

*Integration Framework for
Map Services and Geo-Science Grids*

By Ahmet Sayar

Research Committee:

- *Prof. Geoffrey C. Fox (Principal Advisor)*
- *Prof. Randall Bramley*
- *Prof. Kay Connelly*
- *Prof. Melanie Wu*

*Indiana University - April, 2007
Computer Science - Community Grids Lab (CGL)*

Introduction

Geographic Information Systems (GIS) [2][9] and specifically Map Services are very crucial for the spatial decision making and situation assessment. Map Services play the key role in the situation assessment through accessing and rendering the data to create comprehensible representations. Maps are composed of layers created with spatial data sets which are mostly kept in databases or in plain text files. Geo-Science applications are earth related scientific applications and using spatial and temporal data for their simulations. Since Geo-Science applications require quick response times, the GIS services must enable accessing and processing these large data sets in a reasonable time period.

This document defines an architectural framework for integrating Map Services with Geo-Science grids [11][12]. In this framework, Geo-Science grids' requirements and spatial data characteristic introduces performance and interoperability challenges. We also take the extensibility and generality issues into considerations to create an efficient framework to be applied on any Geo-Science Grids.

Map Services and Geo-Science grids integration is done at the layer level. We enable this by introducing 3-layer structured display. In this structure, bottom layers come from Map Services through coverage portrayal services (CPS) [13] or WMS, middle layer set comes from Map Services through Web Feature Services (providing vector data) and top layer set comes from Scientific Plotting (Sci-Plotting) Services. Sci-Plotting services enable simulation outputs of Geo-Science grids to be overlaid on top of the 3-layer structure. Since all the services are created in Web Service principles, it is possible extending the framework with any other third-party service.

In addition to implementing all the services in Web Service principles, in order to provide interoperability, we use Open Geospatial Consortium (OGC) [1] and ISO/TC211 standards. For the data interoperability, we use OGC's GML [10] specification standards. It defines vector data in feature collections. Since XML provides redundant description of the content and the structure of the content by using tag elements, actual data become much larger than its size in raw format and results in poor performance in transferring and rendering the data (See *Figure 3*). In this paper, we deal with the performance issues and introduce innovative techniques at the communication and transaction level. We do not consider the issues at Database level.

We first explain the proposed integration framework in Chapter 1. In Chapter 2, we summarize the challenges in making the integration framework applicable to real Geo-Science grids. Chapter 2.1 mentions about performance issues mostly coming from Geo-Science grids' requirements and spatial data characteristics. Finally Chapter 3 proposes a solution to the performance problems in the integration framework in order to make the framework applicable to Geo-Science Grids.

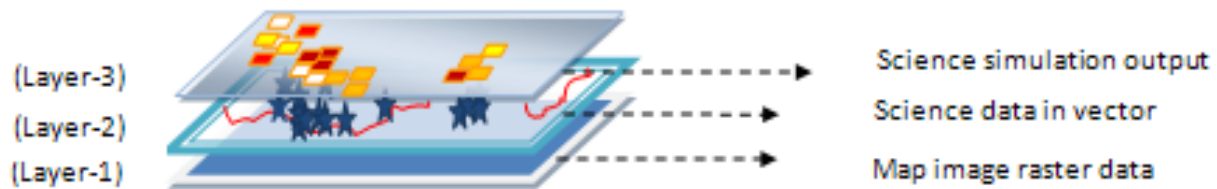
Before proceeding with this document, it is recommended for you to have a look at author's blog [7] and a complementary document about Map Services and GIS [16].

1. Integrating Map Services and Geo-Science Grids in SOA Principles

Integration is done at the layer level. Therefore, the Geo-Science Grids' outputs and Map Servers outputs should be represent-able in layers and synchronized as well. Our OGC compatible Map services provide their data in layers. We also use Google Map servers as Map services by embedding an intermediary middleware. This is explained in another document.

Regarding Geo-Sciences' outputs, we need to convert them into layers to integrate (overlay) them with the Map Services' layers. In order to do that we created Scientific Plotting Web Services. It is based on Dislin plotting libraries. After having wrapped them as Web Services [4], we enabled these libraries to be used by Science community.

There are three set of layers used to build a map with scientific data and information. Layer order is important to create reasonable and human interpretable maps. Please see the Figure 1. *Layer-1* is the bottom layer created from raster data such as Google Earth and coverage data in image format (comes from Web Map Services). *Layer-2* is created from vector data such as state boundaries and seismic data coming from Web Feature Service, and *Layer-3* is created from processed data coming from simulation outputs of the Geo-Scientific applications. *Layer-3* is created at Sci-Plotting Server.



Three-Layer Structure, maps are composed of three classes of layers in above orderings.

Figure 1: Integration output is three-layer structured maps. Layer-3 comes from Sci-Plotting, Layer-2 comes from WFS and Layer-1 comes from WMS or Coverage Portrayal services.

Related to layer structure above there are three major components in the architecture (Figure 2). These are Map Services, Sci-Plotting Services and User Portal (services in grey colors in Figure 2). Regarding the relation of the components and layers; *Layer-1* and *Layer-2* come from Map Services and *Layer-3* comes from Sci-Plotting services. Interactive user portal provides an independent browser based GUI enabling interaction with GIS services while hiding system complexity from the users

1.1. Major Components in the Framework (Figure 2)

Web Feature Service (WFS) (by G. Aydin) [8] provide geographical information as GML feature collections. Data is kept in relational Database and upon request; WFS convert it to GML and returns. WMS interact with a Web Feature Service by submitting database queries encoded in OGC's Filter Encoding and in compliance with the OGC Common Query Language. It is an OGC standard GIS service

implemented in Web Service principles. Basic WFS has three common interfaces. These are “getCapabilities”, “getFeature”, “DescribeFeatureType”.

Web Map Service (WMS) [5][6] enables visualizing, manipulating and analyzing geospatial data through maps shown on browser based interactive GUI. Map Servers typically compose maps in layers. Layers may come from distributed sources: Web Feature Services provide abstract feature representations that can be converted to images, and other Map Servers may contribute map images such as NASA WMS in Figure 2. WMSs can be federated and cascaded to create more detailed and comprehensible map images. WMS is an OGC standard GIS service implemented in Web Service Principles. Basic Web Map Service provides three functionalities. These are “getCapabilities”, “getMap”, “getFeatureInfo”.

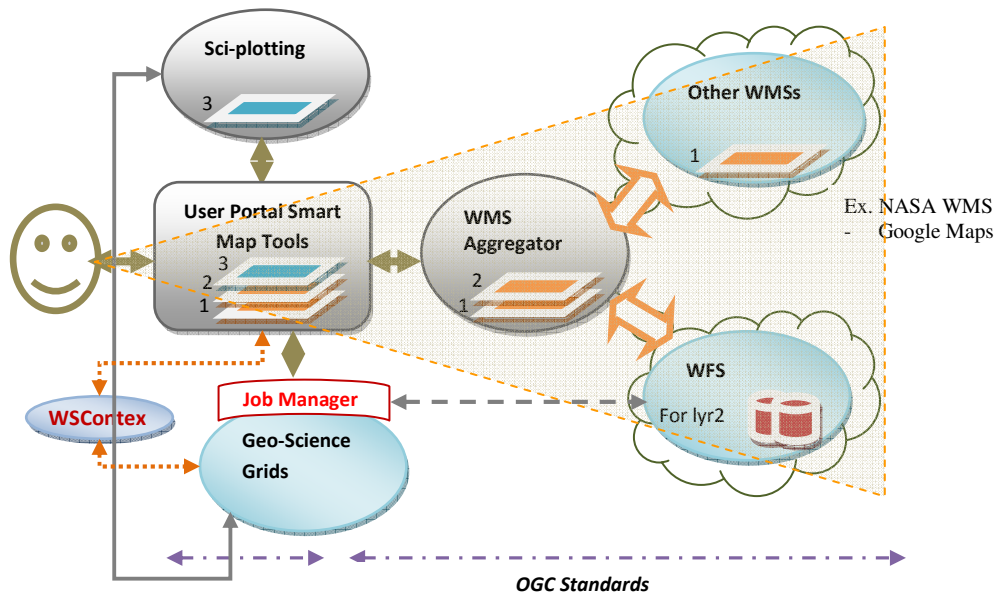


Figure 2: Map Services and Geo-Science Grids integration framework. WFS and Job Manager (interface to Geo-Science Grids) are other CGL-Lab research projects

Aggregator WMS [5][9] is actually a WMS with some extensions. It provides all the interfaces and functionalities that any other OGC compatible WMS provides in OGC standards. Furthermore, Aggregator WMS also provides Google Maps in the image format such as jpeg which can be archived and manipulated depending on the application aims. Aggregator WMS is expected to improve the performance compared to conventional WMS. It uses our innovative pre-fetching, load balancing and caching techniques (see Section 3). It also provides different running modes set before run-time such as streaming or non-streaming data transfer modes.

