Implementing a Semantic Catalogue of Geospatial Data

Helbert Arenas, Benjamin Harbelot and Christophe Cruz
Laboratoire Le2i, UMR-6302 CNRS,Département d'Informatique , Université de Bourgogne, 7 Boulevard Docteur Petitjean, 21078 Dijon, France

Keywords: CSW, OGC, Triplestore, Metadata.

Abstract: Complex spatial analysis requires the combination of heterogeneous datasets. However the identification of a dataset of interest is not a trivial task. Users need to review metadata records in order to select the most suitable datasets. We propose the implementation of a system for metadata management based on semantic web technologies. Our implementation helps the user with the selection task. In this paper, we present a CSW that uses a triplestore as its metadata repository. We implement a translator between Filter Encoding and SPARQL/GeoSPARQL in order to comply to basic OGC standards. Our results are promising however, this is a novel field with room for improvement.

1 INTRODUCTION

There is a growing interest in the development of the SDI (Spatial Data Infrastructure), a term that refers to the sharing of information and resources between different institutions.The term was first used by the United States National Research Council in 1993. It refers to the set of technologies, policies and agreements designed to allow the communication between spatial data providers and users (ESRI, 2010).

Currently vast amounts of information are being deployed in the internet through web services. However, in order to profit of this information, potential users need to first identify relevant and suitable datasets. Later, researchers and decision makers would be able to implement *smart queries*. This is a term first employed by Goodwin (2005). It refers to the combination of heterogeneous data sources in order to solve complex problems (Goodwin, 2005).

In the spatial domain this has been possible, thanks, in a significant part to the standardization efforts by OGC (Open Geospatial Consortium). OGC is an international industry and academic group whose goal is to develop open standards that enable communication between heterogeneous systems (OGC, 2012). The tasks in which OGC is interested are: publishing, finding and binding spatial information. OGC provides standards that allow data providers and users to communicate using a common language. The information is offered through web services such as WFS (Web Feature Service), WMS (Web Map Service) or SOS (Sensor Observation Service). However, in order to identify a dataset of interest the user needs first to identify it, using a catalog service. The OGC standard for catalog services is CSW (Catalogue Service for the Web).

OGC defines the interfaces and operations to query metadata records. There are both commercial and opensource/freeware CSW implementations. Among the commercials we can find ESRI ArcGIS server and MapInfo Manager. Among the opensource implementations we find Constellation, Degree and GeoNetworkCSW. The OGC standards do not indicate specific software components. In the case of CSW, developers are able to select the metadata repository more suitable to their preferences/requirements. However, the OGC CSW standard defines operations, requests and metadata formats that should be supported. For instance, queries submitted to a CSW should be formatted as Filter Encoding or as CQL. The former is a XML encoded query language, while the later is a human readable text encoded query language (OSGeo, 2012)(Vretanos, 2005).

Most common implementations of CSW use a relational database as the metadata records repository. For instance, GeoNetwork currently one of the most popular CSW implementations, uses by default a McKoiDB relational database, although it can connect to MySQL, PostGreSQL and other RDBMS (Dunne et al., 2012). Because of the nature of the metadata repository, currently queries are performed

Arenas H., Harbelot B. and Cruz C..
Implementing a Semantic Catalogue of Geospatial Data.
DOI: 10.5220/0004820101520159
In *Proceedings of the 10th International Conference on Web Information Systems and Technologies* (WEBIST-2014), pages 152-159
ISBN: 978-989-758-023-9

by matching strings to selected metadata elements.

In this paper we propose the use of semantic web technologies to store and query metadata records. By using these technologies we are able to take advantage of inference and reasoning mechanisms not available on relational databases. In Section 2 we review research conducted by other teams in the same field. In Section 3 we describe how we have implemented our model. Finally, in Section 4 we present our conclusions and outline future research.

2 RELATED RESEARCH

Spatial information is offered by different providers through web services that implement standards such as WCS, SOS, WFS or WMS. The deployed services might have heterogeneous characteristics regarding software components, languages, or providers. However, by implementing OGC standards all of them have common request and response contents, parameters and encodings. These common elements allow a user to access different services using a proven, safe strategy . In order to allow datasets to be *discoverable*, they have to be published in a catalogue service that implements the CSW standard. The metadata for the datasets is obtained by the catalogue service with a *harvest* operation. The user in order to discover a specific dataset, submits a query. The server processes the query using a *string matching process*, and sends a response to the user. Once the user has identified the relevant dataset, she is able to obtain the data and perform a specific analysis.

