Ontology-based Geospatial Data Query and Integration

Tian Zhao[1], Chuanrong Zhang[2], Mingzhen Wei[3], Zhong-Ren Peng[4]

(1) University of Wisconsin – Milwaukee, WI
(3) U.S. Geological Survey, Rolla, MO, 65401
(2) University of Connecticut, CT (4) University of Florida, FL

Abstract. Geospatial data sharing is an increasingly important subject as large amount of data is produced by variety of sources, stored in incompatible formats, and accessible through different GIS applications. Past efforts to enable sharing have produced standardized data format such as GML and data access protocols such as Web Feature Service (WFS). While these standards help enabling client applications to gain access to heterogeneous data stored in different formats from diverse sources, the usability of the access is limited due to the lack of data semantics encoded in the WFS feature types. Past research has used ontology languages to describe the semantics of geospatial data but ontology-based queries cannot be applied directly to legacy data stored in databases or shapefiles, or to feature data in WFS services. This paper presents a method to enable ontology query on spatial data available from WFS services and on data stored in databases. We do not create ontology instances explicitly and thus avoid the problems of data replication. Instead, user queries are rewritten to WFS getFeature requests and SQL queries to database. The method also has the benefits of being able to utilize existing tools of databases, WFS, and GML while enabling query based on ontology semantics.

1 Introduction

The National Spatial Data Infrastructure promotes geospatial data sharing to improve data quality, to reduce costs, and to make data more usable to the public [17]. However, much of the existing geospatial data is stored in proprietary formats such as ESRI shapefiles, coverage, and geodatabases, and it is only accessible through vendor-specific geospatial information systems such as ESRI ArcGIS and Intergraph GeoMedia [29, 36]. Data sharing is difficult in this context because different GIS software has incompatible system designs, data models, and database storage structures [9, 28, 34, 35].

To enable sharing of geospatial data, the Open Geospatial Consortium (OGC) has established a series of specifications such as Geographic Markup Language (GML) [11] and Filter encoding, and data exchange protocols such as Web Feature Service (WFS) [13] and Web Map Service (WMS) [12]. These specifications and protocols have provided constructs for describing and accessing geospatial data at the domain level. For example, GML files can encode the geometries of an object as points, lines, or polygons. The WFS server can accept query of features within a bounding box and return the query results in GML format. The resulting GML data can be rendered for visualization based on the geometry encoding. Also, WFS servers accept more complex queries using OGC filter encodings with relational or spatial operators [13]. Although the fast development of these standards and web service technologies has undoubtedly improved the sharing and synchronization of geospatial information across diverse resources, they only can support technical data interoperability and cannot resolve semantic heterogeneity problems in spatial data sharing. WFS and WMS protocols, and GML and OGC filters have no provisions for data sharing at semantics level for applications. For example, a WFS server may name a feature representing a bus route as "Route" or "ROUTE". A *getFeature* query to the WFS server has to spell the name correctly, otherwise no results will return. Similarly, the geometry of the feature can be either a complete route or just a link segment of the route. WFS service client has to know this information in order to formulate *getFeature* requests to retrieve a route by its name. Moreover, a route may have implicit relations with other features such as bus stop but such relation is not specified as a feature property. Even if relations such as bus stops of a route are somehow included as feature properties, they has to be encoded using XML complex types because of the one-to-many relations and users of WFS services may not be able to interpret the meaning of these feature properties based on the XML types alone.

The root of the problem is that structured data such as XML and GML does not have enough constructs to express data semantics. As a result, software developed based on the OGC standards cannot be readily adapted to consider data semantics. Recent research [18, 1, 15, 28, 36] has applied the concepts of semantic web to geospatial data sharing. Semantic web [5] promotes the use of ontology languages such as Resource Description Framework (RDF) and Web Ontology Language (OWL) to provide semantics for data usually found in databases or other structured documents [19, 36]. One important advantage of RDF and OWL ontology is that it is easier to define semantics for data using ontology constructs such as classes and properties. In addition, combined with ontology semantic constraints, the ontology can allow reasoners based on Description Logic (DL) to infer further knowledge from partially specified data and check for data consistencies [2]. Tools like Jena[1] allow inference rules be used to support more powerful query of ontology data. For example, in transit system, a property *transitStop* may be the same as the composition of the *transitPointFeatureEvent* and *stopEvent* properties. This is not yet supported in OWL but one can encode this rule in Jena using its general purpose rule engine. Finally, ontology definitions are extensible so that geospatial applications can use existing domain ontologies as basis to create application ontologies. Thus, by providing a semantic interpretation of the data, RDF and OWL ontology allows software programs to automatically understand structures and meanings of diverse information sources and conduct automatic knowledge inference or reasoning from existing data and documents.

Recent research projects have applied ontology to modeling observations and measurements [30], to enable spatial/temporal/thematic reasoning [1], and to assist the discovery and access of geospatial web services [27]. However, ontology reasoners only can apply to ontology instances such as RDF instances or OWL individuals, which correspond to the database records and WFS feature instances. Transforming all legacy geospatial data to ontology form is time consuming, error prone, and inefficient. Also ontology tools such as Protégé[2] cannot efficiently manipulate ontology instances in large quantity due to memory consumption. Moreover, it is not cost-effective for ontology tools to include the functionalities provided by geodatabases and WFS ser-

[1] http://jena.sourceforge.net/
[2] http://protege.stanford.edu

vices such as transaction management and spatial query. Thus, to efficiently support ontology-based reasoning on geospatial data, it is necessary to keep legacy data stored in geodatabases and other data files while providing an ontology-enabled interface to translate user requests into queries to legacy data stores. To this end, there are projects focused on extracting ontology definitions from database schemas [3] and annotating geospatial data with semantic annotations based on OWL and inference rules [23]. However, it is not clear how the extracted ontology information can help translate user requests into queries to legacy data sources.

In this paper, we propose a new solution to spatial data interoperability at semantic level through an interface based on RDF ontology. We use a real world transportation application, a transit system, as an example for our solution. Comparing with other existing approaches, the most important advantage of our solution is that it does not need to replicate legacy data stored in relational databases, shapefiles, or GML data accessible from WFS services. The interface is to provide an ontology layer for spatial data accessible from WFS services and databases. Users of the interface can query spatial data as if it is defined in terms of some domain and application ontologies. The interface relies on WFS services as data sources for spatial data so that it can use the spatial query functions of WFS servers and use existing WFS client library for feature rendering. The interface can also use relational databases as sources for non-spatial data since this can improve the performance of queries not involving geometries.

The rest of the paper is organized as follows. We discuss related works in Section 2. In Section 3, we give an overview of the proposed interface. Section 4 presents a simple RDF ontology used in our example. In Section 5, we describe the mapping and inference rules that are used to connect ontology definitions with the feature data in databases and WFS servers. Section 6 explains the supported RDF queries, the query semantics, our query rewriting algorithm, and possible extension to handle RDF semantic constraints. Implementation issues are discussed in Section 7.