Sci-Plotting Services: For the core functionality we use Dislin [??] scientific-data plotting libraries. Dislin is a plotting library containing functions for displaying data graphically as curves, graphs, pie-charts, 3-D color plots, surfaces and contours. Some of these services are wrapped as Web Services and integrated into the general visualization system as illustrated in Figure 2. Through these services, Sci-Plotting provide Web Services interfaces to interpret the data in more detail (in graphs or charts) or plot scientific data (in layer set 3) to be overlaid on top of the GIS maps (see Figure 1).

Job Manager (by H. Gadgil) [19] is actually HPSearch project developed at CGL. It is simply a scripting environment for managing distributed workflows. Different Geo-Science applications (PI or Virtual California) require different set of parameters for the application users to trigger the job manager. This set of parameters and their order are defined earlier by the Job manager. Users provide required parameters through the project's user interface module deployed in to the user portal and trigger the application to run. After the application or science grids finish the task, job manager send the output link to the user waiting at the user portal. user's communication with the Geo-Science application is done job manager.

WS-Context [20]'s specification is defined by OASIS (Organization for the Advancement of Structured Information Standards). When multiple Web services are used in combination, the ability to structure execution related data called context becomes important. This information is typically communicated via SOAP [15] Headers. WS-Context provides a definition, a structuring mechanism, and a software service definition for organizing and sharing context across multiple execution endpoints.

Geo-Science Grids: Geo-Science grids (or applications) are based on the data (called as features, spatial data or temporal data) related to the earth. They are defined by coordinates, bounding boxes, spatial reference systems, projections, geometry elements and, some other geo-related attributes and elements. See the Chapter 1.3 for motivating use cases and sample Geo-Science Grids.

1.2. How the components work all together

(Explaining on one of ServoGrid project [22] –Pattern Informatics [21]):

The integration framework enables scientific users to analysis the three set of data. These data sets correspond to layers in three layer structure illustrated in Figure 1, vector data from WFS, raster data from third-party WMS and, plots in image types from Sci-Plotting servers. The process of analysis involves data mining which is made using interactive smart map tools and interactive integration tools at the user portal. User portal is actually an independent browser based GUI enabling interaction with GIS services while hiding system complexity from the users. It is responsible for collecting parameters for invoking Map Services (for layer sets 1 and 2), triggering Geo-Science Grids (for Layer set 3) and integrating (overlying) their outputs. It also enables querying the data corresponds to layer set 2, and analyzing the results interactively. The process is diagrammatically illustrated in Figure 2.

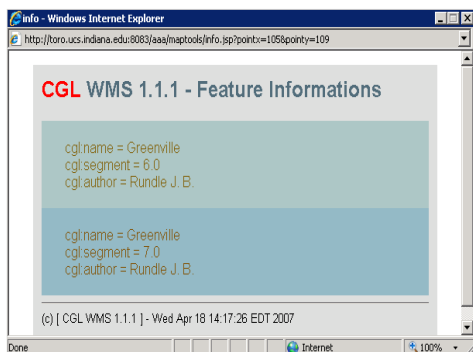
Layer sets 1 and 2 are created asynchronously together at the aggregator WMS. Layer set 3 is created with separate services and procedures. Layer 3 is the top layer overlaid on others and, it is created from the Geo-Science Grids' simulation outputs.

Jobs regarding layer sets 1 and 2 involve most commonly used OGC standard GIS services which are implemented in Web Services principles such as Web Map Services and Web Feature Services. Since the application framework is implemented according to OGC standards, we can easily integrate any other third party OGC GIS services into the framework. Besides, any third party GIS application can use our implementation of WMS and WFS easily.

All the layer sets 1 and 2 are provided by aggregator WMS. It is actually OGC compatible WMS. Aggregator WMS keeps capability metadata providing information about data and service. Users (using integration portal) are informed of available data and layers by Aggregator WMS. Depending on the Geo-Science applications' data-layer requirements, it can upgrade itself by adding the new WFS and/or WMS to the framework and upgrade its capability metadata about its data holdings in layers. OGC standards allow using other service data as if your own by updating your capability and adding enough information about the data. OGC calls it cascading the services. In our framework, we cascade a third party WMS from NASA's OnEarth project. This is called layer set 1 in our proposed framework (see Figure 1 and Figure 2).

Integration portal dynamically update its layer list in the GUI every time it connect to aggregator WMS. It means whenever user opens the browser (since aggregator WMS is defined as default WMS), portal gets the capabilities file from the aggregator WMS through the "getCapabilities" Web Service and, based on this capabilities file it updates its provided layer information in the browser.

Aggregator WMS provides layer sets 1 and 2 with its "getMap" Web Service. Layer 1 is returned in image MIME type such as image/jpeg as DataHandler object attached to SOAP message. Layers belonging to layer set 1 are created from coverage data. Since we have not implemented coverage portrayal Service (CPS) we get them from other WMS (NASA WMS) but we can still add this data as if we provide by using OGC's cascaded WMS properties, as mentioned earlier. Layer 2 is overlaid on layer 1. Layer 2 is created from vector data such as lines and points or any combinations of them. Layer 2 is provided by WFS. WFS keeps these data in the relational Databases.WMS sends a "getCapabilities" request to WFS to learn which feature types WFS can service and what operations are supported on each feature type. Depending on the returned capabilities files of the WFSs to which WMS connected, WMS updates its capability file with returned capability files.



Data used for creating layer set 2 is queried interactively through WMS's "getFeatureInfo" Web Service. When any WMS client sends a getFeatureInfo request to WMS, WMS creates a getFeature request and sends it to WFS. The Web Service address of the WFS is found by looking into capabilities file. After choosing appropriate WFS, WMS transfer feature data from WFS according to architecture defined in Section 2.1.1. Returned feature data is converted to comprehensible format and sent to WMS client. See the sample window popping up at client's browser.

Regarding layer set 3, the user submits a flow for triggering Geo-Science Grid by invoking the job manager Web Service through user portal. The request to job manager includes all the parameters required for execution of the script. The job manager system works in tandem with a context service for communicating with the WMS. The context service is a distributed, high performance registry service useful for storing session and other context related data.

When integration portal user submitted request script, the job manager engine invokes and initializes the various services, namely the Data Filter service, that filters incoming data and reformats it to the proper input format as required by the application runner, and the application runner runs the application on the mined data. Once initialized, the job runner proceeds to execute the WFS Web Service with the appropriate GML (Geographical Markup Language) query as input. This query is exactly the same as the query made by WMS to WFS to create layer set 3. (Since Geo-Science Grids's output will be overlaid on top of the GIS map they must have same bbox values). The WFS then outputs the result of the query onto a topic (data transfer using Naradabrokering) specified by job runner (dashed lines in Figure 2). The Geo-Science grid runs on the returned data and the resulting file is sent to a publicly accessible Web server. The URL of the generated file is then written to the context service and returned to the integration portal by job manager.

When the user triggers the Geo-Science application through integration portal and job manager, job manager sets the context in WS-Context, starts application and get a session id. This session id is also sent to the portal by job manager. Whenever the job is done, job manager sets the context belonging to the session id as "done" at the WS-Context server. User is informed through WS-Context if the job is done or still running. The integration portal constantly polls the context service to see the application's current status. Once it is signaled that the execution is finished, the integration portal invokes the Sci-Plotting server for having it to plot layer set 3. Integration portal provides all the necessary parameters such as the link to the Geo-Science output and some other format related parameters. After all, Sci-Plotting server downloads the result file from the web server and sends it back to integration portal. Integration portal displays the output by overlaying plotted data as a layer at the top of the map (Layer set 3 in Figure 1).

1.3. Motivating Use Cases

1.3.1. iSERVO GIS Grid Research Project

iSERVOGrid project [22] 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) 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 two projects initially in order to apply our integration framework. These are Pattern Informatics (PI) and Virtual California (VC).

1. *Pattern Informatics (PI)*: Pattern Informatics (PI) application is used to produce the well-publicized "hot spot" maps published by SERVO team member Prof. John Rundle and his group at the University of California-Davis. PI analyzes earthquake seismic records to forecast regions with high future seismic activity. It also identifies the characteristic patterns associated with the shifting of small earthquakes from one location to another through time prior to the occurrence of large earthquakes.

