OPEN SOURCE AND OPEN STANDARDS: TOOLS FOR RAPID DEVELOPMENT OF COMMUNITY-ORIENTED GIS

Sally E. Goldin[1], Kurt T. Rudahl[2] and Ploypailin Intapong[3]
[1]KMUTT Geospatial Engineering and Innovation Center (kGeo), 126 Pracha Uthit Road, Bangkok 10140 Thailand
[2]Department of Computer Engineering, King Mongkut's University of Technology Thonburi (KMUTT), 126 Pracha Uthit Road, Bangkok 10140 Thailand,
[3]Software and Computing Innovation Center (Innosoft), KMUTT, 126 Pracha Uthit Rd., Bangkok 10140 Thailand,
[1]seg@goldin-rudahl.com, [2]rudahl@rsgis.net, [3]in.ploypailin@gmail.com

KEY WORDS: Geographic Information Systems, Open Source, Open Standards, Rapid Application Development, Community GIS

ABSTRACT: This paper introduces a stack of three interrelated software tools that implement OGC standards to provide the primary required layers for web-based GIS applications. PostGIS is a sophisticated spatial-relational data base management system which supports a rich set of spatial object types and offers extensive capabilities for manipulating those objects, including queries about spatial relationships, overlays, buffer construction and feature matching. GeoServer is a cross-platform geospatial application server that can manage multiple geographic data sources, both raster and vector, in a wide range of formats. GeoServer abstracts the details of these sources while making their information available to client applications through OGC standard Web Map Service (WMS) and Web Feature Service (WFS) requests. OpenLayers is a JavaScript library for creating and controlling dynamic geographic data displays on a web browser. OpenLayers makes it relatively easy to create WMS and WFS requests and to handle the information returned. Together, these three tools simplify the process of building a web-based GIS application, requiring only commonly-available programming skills (HTML, CSS, JavaScript and SQL).

Our paper starts with a mini-tutorial on these tools. Then, as a case study, we describe our own experience using them to implement a web GIS system for community management of water resources in central Thailand. Our goal is to raise awareness and use of these tools so that organizations with restricted budgets and knowledge can benefit from recent dramatic advances in geospatial information technology.

1. INTRODUCTION

The benefits that can be derived from geospatial information and analysis systems have been widely documented (Seiber, 2006). Unfortunately, many organizations cannot realize these benefits due to the cost and complexity of commercial GIS software. This problem is especially acute for community and local groups who lack funds as well as GIS programming expertise. Available open source tools offer a partial solution.

In this paper, we provide a high-level introduction to three open source software tools that can be used to facilitate the development of web-based GIS systems: PostGIS, Geoserver and OpenLayers. Although these tools come from three different organizations, they work together very well because all three implement standard protocols for the querying and retrieval of raster and vector geospatial information. Our goal in this paper is to help readers understand the capabilities of these tools and their role in a web-based spatial information system. Any web application developer with an intermediate level of experience should possess the requisite knowledge for using these tools. Detailed understanding of GIS operations or representations is not required.

We begin with a quick description of the Web Map Service and Web Feature Service, two standard GIS communication protocols. Next we offer a quick overview of the typical architecture for a system that uses our three-tool stack. Then we examine each of the software tools, surveying its functionality and providing some examples to show how a programmer would work with the tool. Finally, we briefly describe our own experience using these tools to implement a web GIS system for community management of water resources in Thailand.

2. OGC STANDARDS FOR SPATIAL DATA QUERY AND RETRIEVAL

The Open Geospatial Consortium (http://www.opengeospatial.org) is responsible for developing, maintaining and promulgating standard data formats and protocols for geospatial computing. Although OGC currently manages more than thirty standards, two of the most important and influential are the Web Map Service (WMS) and Web Feature Service (WFS).

WMS is a protocol for specifying and requesting spatial data that will be returned from a spatial data source as a raster layer. WMS can be used to retrieve data normally stored in raster form (e.g. satellite imagery) or to assemble

a map from data stored as vectors (e.g. a map of land use polygons). The protocol provides extensive control and selectivity in terms of what is returned and in what format.

WFS is a protocol for specifying and requesting spatial data that will be returned as a set of coordinate-based vector features. Like WMS it allows selection of features based on attributes or spatial relationships.

