WEB SERVICES BASED
WEB MAPPING SERVICES (WMS)

1. INTRODUCTION

Geographical Information Systems (GIS) introduce methods and environments to visualize, manipulate, and analyze geospatial data. The nature of the geographical applications requires seamless integration and sharing of spatial data from a variety of providers. Interoperability of services across organizations and providers is a main goal for GIS and also Grid computing [Berman2003, Foster2004].

To solve the interoperability problems, the Open Geospatial Consortium (OGC) has introduced standards by publishing specifications for the GIS services. OGC is a non-profit, international standards organization that is leading the development of standards for geographic data related operations and services. OGC has variety of contributors from different areas such as private industry and academia to create open and extensible software application programming interfaces for GIS [gis].

GIS services, such as defined by the OGC, are part of a larger effort to build distributed systems around the principles of Service Oriented Architectures (SOA). Such systems unify distributed services through a message-oriented architecture, allowing loose coupling, scalability, fault tolerance, and cross-organizational service collections [Kendall1994]. Web Service standards [Booth2004] are a common implementation of SOA ideals, and Grid computing has converging requirements [Berman2003, Foster2004]. By implementing Web Service versions of GIS services, we can integrate them directly with scientific application grids [Aktas2004, Pierce2003, FoxTech2003].

This Chapter gives the details about the design and architecture of our Web Service refactoring of OGC specifications for the Web Map Services. This is part of a larger effort by our group to investigate translations of GIS services into Web Service standards [crisisgrid, webserviceurl]. Some earlier work in this area is reported in WMS [Sonnet2003]. In these documents they define standard WSDL service interfaces.

2. BACKGROUND

After Web Services gain momentum and wide acceptance, some entities working on GIS started to involve itself in Web Services area and tried to integrate and/or convert their GIS servers and applications into Web Services. Some of these entities are commercial GIS leading companies such as ESRI, Cubewerx, Demis and Intergraph. ESRI produced ArcWeb service package for the GIS Web Services by using UDDI for the catalog and registry services. Cubewerx, Demis and Intergraph provide WMS transparent access to their Web Service mapping applications.

OGC as a GIS standards body published its Web Services Common Implementation Specifications recently. We have some differences with the latest OGC Web Service Specifications (See APPENDIX-A 1). In our implementation of GIS Web Services, return types defined as Strings for the WMS getCapabilities and getFeatureInfo. Returned strings are structured data encoded in XML. Strings are actually XML, plain text, HTML

or GML depending on the requested format. In case of getMap request WMS returns image MIME type such as image/jpeg as DataHandler object attached to SOAP message. OGC has different return types defined for the different types of requests. Our implementation for the return types will be improved and changed in accordance with OGC Web Services Specifications. When we first started to implement GIS Web Services OGC did not have this new specification. We will be adapting our request response object to their schemas. Please see APPENDIX-A 2, 3 and 4 for the sample getCapabilities, getMap and getFeatureInfo requests correspondingly.

Regarding catalog registry services, instead of using OGC WRS (Web Registry Services) [ogcregistry] we utilize an alternative Registry Information Model; the Universal Description, Discovery, and Integration (UDDI) [Bellwood2003]. UDDI is domain-independent standardized method for publishing/discovering information about Web Services. As it is WS-Interoperability (WS-I) compatible, UDDI has the advantage being interoperable with most existing Grid/Web Service standards. WRS is an OGC standard to discover/publish service information of geospatial services. It presents a domain-specific registry capability for geospatial information. UDDI is domain-independent standardized method for publishing/discovering information about Web Services. This work is done by one of my colleagues in Community Grids Labs (CGL), Mehmet Aktas. This work is explained in another Chapter of this document.

Our approach to catalog registry services in GIS applications takes into account the descriptive metadata, i.e. quality of service attributes, into discovery process. The geospatial data being provided by a geospatial service may be fitted with client's request, however, this does not necessarily guarantee whether the service is sufficient for the desired quality of service requirements. The client is able to distinguish geospatial services that match to their requirements by matching Quality of Service attributes of service discovery request and service descriptions.

Regarding to our Web Service based WFS, the format of the request and response objects is String in the form of XML. Each request and response has its own schema file. They are created according to these schema files and given parameters. After creating objects as XML, in the Web Service GIS environment, XML objects are put into extensible SOAP envelope. Requests and responses are carried in the SOAP message over the HTTP. For the architecture details please see the Section 4. This work is explained in another Chapter of this document. This work is leaded by Galip Aydin, one of my colleagues in CGL.

3. WEB SERVICES FOR GIS

Web Services give us a means of interoperability between different software applications running on a variety of platforms. Web Services support interoperable machine-to-machine interaction over a network. Every Web Service has an interface described in a machine-readable format. Web Service interfaces are described in a standardized way by using Web Service Description Language (WSDL) [Christiensen2001]. WSDL files define input and output properties of any service and services' protocol bindings. WSDL files are written as XML documents. WSDL is used for describing and locating Web

Services. Web Services are defined by the four major elements of WSDL, “portType”, “message”, “types” and “binding”. Element portType defines the operations provided by the Web Services and the messages involved for these operations. Element message defines the data elements of the operations. Element types are data types used by the Web Service. Element binding defines the communication protocols. Other systems interact with the Web Service in a manner as described in WSDL using Simple Object Access Protocol (SOAP) messages.

SOAP [Box2000] is an XML based message protocol for exchanging the information in a distributed environment. It provides standard packaging structure for transporting XML documents over a variety of network transport protocols. It is made up of three different parts. These are the envelope, the encoding rules and the Remote Procedure Call (RPC) convention. SOAP can be used in combination with some other protocols such as HTTP. OGC compatible Web Services will be using SOAP over HTTP.

Advantages of the Web Services in GIS area can be grouped into three categories:

Distribution: It will be easier to distribute geospatial data and applications across platforms, operating systems, computer languages, etc. They are platform and language neutral.

Integration: It will be easier for application developers to integrate geospatial functionality and data into their custom applications. It is easy to create client stubs from WSDL files and invoke the services.

Infrastructure: We can take advantage of the huge amount of infrastructure that is being built to enable the Web Services architecture – including development tools, application servers, messaging protocols, security infrastructure, workflow definitions, etc [Sonnet2003]. Some of these features are being developed by using Web Service infrastructure in NaradaBrokering [naradabrokeringurl], message based middleware system, developed in CGL (Community Grids Lab.) at Indiana University. NaradaBrokering aims to provide a unified messaging environment that integrates grid services, Web Services, peer-to-peer interactions and traditional middleware operations. In the near future we will be utilizing these features in GIS visualization systems.

GIS services can be grouped into three different categories; these are data services, processing services and registry, or catalog services [Alameh2003]. Data services are tightly coupled with specific data sets and offer access to customized portions of the data. Processing services provide operations for processing or transforming data in a manner determined by user-specified parameters. Registry or catalog services allow users and applications to classify, maintain, register, describe, search and access information about Web Services. In our development of GIS Web Services for the geophysical applications, we use WFS as data services, IS as catalog-registry services and WMS as processing services. For the architecture details see the Section 4.

4. WEB MAP SERVICES (WMS)

WMS is the key service to the GIS visualization system. WMS produces maps from the geographic data. A map is not the data itself. Maps create information from raw geographic data, vector or coverage data. Maps are generally rendered in pictorial formats such as jpeg (Joint Photographic Expert Group), GIF (Graphics Interchange Format), PNG (Potable Network Graphics). WMS also produces maps from vector-based graphical elements in Scalable Vector Graphics (SVG) [Ferraiolo2003].

WMS provides three main services; these are getCapabilities (Section 4.3.1), getMap (Section 4.3.2) and GetFeatureInfo (Section 4.3.3). GetCapabilities and getMap are required services to produce a map but GetFeatureInfo is an optional service. These services and our implementations are explained in the following subsections.

4.1 Web Services Conversion of WMS services

This section gives the details of the integrations of Web Services technologies into OGC compatible GIS visualization Systems.

We first implemented pure OGC compatible WMS and WFS servers. These servers were communicating over HTTP by making HTTPGET/POST requests. Later, we developed a generic conversion algorithm steps for converting HTTP based visualization systems into service based counterparts. These steps are listed below;

1. Define a WSDL for the OGC Web Services (OWS) as a set of interface definitions for its functionalities.
2. Create appropriate XML Schema for all the requests and responses that OWS provides. These schemas are created according to the attributes and properties of HTTP POST and HTTP GET requests defined in OWS specifications.
3. Create client stubs from the WSDL file of the target OWS.
4. After creating a stand-alone Web Services compatible OGC GIS server, you are ready to bridge this kind of server to other generic OGC servers. (See Section 4.4 for the WMS case)

As a case study, we worked on WMS to apply these conversion algorithms. For the first step you crate service interface (WSDL) by using some Web Service tools. Before doing that you need to implement the functionalities in any language as it is done in HTTP based GIS services.

In developing Web Service versions of the WMS, we have converted existing HTTP GET/P0ST conventions [[Beaujardiere2002] into WSDL interfaces. We have encountered some minor technical problems in this conversion.

Internal implementations of the WMS services are compatible with the current WMS specifications but service interfaces and the way to invoke services are different. Services are invoked through the SOAP over HTTP. Requests are created as XML documents and wrapped into body part of the SOAP request message. These requests are shown in Figure 1-3.

Invoking WMS operations should be according to specifications. OGC compatible requests to WMS are well defined in the WMS specifications [Beaujardiere2002]. Requests must have some parameters whose names, numbers, and values should obey the rules defined in the specifications to be OGC compatible. In this section we define these requests in the form of XML schema files.

These schema files are created to be used during the invocation of the operations implemented as Web Services at the WMS side [Sonnet2003]. Requests are created at the WMS Client side. Clients create these requests after getting the required parameters from the user. When request is ready, client sends this request to WMS as a SOAP message. WMS has deployed Web Services for each service, getMap, getCapabilities and getFeatureInfo. Clients use client stubs created before to invoke these specific Web Services. All these services in WMS take one String parameter. This String parameter is request itself. These requests are actually XML documents in String format.

Below schema files displayed in Figure 1-3 include all the elements and attributes of corresponding OGC HTTPGET/POST requests defined in OGC WMS specifications [Beaujardiere2002]. For further information about the functionalities of the requests in the whole system see the Chapter 4.3. In that Chapter, we explain when and how the requests are used, what is returned when they are called etc.

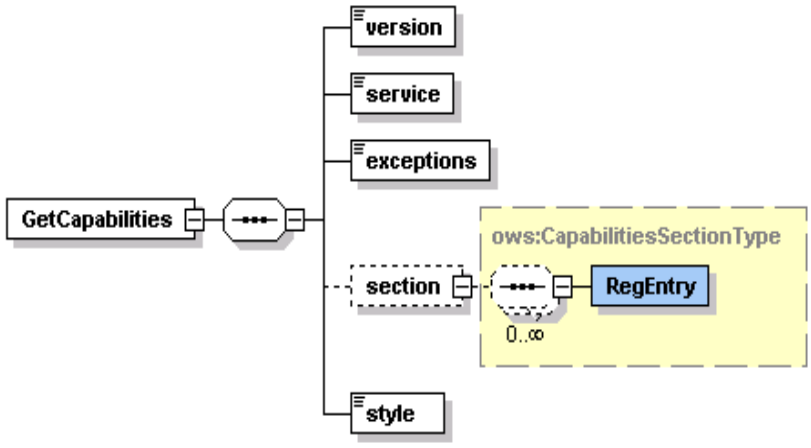


Figure 1 . GetCapabilities Request Schema. See APPENDIX-A 2 for the sample getCapabilities request. See the Chapter 4.3.1for further information.

GetMap request is created for our WMS implementation. We have not implemented styling capability yet. Styling capability will be added soon. WMS supporting styling are called SLD-enabled WMS. The Open GIS Consortium (OGC) Styled Layer Descriptor (SLD) specification [Lalonde2002] defines a mechanism for user-defined symbolization of feature. An SLD-enabled WMS retrieves feature data from

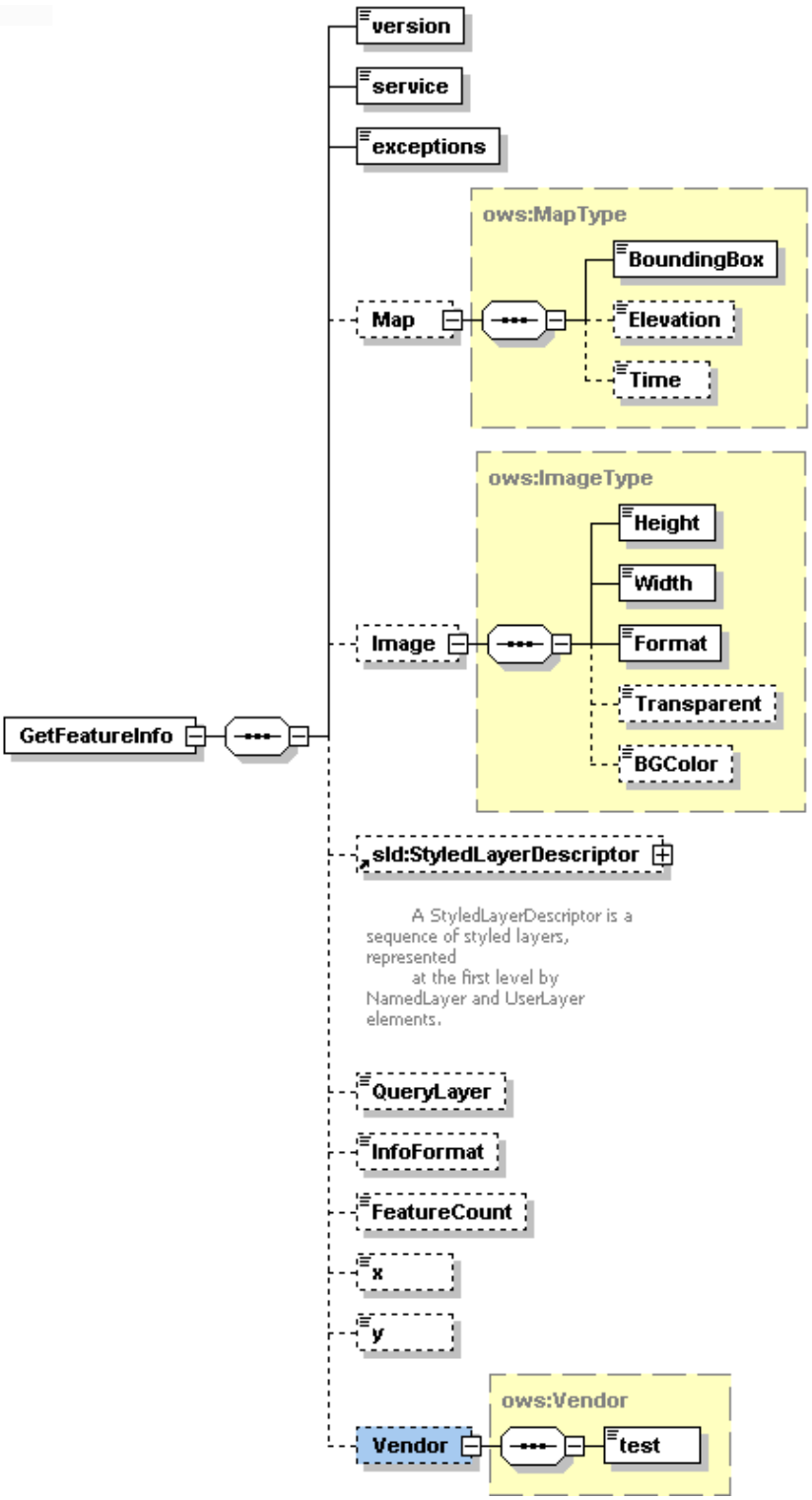


Figure 2 . GetFeatureInfo Request Schema. See APPENDIX-A 3 for the sample getMap request. See the Chapter 4.3.2 for further information.

