Grid Oriented Web Map Services and

Grid & GIS Integration Framework

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 interoperability issues into considerations to create an efficient framework to be applied on any Geo-Science Grids.

Web 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 start the document with innovative 3-layer structured display as the output of the proposed integration framework. We then explain the main components of the framework and how they work all together. We finish the first chapter with the motivating use cases. In Chapter 2, we summarize the research problems and problem statements and, locate our proposed research in the literature. Chapter 3 explains the general problems of distributed GIS and our approaches to the problem solutions in the context of proposed integration framework. Chapter 4 explains the architecture of the key service in the framework (grid oriented Aggregator WMS) which reflects our approach to the problems of the distributed GIS and integration framework mentioned in the previous chapters. Chapter 5 gives the preliminary performance tests. Chapter 6 lists expected contributions and Chapter 7 is the future work.

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 data in layers. We also developed intermediary service enabling Google Map servers to be used as Map services. This is explained in another document [30].

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 [24] 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.

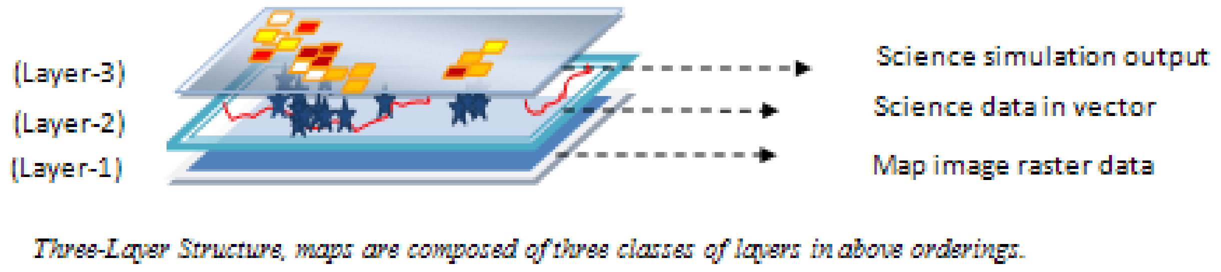

(Layer-3)
(Layer-2)
(Layer-1)
Science simulation output
Science data in vector
Map image raster data
Three-Layer Structure, maps are composed of three classes of layers in above orderings.

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 structures above there are three major components in the architecture (See Figure 2). These are Map Services, Sci-Plotting Services and User Portal (services in grey colors in Figure 2). *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 (see APPENDIX 6). 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"* and *"DescribeFeatureType"* (see APPENDIX 5).

Web Map Service (WMS) [5, 6] enables visualizing, manipulating and analyzing geospatial data through maps displayed on browser based interactive GUI (see APPENDIX 10 and 11). 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. We extend the OGC's WMS standard specifications with Web Service principles. Basic Web Map Service provides three functionalities. These are "getCapabilities", "getMap" and "getFeatureInfo" (see APPENDIX 1, 2 and 3).

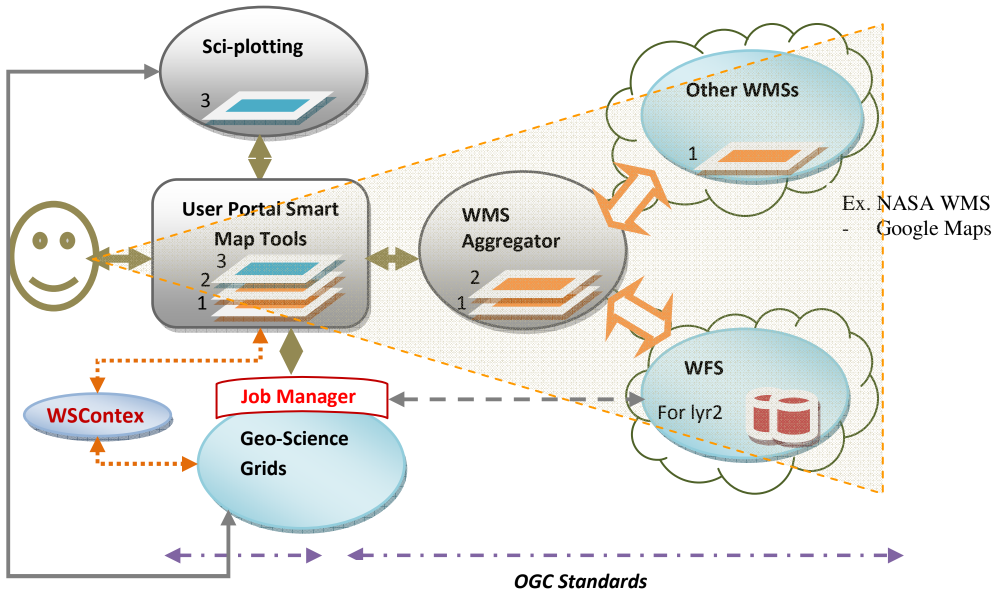


Figure 2: Map Services and Geo-Science Grids integration framework. WFS and Job Manager (interface to Geo-Science Grids) are other CGL-Lab research projects

Aggregator WMS [5, 9] is actually a WMS with some extensions (see APPENDIX 8). It provides all the interfaces and functionalities that any other OGC compatible WMS provides in OGC standards. Furthermore, Aggregator WMS also provides Google Map layers 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 with its enhanced performance capabilities mentioned in the following chapters. Aggregator WMS uses innovative pre-fetching, load balancing and caching techniques (see Section 4) and, 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 technique for managing distributed workflows. Different Geo-Science applications (such as PI and 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 and user portal knows how to invoke it. 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 through job manager. In order to integrate and relate the simulation output with the map services, user invokes the Sci-Plotting server to overlay simulation output over the Map layers created earlier.

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 WMS and, plots from Sci-Plotting servers. The process of data analyzing involves data mining which is made using interactive smart map tools and interactive integration tools at the user portal (see APPENDIX 10 and 11). 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 from end-users for invoking Map Services (for layer sets 1 and 2), triggering Geo-Science Grid (for layer set 3) and integrating (overlaying) its simulation outputs. It also enables querying the layer set 2 data, and analyzing the results interactively. The process is diagrammatically illustrated in Figure 2.

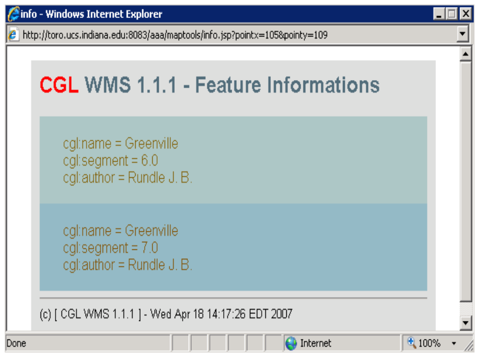
Layer sets 1 and 2 are created asynchronously together at the Aggregator WMS. Layer set 3 is created with separate chain of services and processes. Layer set 3 is the top layer overlaid on others and, it is created from the Geo-Science Grids' simulation outputs.

