http vocabulary in rdf 1.0. URL: https://www.w3.org/TR/2017/NOTE-HTTP-in-RDF10-20170202/ [accessed: 2017-03-10].

[14] Jacek Kopeck. WSDL RDF Mapping : Developing Ontologies from. In *Advances in Conceptual Modeling - Theory and Practice*, pages 312–322. 2006.

[15] Wenwen Li, Chaowei Yang, and Chongjun Yang. An active crawler for discovering geospatial Web services and their distribution pattern âĂŞ A case study of OGC Web Map Service. *International Journal of Geographical Information Science*, 24 (8):1127–1147, 2010. ISSN 1365-8816. .

[16] Fernández López. Overview Of Methodologies For Building Ontologies. *Proceedings of the IJCAI99 Workshop on Ontologies and ProblemSolving Methods Lessons Learned and Future Trends CEUR Publications*, 1999(2):1–13, 1999. ISSN 02698889. . URL http://iwayan.info/Research/Ontology/Tutor_Workshop/Tutorial_4_Analysis.pdf.

[17] Sanjay Kumar Malik, Nupur Prakash, and S a M Rizvi. Ontology Design and Development : Some aspects : An overview. *Semantic Web journal*, pages 1–6, 2010. URL http://www.semantic-web-journal.net/sites/default/files/58_54.pdf.

[18] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila Mcllraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. OWL-S: Semantic Markup for Web Services, . URL: http://www.w3.org/Submission/OWL-S/ [accessed: 2016-04-06].

[19] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila Mcllraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. Owl-s: Semantic markup for web services, . URL: http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/ [accessed: 2017-03-10].

[20] NF Noy and DL McGuinness. Ontology development 101: A guide to creating your first ontology. pages 1–25, 2001. URL http://liris.cnrs.fr/$\sim$amille/enseignements/Ecole_Centrale/Whatisanontologyandwhyweneedit.htm.

[21] Matthew Perry and John Herring. OGC GeoSPARQL-A geographic query language for RDF data. *OGC Candidate Implementation Standard*, page 57, 2012. URL http://www.opengis.net/doc/IS/geosparql/1.0.

[22] Eleanor Rosch. Principles of Categorization. *Readings in Cognitive Science: A Perspective from Psychology and Artificial Intelligence*, pages 312–322, 2013. ISSN 0262621592. .

[23] Kristin Stock, Anne Robertson, and Mark Small. Representing OGC Geospatial Web Services in OWL-S Web Service Ontologies. *International Journal of Spatial data Infrastructures Research*, 6, 2011.

[24] Mike Uschold, Michael Gruninger, Mike Uschold, and Michael Gruninger. Ontologies : Principles , Methods and Applications. *Knowledge Engineering Review*, 11(2):93–136, 1996. ISSN 0269-8889. .

[25] Panagiotis A. Vretanos. Web Feature Service Implementation Specification. Technical report, Open Geospatial Consorium Inc., 2005.

[26] Shotaro Yasunaga, Mitsunori Nakatsuka, and Kazuhiro Kuwabara. Web ontology building system for novice users: A step-by-step approach. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5991 LNAI(PART 2):134–143, 2010. ISSN 03029743. .

[27] Umi Laili Yuhana. Ontology building. University Lecture, 2007. URL: https://yuhanaresearch.files.wordpress.com/2007/04/ontology-building.pdf [accessed: 2017-10-03].