Both WFS and WMS use XML-based schemas to represent queries and returned data. Since XML is explicit, self-describing, and text-based, it provides a universal medium that can be used on any computer or operating system. WFS and WMS make no assumptions about the underlying systems or representations used for storing the data. Indeed, these protocols allow a single query to be satisfied by data retrieved from multiple sources.

WFS, WMS and related standards have greatly improved the interoperability of different GIS systems and moved us closer to a vision of a universally accessible spatial web (OGC, 2003). However, these standards are very complicated. Extensive expertise is needed to build a system that complies with these standards. Fortunately, a number of groups have taken on this task and supplied the results of their work as free, open source software.

3. A TOOL STACK FOR WEB GIS

Computer engineering uses the term "stack" to describe a set of software components arranged in layers. Each component in the stack interacts only with the components above and below it. This architectural approach has been shown to improve robustness and flexibility in software systems design.

In our work we have used a stack that includes three spatially-oriented open source tools (PostGIS, Geoserver and OpenLayers), plus three general tools (the Apache web server, the Jetty servlet engine and the PostgreSQL relational database system). Figure 1 shows how these layers are organized and how they interact.

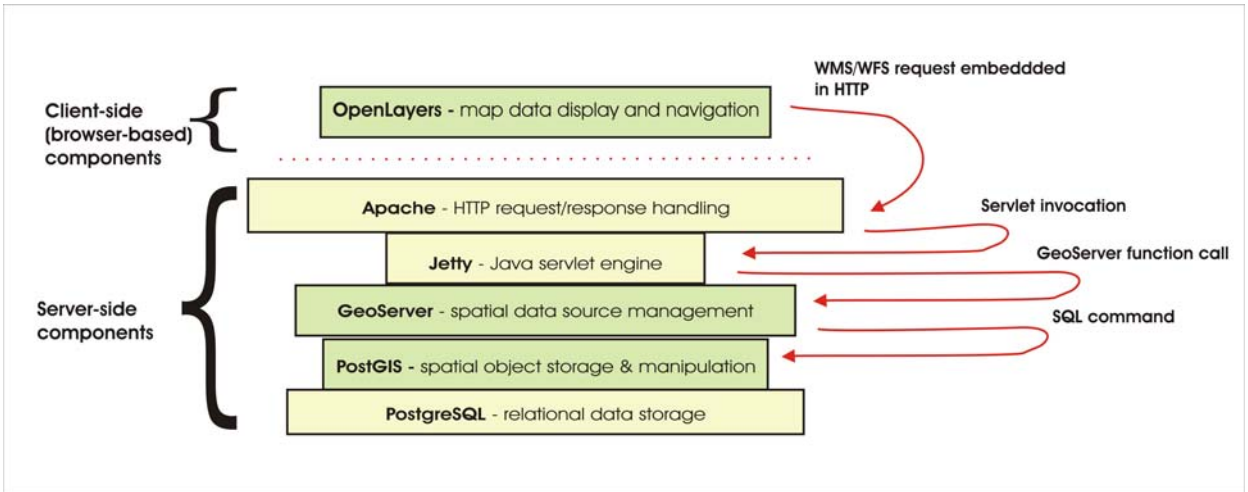

Figure 1: Layers in our web GIS tool stack. Green components offer geospatial capabilities.

For simplicity, the figure shows only the communications that occur when the browser makes a geospatially-relevant request via OpenLayers. The request is transmitted down the stack, transformed into the appropriate type of message at each step. When the browser makes its request, the specifications for the desired spatial data are embedded in a standard HTTP request, which is sent to the Apache web server. Based on configuration parameters set up when we install GeoServer, Apache recognizes that the request should be forwarded to GeoServer. The servlet engine (Jetty in our case) is a manager that can run multiple Java-based web services. It invokes the necessary GeoServer functions. GeoServer consults its list of available geodata sources and determines that the request can be satisfied by data in one or more PostGIS tables. Once the data have been retrieved, they are passed back up the stack in a similar manner.

This very brief description is an oversimplification, of course. The important point to note is that each layer has a well-defined set of services it can provide to the layer above. Furthermore, in many cases the specific component implementing a layer could be replaced with a different component, as long as that the new component provided the same services using the same standard interfaces. For instance, the Jetty servlet engine we have used could be replaced by Tomcat. PostGIS could be replaced by a commercial spatial relational database such as Oracle Spatial. This illustrates the power of well-defined open standards.