Web Feature Service [Vretanos2003] and applies explicit styling information provided by the user in order to render a map.

In our project, since we have just implemented Basic WMS, we have not used elements related to styling in the WMS getMap requests. For defining styling in the getMap request we use StyledLayerDescriptor element. StyledLayerDescriptor has other sub elements and attributes.

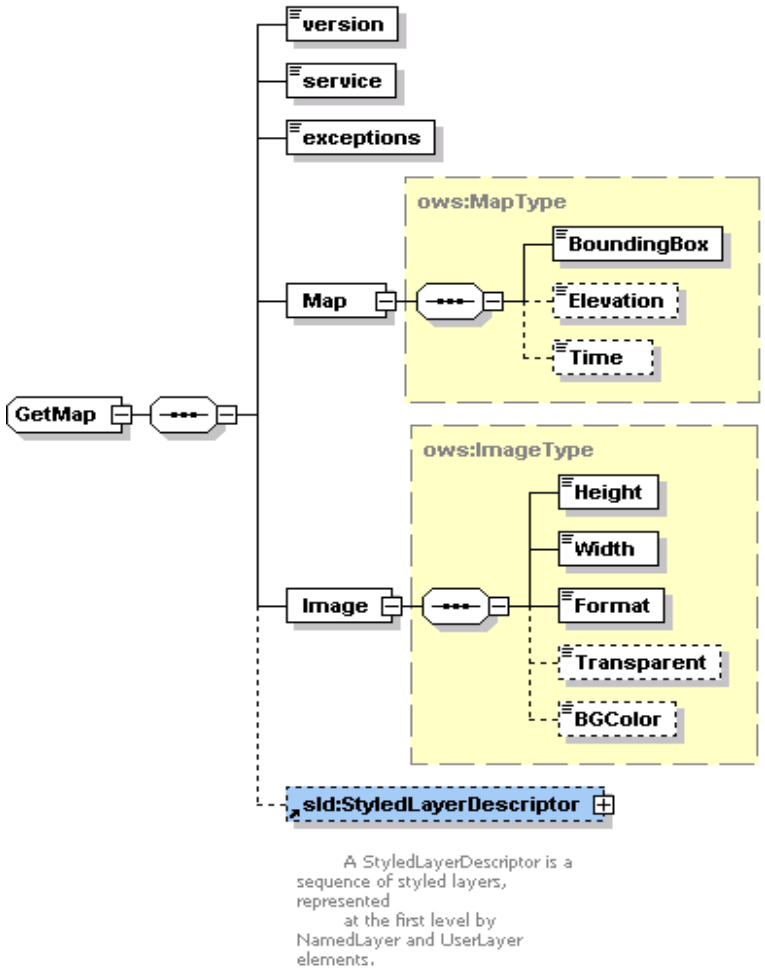


Figure 3 . GetMap Request Schema. See APPENDIX-A 4 for the sample getFeatureInfo request. See the Chapter 4.3.3 for further information.

4.2 Other GIS Components Involved in Visualization System

Our Web Service-compatible WMS depends upon Web Feature Service [crisisgridwfs] and (IS) Information Services [crisisgridfthpis] to accomplish its required tasks. They are ongoing projects in CGL (Community Grids Lab.). This section briefly describes the WMS interactions with these other services.

A general picture of interactions between these three services is displayed in Figure 4. Initial invocations are displayed as black arrows. All the services are implemented as Web Services.

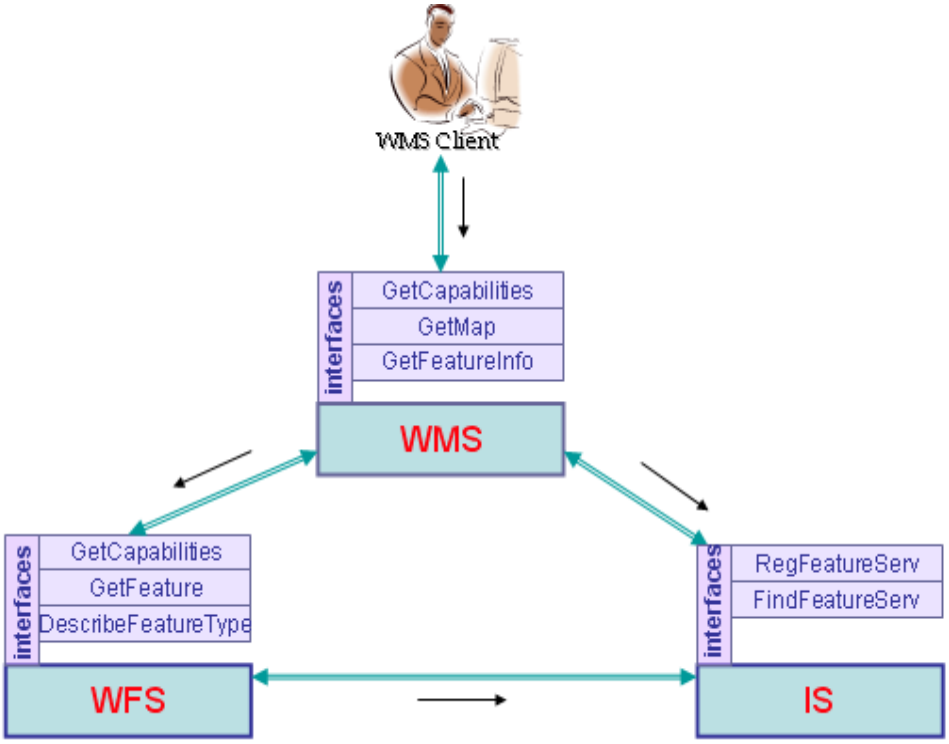


Figure 4 . Basic GIS Components Involved in Visualization System.

4.2.1 Web Feature Service (WFS)

WFS instances store geospatial data and serve them upon request from clients. WFS clients include Web Map Servers and other WFS instances (in case of cascading WFS). WFS provides feature vector data. Vector data are encoded in GML (Geographic Markup Language) [Cox2003], an XML encoding for the transport and storage of geographic information, including both the geometry and properties of geographic features.

According to OpenGIS WFS specification, basic Web Feature Services are getCapabilities, describeFeatureType and getFeature. If WFS is transactional, then it provides two more services. These are "transaction" and "lockFeature" services. Our implementation of WFS is basic WFS, so it does not have transaction and lockFeature capabilities.

Since we have implemented basic WFS, WMS uses basic WFS services: getCapabilities, describeFeatureType, and getFeature. WMS sends a getCapabilities request to WFS to learn which feature types WFS can service and what operations are supported on each feature type. The getCapabilities request can also be mediated by the aggregating Information Services (IS). WMS makes its request to IS to get a specific WFS address that provides needed feature. Please see Section 4.2 for the details about the interconnection between WMS and IS.

When any WMS Client sends a getFeatureInfo request to WMS, WMS creates a getFeature request and sends it to WFS. The URL address of the WFS is found by using IS. After choosing appropriate WFS, WMS makes a getFeature requests to get feature data. A sample request is shown in Figure 5 . The GML file encoded in XML is returned in a SOAP envelope as a response to this request.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<wfs:GetFeature outputFormat="GML2"
xmlns:gml="http://www.opengis.net/gml"
xmlns:wfs="http://www.opengis.net/wfs"
xmlns:ogc="http://www.opengis.net/ogc">
 <wfs:Query typeName="boundary_lines">
<wfs:PropertyName>FNODE</wfs:PropertyName>
<wfs:PropertyName>TNODE</wfs:PropertyName>
<wfs:PropertyName>WORLD</wfs:PropertyName>
<wfs:PropertyName>coordinates</wfs:PropertyName>
<wfs:PropertyName>minx</wfs:PropertyName>
<wfs:PropertyName>miny</wfs:PropertyName>
<wfs:PropertyName>maxx</wfs:PropertyName>
<wfs:PropertyName>maxy</wfs:PropertyName>
  <ogc:Filter>
    <ogc:BBOX>
      <ogc:PropertyName>coordinates</ogc:PropertyName>
      <gml:Box>
        <gml:coordinates>-155,28 -120,50</gml:coordinates>
      </gml:Box>
    </ogc:BBOX>
  </ogc:Filter>
 </wfs:Query>
</wfs:GetFeature>
```

Figure 5 . Sample GetFeature Request from WMS to WFS.

4.2.2 IS (Information-Discovery Services)

For the Visualization Web Services we use a Registry model which is being developed in CGL as a general registry model for Web Services, Fault Tolerant High Performance Information Services (FTHPIS) [crisisgridfthpis]. FTHPIS is a general service registry and discovery model based on UDDI specifications [Bellwood2003]. FTHPIS is described in more detail in the section on UDDI and WS-Context.

4.3 Basic Mapping Server Functionalities

There are three main functionalities for the Basic WMS. These are getCapabilities, getMap and getFeatureInfo. getCapabilities and getMap are mandatory but getFeatureInfo is optional functionalities. There are two general types of WMS. These are Basic WMS and SLD-Enabled WMS. Since we have implemented Basic WMS we will mention basic required fuctionalities. In case of using SLD-Enabled WMS, WE should provide four more additional operations. These are Describe Layer, GetLegend Graphic, GetStyles, and PutStyles.

4.3.1 GetCapabilities

Before a WMS Client requests a map from WMS, it should know what layers WMS provides in which bounding boxes. GetCapabilities request enables WMS Clients to obtain this type of information about the contacted WMS. GetCapabilities request allows the server to advertise its capabilities such as available layers, supported output projections, supported output formats and general service information. After getting this request, WMS returns an XML document with the metadata about the WMS Server. This capability file is kept in the local file system and sent to clients upon getCapabilities request.
After getting the request WMS parses it to derive parameters. If WMS verifies that request, then it sends the capability file to the WMS Client as a SOAP attachment. If WMS encounters any problem during handling of the request than it sends an exception message in SOAP back to the WMS Client. Basic getCapabilities request are pictured out at Figure 6.

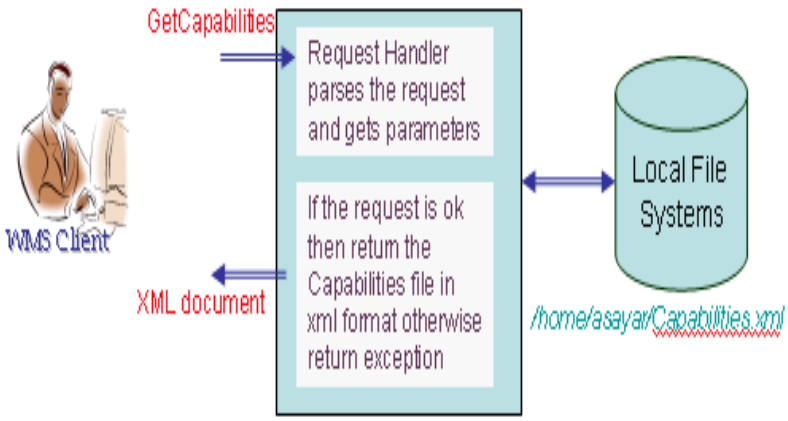


Figure 6 . getCapabilities work flow. See the APPENDIX-A 5 for the sample capabilities file.

4.3.2 GetMap

Another service interface that WMS provides is GetMap request. The getMap service interface allows the retrieval of the map. Chained processes to produce maps are illustrated in Figure 7. This request is done by the client after finishing getCapabilities request and defining the available layers. After getting the getMap request, the WMS goes over the flow depicted in Figure 7 and if everything succeeds, then returns the result as an image in a format defined in the getMap request. All the supported image formats are defined in WMS capability document. Requests for the image formats should be made in accordance with the WMS's capability file. The image is returned back to the WMS Client as an attachment to SOAP message. If the WMS encounters any problem during handling of the request, it sends an exception message in SOAP back to the WMS Client.

WMS first parses the parameters and get their values from the getMap. Depending on these parameters, WMS might need to make some requests to some other WMS services. WMS first determines what layers are requested, in which bounding box, in which form, and so forth. After determining all the request parameters, it makes find_service and getAccess_point requests to IS to determine the WFS providing requested feature data. These requests are done as SOAP messages to IS service interfaces which are implemented as Web Services. GetAccess_point returns the Web Service access point address of the WFS that provides the requested feature. WMS makes getFeature request to the returned WFS and gets the requested feature data in GML format. If the parameter defining returned image format in getMap request is Scalable Vector Graphics (SVG), then WMS creates SVG from returned feature data by using its geometry elements. If the requested image is not in SVG format, WMS first creates the SVG image and then convert it into the desired image formats such as PNG, GIF, or JPEG. Apache Batik provides libraries for this conversion. Batik is a Java(tm) technology based toolkit for applications or applets that use images in the SVG format for various purposes, such as viewing, generation or manipulation. By using these schema files WMS derives geometry elements from the GML file to visualize the feature data. These geometry elements in GML [Cox2003] are basically Point, Polygon, LineString, LinearRing, MultiPoint, MultiPolygon, MultiGeometry, etc.

To create the images from the features returned from the WFS, WMS uses Java Graphics2D and Java AWT libraries. For each layer, a different graphics object is created. As a client, if you assign each layer to different graphics object than Java libraries allow you to overlay these graphic objects.

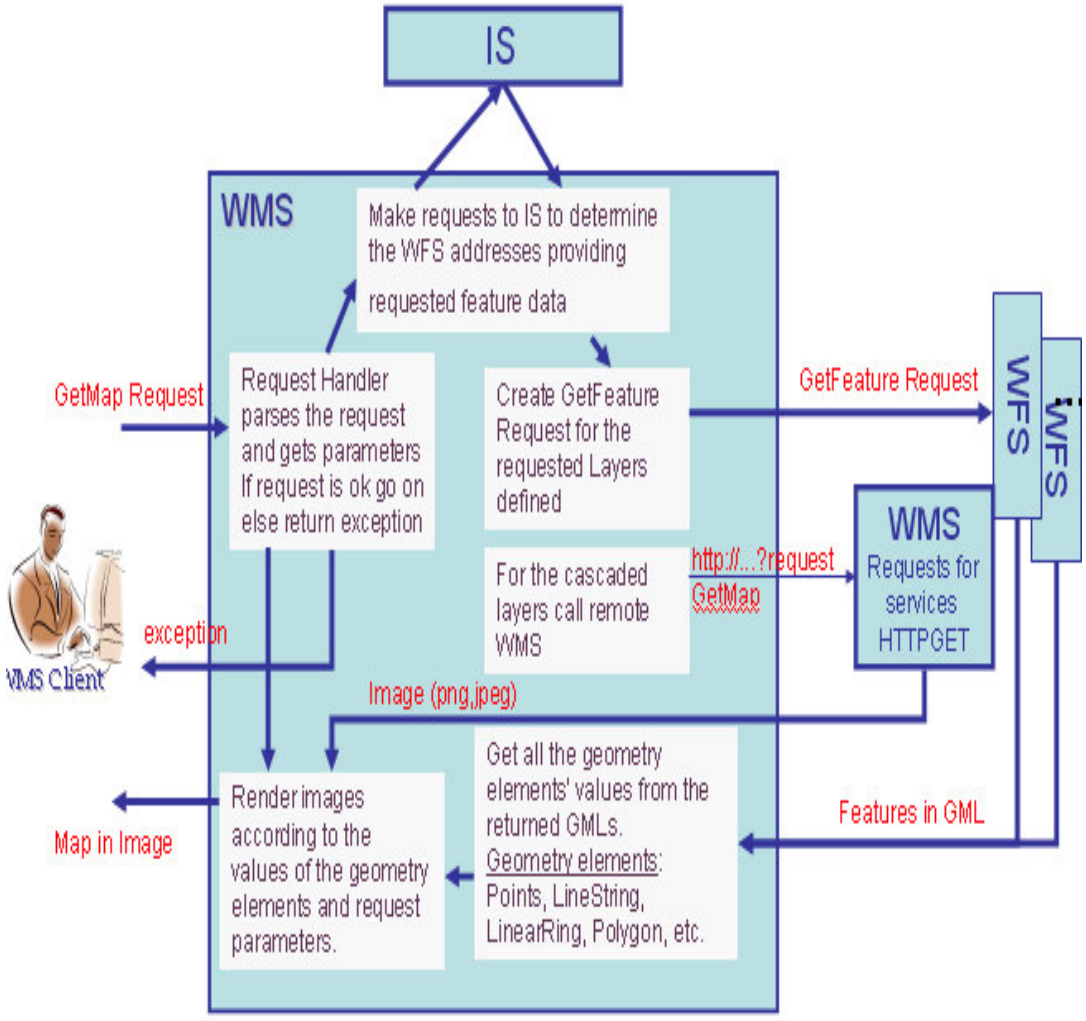


Figure 7. getMap work flow.

How To Create Maps from the layers composed of Raster and Feature data

WMS first parses the parameters and their values from getMap request coming from WMS Client. Depending on these parameters it might need to make some other requests to some other GIS services. WMS first defines what layers are requested in which boundingbox in which form etc. After defining all the request parameters, it makes find_service and getAccess_point requests to IS to determine the WFS providing requested feature data (in case of using CatalogEnabled mode), if WMS works in CatalogDisabled mode then it find out the service addresses from the properties file. Since our implementation of Catalog Service is based on Web Services, to be able to call IS services we should first create service client stubs. getAccess_point returns the wsdl address of the WFS that provides the requested feature. WMS makes getFeature request to this WFS and gets the requested feature data in GML format. If the format parameter in getMap request is Scalable Vector Graphics (SVG), then WMS creates SVG from returned feature data by using its geometry elements. After creating SVG file, we can

easily convert the SVG file into any other image formats such as png, giff, jpeg etc. Apache Batik provides libraries for this conversion. Batik is a Java(tm) technology based toolkit for applications or applets that want to use images in the SVG format for various purposes, such as viewing, generation or manipulation. Creating an SVG file from geometry elements of the received feature data is a little hard. Schema files for the geometry elements are well defined. By using these schema files we created elements to visualize the feature data. These elements are basically Point, Polygon, LineString, LinearRing, MultiPoint, MultiPolygon, and MultiGeometry.

There are basically two main ways to create an image from the simple features. First one is to create SVG file and convert it into any image format. Second is using Java Graphics2D libraries. First create graphics object then overlay another layers created as graphics object. We have been using both ways in different places but we saw that images drawn with graphics2d are with higher quality then the images drawn by SVG conversion. Below you will see a sample code for giving some idea on how to overlay different layers from different sources. In our sample code here, one is coming from HTTP Servlet based WMS server and the other data represented as features are coming from our implementation of Web Service based WFS.

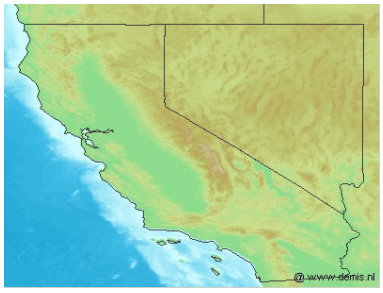
```
URL url = new URL(
                Wmsaddress+"?request=GetMap&width=" +
                width + "&height=" + heigth +
                 "&layers="+layername+
                "&styles=&srs=EPSG:4326&format="+format+"&bbox=" +
                bbox);