Jobs regarding layer sets 1 and 2 involve most commonly used OGC standard GIS services which are implemented in Web Services principles such as WMS and WFS. 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 (see APPENDIX 4) providing information about data and service together. 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 WMS to use other services' data as if its own by updating its capability metadata. OGC names this technique service cascading. 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 (such as 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.


info - Windows Internet Explorer
http://toro.ucs.indiana.edu:8083/aaa/maptools/info.jsp?pointx=105&pointy=109

CGL WMS 1.1.1 - Feature Informations

cgl:name = Greenville
cgl:segment = 6.0
cgl:author = Rundle J. B.

cgl:name = Greenville
cgl:segment = 7.0
cgl:author = Rundle J. B.

(c) [CGL WMS 1.1.1] - Wed Apr 18 14:17:26 EDT 2007

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 the user submits the 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, 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).

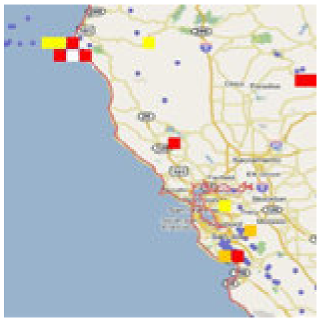
1.3. Motivating Use Cases

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 part of SERVOGrid CCE environment, we initially chose two projects in order to apply our integration framework. These are Pattern Informatics (PI) and Virtual California (VC).

1. *Pattern Informatics (PI)*: 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 by Sci-Plotting server to create layer set 3.

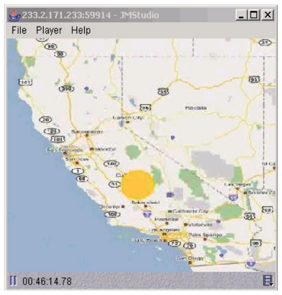


Layer set 3 is created from the PI simulation outputs. Sci-Plotting server gets the output through HTTP protocol and plot the image as layer. The result is a map which shows the fluctuations in seismic activity which are found to be related to the preparation steps for large earthquakes. This map layer 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):* VC is earthquake simulation model for the California. The simulation takes into account the gradual movement of faults and their interaction 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 as 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 -Java Media Framework- 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.



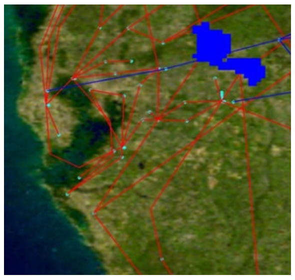
Layer 3 is created at the Sci-Plotting server with the data set coming from VC's simulation. The required parameters for creating layer-set 3 are given by the user through GUI of integration portal. Periodicity of the data for creating movie frames is defined by user interface. We plot layer-set 3 in 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]. For the sample request creating movie streams and brief architecture illustrating how map movies are created see APPENDIX 7 and 9.

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 set 1 we used NASA OnEarth Map server layers, as layer set 2 we used gas pipelines and electric power lines represented as feature data in GML (see APPENDIX 6). We kept them in Database and provided to the Map Server through WFS Web Services. After getting maps covering gas pipelines and electric power lines on the NASA satellite map images, we start running IEISS simulation code through the user portal's smart map and integration tools.



As Layer set 3, 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 and Problem Statement

Extensibility, Interoperability and Performance of the Framework

Title categorizes the challenges of the proposed integration framework. This chapter first explains our approaches to overcome the listed challenges, especially the performance issues, and then, explains our primary research problems.

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 format increases the size further. There is a trade off here that we want to solve in the following chapters.

Performance: For the sake of interoperability and extensibility we use CDF represented and formulated in XML (we use GML in our motivating domain). However, it burdens the data transfer and rendering times and sometimes even makes them impossible. 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. Problem Statement in Literature

Our main goal is creating a general framework mapping distributed and heterogeneous data resources into global Science Grids in a performance efficient manner. To achieve this, we need to access and integrate the data and present it to the Science Grids in accordance with their general needs. We also need to create a general view-based integration interface for clients to utilize data, enable integration and interoperation of data and Science Grids at view-level (layer-level) by hiding the complexities of the system.

On our way to achieving our goals summarized above, we face some general and domain specific problems. Here we list them at the theoretical bases. In the following Chapter (Section 2.2), we compare and contrast our research work with the related works in the literature.

In science domains information/data is inevitably distributed among several data resources. So we need to access and integrate data and specialized computational capabilities (visualization, statistical analysis, data mining etc.) of wide range of relational and un-relational data sources. Federating heterogeneous remote data, transferring large structured data and inter-operating and inter-relating GIS data and Geo-Science grids outputs are the main problems in that respect. Transferring, rendering and displaying the large heterogeneous science data for the decision makers and end-users are CPU and time consuming processes. However, Science Grid applications require quick response times. This is a common problem of data integration proposals with respect to performance. In this document we mostly focus on the performance.

Accessing and integrating data from different providers using different heterogeneous technologies is a difficult task which has been worked upon for decades. Furthermore, multiple data sources not only need to be integrated but also transformed into comprehensible representations through cascaded inter-service communications. The attributes of the data/information which are building these comprehensible representations should be interactively accessed and queried.

The literature shows many proposals for integration of data, ranging from federated databases to mediators (such as SRB -Storage Resource Broker) and ontology. The adoption of a conventional integration methodology does not lead to a solution for comprehensible data integration. They do not consider the graphic aspects to represent schemas nor the diversity and richness of semantic representation of the data. One can achieve data integration to some extent by using conventional techniques but one cannot integrate the data at the representation level. We call the representation level as layer-level (see the 3-layer structured display) and propose an architectural framework to integrate data at that level. Our initial solutions will be based on GIS domain.

2.2. Locating the Proposed Research in Literature

In summary our research focuses on creating Grid-oriented GIS Map Services integrating data at the layer-level and providing them to Science Grids through web portals with interactive-smart view tools. Web portals enable Grids simulation can be accessed in a remotely and run seamlessly. Further more interactive-smart map tools hide the complexities of the system and make it easy to be used by scientist or end-users from different backgrounds. We also associate the inputs and outputs of the Geo-Science Grid simulations at the view level of data integration hierarchy and, create three-layer structured views illustrating the association of input and outputs of the Geo-Science simulations for the decision makers and end-users.

The data access and integration part of our system shows similarities to the FDBS and the SRB. However, it has some differences in the level of data/information integration hierarchy.

Federated Database Systems (FDBS) [28]: A federated database enables communication between multiple DBMS or databases in a single SQL statement and makes the location of information transparent to the user. Constituent databases are geographically decentralized and integrated remotely. A federated database (or virtual database) is fully-integrated. To achieve this, a logical composite of all the constituent databases in a federated database system is created.

Aggregator WMS (see Figure 2) actually functions as Federated Database (or virtual database) in DBMS. Both systems make data abstraction. Data abstraction in our framework is representation based and can leverage FDBS data abstraction at the bottom level. For example, any data is abstracted as GML, and GML is abstracted as layer, and layer is abstracted as map images. Even map images are abstracted as composite maps by overlaying on each other.

Data integration techniques depend on the data abstraction techniques used. In order to integrate data, FDBS uses mapping of the Database views represented in schema. The mapping techniques can be generalized into two groups. In "Global as View (GAV)" the global schema is defined in terms of the underlying schemas. In "Local as View (LAV)", the local schemas are defined in terms of the global schema. Since we use global schema for the data abstractions (GML for feature data and image types for layers and maps) our approach looks like LAV.