We run PI code through the user portal and plot the possibilities of the earthquake happenings in color-coded grid over the previously created seismic and earth map (see the below sample). Seismic data are kept in WFS and queried based on the user provided criteria. We use NASA OnEarth Map server as cascaded WMS and get earth satellite image (Layer set 1 in Figure 1). We get earthquake seismic data (Layer set 2 in Figure 1) from the Web Feature Server and overlay it on Layer 1. PI output is column-tabular data in plain text file. It is used for creating layer set 3 at Sci-Plotting server.

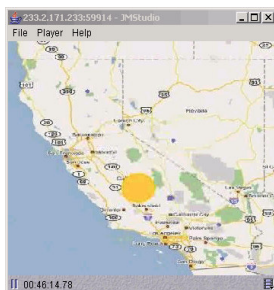


Layer 3 is created from the PI simulation outputs. Sci-Plotting server gets the output through HTTP protocol and plot the image as layer. The result is a map which shows the fluctuations in seismic activity which are found to be related to the preparation steps for large earthquakes. This map layer is overlaid on top of the map (consisting of layer sets 1 and 2) coming from aggregated Map Server. The result PI map shows regions with hotspots where earthquakes are likely to occur during a specified period in the future. Possibilities are ranked from 0(no earthquake happening) to 100(definite earthquake happening) and related to color scale bar. 0 corresponds to light yellow color and 100 corresponds to dark red color.

Communication between user integration portal and job manager is done via WS-Context. Whenever job is done Job Manager notify the user by setting session's specific parameter at WS-Context.

2. *Virtual California (VC)*: Earthquake simulation model for the California. The simulation takes into account the gradual movement of faults and how they interact with each other. It includes 650 segments representing the major fault systems in California, including the San Andreas Fault responsible for the 1906 San Francisco earthquake.

VC has 2-phase run. In the first phase user runs the application by giving required parameters and get the result (the best cost) on his screen. If he likes the cost he runs the second-phase with the returned best cost and some other parameters given through VC GUI to get the forecast values. The result forecast values are played in a movie streams (see the below sample run with JMF Client). Each frame in the stream is actually a three-layer structured static map. Layer sets 1 and 2 are created first as a map by using NASA OnEarth Map server as cascaded WMS through aggregator WMS and, get earth satellite image (Layer set 1 in Figure 1). We get earthquake seismic data (Layer set 2 in Figure 1) from the Web Feature Server again through aggregator WMS and overlay it on Layer 1. VC simulation output (after second phase) is also column-tabular data in plain text file.



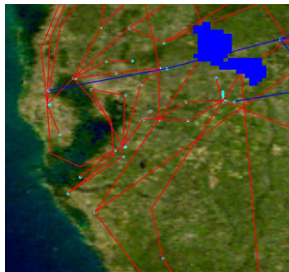
Layer 3 is created at the Sci-Plotting server with the data set coming from VC's simulation. The required parameters for creating layer-set 3 are given by the user through GUI of integration portal. Periodicity of the data for creating movie frames is defined by user interface. We plot layer-set 3 in circular and color coded shapes by using coordinate values, date and time of occurrences and, magnitude values of the simulation output. For the demo and sample output movie streams see the project page [3].

Communication between user integration portal and Job Manager is done via WS-Context. Whenever job is done Job Manager notify the user by setting session's specific parameter at WS-Context.

1.3.2. IEISS GIS Grid Research Project:

The Interdependent Energy Infrastructure Simulation System (IEISS) [23], embodied as analysis software tools developed at Los Alamos National Laboratory with the collaboration of Argonne National Laboratory (ANL), aims at developing a comprehensive simulation study of the nation's interdependent energy infrastructures to address wide variety of intra-and inter-infrastructure dependency questions. The IEISS analysis tool has physical, logical, or functional entities that have variety of attributes and behaviors that mimic its real-world counterpart.

During our research we have worked with IEISS people at ANL to integrate their project to Web Map Services by using our proposed integration framework. Traditionally IEISS is run as a desktop application with input data supplied as XML files collected from various sources, and the result is locally generated. We have used our framework and tried to embed their services web service counterparts. Additionally our user portal for the integration allowed users to select geographical regions on the maps where the simulation is executed. As layer 1 set we used NASA OnEarth Map server layers, as layer 2 set we used gas pipelines and electric power lines represented as feature data in GML. We kept them in Database and provided to the Map Server through WFS service which are all web services. After getting maps covering gas pipelines and electric power lines on the NASA satellite map images, we start running IEISS code from the user portal's smart map tools and integration tools.



As Layer 3 set IEISS people needed to see where the outage areas (blue region) happen and query the area and see the data in text. For example which gas pipeline (blue lines) is broken or what electric lines (red lines) do not provide electricity. IEISS simulation code sends out these data in numbers. We run the code asynchronously and get the output through WS-Context and render the data at Sci-Plotting Service. Sci-Plotting server sends the layer set 3 to user portal and, user integration portal overlay the outage map layer top op the map returned by aggregated Map Server.

2. Research Challenges

Extensibility, Interoperability and Performance of the Framework

Title lists the challenges of the proposed integration framework. This chapter explains our approaches to come up with the listed challenges, especially the performance issues.

Extensibility: We have created our proposed framework in Web Service principles [4]. Each service in the framework is a Web Services. Therefore, they can be used by outsiders and, any services (GIS) created in Web Service principles can be integrated to the framework. Web Services provide loosely coupling of services and extensibility.

Interoperability: In order to provide interoperability we use OGC and ISO/TC211 standards, in addition to creating services in Web Service principles. OGC defines vector data in feature collections and features are encoded in GML. GML is an XML based data representation. As it is very well known, using XML based data representation decreases the performance severely. Geo-Science applications use large data in size. Coding them in XML increases the size. There is a trade off here that we want to solve in the following chapters.

Performance: We consider how to transfer data from WFS to WMS, how to parse and render the XML based large feature data collections efficiently and, how to create efficient communication channels. Regarding the data transfer from WFS to WMS, we have improved the performance by implementing streaming data transfer. Streaming versions of WMS is implemented by using NaradaBrokering publish-subscribe based messaging middleware [14].

Bottleneck of the proposed integration framework is “orange rectangle” (illustrated in Figure 2). From now on, our focus will be on improving this rectangle’s performance. Therefore, we introduce innovative techniques to increase the performance of the system to meet Geo-Science grids’ performance requirements.

2.1. Current Performance Work

Our experience shows that although by using Web Services we can easily integrate several GIS and other services into complex tasks, providing high-rate transportation capabilities for large amounts of data remains a problem because the pure Web Services implementations rely on SOAP messages exchanged over HTTP. This conclusion has led us to an investigation of topic-based publish-subscribe messaging systems for exchanging SOAP messages and data payload between Web Services. We have used NaradaBrokering which provides several useful features besides streaming data transport such as reliable delivery, ability to choose alternate transport protocols, security and recovery from network failures. Detailed information about the data transferring is given in Section 2.1.1.

Regarding the rendering of large XML based scientific data and creating comprehensible representations in map images we use parsers. We have used DOM (Document Object Model), SAX (Simple API for XML – push model) and finally pull parsers. Since we use standard service interfaces and data, using pull parsing technique gives the best performance. Detailed information about pull parsers is given in Section 2.1.2.

2.1.1. Data Transfer:

We make streaming data transfer from WFS to WMS by using Naradabrokering. Naradabrokering is a message oriented middleware (MoM) system which facilitates communications between entities through the exchange of messages.

In case of transferring the GML result set in the form of string causes some problems when the GML is larger than some amount of size. Since the WFS returns the resulting XML document as an `<xsd:string>`,

this has to be constructed in memory and the size will depend on several parameters such as the system configuration and memory allocated to the Java Virtual Machine etc. Consequently there will be a limit on the size of the returned XML documents. For these reasons we have investigated alternative ways for data transport and, researched the use of topic based publish-subscribe messaging systems for streaming the data. Our research on NaradaBrokering shows that it can be used to stream large amount of data between nodes without significant overhead. Additional capabilities such as reliable messaging and support for different transport protocols already inherent in NaradaBrokering show that it is a powerful yet easy to integrate messaging infrastructure. For these reasons we have developed a novel Web Map Service and Web Feature Service that integrate Open Geospatial Consortium specifications with Web Service-SOAP [15] calls and NaradaBrokering messaging system.

In case of streaming through Naradabrokering, the clients make the requests with standard SOAP messages but for retrieving the results a NaradaBrokering subscriber class is used. Through first request to Web Service called *getFeature*, WMS gets the topic (publish-subscribe for a specific data), IP and port on which WFS is streaming requested data. Second request is done by NaradaBrokering Subscriber. Even whole data is not received by WMS; WMS can draw the map image with the returned science data. This depends on the WMS's internal implementation.

2.1.2. Data Parsing and Rendering:

We use Pull parsing technique in our framework to parse XML based geo-science data. Pull Parsing is an approach to validate and parse XML documents.