BufferedImage im = ImageIO.read(url);
Graphics2D g = im.createGraphics();
…
 if(istherePoint)
   String[] points   = getPointsFromFeatureData();
if(isthereLineString)
   String [] LineStrings = getLineStringFromFeatureData();
if(isthereLineRing)
   String [] LineRings = getLineRingFromFeatureData();
if(istherePolygon)
   String [] polygons = getPolygonsFromFeatureData();
…
…
```

Check all the geometry data of the feature, Point, LineString Polygon etc.

```
if(polygons!=NULL){
for(int i=0; i<polygons. length; i++){
   int [][] xypoints = wm.getXYpoints(polygons[i]);
   g.setColor(Color.darkGray);
   g.drawPolygon(xypoints[0], xypoints[1], xypoints[0].length);
}
}
if(LineRings!=NULL){
for(int i=0; i< LineStrings. length; i++){
   int [][] xypoints = wm.getLinesInStr(LineStrings[i]);
   g.setColor(Color.darkGray);
   g.drawPolyline(xypoints[0], xypoints[1], xypoints[0].length);
 }
}
…
g.dispose();
…
```

If you find any geometry data above such as Points, LineStrings, convert the numbers in the GML file for the feature data into appropriate format to draw shapes for representing these geometry elements and display them by using graphics2D object. If you use the same grpahics2D data the layers will be overlaid.

Sample output:



This code shows how to put two layers on each other by using cascading WMS approach defined according to OGC WMS specification. Here we are putting state boundaries layer over California Base map.

WMS gets ‘state boundaries’ data from our implementation of WFS. WFS provides feature data in vector format. This data is encoded in GML. California base map is created from coverage data. Since we have not implemented WCS (Web Coverage Service) so far, we are using third party WMS (Landsat 7 satellite imagery map from WMS at NASA OnEarth) to get maps crated from coverage data. Cascaded WMS returns image in specified format as defined in the getMap request above. Since these two different layers from different WMSs have same boundingbox values and same width and height, they can be overlayed.

Since we have not implemented SLD-enabled WMS we have been using hard coding whenever we needed styling for the geographic data. For example for the river data we have used blue as color of the river data and for the boundary-lines feature data we have used darkGrey. You can see a simple example in the above code about how to make

styling by using graphics2D libraries. When we implemented our WMS as SLD-enabled we will use more complex styling facilities and operations of the graphics class and graphics2D objects.

4.3.3 GetFeatureInfo

This is an optional WMS service. It is not necessary to create a map. It is used only when a user needs further information about any feature type on the map. However, we have found this very useful when building interactive user interfaces to geophysical applications. The GetFeatureInfo method allows us to send additional information (such as earthquake fault dimensions and material properties) to simulation codes that use these as inputs [Aktas2004].

The GetFeatureInfo works as follows: the user supplies an (x, y) cartesian coordinates and the layers of interest and gets the information back in the form of HTML, GML or ASCII format. All these supported formats are defined again in WMS capability file. Figure 8 illustrates the successive process steps done by the WMS to respond to getFeatureInfo requests coming from the WMS Client. To make the presentation more concrete in the figure, we assumed the feature information is requested in text/HTML format. This value is defined in parameter "info_format" in getFeatureInfo request. GetFeatureInfo service interface supports two more info_formats as well. These are plain text and GML formats. Since HTML creation requires a generic XSL [xslurl] file and XSLT transformation, we have chosen this type of requests to demonstrate getFeatureInfo request processing in Figure 8.

All the processes explained in Section 4.3.2 for the getMap until getting requested features from WFS are same for the getFeatureInfo processing. Again all the remote invocations are done by using SOAP messages.

After getting the feature collections data from the WFS, instead of producing a map as explained in Figure 7, WMS lists all the non-geometry elements and attributes in the returned GML file. For the getMap request WMS deals with geometry elements of the returned GML file but for the getFeatureInfo WMS deals with non-geometry elements. From the list of non-geospatial elements, WMS creates a new XML file to be able to transform non-geometry elements into HTML. This XML file is simply another form of GML which includes just non-geometry elements, properties and attributes. To display all of the processes involved in getFeatureInfo handling (Figure 8), we assumed information is requested in HTML format. After creating new XML file from the non-geo elements, WMS creates HTML file from newly created XML file by using generic XSL file and XSLT transformation machine. For the detailed documentation about the getFeatureInfo, please see our project page [crisisgrid].

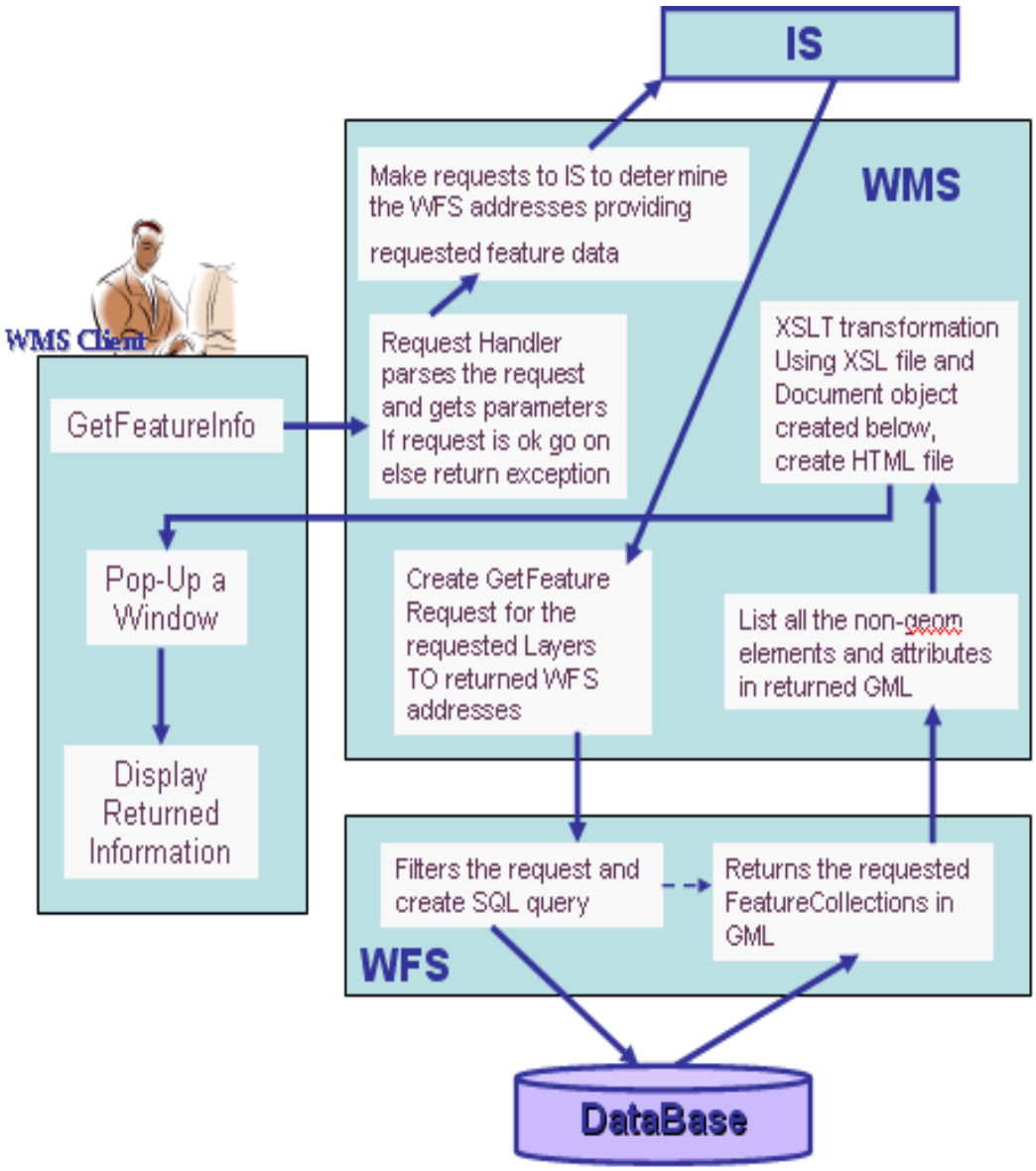


Figure 8 . getFeatureInfo work flow.

4.4 Bridging Web Service Oriented WMS to other WMS Instances

This section explains the architecture to combine Web Services based implementation of WMS systems with the third party WMS systems. Third party systems use HTTP as distributed computing platform.

Cascading WMS is the key issue to enable bridging of these two groups of visualization systems. A cascading WMS is a WMS which aggregates the contents of several individual WMS into one service that can be accessed by clients. Cascading WMS acts like a client to the other WMS and as a server to the clients [Beaujardiere2002]. The

client does not need to keep track of several WMS servers; it only has to be aware of one. The client application does not need to know the ultimate source of all images.

The cascading WMS reports the capabilities of the other WMS as its own and aggregates the contents and capabilities of several distinct WMS servers into one service. For the sample capabilities file please see the APPENDIX-A 5. In most cases, the cascading map server can work on different WMS servers that cannot serve particular projections and formats themselves [Kolodziej2003].

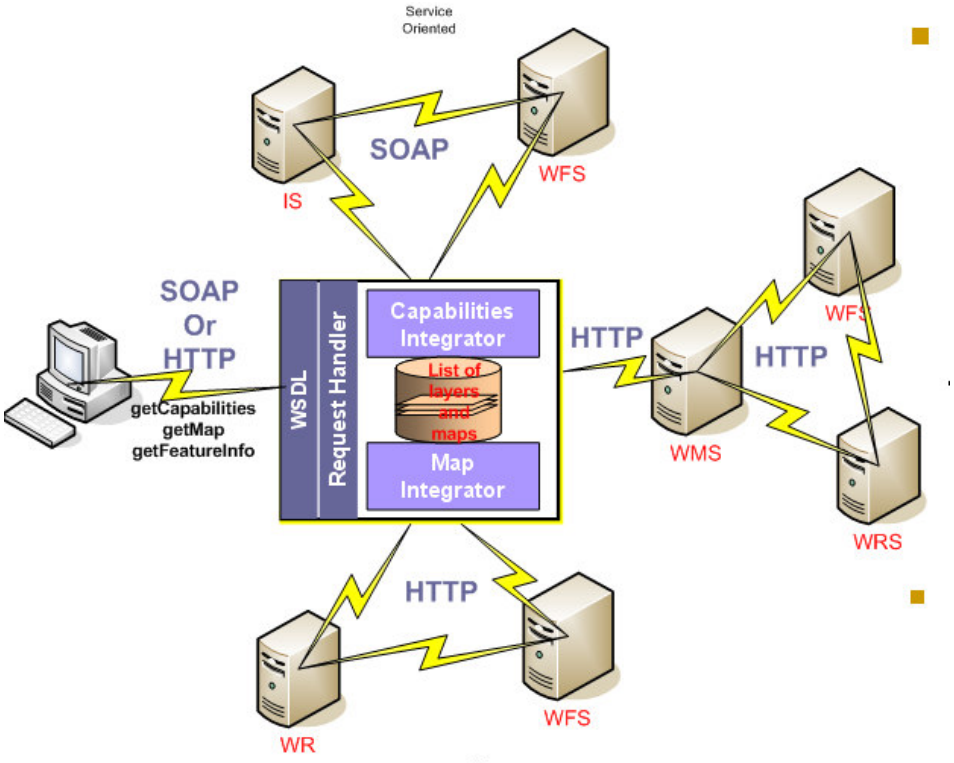


Figure 9 . Bridging of the Web Service-compatible WMS and other WMS.

Clients make their requests to cascaded WMS. Cascaded WMS services are implemented both as Web Services and HTTP Servlet based. It provides separate interface and end points for the service invocations. Clients create their requests and send them in SOAP messages over HTTP in case of using Web Service end points of WMS services or HTTP-GET/POST messages in case of using Servlet end points of WMS services. Please see the Figure 10 for further details. WMS parses coming requests by request handlers. Request handlers derive all the parameters from the request and trigger the responsible modules in the WMS. Figure 9 gives a general depiction.

After getting and parsing the requests WMS defines the requested layers' names. WMS determines if the requested layers are cascaded or not by looking at its capability file. If layer is cascaded than WMS defines the other third party WMS providing requested layer by looking at the capability file. If the layer is not cascaded than WMS determines the addresses of the WFS services that provide these layers by making geo-query to IS. For the cascaded layers, requests to the other (non-Web Service) WMS instances are done over HTTP as defined in OGC specifications, HTTP GET and POST.

As it is shown in Figure 9, proxy cascading WMS integrates SOAP and HTTP based GIS environments. Clients do not have to prepare different versions of requests for the different types of WMSs. Clients just send their requests to the cascading proxy WMS and get the result. In the architecture shown in Figure 9 proxy WMS can be an internal node or an external node of either HTTP based GIS networks or Web Service based GIS networks.

The cascading proxy WMS in the middle of Figure 9 are illustrated and explained in more detail in Figure 10. Different users running on different protocols make their requests to cascading proxy WMS at the same time. WMS handles these differences and responds to clients with a map in requested layers. Interactions of proxy WMS with the WFS are SOAP over HTTP. Interactions with HTTP based WFS will be implemented later.

One raster layer is allowed at one time in one request. After figuring out the address of the WMS (cascaded) providing this layer we make a request to it to get the layer. Addresses are obtained from the Catalog or Information Services in case of WMS running in CatalogEnabled mode, or obtained from WMS's Capabilities file in case of CatalogDisable mode. Please see the Chapter 4.5 for further information about the CatalogDisabled and CatalogEnabled modes.

Here is the sample of proxy cascading WMS's request to HTTP based WMS to get map (or raster layer requested from WMS client)

```
URL url = new URL(
                Wmsaddress+"?request=GetMap&width=" +
                width + "&height=" + heigth +
                 "&layers="+layername+
                "&styles=&srs=EPSG:4326&format="+format+"&bbox=" +
                bbox);

BufferedImage im = ImageIO.read(url);   //returned image or map.