2 Related work

RDF-based data integration The problem of data integration is to combine data of different sources and provide users a unified view of the data [24]. A data integration system often uses a global schema containing mappings from global definitions to local schemas in each data sources. In this context, data query problem can be reduced to the problem of answering queries using materialized views [21], where data sources are described as precomputed views on the global schema. Our method is similar to answering queries using views except that the global schema is defined using RDF ontology and because of this, the views are not just the mapping from data sources on the global schema but also include inference rules to obtain object properties. Also, we need to consider the semantic constraints of the RDF ontology such as subclass and subproperty relations. In this aspect, our method is closely related to [8] that also uses RDF ontology as a medium to provide integrated access to different relational databases. Their approach is to define database schemas as views on RDF ontology entirely including object properties and semantic constraints. One of their focus is to consider databases as incomplete data sources such that missing information is tolerated while query rewriting may result in alternative answers to the same query based on different data sources used. In comparison, our method is more restrictive in the way that mappings may be defined from database tables or WFS features to RDF ontology. In particular, we require each database table and WFS feature to map to one RDF class. Additional RDF classes and object properties are defined through inference rules. This simplifies the query rewriting algorithm and still preserves the flexibility of using RDF ontology to express complex relations of spatial and non-spatial objects. Another related work is D2RQ [7], which is a tool suite that provides RDF interface to relational databases. D2RQ allows users to define a mapping file with rules to establish a RDF ontology that maps RDF classes and properties to database tables and columns. Also, additional properties may be defined over existing RDF classes and properties in a way similar to our inference rules. D2RQ only provides RDF interface for relational databases. We use it to provide access to non-spatial data in databases while spatial data queries are handled by WFS servers.

Geo-spatial ontology Difference in semantics used in different data sources is one of the major problems in spatial data sharing and data interoperability [6, 16]. One possible approach to overcome the problem of semantic heterogeneity is by means of ontology [18, 31, 32]. Cruz *et. al.* [10] studied the problem of ontology-alignment in the context of integrating geospatial data in databases. They developed a semi-automatic method of generating mappings between ontologies of local databases and a global one. The mappings can then be used for query rewriting. Baglioni *et. al* [3] proposed a method on accessing spatial database through ontology layer by semi-automatically building an application ontology from a geographical database and then enrich it with domain ontology by finding correspondence between the classes and properties of the two ontologies. The enriched ontology is said to be used in assisting query answering though it is not clear how semantic queries are translated to spatial SQL to databases. Also related is a project for semi-automatically adding semantic annotations to geospatial data [23], which uses OWL to provide semantic annotation and uses Semantic Web Rule Language (SWRL)[3] to add additional properties between instances based on the existing ontology. More general discussion on developing geospatial ontologies can be found in the work of Arpinar *et. al.* [1], where geospatial semantics are considered for three types of geospatial relations: topological relations, cardinal directions, and proximity relations. Their system architecture would pull geospatial data from sources such as National Map, NASA sources, UCGIS sources, and put it in a massive metadata store, which is then used to populate the ontology-based knowledge base accessible to users via spatial/temporal/thematic reasoners. When geospatial data is solely provided through web services, an additional layer of ontology included in a Service Oriented Architecture can be useful. Paul and Ghosh [27] argue that a domain ontology can be used to provide shared vocabulary for the schemas of WFS servers. User queries to a service broker, which maintains a list of services, can somehow be translated to specific getFeature requests to different service providers – WFS servers. The SPIRIT spatial search engine [22] has shown ontology to be useful in searching web documents with spatial contents. User queries can include a subject, a place name, and a spatial rela-

[3] http://www.w3.org/Submission/SWRL/

tion to the place name. Results are list of documents and their positions on a map. The search engine uses geographical and domain ontologies to disambiguate and expand user queries, to rank documents based on relevance, and to extract metadata from web documents.

The overall assessment is that our approach may extend or complement the related works. For example, some methods [3, 10] generate useful ontologies from geospatial databases, which can be used as the application and/or domain ontologies for our interface. We do not use OWL as in [23] but it may be possible to include OWL ontologies with help of reasoning tools. Also, our method could be one step towards realizing the goal of using reasoners to query data sources through ontology interface as proposed in [1]. The differences are that we do not require data be migrated from sources to knowledge base, thus avoiding the problems of data replication, and we do not support thematic and spatial-temporal queries. Lastly, our method may be applied to both web service discovery and retrieval [27]. In this setting, web services should be published using the class and properties of a domain ontology and service brokers then translate service-retrieval requests to WFS getFeature requests using predefined rules.

3 A RDF-based Spatial Data Interface

Geospatial data contains both spatial attributes such as geometry and non-spatial attributes. The fact that spatial data is distributed among many sources and stored in different formats makes it hard to query spatial data through a single interface. WFS as a standardized protocol designed to alleviate this problem by providing uniform interface to data stores in forms of databases, shapefiles, and GML files. Though there are sophisticated WFS server implementations such as GeoServer[4], software for implementing WFS clients is less than ideal. The client software such as MapBuilder[5] and OpenLayers[6] primarily supports map retrieval, feature rendering, and feature transactions. While the existing WFS clients can locate WFS services and create map layers, the utility of these clients is limited by the lack of data semantics in WFS features. Also, though features may contain descriptive attributes, joint queries based on them are not very efficient since the WFS protocol uses bulky GML tags to encode feature attributes while WFS service is more suitable for spatial querying and transactions. Many identifying attributes of the spatial features can be efficiently stored and accessed through traditional relational databases.

In this paper, we describe an ontology-based interface for querying spatial features. We focus on three aspects:

1. Create an ontology to describe the problem domain of the data supported by the interface.
2. Define mapping and inference rules to connect the ontology with the WFS feature types and database schemas.
3. Rewrite ontology queries to getFeature requests to WFS services and SQL queries to databases to obtain answers.

[4] http://geoserver.org/display/GEOS/GeoServer+Home
[5] http://communitymapbuilder.org/
[6] http://www.openlayers.org/

```
<rdfs:Class rdf:ID="Feature"/>

<rdfs:Class rdf:ID="SpatialFeature">
  <rdfs:subClassOf rdf:resource="#Feature"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Geometry"/>

<rdf:Property rdf:ID="hasID">
  <rdfs:domain rdf:resource="#Feature"/>
</rdf:Property>

<rdf:Property rdf:ID="geometry">
  <rdfs:domain rdf:resource="#SpatialFeature"/>
  <rdfs:range rdf:resource="#Geometry"/>
</rdf:Property>
```

Fig. 1. Domain ontology in RDF schema

For simplicity, we assume that the spatial data in question is distributed among a WFS service and a relational database. The WFS service stores the feature geometries and IDs while the relational database stores the rest of the non-spatial attributes.