FDBMS integrate data by federating databases. We not only integrate the databases but also any other data servers such as WFS, WMS or even SRB servers. We are relatively located at the upper level of the data integration hierarchy.

FDBS provide two-way data flowing (transaction management). In contrast, we enable just one-way data flow (data querying). In our approach, data querying is done by decomposition based on the information kept in capabilities metadata (see APPENDIX 4). That is, query is decomposed to sub-queries and each sub-query is sent to corresponding data server and, result sets are matched as an answer to the main query.

SRB (Storage Resource Broker) [25, 26]: SRB is the implementation architecture of the integrated/ federated digital libraries. They integrate data as digital objects. Digital objects are mostly files but URLs and SQL command string and any string of bits collected from multiple different data sources. Digital objects are geographically decentralized and integrated in a remote fashion.

SRB stores metadata in MCAT [27] as relational databases in central fashion. Instead, we use XML structured capability metadata in distributed fashion. That is, each server keeps its metadata locally and upgrades it dynamically through inter-service communication capability of our system.

Regarding data-flow in the system, SRB provide two-way data flowing (transaction management). In contrast, we emphasize one-way data flow (data querying).

Proposed framework enables two level data integration (layer-level and data-level). At the data-level we can leverage SRB server into our framework by using appropriate wrapper. SRB is not competing but a complementary technology. As it is shown in the figure we use wrappers to integrate SRB servers to the system.

Capability concept in literature: We use capabilities to define service and data together. Communications between servers are achieved through exchanging the capability documents. It enables inter-service communication through well-defined service interfaces and message formats ("getCapabilities"). Capabilities metadata can be updated manually or dynamically and, consists of descriptor, service and provider metadata. Inter-service communication is achieved without a third-party and enables chain of services.

In our system we propose two different groups of servers. They have different capabilities schema and formats. One is for wrapper type of services such as WFS getting any data and serving in the CDF such GML in GIS domain. Another is for layer-level services providing layers such as WMS in GIS domain.

Capability is defined by OGC standard specifications. It looks like Dublin Core metadata [29] format defining resources. Capability like structure is also used in Gannon's approach (XPOLA), for Grid services' security issues, describing dynamic Web/Grid resources.

3. The General Problems of Distributed GIS Systems

Our experience shows that although we can easily integrate several GIS and other services into complex tasks by using Web Services, 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 3.1.

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 3.2.

Since the framework is developed in SOA principles and services are OGC compatible Web Services, we take the extensibility and interoperability issues as granted. We mostly focus on the performance problems of the distributed GIS systems.

3.1. 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 to which WFS streams requested data. Second request is done by NaradaBrokering Subscriber. Even in the case of that the 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.

3.2. 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.

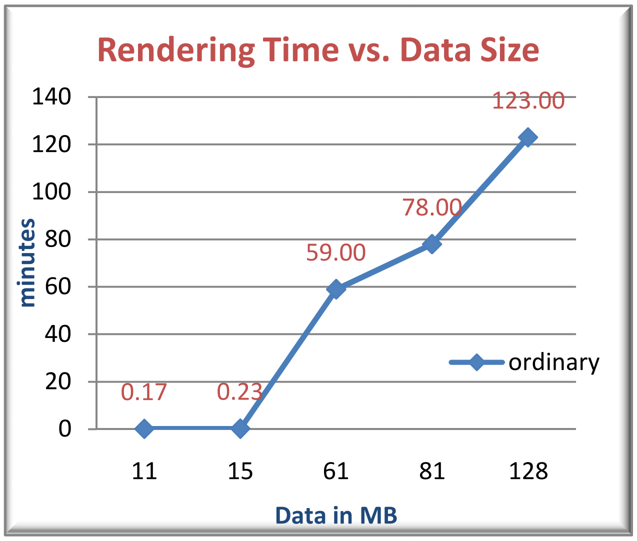
Pull parser only parses what is asked for by the application rather than passing all events up to the client application as SAX parsing does. The pull approach of this parsing model results in a very small memory footprint (no document state maintenance required – compared to DOM), and very fast processing (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 teaches us valuable lessons in terms of the capabilities and limits of our implementation. From the figure 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.

(a) (b)



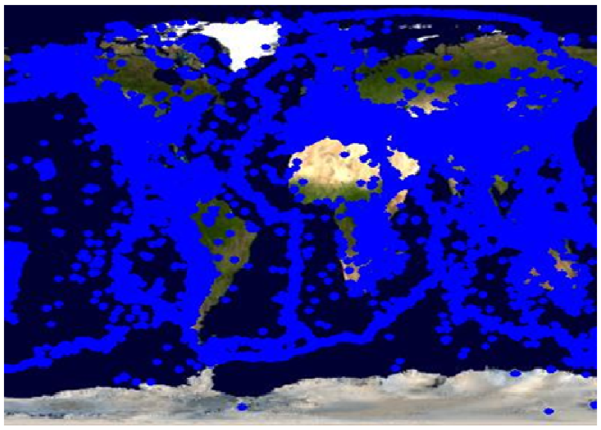


Figure 3: (a) Performance result of the current system. (b) Sample output consisting of Layer-1 and 2. See Figure 1.

The 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,

$$time_{(measured)} = time_{(map\ is\ displayed)} - time_{(client\ makes\ request\ for\ the\ map)}.$$

In order to give more detailed information about measured time in the figure we list all the processes involved below. You can also have a look at Figure 2 while you are reading through the steps.

- **[*time*$_{(client\ makes\ request)}$.]** Client makes requests by interactive smart map tools (Geo-Science Portal) for Aggregator WMS
- Aggregator WMS parse and render requests and define set of actions required based on the requests and its capabilities file.
- Aggregator WMS Creates map images (from the returned data) and returns them to the clients:
    - Defined WFSs and other WMSs to get the requested data (defined in capability file)
    - Creates requests for WFSs and other WSMs
    - Invokes WFSs *getFeature* Web Services for vector data encoded in GML.

  - Invokes other WMSs *getMap* Web Services for raster data rendered in map images
  - Transferring GML data and cascaded map layers (feature collections) from WFS and WMS
  - Parsing and rendering returned GML data sets
  - Aggregating and overlaying layer (layer sets 1 and 2) according to the request
  - Sending the map images to the WMS Client at Geo-Science Portal
- **[*time*$_{(map\ is\ displayed)}$]** Client at the Geo-Science Portal shows its maps on his browser

These steps also a summary of the processes happen in orange rectangle (which is bottleneck of the system). Almost %90 of the *time*$_{(measured)}$ comes from the step 3.5 called "transferring GML data (feature collections) (see APPENDIX 6) and cascaded map layers from WFS and WMS". This explains why orange rectangle becomes a bottleneck in the system. In this case, even if we use the most efficient and fast parsing and rendering algorithms (such as using pull parsing or application and domain specific XPath querying), it won't improve performance very much if the data transfer times are still that much high as shown in the figure.

3.3. Future Directions on Performance Issues

Load balancing, Caching and Pre-fetching

These are the three main means coming to mind to improve the performance further. 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. However, 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).