Pull parser only parses what is asked for by the application rather than passing all events up to the client application as SAX parsing does. The pull approach of this parsing model results in a very small memory footprint (no document state maintenance required – compared to DOM), and very fast processing (fewer unnecessary event callbacks - compared to SAX).

Pull parsing does not provide any support for validation. This is the main reason that it is much faster than its competitors. Since all our services are OGC compatible and created in Web Service principles, we do not necessarily need validation. In OGC, services describe themselves by capability document and servers know each other by exchanging these document. If you are sure that data is valid (as in our case), or if validation errors are not catastrophic to your system, or you can trust validity of the capabilities document of the server you are in contact, then using XML Pull Parsing gives the highest performance results. For example in communication between WFS and WMS, since we know that WFS provides feature data in OGC's GML format, it is very advantageous skipping validation and using "pull parsing". Please see the article where pull parsing is compared with other leading Java based XML parsing implementations [17].

Figure 3 shows current system's performance result improved with streaming data transfer and pull parsing techniques mentioned so far.

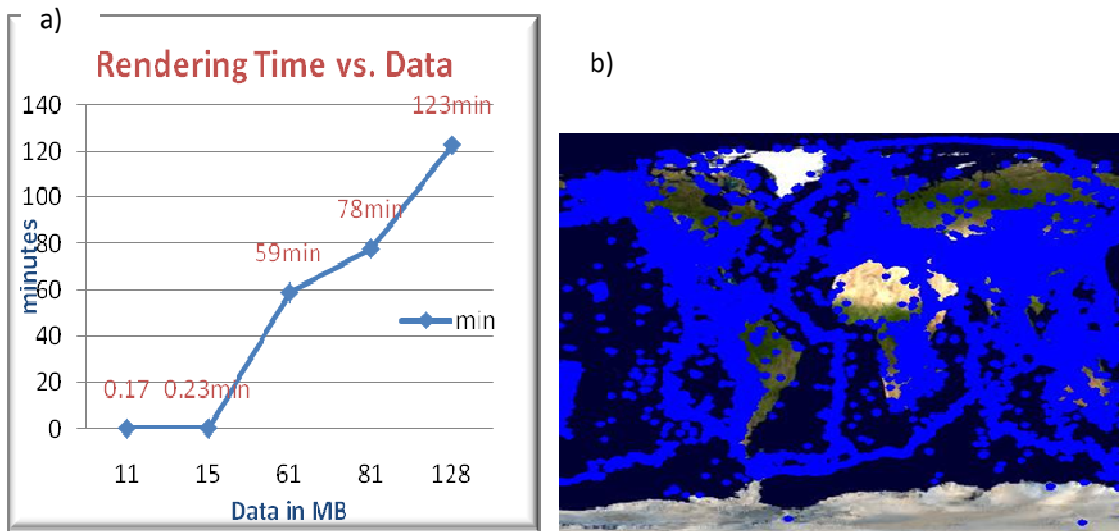


Figure 3: (a) Performance result of the current system. (b) Sample output consisting of Layer-1 and 2. See Figure 1.

This figure teaches us valuable lessons in terms of the capabilities and limits of our implementation. From the below result we draw following conclusions. First, for small data payloads (less than 15MB) the response time is acceptable. However for larger data sets (more than 20MB) the performance decreases sharply and the response time is relatively long. Second, there exists a maximum threshold for the amount of data to be transported from WFS to WMS and rendered at WMS.

Current system test above shows that the performance is still not enough in order to meet Geo-Science grids' performance requirements. As you see, if the spatial data is over 20MB, integration framework is not feasible to use. Time column (y) in the Figure 3 (a) includes querying, transforming, rendering and displaying spatial data. In other words,

$$time_{(at\ the\ column)} = time_{(map\ is\ shown)} - time_{(client\ makes\ request\ for\ the\ map)}$$

2.2. Future Directions on Performance Issues

Load balancing, Caching and Pre-fetching

These are the three main means coming to mind to improve the performance. In this chapter, we take them one by one and analyze their applicability to the integration framework. These concepts change a lot from domain to domain. We analyze these approaches in the domains of spatial data, Map Services and GIS.

Load balancing is the one of the best known techniques to improve the performance but because of the characteristics of the spatial data (variable sized and un-evenly distributed) it is not easy to implement and get efficient performance results by using commonly known load balancing techniques. Since we do not know the workload previously, the classic load balancing algorithms do not work for the variable sized and unevenly distributed data. The work is decomposed into independent work pieces, and the

work pieces are of highly variable sizes. It is not possible to estimate the size of total work at a given worker server. Problem is illustrated in Figure 4 in case of using four worker nodes.

Web Map Services are queried based on some criteria. Bounding box is the key criteria. Bounding box values are in the form of (minx, miny, maxx, maxy). For example, in Figure 4 (a) client needs a map of the earth defined by bounding box (bbox) value of (a,b,c,d).

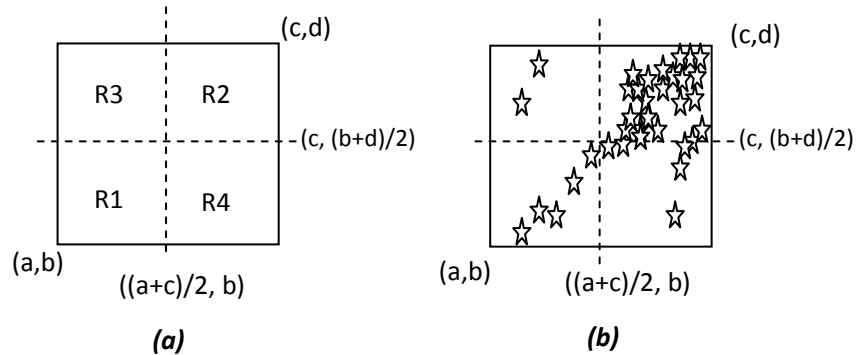


Figure 4: *Classic partitioning can not share the work equally among worker servers. Server assigned the partition of “((a+b)/2, (b+d)/2), (c, d)” gets the most of the work.*

Regarding **caching**, we need to figure out caching algorithm based on what to cache (map images or GML data), how long to keep and how to utilize from it. We plan to cache map images and keep them until next request comes. We utilize from cached data by extracting overlapping region from the cached map (see Figure 5) and prepare request for the remaining parts through rectangulation processes.

Finally, **pre-fetching** is getting the data before it is actually needed and keeping it in local server for the successive requests committed for the same data. In our framework we use archived scientific data. Archived data does not change often. So, it is not reasonable transferring and rendering the same data again and again for every request coming from the different or even the same users (current implementation). In order to solve this problem I am planning to use pre-fetching. Pre-fetching will be done between WMS and WFS based on the predefined periodicity such as once a day or once a week. Periodicity is defined previously depending on the data’s characteristics.

In summary, we can not get expected performance gains when we use load balancing in the way displayed in Figure 4. We therefore propose architecture (Section 3) composed of combination of these three approaches in order to overcome performance and scalability problems in rendering of large size distributed spatial data.

3. Enhanced Web Map Service Architecture for Geo-Science Grids

Dynamic Load Balancing with Caching over pre-fetched data

Web Map Services are enhanced with the performance considerations mentioned in Chapter 2. Architecture basically proposes performance enhancements over “orange rectangle” in Figure 2.

3.1. Brief Architecture

We take the caching and dynamic load balancing into considerations all together. Caching helps us to prevent redoing the jobs of querying and rendering for the data requested before.

Architecture is summarized in three orderly steps. These are

Routine: *Pre-fetching (Runs in a predefined periods asynchronously and independently)*

Run-time:

1. *Caching*
2. *Cached-data extraction and Rectangulation (Figure 5) on un-cached data*
3. *Merging the extracted cached data with the returned map images in response to rectangles*

In order to make these concepts more clear, let's give a concrete example: Example data is map image consists of NASA satellite base maps and earthquake seismic records (in blue dots). See Figure 5.

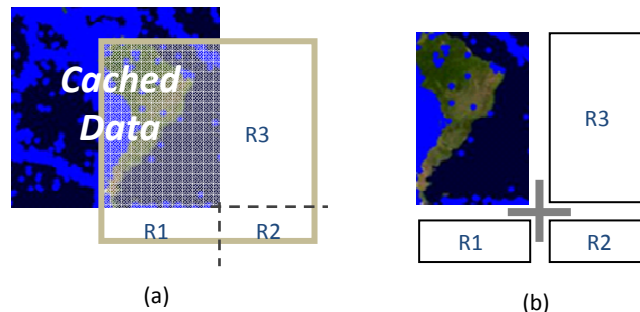


Figure 5: Illustration of the cached data extraction and rectangulation.

