Accessing Distributed WFS Data Through A RDF Query Interface

Tian Zhao[1], Chuanrong Zhang[2], and Weidong Li[2]

[1] University of Wisconsin – Milwaukee
[2] University of Connecticut

Abstract. Geospatial data stored in databases and other formats can be accessed through Web Feature Service (WFS). However, it is not convenient to access data in multiple WFS servers since WFS protocol is geared towards single server. In this paper, we propose an algorithm to query and synthesize distributed WFS data through a RDF query interface, where users can specify data requests to multiple WFS servers using a single RDF query. The algorithm translates each RDF query written in SPARQL-like syntax to multiple WFS get-feature requests, and then convert the WFS results to answers to the original query. A lightweight Web-based prototype is implemented based on this approach.

1 Introduction

In this paper, we propose the design of a RDF query interface for distributed WFS data, which accepts queries in SPARQL-like syntax to provide more flexibility and usability than direct WFS queries.

As a motivational example, imagine a scenario where a developer needs to implement a program to display the flooded streets near the high schools of a city. The programmers can define an interface backed by some predefined WFS queries to fetch data from two servers. For each user request, the interface will request data from the servers and then integrate the results. If the servers use different data definitions, a translation step is needed to reconcile and integrate the data. If the developer needs to support another query such as finding the bridges of major highways, the developer has to perform the above steps again.

While the implementation of two queries shares many similarities, it is not apparent how to re-factor the duplicated code, which include the communication with WFS servers and the transformation and integration of the responses. The difficulty is not with building shared library but with composing specific WFS requests and interpreting and integrating the corresponding results, which may be different from each query. In addition, while queries may involve the same intermediate data, it is not straightforward to implement a caching strategy to improve performance. This is especially critical during emergency when peak data requests can overwhelm data servers.

Our design of RDF query interface aims to improve the productivity for rapid prototyping of WFS query applications. The RDF interface automatically translates user queries formulated in a SPARQL-like syntax to WFS requests sent to multiple servers and then integrates WFS response to answer the original user queries. Using this design, application developers do not need to write code for WFS request and data processing. Instead, they can accomplish the same goal by defining mappings from WFS feature types to RDF definitions and by writing RDF queries.

Note that the data provided by WFS servers may be backed by databases or shapefiles. The cost of converting the data to a uniform format may not be feasible for large or frequently updated data sets.

2 Related Work

In literature, ontology has been used in search tools to help to discover geospatial web services related to certain domain concepts [2]. Tools have also been developed to convert geospatial ontology data to forms that can be accessed via WFS protocol [1]. Given the abundance of data available from geospatial web services and databases, a more interesting direction is to make data from geospatial web services and databases accessible via RDF protocols.

The closest study is the work of Tschirner et al. [3], who proposed a method to convert GML data into ontology data by translating SPARQL queries into WFS requests. Their approach maps a SPARQL query to a WFS request that returns a superset of intended results, transforms the WFS results into ontology data, and then applies the original SPARQL query to obtain the final answer. While this paper shares similar workflow, our approach is different in several ways. Firstly, we do not assume the WFS data is centralized in one server or having a unified definition. This requires the translated WFS requests be separated for each feature type and the final join is done at the client side. We generate multiple OGC filter encoding for each SPARQL query. Secondly, our approach implements a light-weight Web client with roughly 1000 lines of JavaScript without library dependencies for query processing, which is easier for deployment. Lastly, our approach is designed to use a SPARQL-like syntax to bring more convenient query interface to WFS services while Tschirner et al. provide a more complete service of SPARQL endpoint for GML data.

In our prior work [4], we proposed a query rewriting algorithm to translate SPARQL query to WFS requests and database queries using idealized syntax. This paper extends that approach by considering a more realistic subset of SPARQL syntax and by implementing a Web-based prototype that incorporates caching optimization and data rendering.

3 Translate RDF Query to WFS Requests

Typically, a SPARQL query is translated to multiple WFS requests by grouping triples related to the same feature type together so that there is one WFS request per feature type. Any remaining triples are translated to spatial joins to be applied to the results of the WFS requests.

As a concrete example, the below query Q1 is an RDF query in a SPARQL-like syntax for retrieving the streets nearby each high school in New Haven, CT, where a geometry is *nearby* another one if their distance is less than 500 meters (this distance is arbitrarily chosen and can be modified).

```
select  ?s     ?p          where                              (Q1)
        ?s     rdf:type       streets.
        ?p     nh:category     "High School".
        ?s     nearby          ?p
```

In the query, identifiers starting with *?* are variables. The solutions to the variables *?s* and *?p* (which stand for streets and points respectively) between *select* and *where* are the intended results of the query. The lines after *where* are called triples that specify the conditions with which the variable solutions must satisfy. Each triple has the form of *subject predicate object*, which restricts the relation between the subject and object with the predicate. For example, the triple *?s rdf:type streets* says that the solution to ?s must has a type called *streets*. The triple *?p nh:category "High School"* specifies that the category of *?p* is *High School*. Finally, the triple *?s nearby ?p* relates the spatial attributes of *?s* to those of *?p* by the distances between them. Note that users can change the RDF definitions by editing a configuration file.