Im.addFeatureData();   //If there is any other layer in future requested.
```

Here is the sample of proxy cascading WMS's request to Web Service based WMS to get map (or raster layer requested from WMS client). WMS uses Web Service client stubs as you see below. These client stubs are created according to WMS service interfaces described their WSDL files. These interface files should be publicly available for the service to be searched and invoked.

```
String request = createRequestInXML();  //  this request will be wrapped in SOAP.
String service_address = getServiceAdr();  // Web Service address of the remote WMS

WMSServicesSoapBindingStub binding = (WMSServicesSoapBindingStub)
            new WMSServicesServiceLocator().
```

```
                getWMSServices(new URL(service_address));

byte[] map = null;                  // Actual Map returned in bytes.
Object[] attachments = binding.getAttachments();

for (int i = 0; i < attachments.length; i++) {
        AttachmentPart att = (AttachmentPart) attachments[i];
        DataHandler dh = att.getActivationDataHandler();
        BufferedInputStream bis = new BufferedInputStream(dh.getInputStream());
        map = new byte[bis.available()];
        bis.read(map, 0, bs.length);
        bis.close();
}

addFeatureDataOtherLayers(map);
```

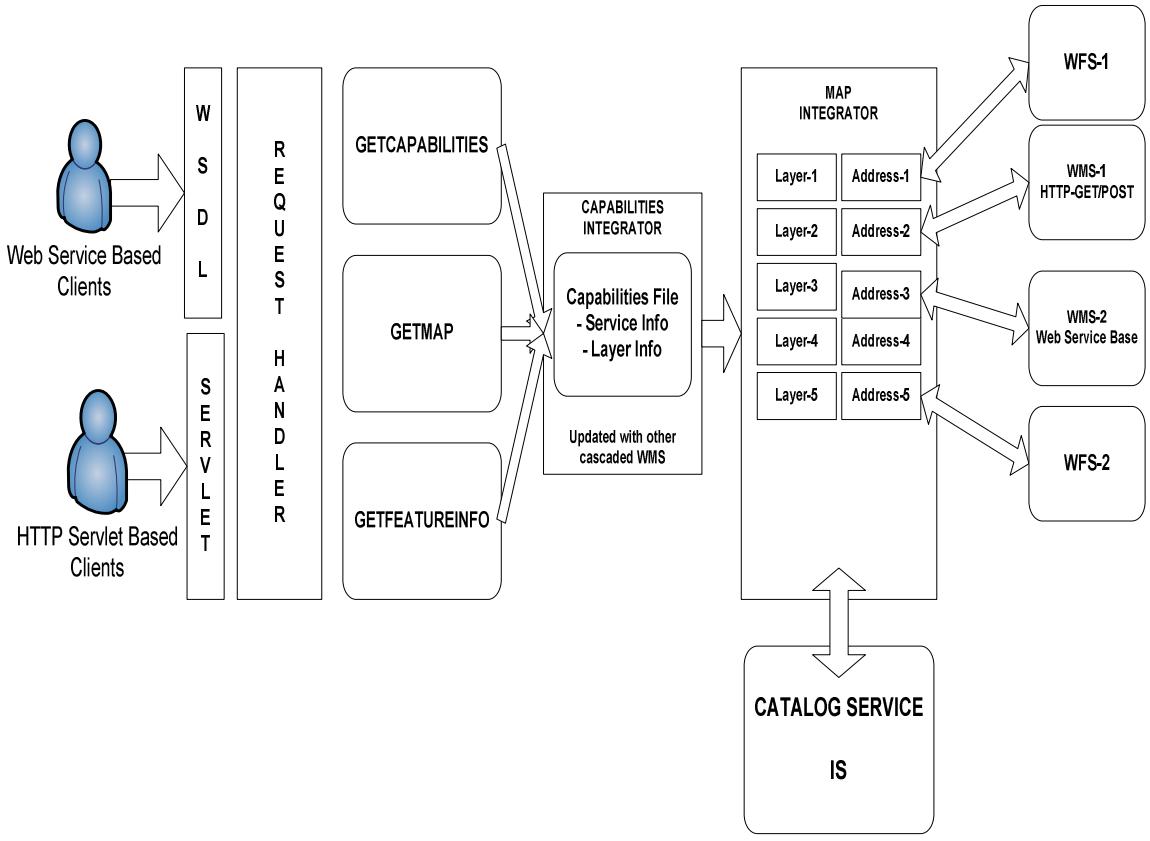


Figure 10. Proxy cascaded WMS for bridging. It is represented as a box in Figure 10.

As it is mentioned proxy cascading WMS provides two interface, one is for serving HPPT-GET/POST requests and other is for serving Web Service based requests. Below, we give sample, actual running service end points for both of these service end poits.

HTTP-GET/POST :

http://gf8.ucs.indiana.edu:8092/wmsstream/WMSServlet

Sample Request :

Parameters are added at the end of service address as (name, value)

http://gf8.ucs.indiana.edu:8092/wmsstream/WMSServlet?request=GetMap&width=400&height=400&layers=Google:Map,California:Faults&styles=&srs=EPSG:4326&format=image/jpeg&bbox=-120,32,-110,40

Web Service :

http://gf8.ucs.indiana.edu:8092/wmsstream/services/WMSServices

Raw result can not be displayed on the browser just by putting the address above in the addrees bar of the browser. Service running above Web Service address can be called by using service ‘s client stubs.

Sample request :

```
String service_address;

request = getRequest();  // Please see the APPENDIX-A 2,3 and 4. sample req
service_address = "http://gf8.ucs.indiana.edu:8092/wmsstream/services/WMSServices";

WMSServicesSoapBindingStub binding;     // Client Binding sub
binding = (WMSServicesSoapBindingStub)
        new cgl.axis.services.wms.WMSServicesServiceLocator().
                        getWMSServices(new URL(service_address));