We define the ontology in terms of a RDF schema. In the schema, RDF classes correspond to the feature types while RDF properties correspond to feature attributes and relations between features. The instances of the RDF classes form a RDF graph and they correspond to the spatial features but we are not going to create these RDF instances explicitly. Instead, the ontology interface provides access to the features through query rewriting. The mapping rules define the correspondence between the RDF properties and the WFS feature attributes and database columns. Simply mapping rules are not very useful other than unifying the access to the two data sources. We use additional inference rules to define some RDF properties between RDF instances in terms of other RDF properties created in the previous step.

4 RDF-based ontology

We begin with a simple domain ontology where `Feature` class contains all spatial and non-spatial objects. All instances of this class have an ID field through the property `hasID`. Spatial objects belong to the `SpatialFeature` class, which is a subclass of `Feature` and has an additional `geometry` property.

We extend the domain ontology with an application ontology to describe a transit system with `Route`, `Stop`, `Link`, `LinkSequence`, `Pattern`, etc. The `Stop` and `Link` are spatial classes with point and line geometry respectively. Other classes describe non-spatial features but they are all closely related to the spatial features. For example, each instance of `Route` class contains several instances of `Link` that make up the route through the property `route_link`. A `Route` instance can also point to several instances of `Stop` that are on the route. A `Stop` instance can refer to stops within

Class	**Superclass**
Route	*Feature*
Pattern	*Feature*
LinkSequence	*Feature*
Stop	*SpatialFeature*
Link	*SpatialFeature*

Property	**Domain**	**Range**
name	*Route*	*string*
intersection	*Stop*	string
routeID	*Pattern*	integer
patternID	*LinkSequence*	integer
linkID	*LinkSequence*	integer
linkOrder	*LinkSequence*	integer
route_link	*Route*	*Link*
route_stop	*Route*	*Stop*
nearby_stop	*Stop*	*Stop*

Fig. 2. Summary of application ontology for transit system

a certain radius via the property `nearby_stop`. We can find the `intersection` of a `Stop` instance as well.

Other classes and properties are not directly related to spatial objects but they are used to infer the properties such as `route_link`. For example, the spatial data for link and stop originally stored as shapefiles may be accessible as Web Features through a WFS server. However, they only contain IDs and their geometries. Thus, we cannot find out the property `route_link` directly by querying the WFS service, and we need other classes and properties to infer `route_link`. The non-spatial data such as Route, Stop (some of its properties), LinkSequence, Pattern often is stored in tables of a relational database and we can use its properties to infer `route_link`. Specifically, we can use the Pattern table to find the patterns in a route and then find the links contained in these patterns through the LinkSequence table (which describes the links traversed in a pattern sequentially).

5 RDF views of WFS features and database tables

The RDF ontology is an abstraction of the spatial and non-spatial data stored in WFS servers and relation databases. In order to have an ontology-based interface to the data, we need a way to map the RDF ontology to the schemas of WFS features and relational tables. This mapping is defined as *RDF views* from the feature and database schemas to the ontology. With RDF views, RDF queries can be rewritten to *getFeature* requests to WFS servers and SQL queries to databases.

The RDF views are defined in two steps. The first step is to create mapping rules to define each WFS feature and relational table as a view over the RDF ontology. In this step, only *datatype properties* are used in the mapping rules. Datatype properties are the properties with ranges of primitive types such as string or integer. The second step is to define inference rules for some additional RDF properties in terms of the datatype properties used in the previous step. The additional properties include the so-called *object properties* whose ranges are RDF class types.

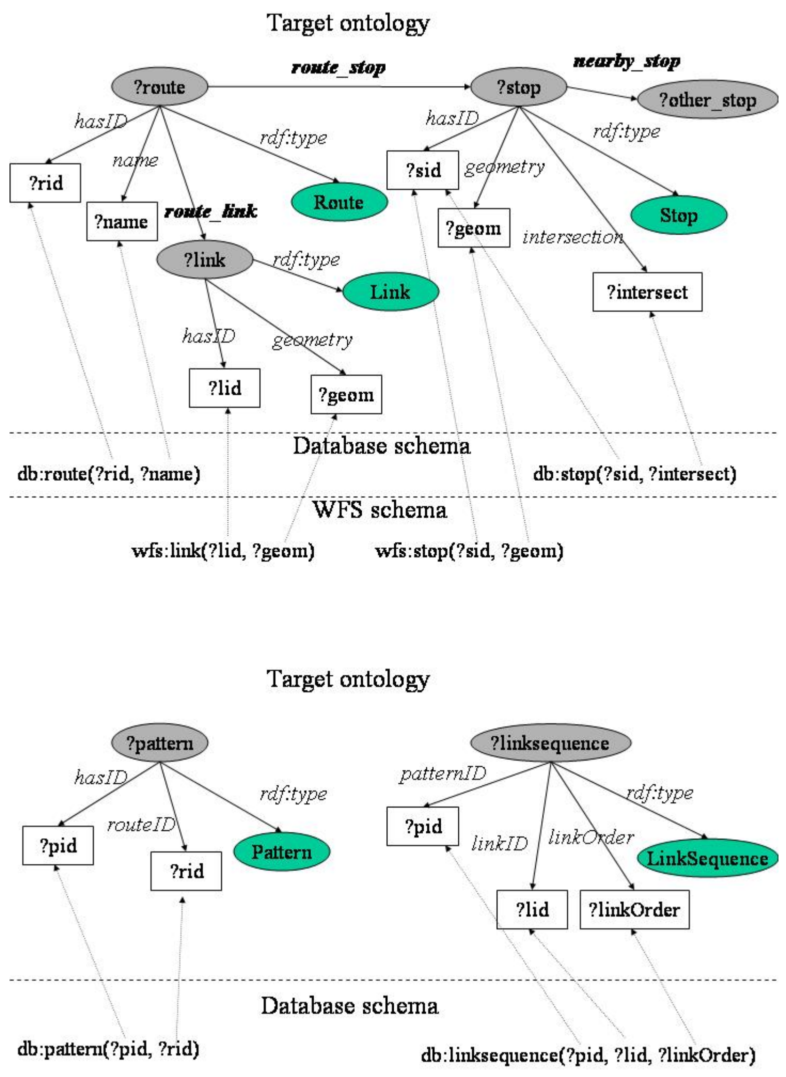


Fig. 3. Schematic mapping from database tables and WFS features to RDF ontology

5.1 Mapping Rules

Definition 1. *A mapping rule from WFS feature or database table to RDF triples has the form of* $p(\overline{X}) : -R(\overline{X}, \overline{Y})$*, where* p *is a predicate corresponding to a WFS feature or a relational table,* R *is a set of RDF triples,* $\overline{X}$ *and* $\overline{Y}$ *are sets of variables.*

Figure 3 illustrates the mapping from relational tables and WFS features to RDF ontology. The mapping rules written in Datalog-like[7] notations are summarized in Figure 4. A mapping rule m has two parts, the left of $:-$ is called the rule head and can be written as $m.head$ and the right of $:-$ is the rule body accessible via $m.body$. The rule head is a predicate corresponding to a WFS feature or database table. For example, `wfs:link(?id, ?geom)` refers to the link feature in a WFS server, where $?id$ and $?geom$ are variables corresponding to the ID and Geometry properties respectively. Note that any string started with ? is a variable. Similarly, `db:Route(?id, ?name)` is a route table in a database where $?id$ and $?name$ are variables representing the ID and name columns of the route.