We get overlapped region of the data (as map image) through cached data extraction process, and remaining region in main query is partitioned through rectangulation process. According to OGC standards in GIS domain, queries are created with location parameter and location is defined in bounding box formats. Bounding box is a formula defining the region as a rectangle through coordinates of bottom left corner and top right corner. Ex $Q(\text{minx}, \text{miny}, \text{maxx}, \text{maxy})$.

Rectangles (R1, R2 and R3) go through the query creation process. Map Service creates `getFeature` request for each rectangle based on their bounding boxes. Other parameters and attributes (required for creating `getFeature` request) are obtained from the main request (big rectangle in Figure 5 a). We append overlapped cached map image with the sub-map images returned to the queries obtained through query processing over the rectangles R1, R2 and R3. Partitioning techniques are also applied on the query ranges when no cached data available (such as in case of first time calls). In that case, partitioning is done by dividing the region (which is defined by main query) into four equal sub-areas.

Load balancing is indirectly applied. Each rectangle obtained through rectangulation represents a worker Map Service. The sub images are obtained from these worker nodes asynchronously. In order to give

more detailed information about the architecture we explain the terms “caching”, “cached data extraction and rectangulation” in the following sub-sections.

3.1.1. Caching

In order to explain our caching techniques clearly we first need to explain the way Map Servers work. Our implementation of WMS is multi-threaded, so it can serve multiple clients at the same time and provides data in the MIME type image format. Therefore, if we store the cached images in local file system it will cause trouble because of that all the threads share the local file system. In order to prevent this, we keep cached images as class objects. Whenever map server needs to use the cached data, converts it into image for a specific client without confusion because of that each client (browser) has its own thread and each thread has its own instances of the classes.

Caching will be utilized just by the next request. In other words, cached image will be kept till the next request comes. In order for the successive request to utilize the caching, its layer numbers and names should be the same, otherwise it will be counted as first time request and caching will not be utilized. According to our caching implementation, we don't need to use any metadata or tool to see if the image is already in hand. There will be only one object (an image class object) cached in the system for one thread (for a session). We do not need any cache space in the local file system.

3.1.2. Cached data extraction and Rectangulation

The methodology here is to remove the regions in the main query which overlap with the cached data and then, create rectangular sub-regions from the remaining main query in the form of bboxes (see Figure 5). After removing the cached region from the requested bbox, we rectangulate the remaining parts as much as equally. Since we get rid of the cached data, the rectangulated regions obtained here look like the first time calls (no cached data left by previous request).

Rectangulation can provide two possible outcomes. The remaining regions of the main query can be divided into two or three rectangles depending upon the number of worker nodes available. Figure 5 illustrates a map figure configured to give three rectangles (R1, R2 & R3). Two rectangles can be formed by merging the R1 & R2 or R3 & R2 to form a single rectangle. We have not tested which one gives better results yet.

The bbox ranges of cached data and main query can be positioned to each other in four possible ways. These are (1) cached data covers main query, (2) cached data is covered by main query, (3) cached data and main query don't overlap and (4) they overlap partially. We explain data extraction and query rectangulation techniques for each group in the same order given above. Depending on their positions to each other, rectangulation techniques change.

Why we need to make rectangulation: Our services are OGC compatible and implemented in Web Service principles. They accept the requests in predefined XML-structured queries such as “*getMap*” Web Service for WMS and “*getFeature*” Web Service for WFS. Queries to WMS and WFS are actually window shape range queries. Range queries are formulated in bbox by OGC standards. After extraction of cached data falling in main query range, the remaining part needs to be converted to rectangular shapes in order to create sub-queries in bbox to get data falling into these regions from WFS. This is why we make rectangulation after cached data extraction from queried-region. Please see the Figure 5 for the sample rectangles obtained through rectangulation process.

3.1.3. Pre-fetching

In the proposed integration framework we use archived data provided by WFS in GML format in XML encoding. Archived data does not change often. So, it is not reasonable transferring and rendering the same data again and again for every request coming from the different or even the same users. In order to solve this problem we plan to use pre-fetching. Pre-fetching will be done between WMS and WFS based on the predefined periodicity depending on the data characteristic.

Pre-fetching is briefly defined as getting the data before it is needed. We accomplish the pre-fetching by the data transfer technique explained in Section 2.1.1. A performance result of the pre-fetching is displayed at Figure 4. Since it is done asynchronous manner and not at the run-time, it does not affect the proposed framework's overall performance. The performance result in Figure 4 also explains the current integration framework's performance (without using proposed architecture). WMS used to request data from WFS every time a new request comes.

Pre-fetching is done between Map Server and Web Feature Server. Through pre-fetching, XML encoded GML data is transferred by using Naradabrokering messaging middleware as explained earlier. The OGC's standard WMS and WFS specification are based on HTTP Get/Post methods, but this type of services have several limitations such as the amount of the data that can be transported, the rate of the data transportation, and the difficulty of orchestrating multiple services for more complex tasks. Web Services help us overcome some of these problems by providing standard interfaces to the tools or applications we develop.

WMS make the requests with standard SOAP messages but for retrieving the results a NaradaBrokering subscriber class is used. Through the "*getFeature*" Web Service, WMS gets the topic name (publish-subscribe for a specific data), IP and port on which WFS streams the requested data. Second request is done by NaradaBrokering Subscriber using the returned parameters.

GML data is provided by streaming WFS (implemented by G. Aydin) [8]. It uses standard SOAP messages for receiving queries from the clients; however, the query results are published (streamed) to a NaradaBrokering topic as they become available. We use JAVA libraries to implement pre-fetching. In order to do that, we define the task and timer. Task defines pre-fetching and, timer defines the running periodicity of the task.

4. Expected Results

Since we build our framework on XML and Web Services technologies, the figure showing current performance of the map rendering is not surprising (*Figure 3*). As you know, XML provides redundant description of the content and the structure of the content by using tag elements. This causes actual data become much larger than its actual size.

In order to satisfy Geo-Science grids' requirements regarding interoperability and extendibility, we should keep using XML and Web Service technologies. However, we should reduce or get rid of the performance bottlenecks of the system stem from XML and Web Service technologies. Because of the poor performance of the system (orange rectangle) regarding rendering and displaying maps consist of layer set 1 and 2 (see Figure 1), we mostly worked on these issues and proposed an approach in Chapter 3.

We also created GIS user portal enabling users to interact with the Map Services and Geo-Science Grids integration framework to run their pre-installed Geo-Science applications. User portal includes smart map tools providing interaction with highly available services while hiding system complexity from the users. You can see some demo running at the project web site [3].

I have tested the proposed architecture over Pattern Informatics [21] Geo-Science Grids. Pattern Informatics application use earthquake seismic data provided by Web Feature Service. I have not completed implementation of the whole architecture explained in Chapter 3. The performance figure below based on preliminary test results. I just got rid of the data transfer time by pre-fetching the data to see the timings of data rendering. Before run-time, I have pre-fetched whole data from WFS by setting the criteria of the query (getFeature) in its widest ranges (ex. Minimum magnitude value is set to 0). WMS responded to the successive requests by using pre-fetched data kept in its local file system, instead of going to WFS.

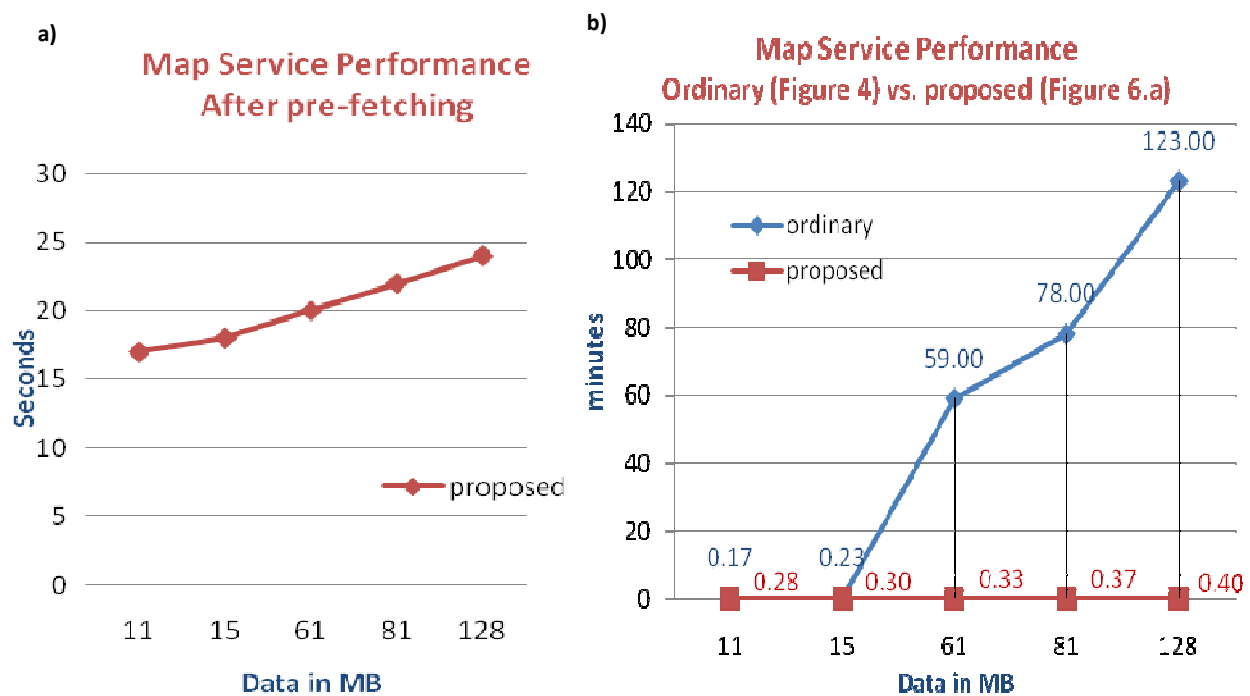


Figure 6: Preliminary Performance evaluation of enhanced Web Map Services with proposed arch.