The *string matching process* is a major limitation in the current SDI. This limitation has been previously identified by other researchers. In (Kammersell and Dean, 2007) the authors aim to integrate heterogeneous datasources. In this research the authors propose the creation of a layer that translates the users query formulated in OWL into WFS XML request format. Later, they propose do the inverse process with the results. Another approach is proposed by (Kolas et al., 2005). Here the authors propose the implementation of five different ontologies: 1) *Base Geospatial Ontology* for basic geospatial concepts resulting from the conversion of GML schemas into OWL. 2) *Domain Ontology*, this is the user‘s ontology. Its purpose is to link user‘s concepts to the Base Geospatial Ontology. 3) *Geospatial Service Ontology*, used to describe services and allow discovery. 4) *Geospatial Filter Ontology*, which is used to formalize filter description and use. 5) *Feature Data Source Ontology*, to represent the characteristics of the features returned from the WFS. Another approach is described by (Harbelot et al., 2013), here the authors suggest the integration of data from OGC services into a triplestore with a focus on the WFS filters. In (Janowicz et al., 2010) (Janowicz et al., 2012), the authors propose the addition of semantic annotations for each level of a geospatial semantic chain process that involves OGC services. For instance, they propose specific semantic annotations at the level of the service OGC Capabilities document that would correspond to all the datasets managed by the service. Other annotations would correspond to specific data layers. Spatial Data with semantic annotations could later be processed and semantically analysed using custom made reasoning services. To achieve this goal they propose the deployment of OGC services capable of interacting with libraries such as *Sapience* which would result in richer data and data descriptions. However there is little development in this direction. At the moment there is little use of semantic annotations on OGC capabilities documents.

In (Gwenzi, 2010) the author describes the CSW limitations by evaluating GeoNetwork. According to the author there are three possible ways to add semantic capabilities to the CSW: 1) Linking keywords to concepts in the *getCapabilities* response. 2) By adding an ontology browser to the GeoNetwork client interface. 3) Using ebRIM extensions to add ontologies to the CSW. In (Gwenzi, 2010) the author implements the third option.

Another experience is presented by Yue et al. (2006). In this work the authors extend the ebRIM CSW specification by: 1) Adding new classes based to existing ebRIM classes; and 2) Adding Slots to existing classes, thus creating new attributes. With these additions the authors are able to store richer metadata records in the catalogue. The authors identified two options to implement an upgraded search functionality: 1) Create an external component without further modification of the CSW schemas; 2) Modify the CSW adding semantic functionalities to the existing CSW schemas. In (Yue et al., 2006) the authors opted for the first option. Yue et al. (2011) extends this work, focusing on geoservices (Yue et al., 2011).

A different approach is used by (Lopez-Pellicer et al., 2010). In this research the goal is to provide access to data stored in CSW as Linked Data. In order to achieve this goal the authors developed CSW2LD, a middle layer on top of a conventional CSW based server. It allows the server to mimic other Linked Data sources and publish metadata records. CSW2LD wraps the following CSW requests: *GetCapabilities*, *GetRecords* and *GetRecordById*.

A very interesting work in progress is described in (Pigot, 2012). This is a website describing a proposal

by a team from the GeoNetwork developer community. The authors intend to perform a major change in GeoNetwork, allowing it to store metadata as RDF facts stored in a RDF repository. They intent to use SPARQL/GeoSPARQL to retrieve data. The website describe technical characteristics of GeoNetwork and mentions fields that require work in order to implement the project. Currently queries in GeoNetwork are formatted as Filter Encoding or as CQL. Any implementation of a RDF metadata repository would need to consider a translation mechanism between the current queries format to SPARQL (a W3C recommendation) (DuCharme, 2011). Currently GeoNetwork handles spatial constraints using GeoTools. In the semantic web domain, spatial queries are performed using GeoSPARQL (Kolas and Batle, 2012). According to the authors it is not clear if GeoSPARQL is mature enough to handle metadata spatial queries. Even more there is no mechanism to translate spatial constraints into GeoSPARQL. Regardless of the advantages that semantic web technologies might bring into CSWs there is scarce research on this topic. By the time we wrote this paper, there was no further development in (Pigot, 2012) and the website was last updated by the end of October of 2012.