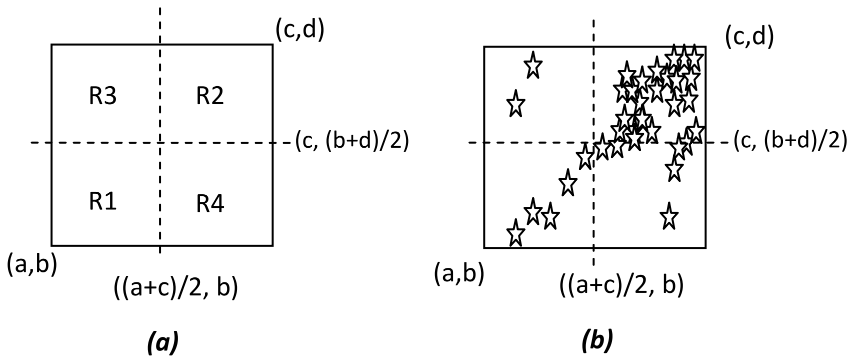


Figure 4: Classic partitioning cannot share the work equally among worker servers. Server assigned the partition of "((a+b)/2, (b+d)/2), (c, d)" gets the most of the work.

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

4. Grid Oriented Web Map Services Architecture

Dynamic Load Balancing with Caching over pre-fetched data

Web Map Service is the key and the most crucial service in GIS and in our proposed Grid integration framework. This Chapter explains the architecture of the grid oriented Web Map Services. This innovative Web Map Service architecture is developed under the considerations of the General GIS problems mentioned in Chapter 3 and Grid integration framework mentioned in Chapter 1.

Proposed Web Map Services are developed in Web Service principles and easily integratable to any Geo-applications created in SOA principles. Web Map Services are created in accordance with OGC and ISO-TC211 standard specifications. Instead of giving all architectural details, we summarize the key enhancements over the general Mapping services in terms of the general GIS problems.

4.1. 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. Load balancing help workload sharing and parallel job run.

Architecture is summarized in three orderly steps. These are

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

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 a map image composed of NASA satellite base maps and earthquake seismic records (in blue dots). See Figure 5.

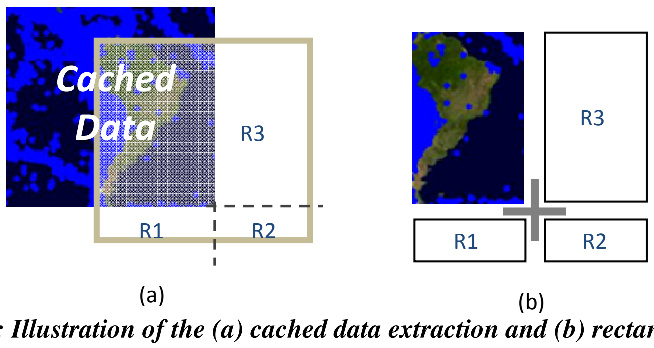


Figure 5: Illustration of the (a) cached data extraction and (b) rectangulation.

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(minx, miny, maxx, 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.

4.1.1. 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.

4.1.2. 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.

4.1.3. 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 3.1. 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 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.

Why we need pre-fetching: We do pre-fetching to get rid of the poor performance of transferring XML based feature data. According to our proposed framework, pre-fetching is done basically for Layer set 2 data. Layer set 2 data is created from the set of GML data. Furthermore, each GML data might be coming from different WFS.

There will be two separate locations for the pre-fetched data. One is temporary which will be active during perfecting the data. Another is stable which will be used for serving the clients' requests. When the data transfer is done to the temporary location, all the data at that location will be moved to stable location. Reading and writing the data files at the stable locations will be synchronized to keep the data files consistent. This cycle will be repeating.

Requests from clients contain some query constraints. So, successive queries after the pre-fetching are handled at the Aggregator WMS (see APPENDIX 8) side at which pre-fetching initialized. Since pre-fetched data is semi-structured XML data (GML), query is done by using parser techniques (we use Pull Parsing) and XPATH queries over pre-fetched data.

One minor challenge with storing the data in file system is that some systems allow a limited size for user created files such as:

LINUX (RedHat)	SOLARIS
2GB{512B block size}	1 TB
8192GB{8KB block size}	2 GB {=<2.5.1}

Even if the operating system does not have limited file size constraint, then file size will be constraint by the hard disk size.

Another challenge is synchronization of the two storages (temporary for fetching the data and stable for answering the client requests while fetching is happening). This challenge is easier to solve.

5. Preliminary Performance Tests and 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 4.

We have tested the proposed architecture over Pattern Informatics [21] Geo-Science Grids (see Chapter 1.3). Pattern Informatics application use earthquake seismic data provided by Web Feature Service. The performance figure below based on preliminary test results. We just got rid of the data transfer time by pre-fetching the data to see the timings of data rendering. Before run-time, we 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 responds to the successive requests by locally querying the pre-fetched data kept in its local file system, instead of going to WFS.

Browser Time out issue: Our proposed framework is integrated with browser based GUI and interactive tools. This enables application framework can be accessed online from anywhere. Since it is browser based, if the querying and transferring the data takes more than a specific time (3-5 minutes for IE and 30 seconds for Safari) the client faces a browser time out risk. In that case, the client gets the "page or site is not responding or unavailable" message. There are some proposed solutions to this kind of problems such as polling and heart-beating until the data transfer is completed. However, this is not our main research focus and instead of spending our efforts on solving this issue, we have been working on increasing the performance of the architecture with the innovative approaches and, at the end, dropped the data transfer times measured in seconds. See the Figure 8 for illustrating the performance results of the Category- 5.

We have implemented services in Web Service principles by using SOAP over HTTP protocol. Due to using HTTP protocol, Web services also have time out issues but easily configured through SOAP binding objects. If the timeout is set

Memory issues: It is an issue because of the parsing large XML structured feature data encoded in GML. DOM allows parsing document to a limited sizes depending on the system setting, operating system and hardware of the server.

Categorizing proposed system Life-cycle from the performance point of view:

1. OGC + DOM Parsing:
   We first developed our system according to OGC public standard specifications using HTTP protocol for server invocations and data transfers and using GML common data model etc.

2. OGC + DOM Parsing + <u>Web Service</u>:
   We have extended the services with Web Service principles and started using SOAP over HTTP protocols for data transfer and service invocations

3. OGC + <u>Pull Parsing</u> + Web Service:
   -Memory and hip size issues are resolved
   In order to parse the data in XML format we started using DOM with small sized data sets. After encountering some memory problems because of using DOM, we started using pull parsing technique to parse the large semi-structured XML based GML data

4. OGC + Pull Parsing + Web Service + <u>Streaming</u>:
   Until this step, we used to work with geo-science applications requiring of working small scale data. These applications were mostly display purposes. When we start adopting our system to Geo-Science Grids we needed to improve our system with large data transfer latency problems. In order to solve these problems, we have integrated NaradaBrokering to our proposed framework (see Section 4 for more information)

5. OGC + Pull Parsing + Web Service + Streaming + <u>Grid Oriented</u>:
   - Browser time out issue is resolved
   Since we propose an interactive system and because of the geo-science applications' characteristics, response times should be restraint in a reasonable range. Till the step 4 we always had timeout problem when the user invokes the application with the large parameters. We solved this problem by using proposed techniques explained in Chapter 4 for Grid oriented Web Map Services. These techniques are summarized as dynamic load balancing with caching over pre-fetched data.