We were measuring the time in minutes or hours (see Figure 3) but after the proposed architecture we measure the time in seconds (Figure 6 a). According to this figure, with the current standard implementation of integration framework and Web Map Services, we can transfer, render and display whole critical data (128 MB) in 123 minutes (Figure 6 b). Whereas, by using new proposed approaches (DLB with cached data extraction and rectangulation and, pre-fetching), the performance results will be measured in seconds.

REFERENCES

- [1] OGC (Open Geospatial Consortium) official web site <http://www.opengeospatial.org/>
- [2] GIS Research at Community Grids Lab, Project Web Site: <http://www.crisisgrid.org>.
- [3] Ahmet Sayar's project web site "Grid Map tools and Web Map Services page" available at <http://complexity.ucs.indiana.edu/~asayar/gisgrids/>
- [4] 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>.
- [5] Jeff De La Beaujardiere, OpenGIS Consortium Web Mapping Server Implementation Specification 1.3, OGC Document #04-024, August 2002.
- [6] Kris Kolodziej, OGC OpenGIS consortium, OpenGIS Web Map Server Cookbook 1.0.1, OGC Document #03-050r1, August 2003.
- [7] Ahmet Sayar's blog page available at <http://ahmetsayar.blogspot.com>
- [8] Vretanos, P. (ed.), Web Feature Service Implementation Specification (WFS) 1.0.0, OGC Document #02-058, September 2003
- [9] Ahmet Sayar, Marlon Pierce, Geoffrey Fox OGC Compatible Geographical Information Services Technical Report (Mar 2005), Indiana Computer Science Report TR610
- [10] Simon Cox, Paul Daisey, Ron Lake, Clemens Portele, Arliss Whiteside, Geography Language (GML) specification 3.0, Document #02-023r4., January 2003.
- [11] Fran Berman, Geoffrey C. Fox, Anthony J. G. Hey., Grid Computing: Making the Global Infrastructure a Reality. John Wiley, 2003.
- [12] Foster, I. and Kesselman, C., (eds.) The Grid 2: Blueprint for a new Computing Infrastructure, Morgan Kaufmann (2004)
- [13] Jeff Lansing., OWS1 Coverage Portrayal Service (CPS) Specifications 1.0.0, Document #02-019r1 February 2002.
- [14] Pallickara S. and Fox G., "NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids" ACM/IFIP/USENIX, Rio Janeiro, Brazil June 2003
- [15] Don Box, David Ehnebuske, Gobal Kakivaya, Andrew Layman, Dave Winer., Simple Object Access Protocol (SOAP) Version 1.1, May 2000,.
- [16] Ahmet Sayar, Research status report available at <http://complexity.ucs.indiana.edu/~asayar/proposal/summaryOfResearchWork.pdf>
- [17] Sosnoski, D. "XML and Java Technologies", performance comparisons of the Java based XML parsers. Available at <http://www-128.ibm.com/developerworks/xml/library/x-injava/index.html>
- [18] Ahmet Sayar, Technical report "Dynamic Load Balancing with Caching for Spatial Data Rendering" <http://complexity.ucs.indiana.edu/~asayar/proposal/loadBalancing5.pdf>
- [19] Harshawardhan Gadgil, Geoffrey Fox, Shrideep Pallickara, Marlon Pierce, Robert Granat, Proceedings of the IEEE/ACM Cluster Computing and Grid 2005 Conference, CCGrid 2005, Cardiff, UK. See also HPSearch Web Site, <http://www.hpsearch.org>.
- [20] Bunting, B., Chapman, M., Hurlery, O., Little M., Mischinkinky, J., Newcomer, E., Webber J, and Swenson, K., Web Services Context (WS-Context) Specification, Version 1.0, July 2003.
- [21] 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).
- [22] Chen, A., Donnellan, A., McLeod, D., Fox, G., Parker, J., Rundle, J., Grant, L., Pierce, M., Gould, M., Chung, S., and Gao, S., Interoperability and Semantics for Heterogeneous Earthquake Science Data, International

Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data, Sanibel Island, FL,
October 2003

- [23] Bush, B.W. and J.H. P. Giguere, S. Linger, A. McCown, M. Salazar, C. Unal, D. Visarraga, K. Werley, R. Fisher, S. Folga, M. Jusko, J. Kavicky, M. McLamore, E. Portante, S. Shamsuddin, *NISAC ENERGY SECTOR: Interdependent Energy Infrastructure Simulation System (IEISS)*, in *NISAC Capabilities Workshop*. 2003:Portland, OR.
- [24] Dislin project web site <http://www.mps.mpg.de/dislin>

APPENDIXES

1. Sample GetCapabilities request to WMS

```
<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>
```

2. Sample getMap request to WMS

```
<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>
    <Elevation>5.0</Elevation>
    <Time>01-01-1987/12-31-1992/P1Y</Time>
  </Map>
  <Image>
    <Height>400</Height>
    <Width>400</Width>
    <Format>video/mpeg</Format>
    <Transparent>true</Transparent>
    <BGColor>0xFFFFFFFF</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:States</ns1:Name>
      <ns1:Description>
        <ns1:Title>California:States</ns1:Title>
        <ns1:Abstract>California:States</ns1:Abstract>
      </ns1:Description>
    </ns1:NamedLayer>
  </ns1:StyledLayerDescriptor>
</GetMap>
```

3. Sample GetFeatureInfo for WMS – very simple one

```
<?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>0xFFFFFFFF</BGColor>
  </Image>
  <QueryLayer>
    Nasa:Satellite, California:Faults, California:States
  </QueryLayer>
  <InfoFormat>text/html</InfoFormat>
  <FeatureCount>999</FeatureCount>
  <x>117</x>
  <y>218</y>
</GetFeatureInfo>
```

4. Sample Capabilities file of WMS - a simple prototype

```
<?xml version='1.0' encoding="UTF-8" standalone="no" ?>
<!DOCTYPE WMT_MS_Capabilities SYSTEM "http://toro.ucs.indiana.edu:8086/xml/capa.dtd">
<Capabilities version="1.1.1" updateSequence="0">
  <Service>
    <Name>CGL_Mapping</Name>
    <Title>CGL_Mapping WMS</Title>
    <OnlineResource xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="simple"
      xlink:href="http://toro.ucs.indiana.edu:8086/WMSServices.wsdl" />
    <ContactInformation>
      ....
    </ContactInformation>
  </Service>
  <Capability>
    <Request>
      <GetCapabilities>
        <Format>WMS_XML</Format>
        <DCPType><HTTP><Get>
          <OnlineResource xmlns:xlink="http://w3.org/1999/xlink" xlink:type="simple"
            xlink:href="http://toro.ucs.indiana.edu:8086/WMSServices.wsdl" />
          </Get></HTTP></DCPType>
        </GetCapabilities>
      <GetMap>
        <Format>image/GIF</Format>
        <Format>image/PNG</Format>
        <DCPType><HTTP><Get>
          <OnlineResource xmlns:xlink="http://w3.org/1999/xlink" xlink:type="simple"
            xlink:href="http://toro.ucs.indiana.edu:8086/WMSServices.wsdl" />
          </Get></HTTP></DCPType>
        </GetMap>
      </Request>
      <Layer>
        <Name>California:Faults</Name>
        <Title>California:Faults</Title>
        <SRS>EPSG:4326</SRS>
        <LatLonBoundingBox minx="-180" miny="-82" maxx="180" maxy="82" />
      </Layer>
    </Capability>
  </Capabilities>
```