4. POSTGIS CAPABILITIES

PostGIS (http://postgis.net) (Obe and Hsu, 2011) is a powerful extension for PostgreSQL (http://www.postgresql.org) that supports storage, retrieval and manipulation of geographic objects. It has been available since 2001. PostGIS offers a broad range of functionality including:

- Definition, storage and retrieval of a wide range of geographic object types (2D and 3D points, polylines, polygons, collections of these three basic types, curves, spheroids, etc.) in special *geometry* columns of normal relational data base tables;
- Access to geographic and non-geographic information via standard SQL;
- Re-projections and on-the-fly coordinate transformations from one spatial reference system to another;
- More than one hundred functions to efficiently compute both simple and complex spatial relationships and quantities;
- Sophisticated functions for importing and exporting spatial data, including from shape files;
- Implementation of most OGC standards for spatial relational data base management.

The functions provided by PostGIS operate in the context of SQL commands to compute distances, create buffers, do polygon overlays and intersections, match features based on spatial similarity, and much more. In fact, PostGIS provides most of the computational capabilities of a typical vector GIS.

Figure 2 offers an example. Note that this figure and the other sample code in this paper are intended to illustrate the relatively small amount of work required (in terms of code volume) to perform sophisticated operations. It is not necessary that the reader understand the details.

In our project, described in section 7, we used the command in Figure 2 when we received a new set of water feature data with the same geographic information as our original data, but with different feature identifiers. The command compares two tables of water features based on the coordinates of their geometries. If the shapes match, it saves the identifier from the ***water*** table in ***foreignid*** column of the ***newwater*** table, so that we could transfer attribute information from ***newwater*** to the correct record of ***water***.

```
update newwater set foreigngid = water.gid from water where
         ST_HausdorffDistance(newwater.geometry,water.geometry) = 0
         and water.subdistrict=newwater.subdistrict;
```

Figure 2: Sample PostGIS command

ST_HausdorffDistance computes the Hausdorff distance (Rockafellar and Wets, 2005) between two features. This is one of half a dozen distance-related functions provided by PostGIS. In our case, we set the cutoff to zero because we are seeking features with identical geometry. The second search criterion, matching associated subdistricts, is a check to discover if we have any data that are duplicates or in error.

Figure 3 shows how we can create and load a new spatial table from a shape file in a single operation, using the ***shp2pgsql*** utility. The spatial information will be stored in a column called ***geometry*** in a new table called ***water***, to be added to the database called ***amphawa***. The **-s** parameter defines the spatial reference system for the table (in this case, UTM zone 47N).

```
shp2pgsql -g geometry -s EPSG:32647 newhydro.shp water | psql -d amphawa
```

Figure 3: Creating a new PostGIS table from a shape file

Various graphical user interfaces are available for use with PostGIS, for people who are not comfortable typing commands.

5. GEOSERVER CAPABILITIES

GeoServer (http://geoserver.org) is a Java-based spatial application server. It provides many functions for delivering spatial information to web applications. GeoServer has been under active development since 2001. GeoServer uses standard access protocols defined by the Open Geospatial Consortium for querying and updating spatial information.

As shown in Figure 1, GeoServer sits between the web server and various sources of spatial data. It interprets WFS and WMS requests and uses the information about its registered data sources to satisfy these requests. One GeoServer instance can manage and access many different sources of data, stored in a variety of different formats: GeoTIFF, raw binary raster with text metadata file, ArcGrid, shapefiles, PostGIS tables, Oracle spatial tables, and other WFS or WMS servers elsewhere on the web. GeoServer provides a web-based control panel for setting up and publishing data sources. Thus no programming is required in order to use its facilities. The ability to respond correctly to WMS and WFS queries is available "out of the box", as soon as GeoServer has been installed and some data sources have been defined.

Other important capabilities provided by GeoServer include:

- *Layer preview capability:* GeoServer comes with a simple OpenLayers-based viewer that allows point-and-click previews of the layers in each data source, along with display of attributes for vector-type sources. This functionality allows programmers to check the data in the spatial database long before the web-based GIS is available.
- *Style definition and viewing:* GeoServer uses the Styled Layer Descriptor (SLD) standard (Lupp, 2007) to specify how each layer should appear in the client. Styles are defined by fairly simple XML documents. The GeoServer control panel includes a style editor to facilitate creating, validating and testing new styles. Using SLD means that the appearance of geographic data on the map can be easily changed without modifying or redeploying client side code.
- *Open Web Services:* In addition to its built-in WMS and WFS services, GeoServer provides a facility for invoking other, user-written services implemented in Java. This simplifies the creation of applications which need to access and modify non-spatial tables in the data base. In our Amphawa Exporer application, for example, we used OWS to implement user management, data export, and photo upload functions.