3 IMPLEMENTATION

A regular implementation of OGC CSW, works as a web service that communicates with a data repository that stores metadata records. According to the OGC standard, the catalog should accept requests formatted as *Filter Encoding*, which is a XML based language, designed to express queries. The web service, translates these queries into a suitable format, such that it can communicate with its data repository. In most of the implementations of CSW, the data repository is relational database.

In this paper, we present a proof of concept implementation, designed to show the benefits of using ontologies in a geospatial data catalog service. We have developed a minimalistic implementation of the CSW standard. A major difference between our system and traditional implementations is that we use a triplestore as our metadata repository. We opted for a Parliament triplestore, because of its spatial capabilities thanks to its support for GeoSPARQL. We developed an ontology in the triplestore, and mapped the metadata records to instances of classes specified in the ontology. Thanks to this, it is possible to use *superclass - subclass* relationships in the metadata search process.

In traditional CSW implementations, a spatial search uses the values of the bounding box of the dataset, as specified in the metadata record. However, the number of users with an understanding of coordinate systems, good enough to allow them to search datasets using only the bounding box coordinate values is quite limited. In our approach, we implement an ontology class called *ToponymUnit*. Instances of this class are geographic features with labels familiar to the users. In our implementation, the user can search for metadata records whose bounding box has specific spatial relations with instances of the class *ToponymUnit*. Using this approach, users can submit queries such as: retrieve metadata records that are *within* the toponym unit known as *France*.

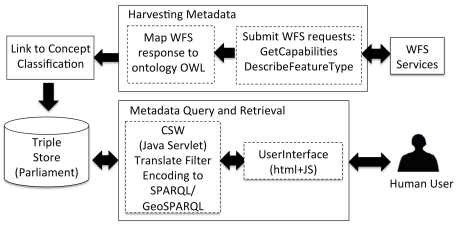

Figure 1: Architecture implementation.

Our current implementation is able to respond to standard *GetRecords* requests submitted as *POST*. The response of our system follows the *csw:SummaryRecord* format. A crucial part of the implementation is the translation of requests from *Filter Encoding* to *SPARQL/GeoSPARQL*.

Figure 1 depicts the processes in the proposed system. In the next subsections we further describe how we obtain the information necessary to construct the metadata records, how we map this information to an ontology, and finally how we perform queries.

3.1 Harvesting Metadata

We focus our research on metadata records for datasets available through services that implement the OGC Web Feature Service (WFS) standard. A service that implements WFS can contain one or many datasets. We identify the datasets available on a WFS and create a metadata record for each one of them. The metadata record is stored as an instance of the class *abc:MetadataRecord* .

In our ontology we implement a class called *geo:Feature* that represent features with spatial representation. Although, a metadata record is an abstract description, it does have a spatial component represented by the bounding box of the dataset it describes. Due to the spatial nature of the metadata records, we define the class *abc:MetadataRecord* as a subclass of *geo:Feature* (See Figure 2).

In our ontology we implemented Dublin Core elements, as properties for instances of the class

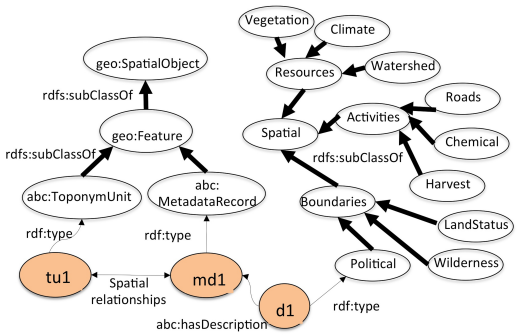

Figure 2: Classes, instances and relationships in the proposed model.