Performance graphs based on the categories listed above (Figure 6, Figure 7 and Figure 8):

Performance graphs are all created from the end-user points of view. In other words graphs shows time values passed from the clients request to final result is obtained. The detailed steps building the time values in the graphs are given before in Chapter 3.2.

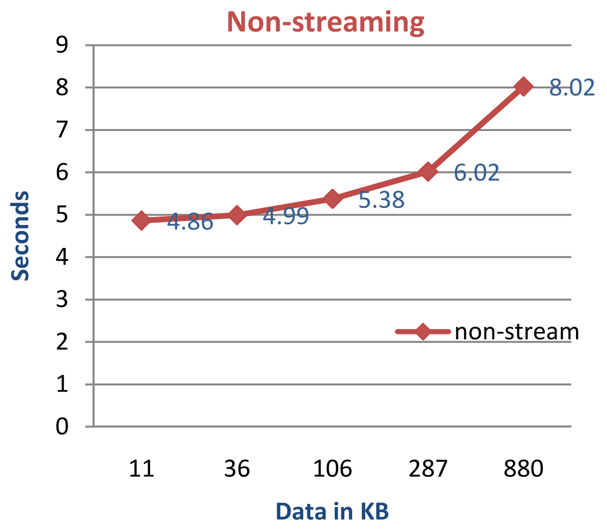


Figure 6: Performance graph of the system for Category-3.

It was ok for small scale data and applications. This graph represents the initial system performance. No advanced techniques for data transfer. Data is transferred as an attachment to SOAP message by using SOAP over HTTP protocol. The system faces browser timeout problem, when the client request the data larger than 1.5MB in size.

Note, the data sizes are in **KB** and timings are **SECONDS**.

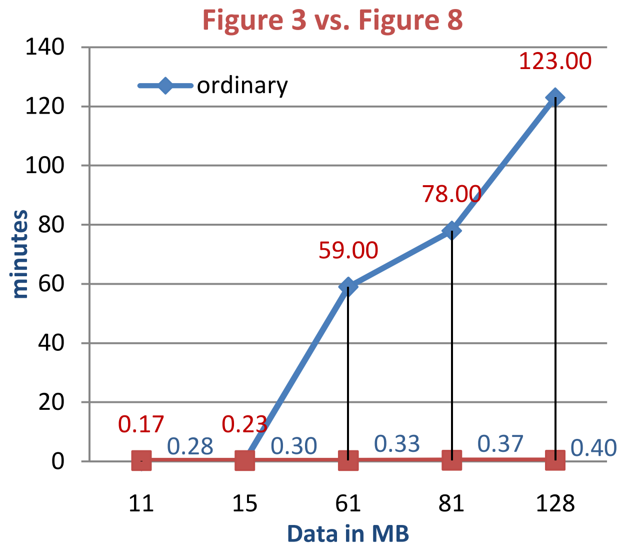


Figure 7: Performance graph of the system for Category-4

(Note the timing. It is measured in minutes)
Still browser time out problem for mapping the data larger than 16MB in size.

Without considering browser time-out. Not whole system just for showing data transfer performance by using streaming techniques through Naradabrokering

We used the techniques explained in Chapter 3.1 and 3.2.

Note, the data sizes are in **MB** and timings are **MINUTES**.

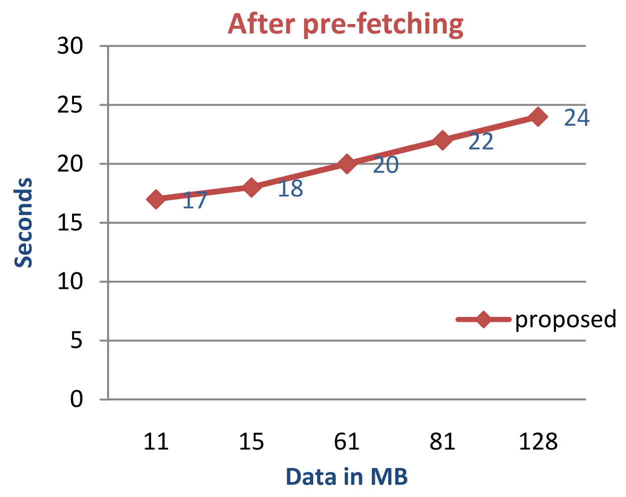


Figure 8: Performance graph of the System for Category-5.

Data is already obtained from the source and successive queries to Aggregator WMS are answered directly from the pre-fetched data without going to cascaded servers.

Performance results are obtained over the most enhances system for the integration framework. We used the techniques summarized in Chapter 3.3.

Note, the data sizes are in **MB** and timings are **SECONDS**.

6. Expected Contributions

GIS is our motivating domain. We merge two important software worlds: GIS and Web Service Architectures. We also analyze the general GIS problems regarding the performance issues and interoperability trade-offs.

Our integration framework and GIS services are interoperable and easy to extend. In order to achieve interoperability we use OGC and ISO/TC211 standard specifications in accordance with the Web Services principles.

Besides implementing services in Web Services principles, we extend OGC capability specifications with the workload management; we create an architectural framework for dynamic capability federation of Map Services enabling workload management and service chaining.

We design Grid-oriented high performance Web Service architecture for distributed Map Services to support accessing and rendering archived distributed and heterogeneous geospatial data. We also develop architectural framework to integrate Map Services with the Geo-science Grids. We do this by introducing innovative 3-layer structured displays. Top layer set in the 3-layer structure comes from our innovative Sci-Plotting Web Services.

In order to create integration framework we needed to create Sci-Plotting services enabling simulation outputs of Geo-Science grids to be overlaid on GIS maps as being comprehensible data layer. Sci-Plotting Services are based on Dislin plotting libraries. Sci-Plotting Web Services also plot charts and graphs through well-defined service interfaces. They can be used any other science community besides GIS.

We create interactive "smart map" and Grid integration tools (see APPENDIX 10 and 11) as the set of portlets for the Science portals. User portal is actually an independent browser based GUI enabling interaction with GIS and Grid services while hiding system complexity from the users.

We also provide enhanced decision support with domain specific metadata languages and interactive mapping tools with query capabilities (*getFeatureInfo (see APPENDIX 3)* Web Service interface of Map Services). We propose an architecture framework to transform heterogeneous and dispersed data into human readable forms (maps, layers, graphs and plotting) and integrate multiple information sources into interactive user interfaces such as digital photography, demographic information, and information from simulations.

We provide capabilities federation through proposed Web Services' capabilities metadata as distinct from data/Database federation/replication approaches. We define the differences of our approaches compared to other data federation system and leverage their systems into our proposed integration framework. We make distinction of data integrations based on the data-lifecycle in the integration hierarchy. We also formalize distributed data access, query, and transformation through capabilities metadata, defining all the data/information sources as interacting Web Services with standard metadata service ports

We define possible bottlenecks and optimization and enhancement opportunities for the distributed heterogeneous information management systems. In that context, the compos-ability nature of data/information services WMS and WFS enable caching and load balancing for obtaining enhanced service outcomes. Capability aggregation, dynamic capability exchanges and updates are the other issues serving optimization and architecture enhancement purposes.