value = binding.getCapability(request);
```

Figure 11 illustrates the proxy cascading WMS mentioned above in Figure 9 and Figure 10. We have combined earthquake seismic data as feature from a WFS server with Landsat 7 satellite imagery map from WMS at NASA OnEarth [NasaOnearth]. WMS from OnEarth provides access to the World map via OGC compatible HTTP GET and POST requests. We are using these clients to set up geophysical simulation runs, as initially described in [Aktas2003].

4.5 Innovations and Lessons Learned

The spatial data between different districts and different departments need to be shared and to be made interoperable. ISO/TC211 and OGC have defined interface specifications and standards to ensure sharing and interoperable capability of the spatial data. By adapting these to Web Service standards, we simplify the interoperation of GIS services with other service domains.

In this Chapter we have described our efforts to build an OGC compatible GIS Services by using Web Service technologies and OGC specifications.

The Web Services model of the GIS systems provides users with just the services and data they need, without having to install, learn, or pay for any unused functionalities. Geophysical applications such as SERVOGrid project involve various kinds of data processors and data providers distributed geographically. By using service oriented GIS architecture, we can integrate new servers into our geophysics applications seamlessly. Web services are platform neutral, operating system neutral, language neutral and easily extendable.

Regarding the system performance,

We first defined system bottlenecks. Because of the characteristics of the geographical applications, system should store, transfer and parse huge XML based data. Transferring data and extracting information (parsing) were two performance shortcomings of the system. All the geographical applications face these types of performance problems. In order to overcome these problems we have used NaradaBrokering messaging middleware for transferring data and pull parsing technique for extracting information from the geographic data.

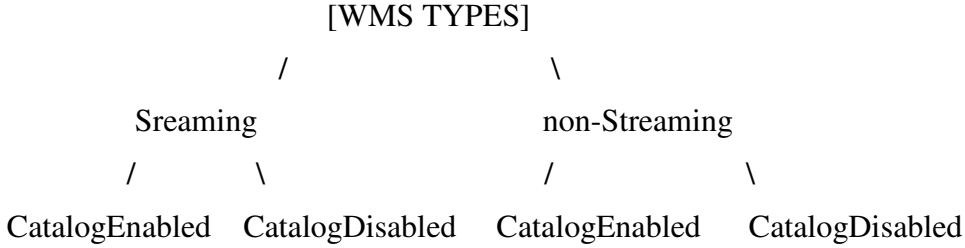
NaradaBrokering is a distributed messaging infrastructure implemented on a network of cooperating brokers. Communication within NaradaBrokering is asynchronous and the system can be used to support different interactions by encapsulating them in specialized events. It supports publish-subscribe messaging models with a dynamic collection of brokers. It is capable to support transport protocols such as TCP, UDP, Multicast, SSL and RTP. It also provides the capability of the communication through firewalls and proxies. It can operate either in a client-server mode like JMS or in a completely distributed JXTA-like peer-to-peer mode. By combining these two disparate models, NaradaBrokering can allow optimized performance-functionality trade-offs for different scenarios. In our GIS visualization system, we use NaradaBrokering in a peer-to-peer mode.

Geographic data are with huge sizes and takes lots of time to transfer in the distributed GIS grid environments. By using streaming WMS and WFS through the NaradaBrokering, we overcome these kinds of common GIS problems. NaradaBrokering places no constrain on either on the size, or rate or scope of the interactions encapsulated within these messages.

XML Pull Parser is a recent development that demonstrates a different approach to XML parsing. It does not provide any support for validation. This is the main reason that it is faster than other alternatives. If you are sure that data is completely valid and validation is done at the server side (as in our case), then using XML Pull Parsing gives the highest performance results. This parsing approach is called pull parsing because the parser only parses what is asked for by the application rather than passing all events up to the client application (as it is done in SAX). The pull approach of this parsing model results in a very small memory footprint, and very fast processing. The pull-parser postpones parsing until a component of the document is accessed, then parses as much of the document as necessary to construct that component.

Regarding the quality of service;

WMS users can select different combination of system properties according to their needs. These properties provided are related to streaming and Catalog-Registry services. Depending on your settings of the variables in the properties file you will end up with four different modes of WMS server. These are listed as below;



- Streaming & CatalogEnabled
- Streaming & CatalogDisabled
- Non-Streaming & CatalogEnabled
- Non-Streaming & CatalogDisabled

In case of using one of CatalogDisabled mode, System gets the WFS addresses from the WMS Capabilities file. When WMS CatalogEnabled mode is used, system gets the WFS address of the requested layer from the Catalog-Registry service. The address of the Catalog-registry service is defined in the properties file of the WMS Capabilities file.

5. WEB MAP CLIENT

We have developed a portlet-based browser client to our Web Service based standard visualization system for testing and the demonstration purposes. A sample WMS Client is shown in Figure 11. Several capabilities are implemented for the user to access and display geospatial data. Our WMS Client enables the user to zoom in, zoom out, measure distance between two points on the map for different coordinate reference systems, to get further information by making getFeatureInfo requests for the attributes of the features on the map, and drag and drop the map to display different bounding boxes. Users can also request maps for the area of interest by selecting predefined options clicking the drop-down list. The user interface also allows the user to change the map sizes from the drop-down lists or enable them to give specific dimensions. Zoom-in and zoom-out features let the user change the bounding box values to display the map in more or less details. Each time user change the bounding box values, user interface shows the updated bounding box values at the each side of the map.

We created generic and application independent WMS Client. It can support more than one geophysics application at the same time. Each geophysics application is bound to a

set of layers. These bindings are defined in structured XML properties file. Users navigate over the applications by selecting set of layers from the dropdown list. A set of layers in the dropdown list is created according to communicated WMS. Binding properties are updated based on the set of supported layers of the communicated WMS.

Our implementation of the client is modular. In order to interact with a specific geophysics application we integrate a plug-in with a modular client. In order to interact with corresponding geophysics application, each component adds various application specific features to WMS Client. Each plug-in can be defined by user it creates a sort of abstraction layer where users can define how to interact with geophysics application.

We created our visualization client to interact with Web Services based visualization systems (architecture is explained in Section 4) but it can also be used for the HTTP based OGC WMSs.

Client interface gives the end users lots of functionality required by the geophysics applications by using Java Server Pages (JSP), Cascading Style Sheets (CSS) and Java Script technologies. We have also developed a portlet version of the WMS Client to be able to deploy in a JSR 168-compatible portlet container. This simplifies distribution of our client application.

WMS services are stateless services. Each time a user makes a request, the WMS Client creates a new request object and invokes remote WMS. All the requests are created according to schema files defined in Section 4.1 and wrapped into the SOAP envelope. After creating SOAP message it is sent over HTTP to the remote WMS.

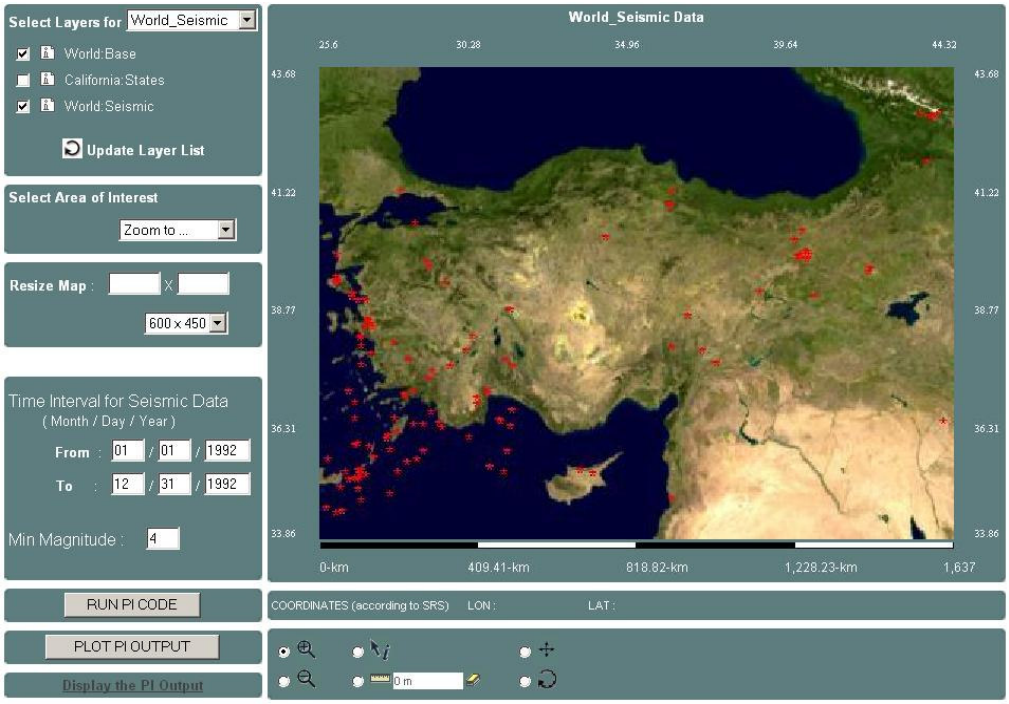


Figure 11 . Project Demo page with the geophysics application. It uses Turkey's Earthquake Seismic data.

We started to upgrade the client and visualization architecture to provide scientific visualizations, real time streaming, and collaborative mappings.

Sample Geophysical Application scenario (PI):

We use proposed visualization architecture for the Pattern Informatics (PI) geophysics applications [Tiampo2002] in the SERVOGrid project [Pierce2003]. SERVOGrid project integrates historical, measured, and calculated earthquake data with simulation codes. SERVOGrid resources are located at various institutions across the country. The SERVOGrid Complexity and Computational Environment (CCE) [FoxTech2003] is an environment to build and integrate different domains of Grid and Web Services into a single cooperating system. As a part of SERVOGrid CCE environment, we chose the PI application which is used to produce the well-publicized "hot spot" grid-values published by SERVO team member Prof. John Rundle and his group at the University of California-Davis. Hot spot values are returned from a remote server running PI algorithms.

In this geophysics application scenario, WMS Client gets the output of the PI server as grid-values, interprets it and overlays it as another layer over the current map. The overlay layer for the hot spots is created by assigning different colors for each grid cell according to their values (Figure 12). These colors represent the different ranges of probabilities of the earthquake for the seismic point in the future. These jobs are done by the PI module deployed in WMS Client. As we mentioned before, each geophysics application has its own module to fulfill the application specific tasks.

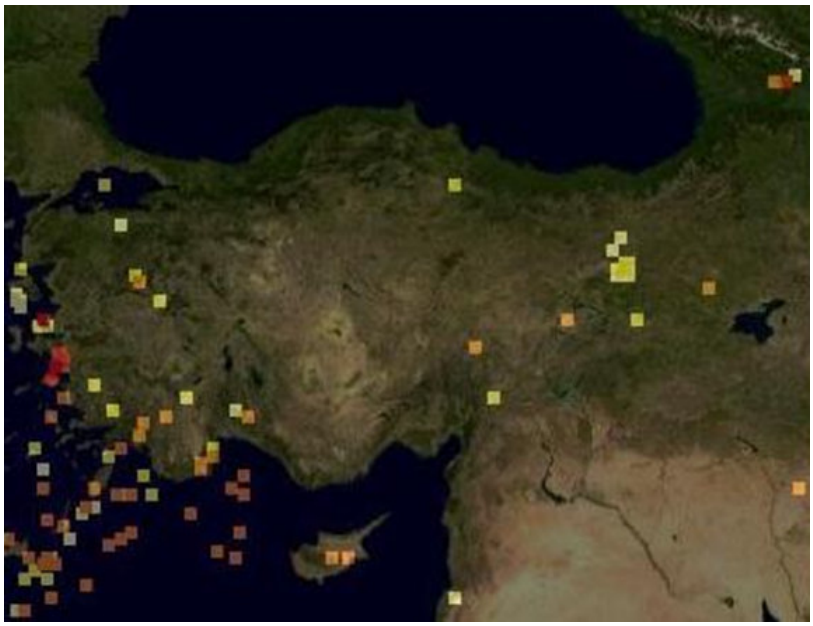


Figure 12. Overlaid layer created by PI module in WMS Client after running PI geophysics application over the map displayed in Figure 11.

5.1 Clients - Using Google Maps as Servlet-based WMS

To be able to integrate Google maps into our OGC compatible WMS, We should be able to get Google maps as an image. But Google API does not allow this. They use XMLHttpRequest and JavaScript (remote scripting). Return type is an object that you can not touch and convert it to an image. This is also prohibited by "Terms of Use" they published in their official Google map page.

As it is mentioned before, WMS returns maps in the form of images such as JPEG, GIF and PNG. WMS Clients get the maps in image formats and overlays them. Usual WMS Clients can not use maps coming from Google Map Servers. To solve this problem and use high performance Google maps in our WMS applications and overlay different map layers coming from the usual WMS with the Google Maps, we created an intermediary Google Mapping Server. It takes WMS compatible requests from the WMS Clients, converts these requests into a new form that real Google Map Server can understand. For the sample OGC compatible getMap request see the Figure 3 and APPENDIX-A 3. In contrast to OGC compatible getMap requests, Google Map server uses requests with different parameters such as zoom level, tile numbers and tile width.

Server side Google is implemented by Sunghoon Ko. He is one of my colleagues in CGL.

Please see the APPENDIX-B 1 for the sample output of overlaying Google Map as a layer in the WMS applications.

5.2 Clients - Integrating AJAX approach into Mapping Services

AJAX [Garret] is an important web development model for the browser based web applications. It uses several technologies which come together and incorporate to create a powerful new model. Technologies forming AJAX model such as XML, JavaScript, HTTP and XHTML are widely-used and well-known. Google Mapping is an example of a high performance AJAX based application.

Web Services are self-contained, self-describing, and modular. Unlike earlier, more tightly coupled distributed object approaches such as Common Objects Request Brokers (CORBA), Web Service systems support an XML message-centric approach, allowing us to build loosely coupled, highly distributed systems that span organizations. Web Services also generalize many of the desirable characteristics of GIS systems, such as standards for providing general purpose specifications for publishing, locating, and invoking services across the Web. Web Services also use widely-used and well-known technologies such as XML and HTTP as AJAX does. Since AJAX and Web Services are XML based structures they are able to leverage each others strength.

AJAX uses HTTP based XMLHttpRequest protocol for the message transfers. Web Services use Simple Object Access Protocol (SOAP) as a communications protocol. In order to be able to integrate these two different computing environments using different

message protocols, we should have found a common protocol or we should have converted the message format from one protocol to another. Since there is no ready to use common protocol to handle messages communications between AJAX and Web Services, we created a new framework converting message formats from one computing environment to another which integrates these two powerful technologies and obtains the best performance results.

This framework is designed for browser based web applications using Web Services. It enables users to utilize AJAX and Web Services advantages.

5.2.1 Generic Integration

In this Section, we describe and illustrate the generic integration framework for integrating AJAX into browser based Web Service applications. There are two main actors in the integration architecture – the visualization client and GIS Web Services. Web Services are invoked by using SOAP - XML based messaging protocol for the message exchange.

How to invoke Web Services in the AJAX model?

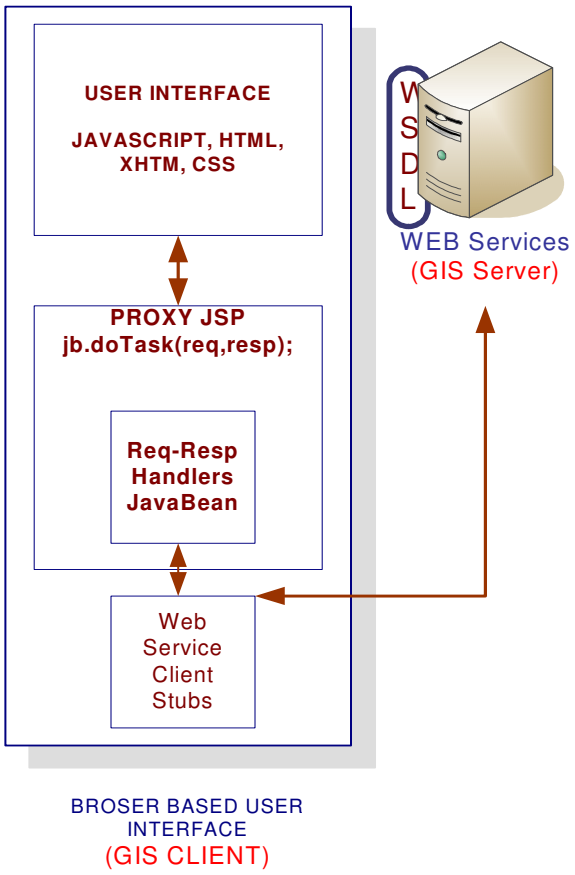


Figure 13: Invoking Web Services from the AJAX applications.

The client browser makes a request to the server broker (via a JSP page), which in turn makes a request to the Web Service by using previously prepared Web Service client stubs. The response from the Web Service is then transformed by the service broker, and presented to the client browser. Below we will go in more detail to explain all these steps.

First create an XMLHttpRequest object to make a remote scripting call.

- var http = new XMLHttpRequest();

Then, define the end point as an URL to make a call. The URL address should be local. This an intermediary proxy service to make appropriate requests for the GIS Web Service.

- var url = "proxy.jsp";

Then, make a call to the local proxy service end point defined above by the user given parameters.

- http.open ("GET", url + "?bbox = " + bbox +…[other parameter-value pairs]……)

proxy.jsp is an intermediary server page to capture request (HttpServletRequest) and response (HttpServletResponse) objects. Proxy JSP includes just one line of codes to forward the HttpServletRequest and HttpServletResponse parameters coming from the first page via XMLHttpRequest protocol.

- jb.doTask(request,response)

"request" and "response" parameters come from the user interface page. This first page includes some JavaScript, XHTML, CSS and JSP to capture the user given parameters and to display the returned result on the screen.

"jb" is a Java class object which handles creating appropriate requests by using its request-response handlers and Web Service client stubs. Request-response handler also handles receiving and parsing response object coming from GIS Web Services interacted with.

After having received response from the GIS Web Service, "jb" object sends the returned result to XMLHttpRequest object in the first page.

- PrintWriter pw = response.getWriter();
- pw.write(response);

XMLHttpRequest object at the user interface page captures this value by making a call as below

- http.onreadystatechange = handleHttpResponse

This generic integration architecture can be applied to any kind of Web services. Since return types of each Web services are different and they provide different service API, you need to handle application specific implementations and requirements in browser based client side.

In Section 5.2.2, we prove the applicability and efficiency of the proposed integration framework by giving two important usage scenarios in GIS domain.

5.2.2 Usage Scenarios

Integration is basically coupling AJAX actions with the Web Services invocations, and synchronizing the actions and returned objects from the point of end users. The usage scenarios in Section 5.2.1 and 5.2.2 use the generic integration architecture illustrated in Figure 13. In the usage scenarios there will be minor difference in the form of extensions. Differences come from the service specific requests created according to the service provider's service API (published as WSDL), or handling returned data to display on the screen. But these are all implementation differences.

5.2.2.1 Google WFS integration with AJAX Approach

WFS provides feature data in vector format and vector data are encoded in GML according to OGC WFS specifications and depending on the parameters given in the "getFeature" request. GML is an XML encoding for the transport and storage of geographic information, including both the geometry and properties of geographic features.

In response to the "getFeature" request, the GML file encoded in XML is returned in a SOAP envelope as a response to this request. After getting a response, the client extracts geometry elements. The most important and commonly used geometry elements are Points, LineStrings, LinearRings, and Polygons. GML is an OGC standard for feature data representation.

Even though Google Mapping API supports just two of them, Points and LineStrings, the other geometry elements can also be converted to these two types with minor updates. Having extracted and obtained geometry elements, these elements are plotted over the Google Map by using "GPoints" and "GPolylines" objects and the "mapOverlay" function of the Google Map API.

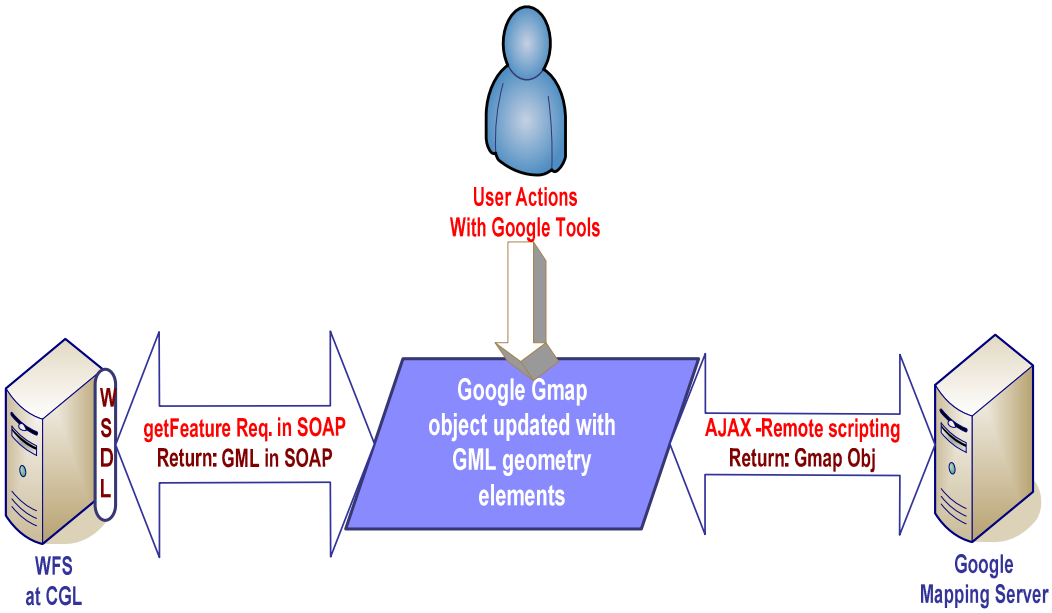


Figure 14: Integration of Google Maps with OGC WFS by using architecture defined in Figure 13. See the APPENDIX-B 2 for the sample output and user interface.

By setting returned GML's non-geometry elements and using 'GMarker" object of the Google API, this architecture also provides the "getFeatureInfo" functionalities of the OGC WMS services. All these tasks are achieved by using XMLHttpRequest API and JavaScript functionalities.

XMLHttpRequest uses DOM for parsing returned structured responses in XML. If returned data is oversized for the server then the DOM parser throws "Out of Memory" exception. In order to overcome this drawback of the DOM and Google Map, we have used Pull Parsing [Little2004]. After parsing and handling GML documents returned from WFS, the result is written into the web browsers response object. Through the responseXML call of the XMLHttpRequest in JavaScript, the browser gets the result and makes appropriate modifications to the data and display on the screen.

XML Pull Parser is a recent development that demonstrates a different approach to XML parsing. It does not provide any support for validation. This is the main reason that it is much faster than its competitors. If you are sure that data is completely valid and validation is done at the server side (as in our case) or in a database, then using XML Pull Parsing gives the highest performance results. In our usage scenario explained above, data comes from the OGC compatible data server (WFS).

This parsing approach is called pull parsing because the parser only parses what is asked for by the application rather than passing all events up to the client application. The pull approach of this parsing model results in a very small memory footprint, and very fast processing. The pull-parser postpones parsing until a component of the document is accessed, then parses as much of the document as necessary to construct that component.

5.2.2.2 Google WMS Integration with AJAX Approach

There are two different path working in parallel by the given user parameters created by the client actions. Actions are interpreted by the browser through the Google Mapping tools. JavaScript captures these actions by ActionListeners and Google Binding APIs and gives to Layer-2 object. Please see the Figure 15.

On the browser user interface class is a JSP page. It includes two JavaScript class-references. One is for the Google Map object and the other is for the WMS map image and bindings to the Google Map object.

Interconnection for creating Layer-2 is done in accordance with the proposed architecture defined above in Figure 13. For Layer-1, a classic Google mapping application is used through the AJAX web application module and XMLHttpRequest protocol. Google handles creating the map by using XMLHttpRequest and given remote JavaScript file in the browser [Beaujardiere2002].

When we use this type of interaction interface to WMS, we can utilize all the OGC compatible functionalities of the WMS such as "getMap", "getCapabilities" and "getFeatureInfo". The client is going to be a thin client; it just takes the map and overlays it over the Google map. Overlay is done by using some advanced JavaScript techniques. The client does not need to make rendering or mapping jobs to create the map image. The map is already returned by the WMS and in a ready to use format such as JPEG or PNG or TIFF. Return type is defined as a parameter in the "getMap" request given to WMS. These images in different formats are converted to a JavaScript object before overlaying.

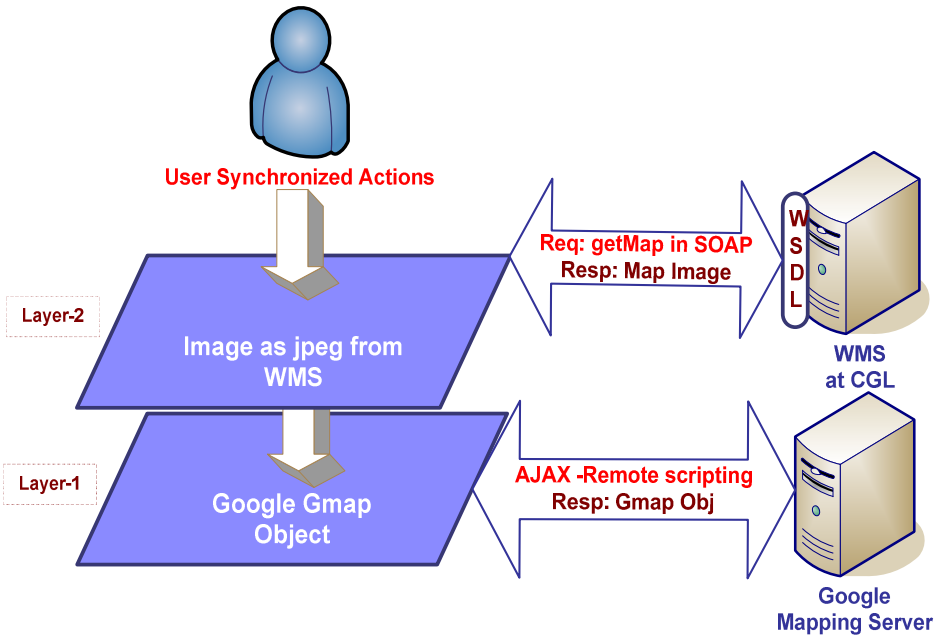


Figure 15: Integration of Google Maps with OGC WMS by using architecture defined in Figure 13. See the APPENDIX-B 3 for the sample output and user interface.

5.3 Innovations and Lessons Learned

WMS clients can be categorized into three groups. These are thin clients, thick clients and application specific customized clients. Application specific customized clients are in the middle of thin and thick groups of clients. For the scientific applications in the geophysics area we have been using application specific customized WMS Clients. Scientific applications require some additional functionalities and additional server interactions. All of this quality of services is granted by the application specific customized WMS Clients. All the servers and services involved scientific application are orchestrated by the WMS Client depending on the characteristics of the application and end users need.

In contrast to thin clients that can just take the map in image format and display on the screen, thick clients are capable of doing image resolution, data format translations and coordinate system conversions. Neither thin nor thick clients are capable of handling scientific application specific tasks.

After having implemented WMS Client we tried to increase the performance of the visualization system. For that reason, we integrated AJAX approach into Web Service based visualization systems.

If the GIS visualization client uses Web Services from the desktop browser application and Web Services are capable of responding fast enough, then using the AJAX model for calling Web Services gives high performance increases. Since both AJAX and Web Services use XML based protocols for the request and responses, they leverage their advantages. This enables application developers to easily integrate AJAX based browser applications into Web Services.

AJAX and Web Services are XML based structures and this property allow developers to utilize their advantages together. The proposed system enables AJAX based high performance web application approaches to utilize web services. If Web Service based applications have web based user interface for end users, then, using this framework makes displaying much faster. Users do not need to wait whole data to be received to render and display the results. Partial displaying is possible without refreshing the whole page. Instead of making request for whole page, only the interested part will be requested. This reduces the workload of the network traffic.

In addition to its advantages, the proposed system has a couple of disadvantages. The proposed integration framework introduces some extra work for the browser based web application developers. Extra work mostly comes from the conversion of parameters to be able to make compatible requests to remote Web Services. In order to make valid requests, the proxy server should be deployed locally and client stubs for Web Service invocations should be created before running the application. Compared to pure AJAX based web application, the performance of the application is reduced by the intermediary proxy server during its conversion and message handling jobs.

6. MAP ANIMATIONS

The geographical information has at the same time, a spatial, thematic and temporary nature. When you have the same layer in different moments in time, it could be interesting to show every static map as a part of an animation, making possible the creation of a little movie composed by a certain number of frames.

6.1 Browser-based Map Animation Techniques

HTML is ideal for creating static websites where text and images are placed at fixed positions. But it doesn't really support dynamic sites, where text, images, and animations are moving around on the screen. Traditionally, these effects were achieved with animated GIF images or Java Applets. Some other techniques recently used in map animations are Quick Time and Macromedia Flash.

An Animated GIF is simply a GIF file that is composed of 2 or more individual GIF images or frames. Each frame is displayed for a set time and a set number of cycles (the times and number of cycles are completely controllable by the creator). The changes in the frames are what gives the graphic the sense of motion. There are no additional plug-in or enhancements to a web browser necessary to see an animated gif.

An Animated GIF is actually many images saved in one. While Animated GIFs can be used for animations, they do not support interactivity. They simply loop images in a predefined order and that's it.

Aside from Animated GIFs, another approach has been used to add dynamic effects to web pages: Java Applets. Applets have often been criticized for "killing" browsers. Sometimes, Java programmers are not as skillful as one might have wished. Some applets are programmed so they eventually take up all resources on the computer, and these results in "freezing" the browser.

There is an API for QuickTime functionalities which is fully accessible through Java. With Java, you can write your own QuickTime-compatible applications, or run Java Applets from QuickTime over the web inside a browser. QuickTime sprites let you add interactivity to your media. QuickTime Sprites are animations that can be made of an image, a short image sequence or a video clip. These images can be moved around the movie frame on a vector path — creating the animation. As the sprite creator you need to create the image (such as a logo), specify the movement path (a spin motion, for example) and add other behaviors (like play a sound). QuickTime does the rest.

One way to escape 'DHTML hell' and provide more interactivity on the client is to work with a Browser "Plug-in" like Java Applets or the increasingly popular Macromedia Flash plug-in. Macromedia Flash is most commonly used for the design and scripting of

animations and other dynamic content for web sites. Over the years Flash evolved to a more powerful development tool and it includes a JavaScript like programming language called action-script.

After giving some background information, we give architecture of our proposed map animation and movie framework in Section 6.2. We have a lot of advantages over the techniques mentioned above.

6.2 Streaming Map movies

We produce map animations in the form of streaming map videos similar to ones you see in the weather cast web sites or weather news. Maps are produced from geographic data provided by WFSs, WCSs and WMSs in the form of vector or raster data. All the GIS services in our system are OGC compatible and Web Services based [Cox2004].

The Open Geospatial Consortium (OGC) defines several related standards for the representation, storage, and retrieval of geographic data and information. Regarding to visualization services it basically defines and publishes standardization and implementation specifications for Web Map Service (WMS), Web Feature Service (WFS), Web Coverage Service (WCS) and Web Registry Service (WRS). The Web Map Service (WMS) [Beaujardiere2002] produce maps from the geographic data. Geographic data are kept in Web Coverage Server (WCS) and/or Web Feature Service (WFS) or other WMSs. WCS stores raster data in image tiles and WFS stores feature vector data in GML formats. WMS produces maps from these raw geographic data upon requests from the WMS clients. These maps are the static representations of geospatial data. Representations are in pictorial formats such as PNG, SVG, JPEG, GIF, etc.

Standard map servers produce static images, but many type of geographic data are time dependent. In order to understand geographic phenomena and characteristics of temporal data it is necessary to examine how these patterns change over time for these types of data. We are therefore investigating the problems of creating streaming video map servers based upon appropriate standard collaboration technologies [Wu2004].

In our approach, visualizing changes over time is achieved by integrating temporal information on a map. Usually the result is a series of static maps showing certain themes at different moments. In addition to creating static maps, WMS also has the ability to combine the static maps correspond to a specific time interval data and combine them in an animated movie. Movies created by WMS are composed of a certain number of frames. Each frame represents a static map that corresponds to a time frame defined in request.

In GIS area there is a huge demand of using multimedia technology over internet scale to support groups collaborative work in which members are distributed at different geological locations, such as distance learning, virtual classroom, video conferencing etc. Because of the characteristics of the geospatial data, one organization or entity can not

have all the geo-data and geo-services available locally. So they need some other data and services from other entities. They require sharing of data. Some times organizations need to make a decision about the geographic data by looking at the map or animated movies. They can achieve this by getting together physically or using collaborative GIS services. This growing need and demand for large scale interactive and collaborative GIS systems present several interesting research challenges in computer science such as: design of good collaboration framework with advantages of high scalability, extensibility, reliability and security, design of synchronization and composition mechanisms to synchronize multiple video/audio streams. In order to achieve these aims, partially or fully, we have been using our Lab's GlobalMMCS collaboration services for the GIS collaborative visualization applications. GlobalMMCS [Lalonde2002, Vretanos2003, Sayar2005] is an integrated video conferencing solution which enables heterogeneous clients to join the same real-time multimedia sessions. It provides a flexible architecture to support even more standards and applications. We also inherit many additional features such as replay and collaborative annotation and whiteboard systems.

All the services participating in the proposed architecture are Web Services. System utilizes all the advantages of the Web Services such as easy integration, using widely acceptable technologies, cross-platform, cross-language etc. However, because of the characteristics of the geographic data and Web Services message exchange protocol, SOAP, there are performance limitations if we use the current SOAP approaches to integrate Geographic Information Systems (GIS) applications with Web Service based collaboration systems, especially for the multimedia GIS applications such as displaying streaming map movies. To overcome this type of performance issues we created streaming version of Open Geospatial Consortiums (OGC)'s Web Map Service (WMS) for creating both static maps and map movie streams composed of more than one static map. Streaming data transfer is enabled by using Community Grids Lab's NaradaBrokering messaging middleware. The NaradaBrokering messaging substrate enables scalable, fault-tolerant, distributed interactions between entities, and is based on the publish/subscribe paradigm. The NaradaBrokering substrate provides support for transport protocols such as TCP, Parallel TCP, UDP, Multicast, HTTP and SSL; it also facilitates communications across NAT and firewall/proxy boundaries.

Animating static maps are done by the help of Hasan Bulut. He is one of my colleagues in CGL.

6.3 Innovations and Lessons Learned – Future Work

There is no much work done on this area. This is also an ongoing work in our Lab. We have implemented a couple of good sample applications of streaming map movies and used Java Media Framework (JMF) libraries and JMF Studio to display the movie streams published to a multicast IP address and port number. Broadcast and Unicast addresses are also supported by the JMF Studio. Server creating movie streams sends streams to a multicast address, clients who wants to see the map movie streams should

subscribe to this address. In our basic proof of concept application, clients use JMF studio to subscribe to the Real Time Protocol (RTP) session that they are interested in.

We will continue to integrate the streaming map server with Global-MMC's archiving capabilities. This will enable useful functionality, such as allowing users to select a movie from the archive. To make the WMS available for the collaborative conferences or online education, admin user will be able to update map streams on the fly while it is playing. This is called annotation feature for the collaboration. For this purpose, we plan to integrate e-Sports [Gang2005] whiteboard drawing tools.

We will be creating movies at the client side even in case of collaborative movie creating environment. There will be significant performance gain when we use this approach. Clients can archive both previously created frames and movies. If a client needs same type of frames for the same matching time intervals then it does not need to go back to WMS and spend time for getting the movie frames
Communication and request/response message passing between WMS and clients will be done through the NaradaBrokering (message based middleware system).

Right now for publishing collaborative map videos streams, we are using multicast addresses but in the future we will be using publish/subscribe properties of NB. Approaches to collaboration have tended to use IP Multicast to deal with the content distribution problem. Multicast provides a powerful, elegant and flexible framework for implementing collaborative systems. In this scenario, participants agree upon a multicast group and collaborate by exchanging data over this group; the system relies on MBONE to manage this interchange. Far more powerful framework for collaboration is the publish/subscribe paradigm. In publish/subscribe system the routing of messages from the publisher to subscriber is within the purview of the message oriented middleware (MOM), which is responsible for routing the right content from the producer to right consumers [5].

In the future we will be using Access Grid for the collaborative video conferencing for the streaming map videos. Access Grid uses IP-Multicast for video conferencing. The advantage of this solution is its simplicity and ease of use for the end user but it requires much more network bandwidth then many users can accommodate.

APPENDICES – A

1. WMS Web Services Interface Definition (as WSDL file)