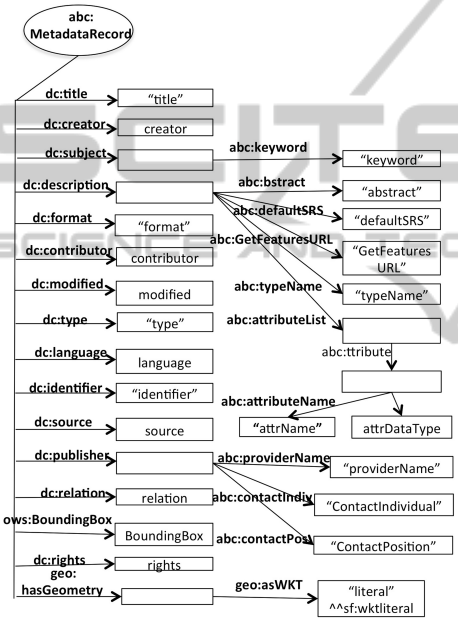

Figure 3: Properties of instances of the class *abc:MetadataRecord*

abc:MetadataRecord. We have developed a harvest tool that queries the WFS and constructs triples with the responses. Our tool is a Java application that makes use of two requests that are part of the core OGC WFS standard: *GetCapabilities* and *DescribeFeatureType*. From the *GetCapabilities* request, we obtain a general description of the catalog service and information regarding the available datasets on it. With the *DescribeFeatureType* request, we can obtain the list of attributes for the actual features that compose the dataset. We have tested our tool with 17 services that implement WFS, having as a result 2690 metadata records. Figure 3 depicts the Dublin Core elements as properties of the class *abc:MetadataRecord*.

The mapping process between the WFS request responses and ontology properties is done with our harvest tool. The WFS respond to requests with a XML document. Our harvest tool, navigates through the response document, identifies relevant elements and maps them to specific properties in our ontology (See Figures 1 and 3).

For instance, the following code is part of a *GetCapabilities* response containing information regarding the WFS publishing entity.

```
 <ows:ServiceProvider>
<ows:ProviderName>
Provider X
</ows:ProviderName>
<ows:ServiceContact>
<ows:IndividualName>Helbert<ows:IndividualName/>
<ows:PositionName>GIS Manager<ows:PositionName/>
</ows:ServiceContact>
</ows:ServiceProvider>
```

This information is mapped into the *dc:publisher* Dublin Core element (See Figure 3).

```
abc:md1 a abc:MetadataRecord .
abc:md1 dc:publisher _:B01 .
_:B01 abc:ProviderName "Provider X" .
_:B01 abc:PositionName "GIS Manager" .
_:B01 abc:IndividualName "Helbert" .
```

The following XML code, is another part of a *GetCapabilities* response.

```
 <FeatureType xmlns:example=
 "http://www.example-provider.org/example">
<Name>example:name</Name>
<Title>Example dataset title</Title>
<Abstract>Example abstract</Abstract>
<ows:Keywords>
<ows:Keyword>example keyword1</ows:Keyword>
</ows:Keywords>
<DefaultSRS>
urn:x-ogc:def:crs:EPSG:4326
</DefaultSRS>
<ows:WGS84BoundingBox>
<ows:LowerCorner>-5.84 37.75</ows:LowerCorner>
<ows:UpperCorner>11.02 54.63</ows:UpperCorner>
</ows:WGS84BoundingBox>
</FeatureType>
```

From this segment of the response, we can obtain information for: *dc:title*, *dc:subject*, *dc:description* and *ows:BoundingBox*.

```
abc:md1 a abc:MetadataRecord .
abc:md1 dc:title "Example dataset title" .
abc:md1 dc:subject _:B02 .
_:B02 abc:keyword "example keyword1" .
abc:md1 dc:description _:B03 .
_:B03 abc:abstract "Example abstract" .
_:B03 abc:defaultSRS "EPSG:4326" .
abc:md1 geo:hasGeometry _:B04 .
_:B04 geo:asWKT "POLYGON((-5.84 37.75,
-5.84 54.63, 11.02 54.63, 11.02 37.75,
-5.84 37.75))"^^sf:wktLiteral .
```