7. Future Work

We mostly focused on structured (different data models), syntactical (different languages and data representations) and systemic (different hardware and operating systems) heterogeneity issues in the integration framework and, the feasibility of such an integration framework. Since we applied our proposed integration framework in OGC compatible GIS domain we did not take the semantic heterogeneity of the data into consideration. General data model and semantics of the data are defined by OGC. We will be extending our framework by researching on semantic issues to make our framework applicable over all Science domains such as Astronomy and Chemistry.

Our motivating domain was GIS and, we have used GML as common data model, WMS for display services, WFS for data services providing data in common data model, and OGC's *capability* definitions for the capability metadata for the services. When we try to apply the framework to other domains such as Astronomy or Chemistry domains, we will need to update the system with the new data model and online services. We will be summarizing the architectural requirements and instructions to create a similar framework for different science domains. If it is possible, (going one step further) we will try to create a generic framework with the generic services and common data model running over different science domains.

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

[25] Implementing and using SRB, Proc UK e-Science All Hands Meeting 2003, © EPSRC Sept 2003, ISBN 1-904425-11-9.

[26] Rajasekar, A., M. Wan, and R. Moore, (2002), "MySRB & SRB – Components of a Data Grid," The 11th International Symposium on High Performance Distributed Computing (HPDC-11) Edinburgh, Scotland, July 24-26, 2002.

[27] MCAT, (2000) "MCAT:Metadata Catalog", SDSC. Available at http://www.sdsc.edu/srb/index.php/MCAT

[28] Sheth, A.; Larson, J.: Federated Database Systens for Managing Distributed, Heterogeneous and Autonomous Databases. *ACM Computing Surveys*, Vol. 22, No. 3, 1990, pp. 183-236.

[29] Dushay, Naomi. & Hillmann, Diane (2003). "Analyzing Metadata for Effective Use and Re-Use." DC-2003 Dublin Core Conference: Supporting Communities of Discourse and Practice . Metadata Research and Applications. September 28 -October 3, 2003.

[30] Ahmet Sayar, Marlon Pierce, and Geoffrey C. Fox, "Integrating AJAX Approach into GIS Visualization Web Services.", IEEE International Conference on Internet and Web Applications and Services, ICIW'06, February 2006

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>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: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 – For the California fault data

```
<?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>
```