The rule body is a set of RDF triples written in N3[8] notations, where each triple has the form of `subject predicate object`, and $subject$ and $object$ can be variables that correspond to RDF instances or primitive values such as strings, and $object$ can also be RDF types or constants. The predicate corresponds to the RDF properties such as `hasID` and `geometry`. For each database table and each WFS feature type, we create a corresponding RDF class. The mapping rules map the database tables and WFS feature types to the corresponding RDF classes. Consequently, the RDF triples in a body of a rule always have the same subject. For example, the only subject in Rule (M1) is `?stop` while the subject in (M2) is `?link`.

In our definition, a variable $?x$ appearing in the rule head corresponds to the value of a WFS feature property or a database table cell, while a variable $?y$ only appearing in the rule body but not in rule head corresponds to RDF instances. Since we don't create RDF instances explicitly, the variables such as $?y$ are never materialized.

5.2 Inference rules

The mapping rules connect WFS features and database tables to RDF ontology but only the datatype properties of the ontology are used. RDF ontology is more flexible in that it can define object properties to connect RDF instances. For example, we can define `route_link` property to specify the links contained in a route. This is useful because a route does not have geometry of its own. If we want to display a route on a map based on the route name, we cannot rely on the datatype properties of the `Route` class. Instead, we can use `route_link` property to find out the links in a route with certain name and render the geometries of these links on a vector layer of a map.

To connect object properties such as `route_link` to WFS features and database tables, we use a set of inference rules to derive object properties based on the datatype properties. An inference rule i also written in Datalog-like notations has two parts: the head written as $i.head$ and the body written as $i.body$. The head of the rule has the form of a RDF triple while the body is a set of RDF triples plus some filters.

Definition 2. *An inference rule has the form of* $r(\overline{X}) : -R(\overline{Y}, \overline{Z}), F(\overline{Z})$*, where* r *is a RDF triple,* R *is a set of RDF triples,* F *is an optional set of filters, and* $\overline{X}$*,* $\overline{Y}$*,* $\overline{Z}$ *are sets of variables.* $\overline{X}$ *is a subset of* $\overline{Y}$ *and they contain variables referring to RDF instances.* $\overline{Z}$ *is a set of variables referring to datatype values or geometries.*

[7] http://www.ccs.neu.edu/home/ramsdell/tools/datalog/datalog.html
[8] http://www.w3.org/2000/10/swap/Primer.html

```
(M1) wfs:stop(?geom, ?sid)  :- ?stop rdf:type Stop,
                               ?stop hasID ?sid,
                               ?stop geometry ?geom.
(M2) wfs:link(?geom, ?lid)  :- ?link rdf:type Link,
                               ?link hasID ?lid,
                               ?link geometry ?geom.
(M3) db:stop(?sid, ?intersect)
                            :- ?stop rdf:type Stop,
                               ?stop hasID ?sid,
                               ?stop intersection ?intersect.
(M4) db:route(?rid, ?name)  :- ?route rdf:type Route,
                               ?route hasID ?rid,
                               ?route name ?name.
(M5) db:pattern(?pid, ?rid) :- ?pattern rdf:type Pattern,
                               ?pattern hasID ?pid,
                               ?pattern routeID ?rid.
(M6) db:linksequence(?pid, ?lid, ?linkOrder)
                            :- ?linkseq rdf:type LinkSequence,
                               ?linkseq patternID ?pid,
                               ?linkseq linkID ?lid,
                               ?linkseq linkOrder ?linkOrder.
```

Fig. 4. Mapping rules from WFS features and relational tables to RDF ontology

```
(I1)                                 (I2)
?route route_link ?link :-
   ?route rdf:type Route,
   ?route hasID ?rid,
   ?pattern rdf:type Pattern,       ?stop nearby_stop ?other :-
   ?pattern hasID ?pid,                ?stop rdf:type Stop,
   ?pattern routeID ?rid,              ?other rdf:type Stop,
   ?linkseq rdf:type LinkSequence,     ?stop geometry ?geom,
   ?linkseq hasID ?lid,                ?other geometry ?g,
   ?linkseq patternID ?pid,            filter(
   ?link rdf:type Link                    DWithin(?g, ?geom, 1)
   ?link hasID ?lid.                   ).
```

Fig. 5. Inference rules where `filter(DWithin(?g, ?geom, 1))` is a filter to specify that ?g is within a distance of 1 from ?geom.

Figure 5 shows two inference rules where Rule I1 says that Property `route_stop` relates a route to a link if the route ID matches the route ID of a pattern, the pattern's ID matches the pattern ID of a link-sequence, and the link-sequence's link ID matches the link's ID. The inference rule for `route_stop` can be defined similarly. Note that the filter in Rule I2 is similar to the OGC filters and the rule says that a stop one has a nearby stop two if stop two's geometry is within a distance of 1 from stop one's geometry. Here we omit the unit of distance.

To simplify query rewriting algorithm, we require an object property to appear in the head of at most one inference rule.

6 RDF queries

We use SPARQL[9] – a query language for RDF data to write our spatial queries. The SPARQL queries have to be rewritten to WFS getFeature requests and SQL queries. The queried results can then be displayed or rendered on a map.

Definition 3. *We consider SPARQL queries in the form of*

$$select\ \overline{X}\ where\ R(\overline{Y},\overline{Z}),\ F(\overline{Z})$$

where R is a set of RDF triples, F is an optional set of filters, and $\overline{X}$, $\overline{Y}$, and $\overline{Z}$ are set of variables. $\overline{X}$ is a subset of $\overline{Z}$. $\overline{Y}$ is a set of variables corresponding to RDF instances while $\overline{Z}$ is a set of variables corresponding to datatype values and geometries.

We restrict the set of variables that a SPARQL query selects to be datatype variables or geometries. The reason is that we are mainly interested in the attributes of spatial or non-spatial features, not the virtual RDF instances. We call the set of RDF triples in a SPARQL query q its body and write it as $q.body$.

As an example, suppose we want to find the geometry of a route by the route name "Summit". The SPARQL query can be written as

```
(Q1) select ?geom where ?route rdf:type Route.
                        ?route route_link ?link.
                        ?link geometry ?geom.
                        ?route name ?name.
                        filter(?name = "Summit").
```

Here we use the property `route_link` to find the links of a route with name "Summit". Another example is to find the intersection address of a bus stop based on the x, y coordinates of the stop (-88, 43).

```
(Q2)
select ?intersect where ?stop rdf:type Stop.
                        ?stop intersection ?intersect.
                        ?stop geometry ?geom.
                        filter(DWithin(?geom, (-88,43), 0.1)).
```

To display the nearby stops of a given stop, we can use the query below:

```
(Q3) select ?g where ?stop rdf:type Stop.
                     ?stop geometry ?geom.
                     filter(DWithin(?geom, (-88,43), 0.1)).
                     ?stop nearby_stop ?other.
                     ?other geometry ?g.
```

Note that a user interface can hard-wire some of the queries into the interface so that users do not have to write them explicitly. The queries such as Q2 and Q3 are inconvenient for users to write directly because the geometries of interests are difficult to specify precisely. The interface, however, can rely on mouse click to select stops and then query for the nearby stops or intersection addresses.