Additionally, our harvesting tool, submits a *DescribeFeatureType* request for each layer of information found in the WFS. From the response we are able to obtain a list of attributes for the spatial features that compose the dataset. The following XML code depicts part of a response to a *DescribeFeatureType* request.

```
<xsd:complexType name="country_boundsType">
<xsd:complexContent>
<xsd:extension base="gml:AbstractFeatureType">
<xsd:sequence>
<xsd:element maxOccurs="1" minOccurs="0"
name="THE_GEOM" nillable="true"
type="gml:MultiSurfacePropertyType"/>
<xsd:element maxOccurs="1" minOccurs="0"
name="AREA" nillable="true"
type="xsd:double"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
```

Our harvesting tool, browses through the response, identifies relevant elements and create an attribute list within the dublin core element *dc:description* , and creates the following triples:

```
abc:md1 a abc:MetadataRecord .
abc:md1 dc:description _:B03 .
_:B03 abc:attributeList _:B05 .
_:B05 abc:attribute _:B06 .
_:B06 abc:attributeName "THE_GEOM" .
_:B06 abc:attributeDataType
gml:SurfacePropertyType .
_:B05 abc:attribute B07 .
_:B07 abc:attributeName "AREA" .
_:B07 abc:attributeDataType xsd:double .
```

3.2 Link to Concept Classification

By using semantic web technologies we are not limited to string matching queries. We can also use inference mechanisms based on subsuming and established relationships between terminology and concepts. To test these capabilities, we have implemented a taxonomy with *domain ontology* classes. The relationships between these concepts are of the type *subclassOf*. Individual datasets are represented as instances of the domain ontology classes. Each dataset is described by a metadata record, which is an instance of the class *abc:MetadataRecord* . The link between these two instances is given by the property *abc:hasDescription* (See Figure 2). The following triples depict the relation between an instance of *abc:MetadataRecord* and an instance of the domain ontology class *abc:Political*.

```
abc:d1 a abc:Political .
abc:md1 a abc:MetadataRecord .
abc:d1 abc:hasDescription abc:md1 .
```

Figure 2 depicts our proposed domain ontology. Using the *domain ontology* we can make inferences regarding the class membership. From the previous example, because $abc : Political$ is a subclass of $abc : Boundaries$ and $abc : Spatial$, we can infer that $abc : md1$ is also a member of these two classes.

The goal of this paper is to show potential uses of semantic web technologies for metadata record management. At the moment our focus is not on the automatic classification of datasets within a domain ontology. This task can be achieved with a variety of methods for instance: Naive Bayes, Decision Rules, Neural Networks, among others. For this experiment, we decided to create an instance of the class $abc : Spatial$ for each metadata record, and later add further specification of the instance randomly, only in order to test the system query capabilities. Our future development plans include the implementation of sophisticated methods for the dataset classification. An interesting work in this field, although not in the spatial domain is (Werner et al., 2012).

3.3 Toponym Elements

In order to facilitate the identification of suitable spatial datasets we define the class *abc:ToponymUnit* as a subclass of *geo:Feature* . Instances of the class *abc:ToponymUnit* are geographic features with known, accepted names. Our system enables users to integrate into their metadata queries spatial relations between the spatial components of metadata records (bounding box) and toponym instances (See Figure 2).

To populate the class *abc:ToponymUnit*, we use a country political boundaries dataset from Esri and DeLorme Publishing Company, Inc. under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States License (ESRI, 2011). The dataset is a shapefile with 668 multipolygon features. Our goal, is to create instances of the class *abc:ToponymUnit*, each instance representing an area with a known political designation.

However, before translating the political boundaries dataset into triples, it was necessary to perform the following steps: 1) Convert the multipolygon features to polygon ones. 2) Delete polygons that we considered too small for practical purposes. 3) Simplify the remaining polygons by reducing the number of vertices. 4) Translate the political boundaries dataset into instances and triples using a customized Java program, implemented with Jena and GeoTools libraries. and finally 5) Upload the triples into our triplestore. The pre-loading processing was done using Quantum GIS and GeoTools. The final result is

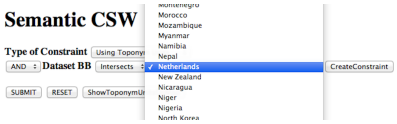

Figure 4: HTML user interface: Defining a constraint using a spatial relationship with an instance of th class *abc:ToponymUnit*

3037 instances of the class *abc:ToponymUnit*.