Pre-processing The pre-processing phase separates the triples into four groups based on the triple predicates. For example, the triples in Query Q1 can be separated as below, where *?p nh:category ?c* and *?c == "High School"* are automatically generated from *?p nh:category "High School"*.

type triple	*?s rdf:type streets.*
property triple	*?p nh:category ?c.*
filter triple	*?c == "High School".*
spatial join triple	*?s nearby ?p*

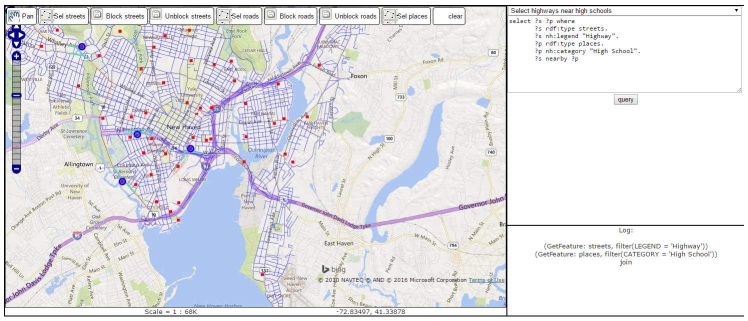
Query rewriting Based on the separated triples, the rewriting phase includes five steps: (1) identify the set of feature variables that correspond to feature types; (2) find the set of triples related to each feature variable; (3) find the set of feature types for each feature variable; (4) find the set of filter expressions for each pair of feature variable and its type; (5) construct a set of WFS get-feature requests for each feature variable.

For the query Q1, the feature variables are ?s and ?p and their feature types are `new_haven_streets` and `new_haven_places` respectively. The following get-feature requests can be generated.

`getFeature`(*new_haven_streets*)
`getFeature`(*new_haven_places*, CATEGORY *PropertyIsEqualTo* 'High School')

Post-processing After receiving the responses from the get-feature requests, we perform spatial joins on the retrieved geospatial features if necessary. The results of the spatial join are filtered by the types of the features identified from the selection variables of the SPARQL query. The features retrieved from the get-feature requests are cached, which greatly improves performance by avoiding network and server overhead.

Implementation Figure 1 is a screen shot of our prototype, which is available at `boyang.cs.uwm.edu:8080/newHaven/bing.html`. It parses each SPARQL query in text form and generates a set of get-feature requests, which are sent as AJAX calls to WFS servers and the responses are joined before being displayed on a map. There are some pre-defined queries though they are not hard-wired in any way and users can revise the query in the textbox. The query interface actions are logged below the textbox so that user can track the query progress.



```
select ?s ?p where
       ?s rdf:type streets.
       ?s nh:legend "Highway".
       ?p rdf:type places.
       ?p nh:category "High School".
       ?s nearby ?p
```

```
Log:

(GetFeature: streets, filter(LEGEND = 'Highway'))
(GetFeature: places, filter(CATEGORY = 'High School'))
join
```

Fig. 1. Highways near high schools

4 Conclusion

In this paper, we present an algorithm to convert RDF queries to WFS requests so that users can query distributed WFS features as if they were RDF instances. The algorithm avoids the cost of converting features to RDF objects while retaining the benefits of RDF queries. As future work, we will extend the algorithm to include static checking capability to detect semantic errors before runtime.

References

1. J. Jones, W. Kuhn, C. Keler, and S. Scheider. Making the web of data available via web feature services. In *AGILE 2014*, 2014.
2. W. Li, C. Yang, D. Nebert, R. Raskinc, P. Houser, H. Wu, and Li Z. A semantic-based web service discovery and chaining for building an arctic spatial data infrastructure. *Computers & Geosciences*, 37:1752–1762, 2011.
3. Sven Tschirner, Ansgar Scherp, and Steffen Staab. Semantic access to inspire how to publish and query advanced gml data. In *Workshop in Conjunction of 10th International Semantic Web Conference*, 2011.
4. T. Zhao, C. Zhang, M. Wei, and Z.-R Peng. Ontology-based geospatial data query and integration. In *Lecture Notes in Computer Science LNCS5266: Geographic Information Science, 5266*, pages 370–392, 2008.