[9] http://www.w3.org/TR/rdf-sparql-query/

WFS feature instances

```
   wfs:stop(10, (-88,42))         wfs:link(22, ((-88,42), (-88,43)))
   wfs:stop(11, (-88,43))         wfs:link(23, ((-88,43), (-87,43)))
```

database instances

```
       db:route(25, "Summit")          db:pattern(11, 25)
       db:stop(10, "Main at Oakland")  db:linksequence(11, 22, 1)
       db:stop(11, "Main at Adams")    db:linksequence(11, 23, 2)
```

target RDF instances

```
_stop10 rdf:type     Stop;               _route25 rdf:type Route;
        hasID        10;                          hasID    25;
        intersection "Main at Oakland";           name     "Summit".
        geometry      (-88,42).
                                         _pattern11 rdf:type Pattern;
_stop11 rdf:type     Stop;                          hasID    11;
        hasID        11;                            routeID  25.
        intersection "Main at Adams";
        geometry      (-88,43)           _seq11_1  rdf:type  LinkSequence;
                                                   linkID    22;
_link22 rdf:type Link;                             linkOrder 1;
        hasID    22;                               patternID 11.
        geometry ((-88,42), (-88,43)).
                                         _seq11_2  rdf:type  LinkSequence;
_link23 rdf:type Link;                             linkID    23;
        hasID    23;                               linkOrder 1;
        geometry ((-88,43), (-87,43)).             patternID 11.
```

Fig. 6. WFS/database instances and the corresponding RDF instances written in N3 notations. For example, `_route25` is a node of Route type and it has ID 25 and name "Summit".

6.1 Query semantics

Given a set of RDF views connecting a RDF ontology and a set of WFS and database schemas, we can always convert a set of WFS and database instances – the source instances, to a set of RDF instances – the target instances. The goal here is to allow querying on the target instances without actually creating them.

Thus, the problem we want to solve is

> to find the answers to a SPARQL query Q1 on the RDF ontology by rewriting the query to WFS/SQL queries Q2 so that the answer to Q2 on source instances is the same as the answer to Q1 on target instances.

As an example, consider the WFS and database instances shown in Figure 6, where `wfs:stop(10, (-88,42))` represents a feature instance of bus stop that has ID 10 and point geometry $(-88, 42)$. Similarly, the line geometry of `wfs:link(22, ((-88,42),(-88,43)))` is $((-88, 42), (-88, 43)))$. Also, a database instance written as `db:linksequence(11, 22, 1)` is a link sequence with pattern ID 11, link ID is 22, and link order 1.

We use the mapping rules to create target RDF instances shown in Figure 6. Applying the inference rule I1 and I2 to the sample data, we obtain additional properties for the instances of Route and Stop class as shown in Figure 7. Figure 8 illustrates the

```
_route25 rdf:type    Route;    _stop10 rdf:type     Stop;
         hasID       25;               hasID        10;
         name        "Summit";         intersection "Main at Oakland";
         route_link  _link22;          geometry     (-88,42);
         route_link  _link23;          nearby_stop  _stop11.
                               _stop11 rdf:type     Stop;
                                       hasID        11;
                                       intersection "Main at Adams";
                                       geometry     (-88,43);
                                       nearby_stop  _stop10.
```

Fig. 7. RDF node with properties derived with inference rules.

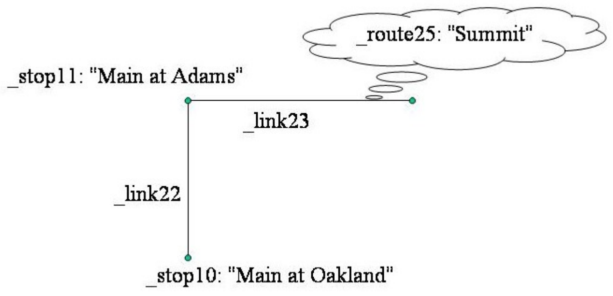


Fig. 8. Route _route25 includes the links _link22, _link23 and the stops _stop10 and _stop11.

relationship between the route nodes, link nodes, and stop nodes. If we directly apply query Q1 to the target instances in Figure 6 and 7, then we can find that a route by the name of "Summit" has two links `_link22` and `_link23`, which have the line geometries of `((-88,42), (-88,43))` and `((-88,43), (-87,43))`. We want to rewrite Q1 to WFS requests and SQL queries so that the same answers are returned from the WFS servers and databases.

6.2 Query Rewriting

The query rewriting algorithms have two parts. The first part shown in Figure 9 applies inference rules to the body of a SPARQL query (*target query*) so that RDF triples with object properties are replaced by RDF triples with datatype properties. An inference rule i is applicable to a triple t if $i.head$ matches t via a variable substitution s such that $s(i.head) = t$.

Definition 4. *A substitution s is a variable mapping from the set V to another set V' such that for each $v \in V$, $s(v) \in V'$. $s(t)$ is t with every variable in t replaced by $t(v)$.*

Note that an inference rule may define an object property in terms of not only datatype properties but also other object properties. This problem can be solved by

```
input: target query q
       a set of inference rules I

for each triple t in q.body
  for each inference rule i in I
     if there exists a substitution s such that s(i.head) = t
     then replace t in q.body with s(i.body)
  end for
end for

output: q' where q'.body has only triples in RDF mapping
```

Fig. 9. Algorithm 1: apply inference rule to SPARQL query.

```
input: target query q
       a set of mapping rules M
initialize: apply algorithm 1 to q to obtain q'
            group triples in q'.body by subject name
            resulting a set of triple groups L

for each triple group t in L
   for each mapping rule m in M
      if there exists a substitution s such that
              s(m.body) contains all triples in t
      then replace t in q' with s(m.head)
   end for
end for

output: q' where q'.body contains database queries and/or
        WFS getFeature requests
```

Fig. 10. Algorithm 2: query rewriting.

recursively applying the inference rules to the query body until no object property remains. We do not to consider this case for simplicity

Also note that the inference step is similar to *backward chaining* technique of automatic deduction used in logic programming systems such as Prolog [33]. However, we do not define recursive inference rules and the head of each inference rule is distinct. The restrictions ensure that Algorithm 1 always terminates and there is no need for backtracking (to try alternative rules of the same head).

The second part shown in Figure 10 applies algorithm 1 to the target query, and then rewrites the resulting query to WFS *getFeature* requests and SQL queries.