3.4 Metadata Query

In order to test our implementation, we need to enable users to submit queries to the catalog service and browse through the query results. This process is composed by the following four sub-processes : 1) Definition of the query. 2) Mapping the user request into a suitable metadata repository query format. 3) Mapping the triplestore response into an allowed OGC standard. 4) The visualization of the results in the user interface (See Figure 1).

3.4.1 Definition of the Query

The OGC standard query language for services implementing CSW is called Filter Encoding. This is a language based on XML . Queries with Filter Encoding might become long XML documents that require a strict syntax, therefore it is necessary a software application that helps the user to compose them. In our system this task is done in a user interface deployed as a web page. It uses a combination of HTML and JavaScript, to enable users to compose complex queries.

In order to help the user to compose queries, the web site requests from the Server: 1) The list of domain ontology classes, 2) The list of labels associated to instances of *abc* : *ToponymUnit*. It uses this information to populate combo boxes, allowing the user to compose constraints (See Figure 4).

The JavaScript application receives the user input and formats it as a XML query document following the Filter Encoding specification. The application allows multiple constraints to be linked using the operators *AND* and *OR*. When the query is completed the user submits it as a *POST*. The following XML code represents a query as formatted by the JavaScript running on the website.

```
<GetRecords><Query>
<Constraint><Filter><And>
<PropertyIsLike>
<ValueReference>dc:title</ValueReference>
<Literal>water</Literal>
</PropertyIsLike>
<DescribesInstanceOf>abc:Harvest
</DescribesInstanceOf>
<sfIntersects>
<ValueReference>BoundingBox</ValueReference>
<ToponymUnit>Netherlands</ToponymUnit>
</sfIntersects>
</And></Filter></Constraint>
</Query></GetRecords>
```

3.4.2 From Filter Encoding to SPARQL/GeoSPARQL

We are using as our metadata repository a triplestore, therefore it is necessary to translate request from Filter Encoding to SPARQL/GeoSPARQL. This task is accomplished by our Servlet application. Once the XML query arrives, the application proceeds to decompose it, into its constituent constraints.

In a SPARQL query, we distinguish three components. 1) The specific elements or nodes we are requesting; 2) A set of triples that define a pattern the SPARQL engine is going to look for; and 3) The filter component, where we define a set of boolean value conditions for the triples that match the previously defined pattern.

Our servlet application translates each constraint separately into the respective set of triples pattern and filter conditions. Using the interface the user is able to define three types of constraints:

1. Alphanumeric attributes in the metadata record: The user can select one attribute in the metadata record, and perform a string matching, using the operators *PropertyIsEqualTo* and *PropertyIsLike*. In the later case the SPARQL implementation will require the definition of a filter component:

   ```
   <PropertyIsLike>
   <ValueReference>dc:title</ValueReference>
   <Literal>water</Literal>
   </PropertyIsLike>
   ```

   would be translated to the triple pattern:

   ```
   ?md a abc:MetadataRecord.
   ?md dc:title ?xTitle.
   ```

   with the filter component:

   ```
   (regex(?xTitle,"water","i"))
   ```

2. Domain ontology class membership: Each metadata record describes an entity with a class membership. This type of constraint allows the user to identify the class membership of the entity. For example, the constraint:

   ```
   <DescribesInstanceOf>abc:Harvest
   </DescribesInstanceOf>
   ```

   Is translated as the SPARQL triples:

   ```
   ?md a abc:MetadataRecord.
   ?ds abc:hasDescription ?md.
   ?ds a abc:Harvest.
   ```

3. Using toponym elements and spatial relationships: In this case the user identifies a toponym of interest, then she defines a spatial relationship between the bounding box of the metadata record and the geometry of the selected toponym. We implement the spatial operators *Disjoint*,*Intersects*, *Contains* and *Within* (See Figure 4). For example, the following XML extract, indicates that the metadata records bounding box should intersect the geometry of *Netherlands*:

```
<sfIntersects>
<ValueReference>BoundingBox</ValueReference>
<ToponymUnit>Netherlands</ToponymUnit>
</sfIntersects>
```