6. OPENLAYERS CAPABILITIES

JavaScript (Flanagan, 2011) is the programming language most often used to provide interactive behavior in web applications. JavaScript programs are executed on the client, within the web browser. They are largely event driven. That is, the JavaScript program associates different actions with different events such as a mouse click, a button press or the initial loading of a web page and triggers those actions when the event occurs. A JavaScript program can control what is visible on the screen by showing, hiding or changing the characteristics of the page elements (text, tables, page regions, buttons, etc.), which are defined in HTML.

OpenLayers (http://openlayers.org) (Hazzard, 2011) is an open source JavaScript library for loading, displaying and rendering maps and other geographic data from multiple sources. The first version of OpenLayers was released in 2006, and it has been widely used since. OpenLayers provides functions for creating a map, populating it with data, zooming in and out, panning the visible area, creating markers, getting geographic coordinates in response to clicks on the map, and so on. Most importantly, it can create WFS or WMS requests to get data to display on the map.

Figure 4 shows how simple it is to create a map in OpenLayers.

```
var options =
{
        restrictedExtent: restrictarea,
        maxExtent: bounds,
        maxResolution: 911.8359375,
        projection: "EPSG:32647",
        units: 'm',
        numZoomLevels:11
};
map = new OpenLayers.Map('map',options);
```

Figure 4: Creating a map in OpenLayers

In this example, ***restrictarea*** and ***bounds*** are variables that have been previously set to limit the largest area that can be viewed. The ***'map'*** argument in the final line is the id of the HTML ** ** element to hold the map.

OpenLayers includes pre-defined widgets for controlling zooming and panning, for displaying scale bars, and so on. Figure 5 shows some code which creates instances of these widgets in our application and then assigns them to locations on our customized control panel. It is necessary to call the function ***map.addControls()*** to make these widgets actually appear.

Figure 6 illustrates the process of loading and displaying a layer via WMS. In this case, the layer is a gray scale satellite image which will be used as the base map. Vector layers will be displayed on top of this image.

```
var panZoomBarControl = new OpenLayers.Control.PanZoom(
     {div:document.getElementById("panZoomBar")});
var navToolbarControl = new OpenLayers.Control.NavToolbar(
     {div:document.getElementById("navigationTool")});
var scaleline = new OpenLayers.Control.ScaleLine(
     {div:document.getElementById("scaleline"),
      singleLine: true, divisions: 2,  subdivisions: 2 });
```

Figure 5: Creating navigation control widgets

```
var amphawa_ETMB4 = new OpenLayers.Layer.WMS('Amphawa Base Layer', mapurl,
         {layers: 'Amphawa:EtmB4'},{isBaseLayer: true});
map.addLayer(amphawa_ETMB4);
```

Figure 6: A simple WMS request using OpenLayers

Finally, Figure 7 provides an extended example, showing the construction and execution of a WFS request. In this case, the code is requesting water features which are within a specified distance from any of a set of villages selected by the user. The first section constructs a spatial filter based on a fixed radius from each village in the set. The second section executes the request, indicating that WFS is the protocol that should be used and specifying various details about how the result should be rendered on the map. The final two lines display the returned result.

```
function findFeatureByVillage(selectedRadius,villageList)
{
  var i;
  for(i=0; i<villageList.length; i++)
  {
     var longlat = (villageList[i].split(","));
     filterVillage[i] = new OpenLayers.Filter.Spatial({
           type: OpenLayers.Filter.Spatial.DWITHIN,
           distance:selectedRadius,
           distanceUnits: 'm',
           projection: "EPSG:32647",
           value: new OpenLayers.Geometry.Point(longlat[0], longlat[1])
     });
  }
  var highlightlayer = new OpenLayers.Layer.Vector("highlightfeature",
     {
              'displayInLayerSwitcher':false,
               strategies: [new OpenLayers.Strategy.BBOX()],
               protocol: new OpenLayers.Protocol.WFS(
               {
                  url: urlWFS,
                  featureType: "water",
                  featureNS: "http://www.kgeo.org/amphawa",
                  geometryName: null
               }),
               styleMap: new OpenLayers.StyleMap(
               {
                  strokeWidth: 5,
                  strokeColor: "#FF0000"
               }),
               filter: new OpenLayers.Filter.Logical(
               {
                  type: OpenLayers.Filter.Logical.OR,
                  filters:filterVillage
               })
     });
  highlightlayer.setVisibility(true);
  map.addLayer(highlightlayer);
}
```

Figure 7: Spatial query for vector features using WFS

While this may seem quite complicated, any web developer familiar with JavaScript could create this code. If we had to create a program to actually write the XML underlying this WFS request, it would be many pages long.