For example, we can apply inference rule I1 to the query Q1 to replace `?route route_link ?link.` with the body of Rule I1. Below is the result with some redundant triples removed.

```
(Q1') select ?geom where ?route rdf:type Route.
                         ?route hasID ?rid.
                         ?route name ?name.
```

```
                                filter(?name = "Summit").
                                ?pattern rdf:type Pattern,
                                ?pattern hasID ?pid,
                                ?pattern routeID ?rid,
                                ?linkseq rdf:type LinkSequence,
                                ?linkseq patternID ?pid,
                                ?linkseq hasID ?lid,
                                ?link rdf:type Link
                                ?link hasID ?lid.
                                ?link geometry ?geom.
```

Next, we apply mapping rules to replace the query body with WFS requests and SQL queries. For Query `Q1'`, we apply the set of mapping rules M2, M4, M5, and M6. After substitution, we get query Q1".

```
(Q1'') select ?geom where wfs:link(?geom, ?lid),
                          db:pattern(?pid, ?rid),
                          db:linksequence(?pid, ?lid, ?linkOrder),
                          db:route(?rid, ?name),
                          filter(?name = "Summit").
```

At this point, the query rewriting is complete where the body of Query `Q1''` has a WFS *getFeature* request for the link feature and and a database SQL query for the pattern, linksequence, and route tables.

The order of execution of this query may be significant, however, since it is not efficient to send the WFS *getFeature* request without obtaining link IDs of the route "Summit" first. There could be large number of link features and the resulting GML file returned by the *getFeature* request can be slow to download and parse. This requires us to execute the database query that corresponds to:

```
    db:pattern(?pid, ?rid),
    db:linksequence(?pid, ?lid, ?linkOrder),
    db:route(?rid, ?name),
    filter(?name = "Summit").
```

This fragment can be translated to SQL of the form

```
    select distinct l.lid
    from patterns p, linksequence l, route r
    where p.pid = l.pid and
          r.rid = p.rid and
          r.name = "Summit";
```

The retrieved link IDs are used to send *getFeature* request to the WFS server for the geometries of the feature `link`.

Similarly, Query Q2 can be rewritten as

```
(Q2')
select ?intersect where db:stop(?sid, ?intersect),
                        wfs:stop(?sid, ?geom),
                        filter(DWithin(?geom, (-88,43), 0.1)).
```

In this case, it is more efficient to query WFS feature stop for the stop ID where the geometry of the stop is with distance 0.1 of the point (-88,43). The retrieved stop ID is used to query database for the address of the stop intersection.

Lastly, Query Q3 can be rewritten as

```
(Q3') select ?g where wfs:stop(?id, ?geom),
                      filter(DWithin(?geom, (-88,43), 0.1)),
                      wfs:stop(?sid, ?g),
                      filter(DWithin(?g, ?geom, 1)).
```

In this case, two WFS requests are needed. The first request finds the geometry $?geom$ of a stop around the point (-88,43) and the second request finds all the stops that are within the distance of 1 from $?geom$. The geometries of all requested stops are returned.

Note that the execution of WFS and database queries may be more efficient if at least one variable of each WFS query is given values by the previous queries or such variable is constrained by some OGC filters. This is somewhat related to query rewriting in the context of data integration where data sources may have restrictions on possible access path to data [20].

Also note that in general, query rewriting using views should consider whether the rewritings are equivalent or maximally contained and whether the views are assumed to be complete or not [20]. In our case, the views (database or WFS schemas) on RDF schemas are complete and we seek equivalent rewriting. The views are relatively simple since there can be at most one database view and one WFS view on each RDF class. Because of these restrictions, each database table or WFS feature type has unique mapping in a RDF class. So if a RDF ontology R is created from a database and a WFS based on the above mapping and inference rules, and Q' is a rewriting of Q using Algorithm 2, then Q' should always produce the same result as executing Q on R. Also, though the complexity for query rewriting when queries and views are expressed as conjunctive queries is NP-complete [25], our algorithm is polynomial time because of the restrictions on views.

6.3 Semantic constraints

RDF schema[10] includes some constructs of semantic constraints to assist ontology reasoning. The constructs include `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:domain`, and `rdfs:range`, etc. These restrictions can be useful in stating queries. For example, a RDF query for the instances of `SpatialFeature` class that are within a certain bounding box can be automatically translated to queries for instances of `Stop` and `Link` since they are both subclasses of `SpatialFeature`.

In fact, the ontology defined in Figure 1 and 2 already contains some semantic constraints. The class `SpatialFeature` is a subclass of `Feature`, while the former has the subclasses `Link` and `Stop`, and the latter has the subclasses `Route`, `Pattern`, `LinkSequence`, etc. Also, we specified the domain and range of the property `route_link` to be `Route` and `Link`.

Because of these constraints some queries can be simplified. For example, Query `Q1` does not need to say that the type of `?route` is `Route` because the domain of `route_link` already has that restriction.

```
simplified (Q1)

select ?geom where ?route route_link ?link.
```

[10] http://www.w3.org/TR/rdf-schema/

```
                       ?link geometry ?geom.
                       ?route name ?name.
                       filter(?name = "Summit").
```

Nothing needs to be changed in the algorithm for this example because the inference rule for `route_link` will add triples to specify the types of `?route` and `?link`.

In general, however, we need to include more inference rules to incorporate the use of semantic constraints. That is, we need to include inference rules to consider subclass and subproperty relations and the domain and range restrictions. We need to modify Algorithm 1 so that for each triple of the form `?s rdfs:type SpatialFeature`, we add additional triples `?s rdfs:type Stop` and `?s rdfs:type Link` since `Stop` and `Link` are subclasses of `SpatialFeature`. Similar treatment should be applied to `Feature`. Also, for any triple `?s p ?o`, if the property `p` has domain and range constraints, we should add type restrictions for `?s` and `?o`; if `p` has a subproperty `p'`, then we should add another triple `?s p' ?o`. Algorithm 2 remains the same.

More semantic constraints can be found in another ontology language OWL[11], an extension of RDF. For example, classes and properties in OWL can be declared as synonyms using `owl:sameAs`. Note that OWL-DL – a subset of OWL is based on description logics. Query rewriting with views is more difficult when the views are expressed in description logics and conjunctive queries over description logics. Query rewriting may be possible with some restrictions on the views [4]. Also, when description logics are combined with inference rules, the problem may be undecidable. Some restricted systems have decidable algorithms but they do not deal with query rewriting. Examples include $\mathcal{AL}$-Log [14], which is an integrated system for knowledge representation based on description logics and Datalog, and Motik *et.al.* [26] proposed a decidable combination of OWL-DL with function-free Horn rules.