```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="http://services.wms.ogc.cgl"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://services.wms.ogc.cgl"
    xmlns:intf="http://services.wms.ogc.cgl"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tns1="http://www.w3.org/1999/XMLSchema"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    - <!--
    WSDL created by Apache Axis version: 1.2beta3
    Built on Aug 01, 2004 (05:59:22 PDT)  -->
- <wsdl:message name="getFeatureInfoResponse">
  <wsdl:part name="getFeatureInfoReturn" type="soapenc:string" />
    </wsdl:message>
- <wsdl:message name="getMapResponse">
  <wsdl:part name="getMapReturn" type="tns1:anyType" />
    </wsdl:message>
- <wsdl:message name="getCapabilityResponse">
  <wsdl:part name="getCapabilityReturn" type="soapenc:string" />
    </wsdl:message>
- <wsdl:message name="getMapRequest">
  <wsdl:part name="request" type="soapenc:string" />
    </wsdl:message>
- <wsdl:message name="getFeatureInfoRequest">
  <wsdl:part name="request" type="soapenc:string" />
    </wsdl:message>
- <wsdl:message name="getCapabilityRequest">
  <wsdl:part name="request" type="soapenc:string" />
    </wsdl:message>
- <wsdl:portType name="WMSServices">
- <wsdl:operation name="getMap" parameterOrder="request">
  <wsdl:input message="impl:getMapRequest" name="getMapRequest" />
  <wsdl:output message="impl:getMapResponse" name="getMapResponse" />
    </wsdl:operation>
- <wsdl:operation name="getCapability" parameterOrder="request">
  <wsdl:input message="impl:getCapabilityRequest" name="getCapabilityRequest"
    />
  <wsdl:output message="impl:getCapabilityResponse"
    name="getCapabilityResponse" />
    </wsdl:operation>
- <wsdl:operation name="getFeatureInfo" parameterOrder="request">
```

```
<wsdl:input message="impl:getFeatureInfoRequest"
    name="getFeatureInfoRequest" />
<wsdl:output message="impl:getFeatureInfoResponse"
    name="getFeatureInfoResponse" />
  </wsdl:operation>
  </wsdl:portType>
- <wsdl:binding name="WMSServicesSoapBinding" type="impl:WMSServices">
<wsdlsoap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http" />
- <wsdl:operation name="getMap">
<wsdlsoap:operation soapAction="" />
- <wsdl:input name="getMapRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://services.wms.ogc.cgl" use="encoded" />
  </wsdl:input>
- <wsdl:output name="getMapResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://services.wms.ogc.cgl" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="getCapability">
<wsdlsoap:operation soapAction="" />
- <wsdl:input name="getCapabilityRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://services.wms.ogc.cgl" use="encoded" />
  </wsdl:input>
- <wsdl:output name="getCapabilityResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://services.wms.ogc.cgl" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="getFeatureInfo">
<wsdlsoap:operation soapAction="" />
- <wsdl:input name="getFeatureInfoRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://services.wms.ogc.cgl" use="encoded" />
  </wsdl:input>
- <wsdl:output name="getFeatureInfoResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://services.wms.ogc.cgl" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
- <wsdl:service name="WMSServicesService">
- <wsdl:port binding="impl:WMSServicesSoapBinding" name="WMSServices">
<wsdlsoap:address
    location="http://gf8.ucs.indiana.edu:8092//wmsstream/services/WMSServi
    ces" />
  </wsdl:port>
  </wsdl:service>
```

```
</wsdl:definitions>
```

2. Sample getCapabilities request encoded in XML to be put inside the SOAP envelope