7. AMPHAWA EXPLORER: A CASE STUDY FOR OPEN SOURCE WEB GIS

Over the past nine months, we have utilized the open source tools described in this paper to implement *Amphawa Explorer*, a specialized web GIS designed to assist communities in managing their local water resources. For details of the project objectives and functionality, see Goldin et al. (2014). We built a spatial database using PostGIS which included water features from ground surveys by our team, land use, roads, village locations, tourism sites and subdistrict boundaries. The application allows users to control what information is displayed on the map, to search for water features based on their locations or attributes, to view and edit attribute values for water and several other data layers, to associate text or photographic notes with selected locations, and to export selected subsets of water data as spreadsheet or shapefiles. The system also provides non-spatial functions such as online help and user management.

In addition to the first author, who served as system architect, database designer and project manager, one full time and two part time programmers worked on implementing Amphawa Explorer. None of these programmers had any previous experience working on geospatial applications or using any of the open source packages we employed. Nevertheless, the full web GIS was up and running in approximately six months. This is quite rapid development for a system of this complexity, especially considering the learning curve involved in mastering new tools.

Without the support of PostGIS, GeoServer and OpenLayers, we believe that the implementation would have required significantly more time and effort. Furthermore, if we had not used these tools, the resulting system would have been far less flexible and maintainable. At this point, we can add new data layers or new spatial queries with minimal effort. We can respond to customers' requests for changes in style or symbolism with little or no programming, thanks to GeoServer's style support.

Finally, we can adapt the basic application framework from Amphawa Explorer to systems we create for new clients. In fact, we now realize that due to our lack of experience, we did not take advantage of many features offered by OpenLayers, GeoServer and PostGIS, coding functionality from scratch when we could have used built-in capabilities of our tools instead. Given what we have learned, we expect that development of our next web GIS project will be even faster and easier.

8. CONCLUSION

Many groups, especially communities and non-profit organizations, cannot realize the benefits of geospatial computing due to the cost and complexity of commercial GIS software. Modern open source tools such as those discussed in this paper can help solve this problem. Broader awareness and use of these tools can help organizations with restricted budgets and knowledge benefit from recent dramatic advances in geospatial information technology.

REFERENCES

de la Beaujardiere, J. (ed), 2006. OpenGIS® Web Map Server Implementation Specification. Retrieved on 15 August 2014 from http://www.opengeospatial.org/standards/wms.

Flanagan, D., 2011. *JavaScript: the Definitive Guide, Sixth Edition*. Sebastapol, CA: O'Reilly

Goldin, S.E., Rudahl, K.T., Varnakovida, P. and Tangsupvattana, A. , 2014. Building community networks for preserving local resources with web GIS. In the *Proceedings of AsiaGIS 2014*, Chiang Mai, Thailand, 16-17 June, 2014.

Hazzard, E. , 2011. *OpenLayers 2.10 Beginners Guide*. Birmingham, UK: Packt Publishing.

Lupp, M. (ed), 2007. Styled Layer Descriptor of the Web Map Service Implementation Specification, V. 1.1.0. Retrieved on 15 August 2014 from http://www.opengeospatial.org/standards/sld.

Obe, R.O. and Hsu, L.S., 2011. *PostGIS in Action*. Stamford, CT: Manning Publications Co.

OGC, 2003. The Spatial Web. OGC White Paper, Retrieved 15 August 2014 from http://portal.opengeospatial.org/files/?artifact_id=3859.

Rockafellar, R.T. and Wets, R.J., 2005. *Variational Analysis*. Heidelberg: Springer-Verlag, p.117.

Seiber, R., 2006. Public participation geographic information systems: a literature and framework. *Annals of the Association of American Geographers*, (96:3), pp. 491-507.

Vretanos, P.A. (ed), 2010. OpenGIS® Web Feature Service 2.0.0 Interface Standard. Retrieved on 15 August 2014 from http://www.opengeospatial.org/standards/wfs.