5. Sample GetFeature Request for WFS - for earthquake fault data:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<wfs:GetFeature outputFormat="GML2"
gml=http://www.opengis.net/gml
wfs=http://www.opengis.net/wfs
ogc="http://www.opengis.net/ogc">
  <wfs:Query typeName="fault">
    <wfs:PropertyName>name</wfs:PropertyName>
    <wfs:PropertyName>segment</wfs:PropertyName>
    <wfs:PropertyName>author</wfs:PropertyName>
    <wfs:PropertyName>coordinates</wfs:PropertyName>
  <ogc:Filter>
    <ogc:BBOX>
      <ogc:PropertyName>coordinates</ogc:PropertyName>
      <gml:Box>
        <gml:coordinates>-150,30 -100,50</gml:coordinates>
      </gml:Box>
    </ogc:BBOX>
  </ogc:Filter>
</wfs:Query>
</wfs:GetFeature>
```

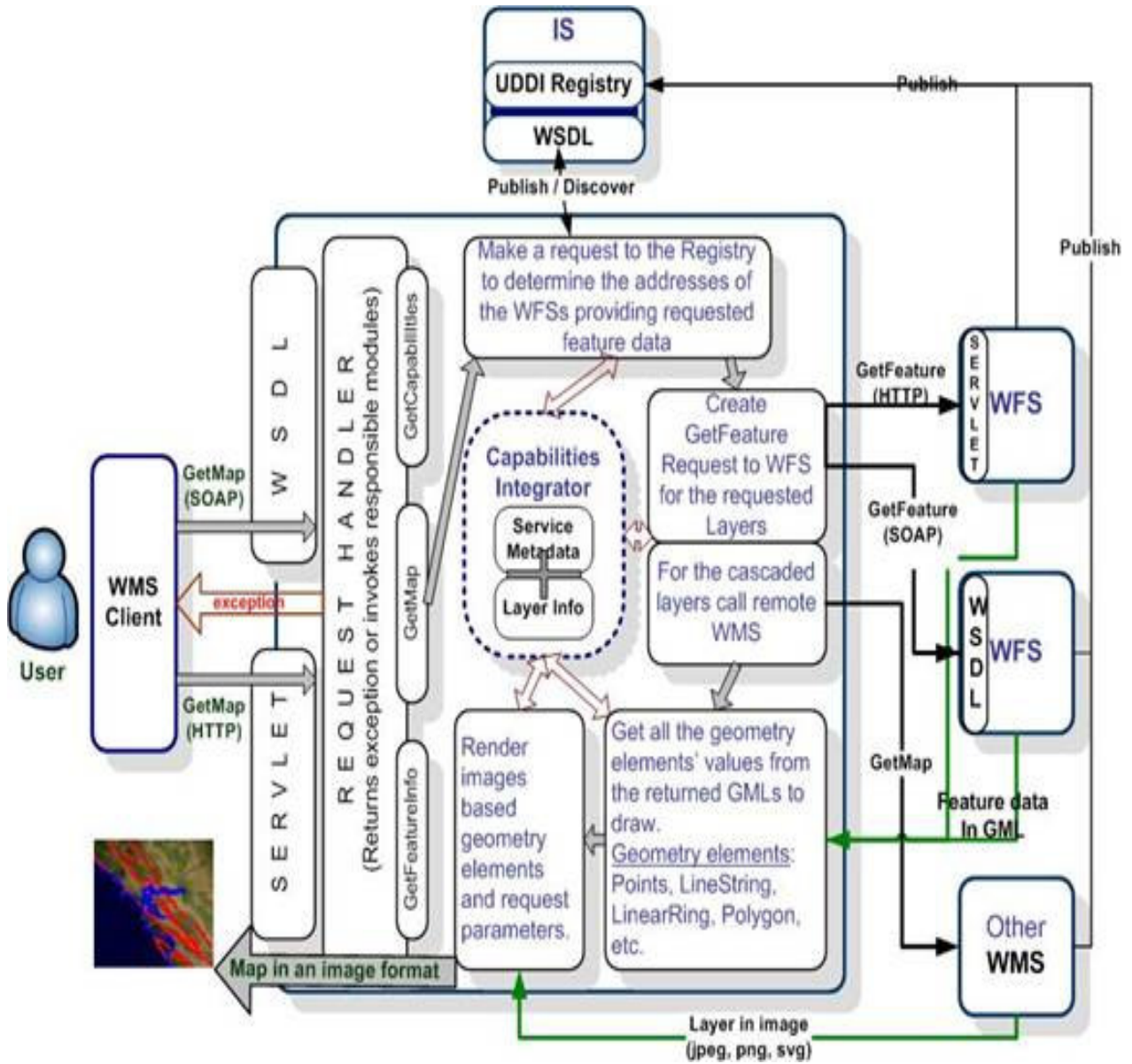
6. Sample GML document - for earthquake fault data. It might include thousands of feature.

```
<wfs:FeatureCollection
  xmlns:wfs=http://www.opengis.net/wfs
  xmlns:gml=http://www.opengis.net/gml
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xsi:schemaLocation="http://crisisgrid.org/schemas/fault_new.xsd">
  <gml:boundedBy>
    <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#27354">
      <gml:coordinates decimal="." cs="," ts=" ">
        -150,30 -100,50
      </gml:coordinates>
    </gml:Box>
  </gml:boundedBy>
  <gml:featureMember>
    <fault>
      <name>Bartlett Springs</name>
      <segment>0.0</segment>
      <author>Rundle J. B.</author>
      <gml:lineStringProperty>
        <gml:LineString srsName="null">
          <gml:coordinates>
            -123.05,39.57 -122.98,39.49
          </gml:coordinates>
        </gml:LineString>
      </gml:lineStringProperty>
    </fault>
  </gml:featureMember>
  <gml:featureMember>
    <fault>
      <name>Bartlett Springs</name>
      <segment>2.0</segment>
      <author>Rundle J. B.</author>
      <gml:lineStringProperty>
        <gml:LineString srsName="null">
          <gml:coordinates>
            -122.91,39.41 -122.84,39.33
          </gml:coordinates>
        </gml:LineString>
      </gml:lineStringProperty>
    </fault>
  </gml:featureMember>
</wfs:FeatureCollection>
```

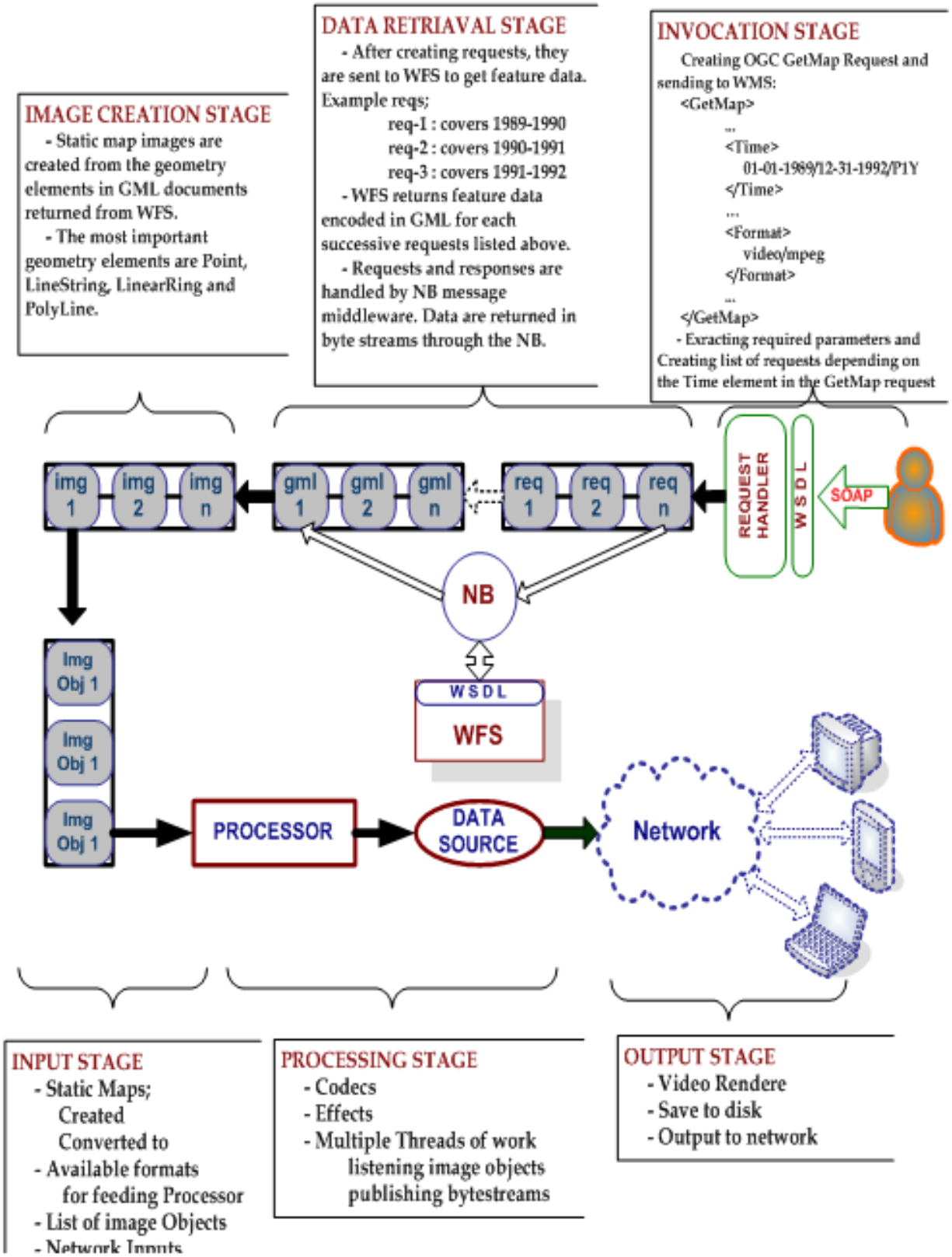

7. Sample getMap request for creating movie streams