The constraint is translated as the following triple pattern:

```
?md a abc:MetadataRecord.
?md geo:hasGeometry ?boundingbox.
?boundingbox geo:asWKT ?boundingbox_wkt.
?topoUnit a abc:ToponymUnit.
?topoUnit abc:CountryName "Netherlands".
?topoUnit geo:hasGeometry ?topoGeo.
?topoGeo geo:asWKT ?topoWKT.
```

plus the additional filter component:

```
(geof:sfIntersects
(?boundingbox_wkt,?topoWKT))
```

Once all the constraints have been translated, the triples and filter elements are merged into a syntactically correct SPARQL/GeoSPARQL query, which is submitted to the triplestore. The following code, depicts the SPARQL query resulting from combining all the triple patterns and filter components from the previous examples:

```
SELECT DISTINCT ?md ?xTitle
Where
{?md a abc:MetadataRecord.
?md dc:title ?xTitle.
?md geo:hasGeometry ?boundingbox.
?boundingbox geo:asWKT ?boundingbox_wkt.
?ds abc:hasDescription ?md.
?ds a abc:Harvest.
?topoUnit a abc:ToponymUnit.
?topoUnit abc:CountryName "Netherlands".
?topoUnit geo:hasGeometry ?topoGeo.
?topoGeo geo:asWKT ?topoWKT.
FILTER((regex(?xTitle,"water","i"))&&
(geof:sfIntersects(?boundingbox_wkt,?topoWKT)))}
```

The response from the triplestore is then formatted by the servlet as *csw:SummaryRecord* and sent to the client website. The results are visualized in the website allowing the user to examine the metadata records (See Figure 5).

3.5 Smart Queries

A *smart query* requires the combination of diverse datasources as described in (Goodwin, 2005). However, first the researcher must be able to identify the most suitable dataset for the analysis. Our implementation aims to help users in this task. By using a domain ontology, we improve the user's query capabilities. Our use of toponyms, allows the user to select areas of interest by name, and establish specific spatial relationships with the dataset of interest. The actual features of the dataset can later be obtained using the value of *abc:GetFeaturesURL*, in the metadata record.

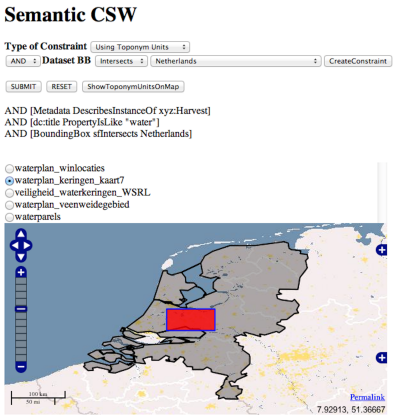

Figure 5: HTML user interface showing the results of the example query.

4 CONCLUSIONS

In this work we present a simplified CSW implementation with a triplestore as a metadata repository. Our implementation has a working translator that is able to convert Filter Encoding queries into SPARQL/GeoSPARQL ones. The system allows complex queries that can take advantage of inference mechanisms provided by Semantic Web technologies.

At this point, our system only uses inference based on class to subclass relationships. However, we plan to extend these capabilities to include relationships between concepts and automatic class membership determination using a domain ontology.

Our approach to capture metadata information is generic, takes advantage of the OGC standard interfaces. With our harvesting tool we were able to create 2690 metadata records. However the information supplied by the WFS publishing entities has limitations and is in many cases incomplete. Our metadata records contain 1384 distinct keywords including 383 actual URLs. However in no case the URLs referred to any ontology or formalized vocabulary. From the URLs, 217 were links to html documents, and 52 to XML documents. In both cases the documents contained extended metadata descriptions of the datasets. All

the datasets with URL of extended descriptions were provided by one single WFS deployment (giswebservices.massgis.state.ma.us/geoserver/wfs?), the rest of the keywords were strings with no formal semantics associated. Our metadata harvesting tool also obtained information regarding the names of the attributes of the dataset. In total we have obtained 6331 individual attribute names, all of them were strings with no formal semantics associated.

The use of extended descriptions in XML and HTML documents is not a standard practice among the WFS publishing entities. However, in case we find more documents of this kind, we can upgrade the harvesting tool in order to allow it to get information from the associated documents.