7 Implementation issues

Though the query rewriting algorithm can translate SPARQL queries to WFS requests and SQL queries, it is easier to do this using an existing application – D2R Server [7] that can create virtual RDF graphs from one or more relational databases. With D2R server, we no longer need to send SQL queries to databases directly. In fact, we can modify the Algorithm 2 slightly to make it work with D2R Server. D2R Server accepts SPARQL queries and through a set of mapping rules similar to what we have described, it translates the SPARQL queries to database queries and gets results back as RDF triples. So we first rewrite some RDF triples in the body of initial query to WFS getFeature requests. For example, Query Q1 can be rewritten to

```
select ?geom where ?route rdf:type Route.
                   ?route route_link ?link.
                   ?route name ?name.
                   filter(?name = "Summit").
                   ?link  hasID ?lid.
                   wfs:link(?lid, ?geom).
```

[11] http://www.w3.org/TR/owl-features/

Since we are interested in the link IDs of the route "Summit", we can send the following SPARQL query to the D2R Server:

```
(Q4) select ?lid where ?route rdf:type Route.
                       ?route route_link ?link.
                       ?route name ?name.
                       filter(?name = "Summit").
                       ?link  hasID ?lid.
```

The above query retrieves link IDs from database and we assume that the D2R Server has the same RDF ontology, which includes the RDF property `route_link`.

```
<wfs:GetFeature service="WFS" version="1.0.0"
  xmlns:wfs="http://www.opengis.net/wfs"
  xmlns:ogc="http://www.opengis.net/ogc" ...
  <wfs:Query typeName="wfs:link">
    <ogc:Filter>
       <OR>
          <PropertyIsEqualTo>
               <PropertyName>id</PropertyName>
               <Literal>22</Literal>
          </PropertyIsEqualTo>
          ...
       </OR>
    </ogc:Filter>
  </wfs:Query>
</wfs:GetFeature>
```

Fig. 11. A WFS GetFeature request to retrieve instances of feature Link with ID filters

After obtaining the link IDs, a getFeature request similar to Figure 11 can be sent to WFS server to retrieve the geometries of the links. Notice that Query Q4 has an additional triple in its body `?link hasID ?lid` to retrieve link IDs. This can be generalized because the WFS features can be identified by their IDs. So in general, if the initial query involves a WFS feature, we require both the WFS getFeature request and the SPARQL query to D2R server to return IDs of the feature. If a WFS request is sent first, then the resulting feature IDs are used in the filters of the SPARQL query to D2R server. The architecture of the RDF interface is shown in Figure 12.

A sample demonstration of the interface[12] is shown in Figure 13, which shows Waukesha county's route links and streets as a WFS and a WMS layer, and Figure 14, which shows the links and bus stops of the route "Summit" retrieved from a WFS server using the link IDs and stop IDs retrieved from a D2R server. Also, as shown in Figure 14, users can click on a bus geometry (highlighted) and retrieve the intersection "SUMMIT AVE at SYLVAN TER" associated with the stop.

[12] http://jiangxi.cs.uwm.edu:8080/waukesha/gis.html

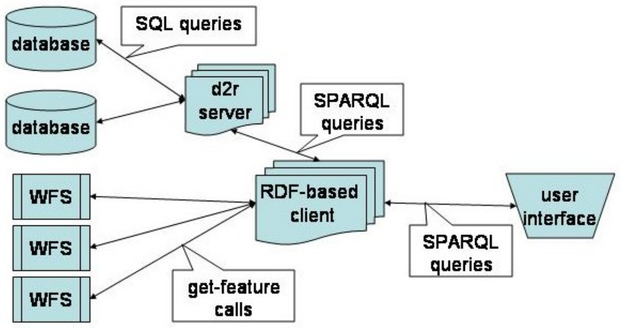


Fig. 12. RDF interface architecture

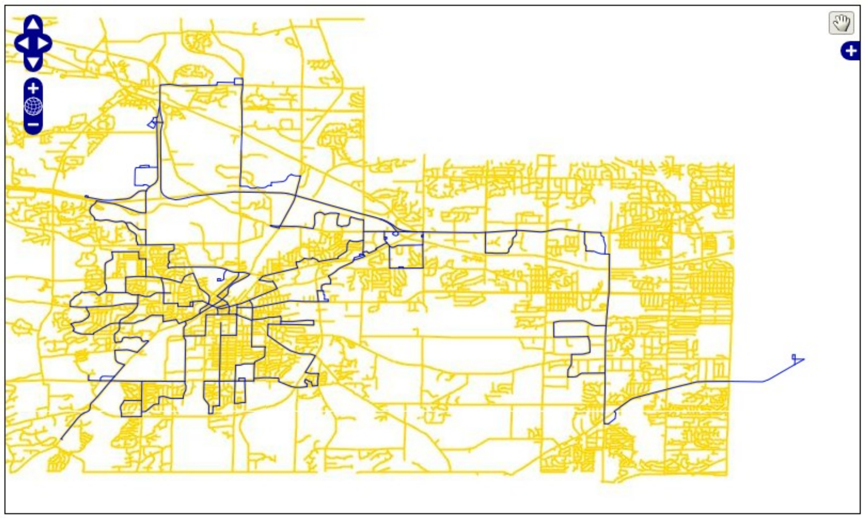
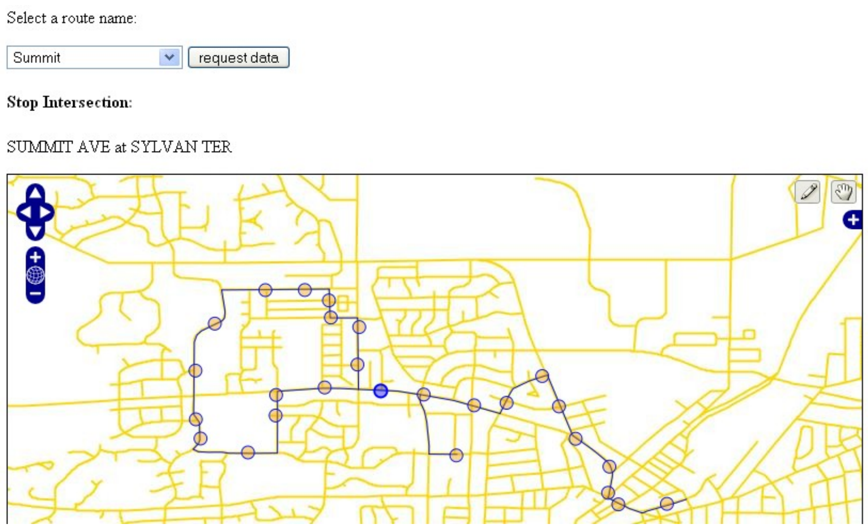


Fig. 13. Transit routes of Waukesha county

8 Conclusion and Future work

We have presented a new method to provide integrated access to distributed geospatial data using RDF ontology and query rewriting. This method is more efficient than converting all data to ontology instances because it avoids the costs and consistency problems of data replication. Also, ontology interface can still use existing tools for conducting spatial query on geometries and relational queries on non-spatial data, and for rendering spatial features encoded in GML. Using the proposed method, user queries can be more straightforward because the application ontology can encode data semantics not available in WFS feature types or database schemas.