```
<?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>
    <Elevation>5.0</Elevation>
    <Time>01-01-1987/12-31-1992/P1Y</Time>
  </Map>
  <Image>
    <Height>400</Height>
    <Width>400</Width>
    <Format>video/mpeg</Format>
    <Transparent>true</Transparent>
    <BGColor>0xFFFFFFFF</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:States</ns1:Name>
      <ns1:Description>
        <ns1:Title>California:States</ns1:Title>
        <ns1:Abstract>California:States</ns1:Abstract>
      </ns1:Description>
    </ns1:NamedLayer>
    <ns1:NamedLayer>
      <ns1:Name>World:Seismic</ns1:Name>
      <ns1:Description>
        <ns1:Title>World:Seismic</ns1:Title>
        <ns1:Abstract>World:Seismic</ns1:Abstract>
      </ns1:Description>
    </ns1:NamedLayer>
  </ns1:StyledLayerDescriptor>
</GetMap>
```

8. Detailed WMS Architecture



9. Detailed Movie creation architecture from the static map images:



10. GUI for integration framework Standard Map Tools User Interface of Integration Framework:

California earthquake fault data is overlaid Google Map.

This GUI is a standard map tools for the integration GUI see the next one showing PI interface

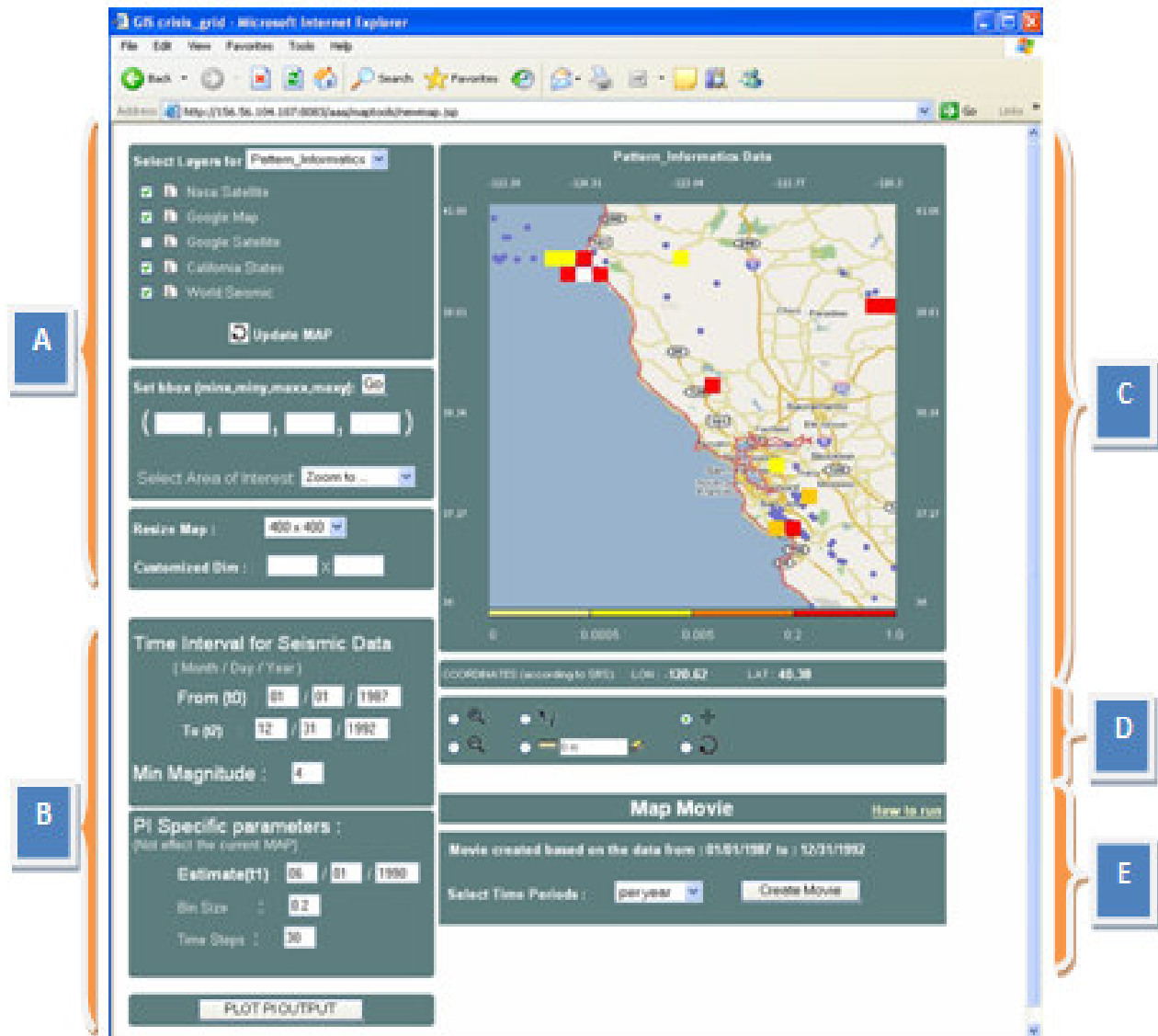
The screenshot shows a web-based map interface titled "California_Faults Data". The main map area displays a map of California with various fault lines overlaid in red and blue. The map is surrounded by a coordinate grid. Below the map, there is a distance scale and coordinate information.

Annotations and Callouts:

- Sub layers of selected main layer. These are defined in capabilities file of the WMS**: Points to the "Select Layers for" dropdown menu.
- Changes in the list of sub layer to take effect**: Points to the "Update MAP" button.
- Main layers based on the project names and/or initially connected master WMS capabilities file**: Points to the list of layers (Nasa: Satellite, Google Map, Google Satellite, California Faults, California States).
- Coordinate Values, dynamically updated**: Points to the "COORDINATES" section showing LON: -123.6 and LAT: 37.42.
- Like a terrain service. Predefined regions listed in names**: Points to the "Select Area of Interest: Zoom to ..." dropdown.
- Resizing the map. Two options- predefined and customized**: Points to the "Resize Map" and "Customized Dim" controls.
- Optional, if you want to get a map in a specific bounding box values. Click 'Go' after selection**: Points to the "Set bbox" input fields and the "Go" button.
- Map tools, they can be selected one at a time**: Points to the map navigation controls (pan, zoom, etc.).
- Distance scale, dynamically updated based on coordinates and bounding box**: Points to the distance scale below the map.
- Latitude and longitude values of the mouse on the map. Dynamically updated**: Points to the coordinate display area.

11. Extended GUI for integration of WMS and Geo-Science Grids

B and E parts are extensions to the standard map tools



Layer-1 and 2 are manipulated through the parts A, C and D. Layer-3 is manipulated through part B and utilize the parameters given in part A. Part C is the output screen and enables interactive manipulation of the layers and interactive query of the data rendered in layer-2. If the data rendered in layer-2 is time series data, then part E enables creating movies. Part A enables users to set bbox, map size, specific region to zoom-in, and the layers to be overlaid and project to work with. Part D consists of map tools enabling zoom-in, zoom-out, drag and drop, and data query of the map displayed in Part C. Part B enables users to enter parameters specific to Geo-Science application. For example for the Pattern Informatics application, users should enter the parameters “bin-size”, “time-steps”. Users can easily move to another project that they want to work by using drop-down list at the top-left corner.