The results of our current implementation are promising, in the near future we will implement an automatic classification of metadata records based on harvested metadata using a domain ontology.

ACKNOWLEDGEMENTS

This research is supported by: 1) Conseil régional de Bourgogne. 2) Direction Générale de l'Armement, see: http://www.defense.gouv.fr/dga/.

REFERENCES

DuCharme, B. (2011). *Learning SPARQL*. O'Reilly Media, Inc.

Dunne, D., Leadbetter, A., and Lassoued, Y. (2012). ICAN Semantic Interoperability Cookbooks. Technical report, International Coastal Atlas Network.

ESRI (2010). GIS Best Practices: Spatial Data Infrastructure (SDI). http://www.esri.com/library/bestpractices/spatial-data-infrastructure.pdf. Accessed: July 2013.

ESRI, D. (2011). World administrative units. http://resources.arcgis.com/content/data-maps/10.0/world. Accessed on May 2013.

Goodwin, J. (2005). What have ontologies ever done for us - potential applications at a national mapping agency. In *OWL: Experiences and Directions (OWLED)*.

Gwenzi, J. (2010). Enhancing spatial web seach with semantic web technology and metadata visualization. Master of science, University of Twente.

Harbelot, B., Arenas, H., and Cruz, C. (2013). Semantics for Spatio-Temporal "Smart Queries". In *Poster presentation in the 9th. International Conference on Web Information Systems and Technologies*, Aachen, Germany.

Janowicz, K., Schade, S., Broring, A., Kebler, C., Maue, P., and Stasch, C. (2010). Semantic Enablement for Spatial Data Infrastructures. *Transactions in GIS*, 14(2):111–129.

Janowicz, K., Scheider, S., Pehel, T., and Hart, G. (2012). Geospatial Semantics and Linked Spatiotemporal Data - Past, Present and Future. *Semantic Web - Interoperability, Usability and Applicability*, 3(4):1–10.

Kammersell, W. and Dean, M. (2007). Conceptual Search: Incorporating Geospatial Data into Semantic Queries. In Scharl, A. and Tochtermann, K., editors, *The Geospatial Web*, Advanced Information and Knowledge Processing, pages 47–54. Springer London. 10.1007/978-1-84628-827-2_5.

Kolas, D. and Batle, R. (2012). GeoSPARQL User Guide. http://ontolog.cim3.net/file/work/SOCoP/Educational/GeoSPARQL User Guide.docx Accessed on May 2013.

Kolas, D., Hebeler, J., and Dean, M. (2005). Geospatial Semantic Web: Architecture of Ontologies. pages 183–194.

Lopez-Pellicer, F. J., Florczyk, A., Renteria-Aguaviva, W., Nogueras-Iso, J., and Muro-Medrano, P. R. (2010). CSW2LD: a Linked Data frontend for CSW.

OGC (2012). OGC Institutional Web Site. http://www.opengeospatial.org/. Accessed: September 2013.

OSGeo (2012). CQL. http://docs.geotools.org/latest/userguide/library/cql/cql.html. Accessed on November 2012.

Pigot, S. (2012). Using RDF as Metadata Storage. http://trac.osgeo.org/geonetwork/wiki/rdfstore. Accessed on May 2013.

Vretanos, P. A. (2005). Filter Encoding Implementation Specification. online. Accessed on May 2013.

Werner, D., Cruz, C., and Nicolle, C. (2012). Ontology-based Recommender System of Economic Articles. In *WEBIST 2012*, pages 725–728.

Yue, P., Di, L., Yang, W., Yu, G., and Zhao, P. (2006). Path planning for chaining geospatial web services. In *Proceedings of the 6th international conference on Web and Wireless Geographical Information Systems*, W2GIS'06, pages 214–226, Hong Kong, China. Springer-Verlag.

Yue, P., Gong, J., Di, L., He, L., and Wei, Y. (2011). Integrating Semantic Web Technologies and Geospatial Catalog Services for Geospatial Information Discovery and Processing in Cyberinfrastructure. *GeoInformatica*, 15:273–303. 10.1007/s10707-009-0096-1.