```
<?xml version="1.0" encoding="UTF-8"?>
<GetCapabilities xmlns="http://www.opengis.net/ows">
        <version>1.1.1</version>
        <service>wms</service>
        <exceptions>application_vnd_ogc_se_xml</exceptions>
        <style>full</style>
</GetCapabilities>
```

3. Sample getMap request encoded in XML to be put inside the SOAP envelope

```
<?xml version="1.0" encoding="UTF-8"?>
<GetMap xmlns="http://www.opengis.net/ows">
        <version>1.1.1</version>
        <service>wms</service>
        <exceptions>application_vnd_ogc_se_xml</exceptions>
        <Map>
                <BoundingBox decimal="." cs="," ts=" ">-124.85,32.26,-113.56,42.75</BoundingBox>
        </Map>
        <Image>
                <Height>400</Height>
                <Width>400</Width>
                <Format>image/jpg</Format>
                <Transparent>true</Transparent>
                <BGColor>0xFFFFFF</BGColor>
        </Image>
        <ns1:StyledLayerDescriptor version="1.0.20" xmlns:ns1="http://www.opengis.net/sld">
                <ns1:NamedLayer>
                        <ns1:Name>Nasa:Satellite</ns1:Name>
                        <ns1:Description>
                                <ns1:Title>Nasa:Satellite</ns1:Title>
                                <ns1:Abstract>Nasa:Satellite</ns1:Abstract>
                        </ns1:Description>
                </ns1:NamedLayer>
                <ns1:NamedLayer>
                        <ns1:Name>California:Faults</ns1:Name>
                        <ns1:Description>
                                <ns1:Title>California:Faults</ns1:Title>
                                <ns1:Abstract>California:Faults</ns1:Abstract>
                        </ns1:Description>
                </ns1:NamedLayer>
                <ns1:NamedLayer>
                        <ns1:Name>California:States</ns1:Name>
```

```
                        <ns1:Description>
                                <ns1:Title>California:States</ns1:Title>
                                <ns1:Abstract>California:States</ns1:Abstract>
                        </ns1:Description>
                </ns1:NamedLayer>
        </ns1:StyledLayerDescriptor>
</GetMap>
```

4. Sample getFeatureInfo request encoded in XML to be put inside the SOAP envelope

```
<?xml version="1.0" encoding="UTF-8"?>
<GetFeatureInfo xmlns="http://www.opengis.net/ows">
        <version>1.1.1</version>
        <service>wms</service>
        <exceptions>application_vnd_ogc_se_xml</exceptions>
        <Map>
                <BoundingBox decimal="." cs="," ts=" ">-124.85,32.26,-113.56,42.75</BoundingBox>
        </Map>
        <Image>
                <Height>300</Height>
                <Width>400</Width>
                <Format>image/jpg</Format>
                <Transparent>true</Transparent>
                <BGColor>0xFFFFFF</BGColor>
        </Image>
        <QueryLayer>Nasa:Satellite,California:Faults,California:States</QueryLayer>
        <InfoFormat>text/html</InfoFormat>
        <FeatureCount>999</FeatureCount>
        <x>117</x>
        <y>218</y>
</GetFeatureInfo>
```

5. WMS Capabilities File

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSPY v2004 rel. 4 U (http://www.xmlspy.com)-->
<WMS_Capabilities xmlns="http://www.opengis.net/wms" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wms
C:\capabilities_1_3_0.xsd" version="1.3.0" updateSequence="String">
        <Service>
                <Name>WMS</Name>
                <Title>Pervasive WMS</Title>
                <Abstract>wms reference implementation</Abstract>
                <KeywordList>
                        <Keyword >pervasive</Keyword>
```

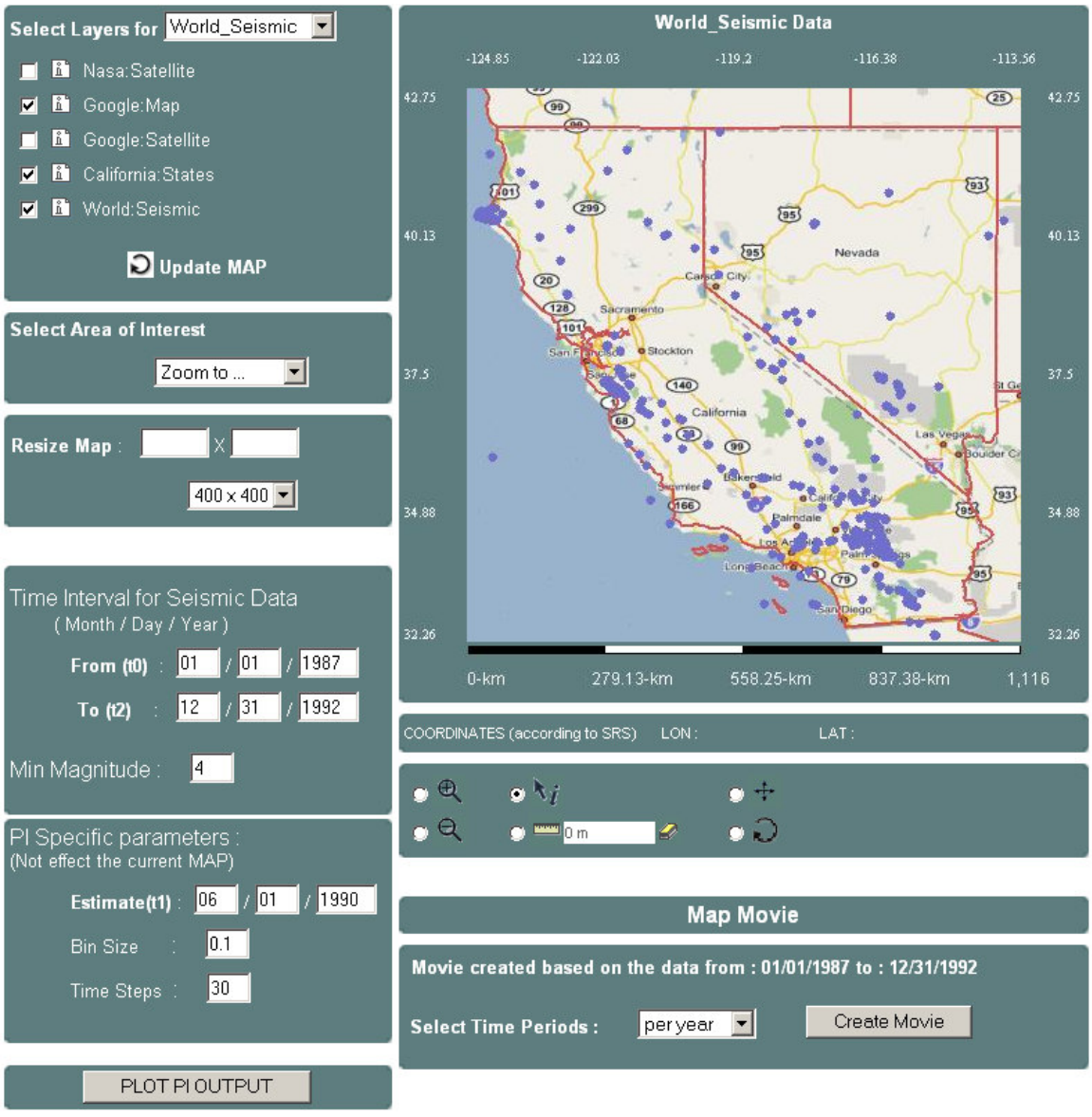
```
                <Keyword >wms</Keyword>
        </KeywordList>
        <OnlineResource xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="simple"
        xlink:href="http://toro.ucs.indiana.edu:8086/WMSServices.wsdl"/>
        <!-- the following service information is optional -->
        <ContactInformation>
                <ContactPersonPrimary>
                        <ContactPerson>Ahmet Sayar</ContactPerson>
                        <ContactOrganization>Pervasive Tech Lab</ContactOrganization>
                </ContactPersonPrimary>
                <ContactPosition>Research Assistant</ContactPosition>
                <ContactAddress>
                        <AddressType>XXXX</AddressType>
                        <Address>501 N. Morton St. Rm 222</Address>
                        <City>Bloomington</City>
                        <StateOrProvince>IN</StateOrProvince>
                        <PostCode>47404</PostCode>
                        <Country>USA</Country>
                </ContactAddress>
                <ContactVoiceTelephone>1(812)8560752</ContactVoiceTelephone>
                <ContactFacsimileTelephone>1(812)8567972</ContactFacsimileTelephone>

<ContactElectronicMailAddress>asayar@cs.indiana.edu</ContactElectronicMailAddress>
        </ContactInformation>
</Service>
<Capability>
        <Request>
                <GetCapabilities>
                        <Format>application/vnd.ogc.wms_xml</Format>
                        <DCPType>
                                <!-- Currently there is just one DCPT environment supported HTTP.
                                                In the near future there will be web services
                                                support by the Open-GIS.
                                                Whenever they update their standard schemas, I
                                                will update my capabilities document.
                                        Since I am going to use web mep services I do not need
                                        these informations -->
                                <HTTP><Get><OnlineResource /></Get>
                                        <Post>  <OnlineResource /></Post>
                                </HTTP>
                        </DCPType>
                </GetCapabilities>
                <GetMap>
                        <Format>image/gif</Format>
                        <Format>image/png</Format>
                        <Format>image/jpg</Format>
                        <Format>image/tif</Format>
                        <Format>image/bmp</Format>
                        <Format>image/svg+xml</Format>
                        <DCPType>
                                <!-- Currently there is just one DCPT environment supported HTTP.
                                                In the near future there will be web services
                                                support by the Open-GIS.
                                                Whenever they update their standard schemas, I
                                                will update my capabilities document.
                                        Since I am going to use web mep services I do not need
                                        these informations -->
                                <HTTP><Get><OnlineResource /></Get>
                                        <Post>  <OnlineResource /></Post>
                                </HTTP>
                        </DCPType>
                </GetMap>
```

```
</Request>
<Exception>
        <Format>application/vnd.ogc.se_xml</Format>
        <Format>application/vnd.ogc.se_inimage</Format>
        <Format>application/vnd.ogc.se_blank</Format>
</Exception>
<Layer queryable="0" cascaded="1" opaque="0" noSubsets="0" fixedWidth="1"
                                                                        fixedHeight="1">
        <Name>pervasive WMS-demo Layers</Name>
        <Title>pervasive WMS-demo Layers</Title>
        <Abstract>pervasive WMS-demo Layers</Abstract>
        <KeywordList>
                <Keyword>pervasive</Keyword>
                <Keyword>WMS</Keyword>
                <Keyword>layer</Keyword>
        </KeywordList>
        <CRS>EPSG:4326</CRS>
        <EX_GeographicBoundingBox>
                <westBoundLongitude>-150</westBoundLongitude>
                <eastBoundLongitude>100</eastBoundLongitude>
                <southBoundLatitude>30</southBoundLatitude>
                <northBoundLatitude>50</northBoundLatitude>
        </EX_GeographicBoundingBox>
        <MinScaleDenominator>0</MinScaleDenominator>
        <MaxScaleDenominator>100000000</MaxScaleDenominator>

        <!-- WORLD SEISMIC -->
        <Layer queryable="0" cascaded="1" noSubsets="0">
                <Title>World_Seismic</Title>
                <Abstract>Seismic data for the world</Abstract>
                <CRS>EPSG:4326</CRS>
                <Layer queryable="1" cascaded="1" noSubsets="0" fixedWidth="0"
                                                                        fixedHeight="0">
                        <Name>Nasa:Satellite</Name>
                        <Title>Nasa:Satellite</Title>
                        <EX_GeographicBoundingBox>
                                <westBoundLongitude>-150</westBoundLongitude>
                                <eastBoundLongitude>-100</eastBoundLongitude>
                                <southBoundLatitude>30</southBoundLatitude>
                                <northBoundLatitude>50</northBoundLatitude>
                        </EX_GeographicBoundingBox>
                        <BoundingBox CRS="EPSG:26986" minx="189000"
                miny="834000" maxx="285000" maxy="962000" resx="1" resy="1" />
                        <MinScaleDenominator>0</MinScaleDenominator>
                        <MaxScaleDenominator>100000000</MaxScaleDenominator>
                </Layer>
                <Layer queryable="1" cascaded="1" noSubsets="0" fixedWidth="0"
                                                                        fixedHeight="0">
                        <Name>Google:Map</Name>
                        <Title>Google:Map</Title>
                        <EX_GeographicBoundingBox>
                                <westBoundLongitude>-150</westBoundLongitude>
                                <eastBoundLongitude>-100</eastBoundLongitude>
                                <southBoundLatitude>30</southBoundLatitude>
                                <northBoundLatitude>50</northBoundLatitude>
                        </EX_GeographicBoundingBox>
                        <BoundingBox CRS="EPSG:26986" minx="189000"
                miny="834000" maxx="285000" maxy="962000" resx="1" resy="1" />
                        <MinScaleDenominator>0</MinScaleDenominator>
                        <MaxScaleDenominator>100000000</MaxScaleDenominator>
                </Layer>
                <Layer queryable="1" cascaded="1" noSubsets="0" fixedWidth="0"
```

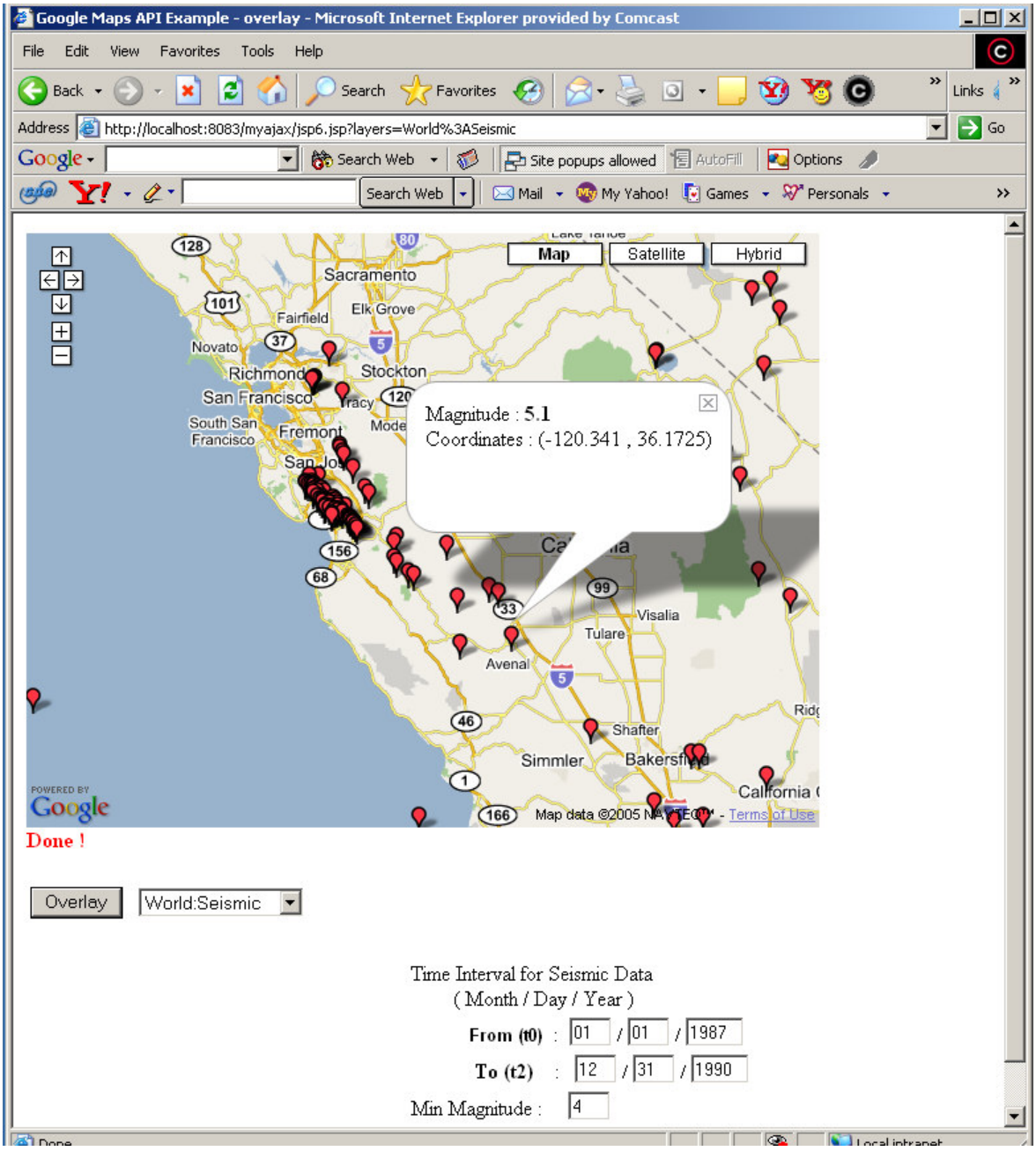
```
                                                                                fixedHeight="0">
                                        <Name>Google:Satellite</Name>
                                        <Title>Google:Satellite</Title>
                                        <EX_GeographicBoundingBox>
                                                <westBoundLongitude>-150</westBoundLongitude>
                                                <eastBoundLongitude>-100</eastBoundLongitude>
                                                <southBoundLatitude>30</southBoundLatitude>
                                                <northBoundLatitude>50</northBoundLatitude>
                                        </EX_GeographicBoundingBox>
                                        <BoundingBox CRS="EPSG:26986" minx="189000"
                                miny="834000" maxx="285000" maxy="962000" resx="1" resy="1" />
                                        <MinScaleDenominator>0</MinScaleDenominator>
                                        <MaxScaleDenominator>100000000</MaxScaleDenominator>
                                </Layer>
                        </Layer>
                </Layer>
        </Capability>
</WMS_Capabilities>
```