Our method uses RDF ontology but since RDF is a subset of OWL, the method can


Select a route name: Summit

Stop Intersection:

SUMMIT AVE at SYLVAN TER

Fig. 14. Route “Summit” and its stops

be directly applied to OWL ontology though extension may be needed to take advantage of the semantic constraints of OWL. OWL has more semantic constructs such as class equivalence/intersection/union, transitive/reflexive object properties, and the restrictions of universal/existential quantification on object properties. Our query rewriting algorithm needs to be extended to handle semantic constraints encoded with these constructs. However, it may be possible to use some existing tools such as Jena in this extension.

To evaluate the effectiveness of our method, we may need to implement a more complete ontology interface where user can query any classes and properties defined in domain and application ontology. The evaluation criteria may include the expressiveness of the interface – how many kinds of queries the interface can handle. The criteria should also include the efficiency of the interface – how fast the results are returned in comparison to the alternative ways of data integration such as data warehousing approach where all data are converted to ontology instances. Finally, we may want to evaluate the flexibility of the method – how easy it is to combine this method with other GIS applications. For example, we can provide this ontology interface as a web service where other web-based applications can use it as a source of querying and rendering geospatial data.

As future work, we plan to implement the query rewriting algorithms to accept arbitrary SPARQL queries. Also, it may be useful to have ontology browser to display instances of ontology classes, thus users can navigate to other instances following links of object properties. The browser could be implemented on top of the query rewriting module. However, it is not clear how the contents of the browser should be presented since GML data of geometries is clearly not interesting while rendering all returned spatial instances could be very inefficient.

9 Acknowledgments

We thank the anonymous reviewers for their valuable comments. This research was partially supported by NSF award BCS-0616957.

References

1. I Budak Arpinar, Amit Sheth, Cartic Ramakrishnan, E Lynn Usery, Molly Azami, and Mei-Po Kwan. Geospatial ontology development and semantic analytics. *Transactions in GIS*, 10(4):551–575, 2006.
2. Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Application*. Cambridge University Press, New York, NY, USA, 2003.
3. Miriam Baglioni, Maria V. Masserotti, Chiara Renso, and Laura Spinsanti. Building geospatial ontologies from geographical databases. In *Proceedings of GeoSpatial Semantics, the Second International Conference*, pages 195–209, 2007.
4. Catriel Beeri, Alon Y. Levy, and Marie-Christine Rousset. Rewriting queries using views in description logics. In *PODS '97: Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 99–108, 1997.
5. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, pages 34–43, 2001.
6. Y. Bishr. Overcoming the semantic and other barriers to GIS interoperability. *International Journal of Geographical Information Science*, 12(4):299–314, 1998.
7. Christian Bizer and Andy Seaborne. D2RQ -treating Non-RDF databases as virtual RDF graphs. In *Proceedings of the 3rd International Semantic Web Conference (ISWC2004)*, 2004.
8. Huajun Chen, Zhaohui Wu, Heng Wang, and Yuxin Mao. RDF/RDFS-based relational database integration. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, page 94, 2006.
9. J. Choichi. Constaint-based interoperability of spatiotemporal databases. *Geoinformatica*, 3(3):211–243, 1999.
10. I. F. Cruz, W. Sunna, and A. Chaudhry. Semi-automatic ontology alignment for geospatial data integration. In *Proceedings of the 3rd International Conference, GIScience*, pages 51–66, October 2004.
11. OGC document 02-023r4. Geography Markup Language (GML), version 3.00, 2003.
12. OGC document 04-024. Web Map Service, Version 1.3, 2004.
13. OGC document 04-094. Web Feature Service, Version 1.1.0, 2005.
14. Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. AL-log: Integrating datalog and description logics. *Journal of Intelligent Information Systems*, 10(3):227–252, 1998.
15. M. Egenhofer. Toward the Semantic Geospatial Web. In *Tenth ACM International Symposium on Advances in Geographic Information Systems*, pages 1–4, 2002.
16. S.I. Fabrikant and B.P. Buttenfield. Formalizing semantic spaces for information access. *Annals of the Association of American Geographers*, 91, 2001.
17. FGDC. National Spatial Data Infrastructure, 2007.
18. F. Fonseca, M. Egenhofer, P. Agouris, and G. Camara. Using ontologies for integrated geographic information systems. *Transactions in GIS*, 6(3):231–257, 2002.
19. G. Hobona G, D. Fairbairn, and P. James. Semantically-assisted geospatial workflow design. In *Proceedings of 15th ACM International Symposium on Advances in Geographic Information Systems*, pages 194–201, November 2007.
20. Alon Y. Halevy. Theory of answering queries using views. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 29(4):40–47, 2000.

21. Alon Y. Halevy. Answering queries using views: A survey. *VLDB Journal: Very Large Data Bases*, 10(4):270–294, 2001.
22. C.B. Jones, A.I. Abdelmoty, D. Finch, G. Fu, and S. Vaid. The SPIRIT spatial search engine: architecture, ontologies and spatial indexing. In *Proceedings of the 3rd International Conference, GIScience*, pages 25–139, October 2004.
23. Eva Klien. A rule-Based strategy for the semantic annotation of geodata. *Transactions in GIS*, 11(3):437C452, 2007.
24. Maurizio Lenzerini. Data integration: a theoretical perspective. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246, 2002.
25. Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proceedings of the 14th ACM Symposium on Principles of Database Systems*, pages 95–104, 1995.
26. Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for owl-dl with rules. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):41–60, JUL 2005.
27. Manoj Paul and S. K. Ghosh. An approach for service oriented discovery and retrieval of spatial data. In *Proceedings of the 2006 international workshop on Service-oriented software engineering*, pages 88–94, 2006.
28. Z.-R. Peng. A proposed framework for featurelevel geospatial data sharing: A case study for transportation network data. *International Journal of Geographic Information Sciences*, 19(4):459–481, 2005.
29. Z.-R. Peng and C. Zhang. The roles of geography markup language, scalable vector graphics, and web feature service specifications in the development of internet geographic information systems. *Journal of Geographical Systems*, 6(2):95–116, 2004.
30. Florian Probst. Ontological analysis of observations and measurements. In *Proceedings of the 4th International Conference, GIScience*, pages 304–320, 2006.
31. H. Pundt and Y. Bishr. Domain ontologies for data sharing - an example from environmental monitoring using field GIS. *Computers and Geosciences*, 28(1):95–102, 2002.
32. B. Smith and D. Mark. Geographic categories: An ontological investigation. *International Journal of Geographic Information Science*, 15(7):591–612, 1998.
33. Leon Sterling and Ehud Shapiro. *The Art of Prolog*. MIT Press, 1994.
34. C. Zhang and W. Li. The roles of Web Feature and Web Map Services in real time geospatial data sharing for time-critical applications. *Cartography and Geographic Information Science*, 32(4):269–283, 2005.
35. C. Zhang, W. Li, Z.-R. Peng, and M. Day. GML-based interoperable geographical database. *Cartography*, 32(2):1–16, 2003.
36. C. Zhang, W. Li, and T. Zhao. Geospatial data sharing based on geospatial semantic web technologies. *Journal of Spatial Science*, 52(2):35–49, 2007.