APPENDIX – B

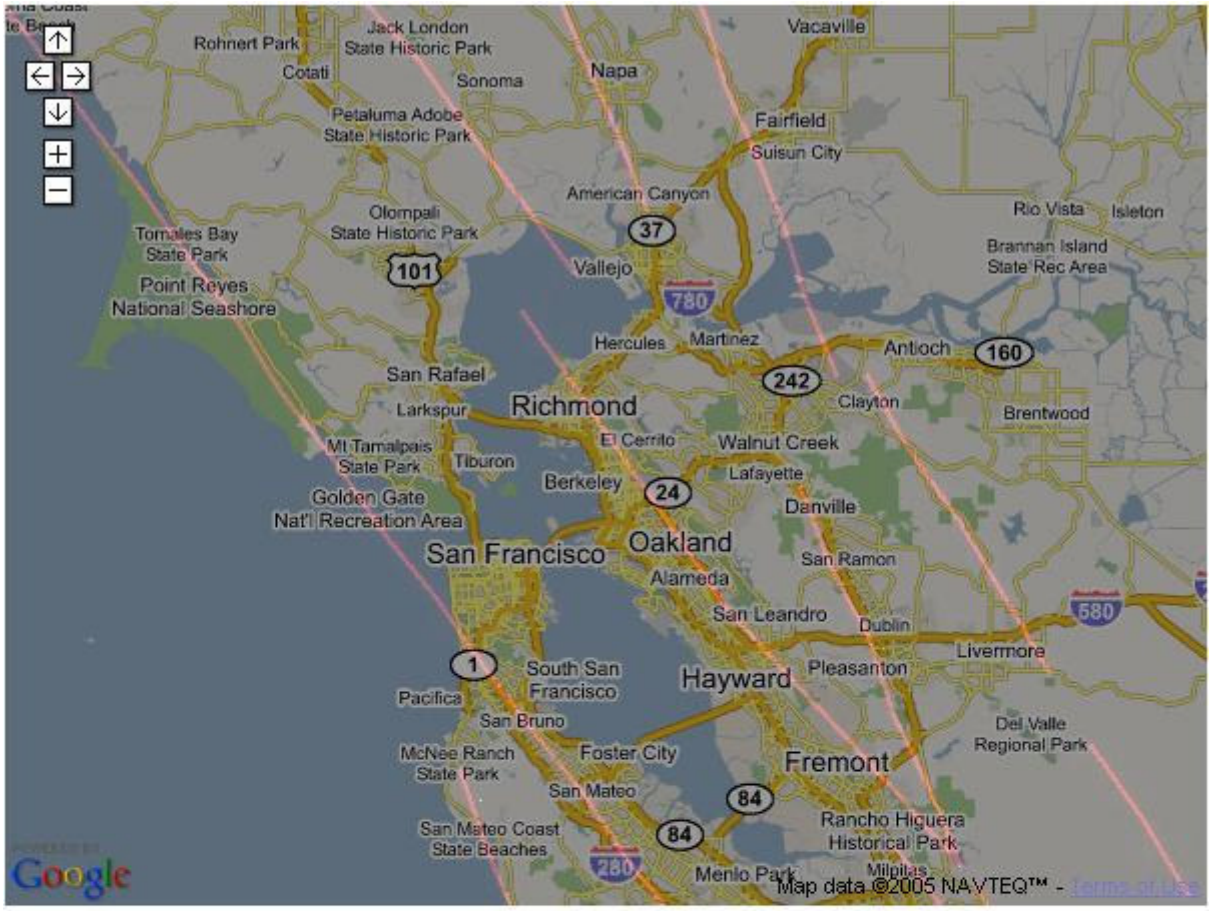
1. WMS Client User Interface - Using Google Maps as Servlet-based WMS



2. WMS Client User Interface - Google WFS integration with AJAX Approach



3. WMS Client User Interface - Google WMS Integration with AJAX Approach

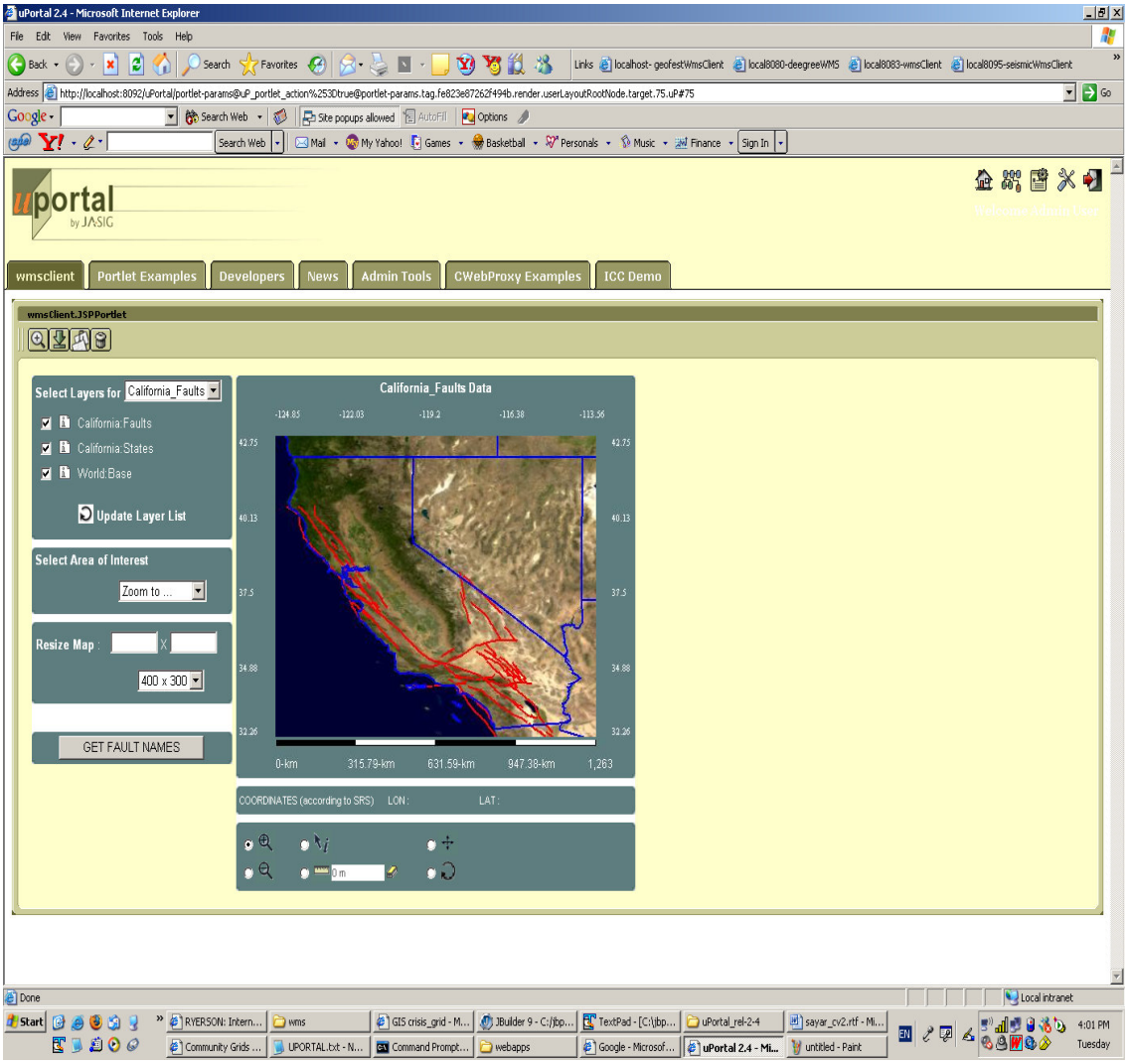


Opacity:	20%	40%	60%	80%	100%

SELECT GOOGLE BASE MAP	SELECT LAYER(S) CGL OGC WMS
◉ Google Road Map	☑ California Fault Layer
○ Google Satellite Map	
○ Google Hybrid Map	

TO GET FURTHER INFO ABOUT THE FEATURE :
SELECT ☐ & CLICK ON THE FEATURE.

4. WMS Client User Interface - Portal Version



REFERENCES

[Aktas2004] Aktas M., Aydin G., Donnellan A., Fox G.C, Granat R., Grant L., Lyzenga G., McLeod D., Pallickara S., Parker J., Pierce M., Rundle J., Sayar A., and Tullis T. "iSERVO: Implementing the International Solid Earth Research Virtual Observatory by Integrating Computational Grid and Geographical Information Web Services" Technical Report December 2004, to be published in Special Issue for Beijing ACES Meeting July 2004.

[Alameh2003] Alameh N., Chaining Geographic Information Web Services, IEEE Internet Computing, Sept-Oct 2003, 22-29.

[Beaujardiere2002] Jeff De La Beaujardiere, OpenGIS Consortium Web Mapping Server Implementation Specification 1.3, OGC Document #04-024, August 2002.

[Bellwood2003] Bellwood, T., Clement, L., and von Riegen, C. (eds) (2003), UDDI Version 3.0.1: UDDI Spec Technical Committee Specification. Available from http://uddi.org/pubs/uddi-v3.0.1-20031014.htm.

[Berman2003] Fran Berman, Geoffrey C, Fox, Anthony J. G. Hey., Grid Computing: Making the Global Infrastructure a Reality. John Wiley, 2003.

[Booth2004] Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. "Web Service Architecture." W3C Working Group Note, 11 February 2004. Available from http://www.w3c.org/TR/ws-arch.

[Box2000] Don Box, David Ehnebuske, Gobal Kakivaya, Andrew Layman, Dave Winer., Simple Object Access Protocol (SOAP) Version 1.1, May 2000.

[Cox2003] Simon Cox , Paul Daisey, Ron Lake, Clemens Portele, Arliss Whiteside, Geography Language (GML) specification 3.0, Document #02-023r4., January 2003.

[crisisgrid] GIS Research at Community Grids Lab, Project Web Site: http://www.crisisgrid.org.

[crisisgridwfs] Aydin G., SERVOGrid WFS implementation web page: http://www.crisisgrid.org/html/wfs.html

[crisisgridfthpis] Aktas M., SERVOGrid Information Services Web Site, http://grids.ucs.indiana.edu/~maktas/fthpis.

[Christiensen2001] Christiensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, Web Service Description Language (WSDL) Version 1.1, March 2001.

[Ferraiolo2003] Ferraiolo, Dean Jackson, Scalable Vector Graphics (SVG) Sprcification 1.1., January 2003.

[Foster2004] Foster, I. and Kesselman, C., (eds.) The Grid 2: Blueprint for a new Computing Infrastructure, Morgan Kaufmann (2004).

[FoxTech2003] Geoffrey Fox, et al, Complexity Computational Environnent (CCE) Architecture. Technical Report available from http://grids.ucs.indiana.edu/ptliupages/publications/CCE%20Architecture.doc

[Gang2005] Zhai G, Fox G., Pierce M., Wu W., Bulut H. eSports: Collaborative and Synchronous Video Annotation System in Grid Computing Environment IEEE International Symposium on Multimedia (ISM2005) December 12-14, 2005 Irvine, California, USA.

[Garret] Jesse James Garret, Ajax: A New Approach to Web Applications. http://www.adaptivepath.com/publications/essays/archives/000385.php

[gis] OGC (Open Geospatial Consortium) official web site http://www.opengeospatial.org/

[Kendall1994] A Note on Distributed Computing, S. C. Kendall, J. Waldo, A. Wollrath, G. Wyant, A Note on Distributed Computing, Sun Microsystems Technical Report TR-94-29, November 1994. Available from http://research.sun.com/techrep/1994/abstract-29.html.

[Kolodziej2003] Kris Kolodziej, OGC OpenGIS consortium, OpenGIS Web Map Server Cookbook 1.0.1, OGC Document #03-050r1, August 2003.

[Lalonde2002] Lalonde, W. (ed.), Styled Layer Descriptor(SLD) Implementation Specification 1.0.0, OGC Document #02-070, August 2002.

[Little2004] Mark Little, Eric Newcomer, Greg Pavlik., OASIS Web Services Context Specifications (WS-Context) 0.8. November 2004.

[naradabrokeringurl] Message based middleware project at Community Grids Lab, Project Web Site: http://www.naradabrokering.org/

[NasaOnearth] Project OnEarth at NASA JPL (Jet Propulsion Lab) http://onearth.jpl.nasa.gov/

[Nebert2004] Douglas Nebert, Arliss Whiteside, OpenGIS Consortium Catalogue Services Specifications 2.0. OGC Document# 04-021r2, May 2004.

[ogcregistry] Open GIS Consortium Inc. OWS-1 Registry Service. 2002-07-26.

[Pallickara2005] Pallickara S., Bulut H., Burnap P., Fox G., Uyar A., Walker D. Support for High Performance Real-time Collaboration within the NaradaBrokering Substrate Technical Report May 2005.

[Pierce2003] Marlon Pierce, Choonhan Yoon and Geoffrey Fox: Interacting Data Services for Distributed Earthquake Modeling. ACES Workshop at ICCS June 2003 Australia.

[Sayar2005] Sayar A., Pierce M., Fox G.C. OGC Compatible Geographical Information Services Technical Report (Mar 2005), Indiana Computer Science Report TR610

[Sonnet2003] Jérôme Sonnet, Charles Savage. OGC Web Service Soap Experiment Report 0.8 Document#03-014, Jan 2003.

[Tiampo2002] Tiampo, K. F., Rundle, J. B., McGinnis, S. A., & Klein, W. Pattern dynamics and forecast methods in seismically active regions. Pure Ap. Geophys. 159, 2429-2467 (2002).

[Vretanos2003] Vretanos, P. (ed.), Web Feature Service Implementation Specification (WFS) 1.0.0, OGC Document #02-058, September 2003.

[xslurl] W3C XSL Web Site : http://www.w3.org/Style/XSL/

[webserviceurl] Web Services Technologies http://www.w3.org/2002/ws/
[Wu2004] Wenjun Wu, Geoffrey Fox, Hasan Bulut, Ahmet Uyar, Harun Altay “Design and Implementation of A Collaboration Web-services system”, Journal of Neural, Parallel & Scientific Computations (NPSC), Volume 12, 2004.