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.**

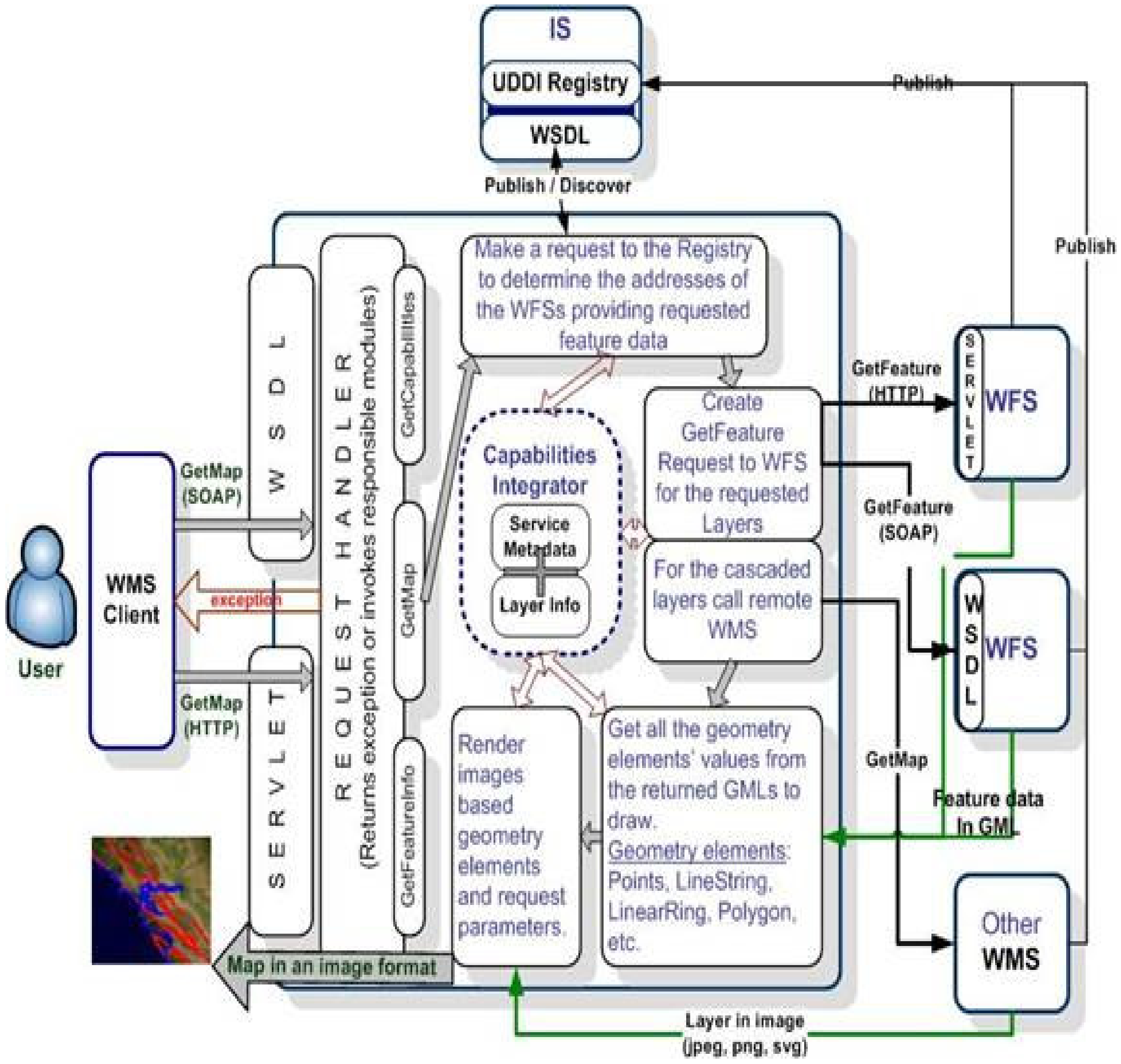
```
<wfs:FeatureCollection
      xmlns:wfs=http://www.opengis.net/wfs
      xmlns:gml=http://www.opengis.net/gml
      xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
      xsi:schemaLocation="http://crisisgrid.org/schemas/fault_new.xsd">
      <gml:boundedBy>
            <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#27354">
                  <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>0xFFFFFF</BGColor>
        </Image>
        <ns1:StyledLayerDescriptorversion="1.0.20"xmlns:ns1="http://www.og.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 <u>Aggregator</u> WMS Architecture


IS
UDDI Registry
WSDL
Publish
Publish / Discover
Publish
User
WMS Client
GetMap (SOAP)
GetMap (HTTP)
exception
W S D L
S E R V L E T
R E Q U E S T H A N D L E R (Returns exception or invokes responsible modules)
GetCapabilities
GetMap
GetFeatureInfo
Make a request to the Registry to determine the addresses of the WFSs providing requested feature data
Capabilities Integrator
Service Metadata
Layer Info
Create GetFeature Request to WFS for the requested Layers
For the cascaded layers call remote WMS
Render images based geometry elements and request parameters.
Get all the geometry elements' values from the returned GMLs to draw.
Geometry elements: Points, LineString, LinearRing, Polygon, etc.
Map in an image format
GetFeature (HTTP)
GetFeature (SOAP)
GetMap
S E R V L E T
WFS
W S D L
WFS
Feature data In GML
Other WMS
Layer in image (jpeg, png, svg)

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

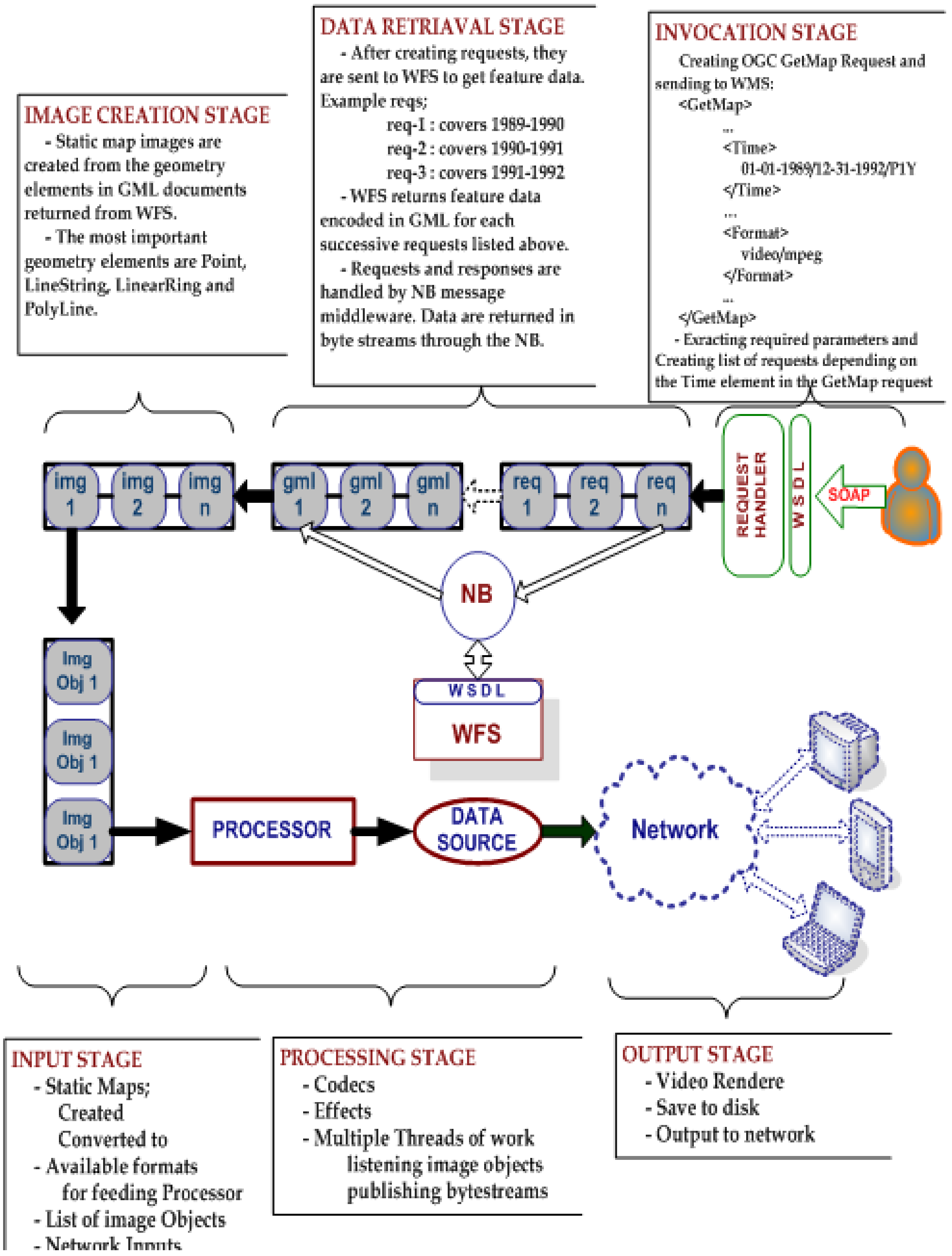


IMAGE CREATION STAGE
- Static map images are created from the geometry elements in GML documents returned from WFS.
- The most important geometry elements are Point, LineString, LinearRing and PolyLine.

DATA RETRIAVAL STAGE
- After creating requests, they are sent to WFS to get feature data. Example reqs;
  req-1 : covers 1989-1990
  req-2 : covers 1990-1991
  req-3 : covers 1991-1992
- WFS returns feature data encoded in GML for each successive requests listed above.
- Requests and responses are handled by NB message middleware. Data are returned in byte streams through the NB.

INVOCATION STAGE
Creating OGC GetMap Request and sending to WMS:

```
<GetMap>
      ...
      <Time>
         01-01-1989/12-31-1992/P1Y
      </Time>
      ...
      <Format>
         video/mpeg
      </Format>
      ...
</GetMap>
```

- Exracting required parameters and Creating list of requests depending on the Time element in the GetMap request

INPUT STAGE
- Static Maps;
  Created
  Converted to
- Available formats for feeding Processor
- List of image Objects
- Network Inputs

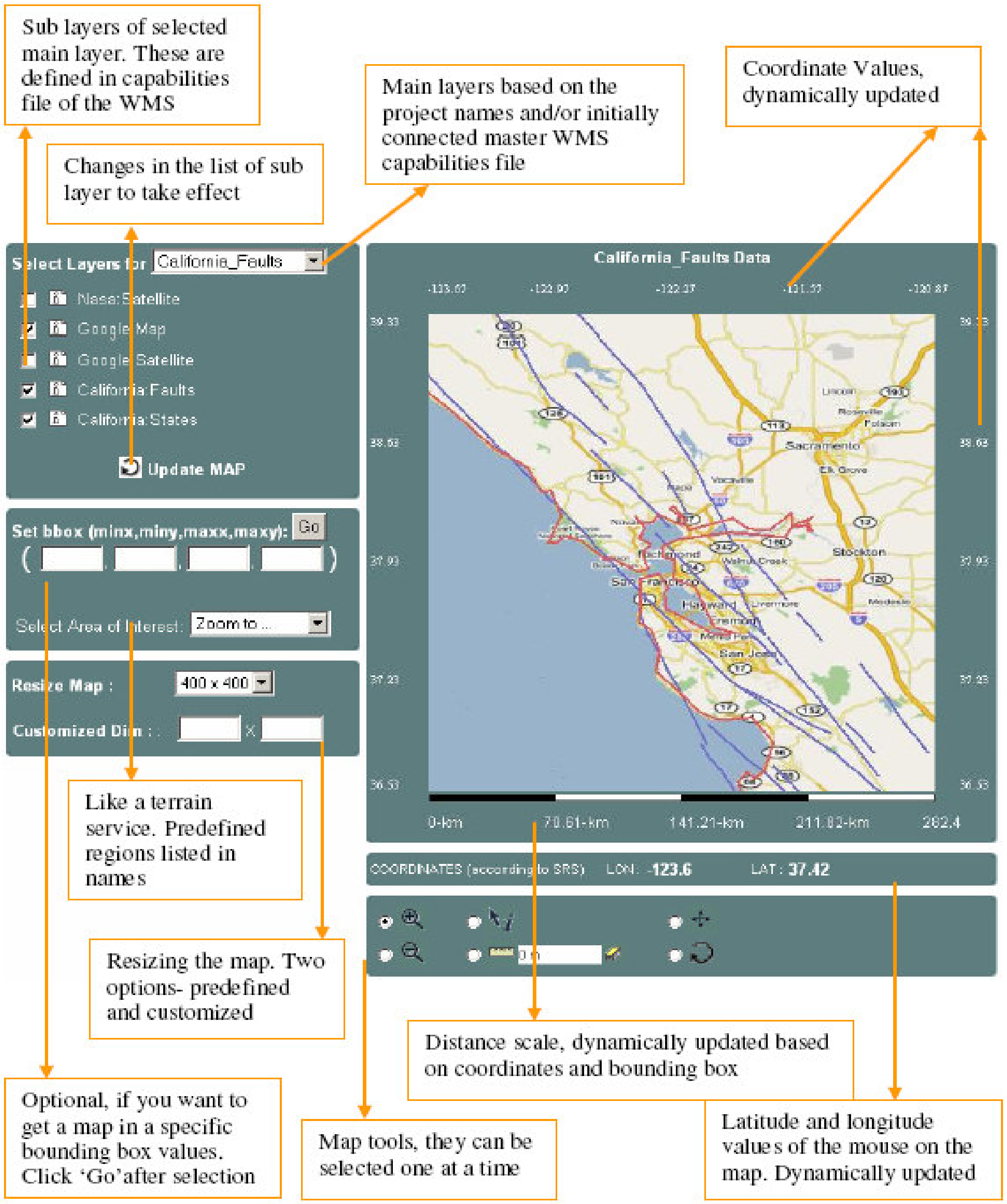
PROCESSING STAGE
- Codecs
- Effects
- Multiple Threads of work listening image objects publishing bytestreams

OUTPUT STAGE
- Video Rendere
- Save to disk
- Output to network

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

California earthquake fault data is overlaid Google Map.

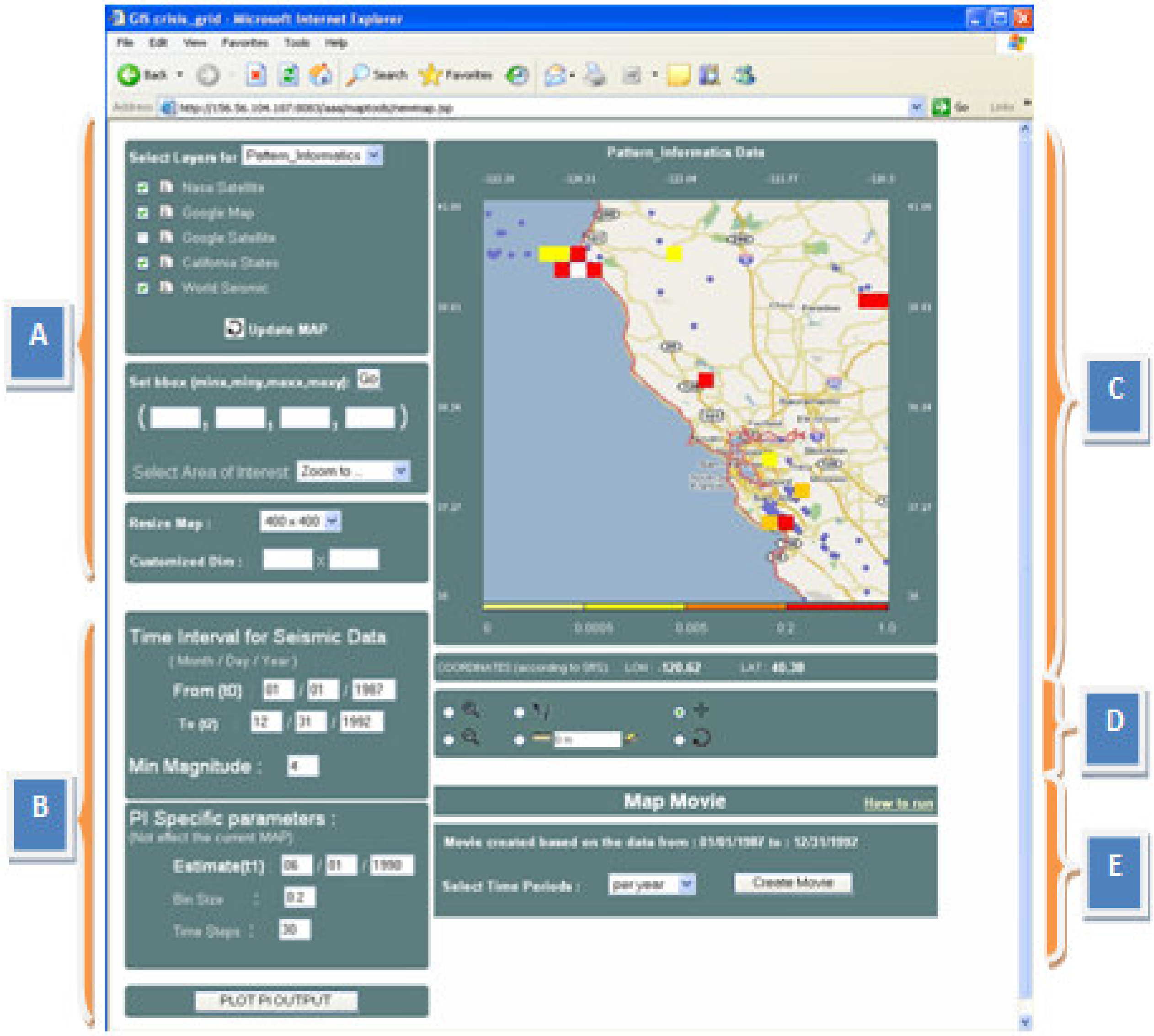
This GUI is a standard map tools for the integration GUI see the next one showing PI interface



Sub layers of selected main layer. These are defined in capabilities file of the WMS
Changes in the list of sub layer to take effect
Main layers based on the project names and/or initially connected master WMS capabilities file
Coordinate Values, dynamically updated
Select Layers for California_Faults
Nasa:Satellite
Google Map
Google Satellite
California:Faults
California:States
Update MAP
California_Faults Data
Set bbox (minx,miny,maxx,maxy): Go
Select Area of Interest: Zoom to ...
Resize Map : 400 x 400
Customized Dim : x
COORDINATES (according to SRS) LON: -123.6 LAT: 37.42
0-km 70.61-km 141.21-km 211.82-km 282.4
Like a terrain service. Predefined regions listed in names
Resizing the map. Two options- predefined and customized
Distance scale, dynamically updated based on coordinates and bounding box
Optional, if you want to get a map in a specific bounding box values. Click 'Go' after selection
Map tools, they can be selected one at a time
Latitude and longitude values of the mouse on the map. Dynamically updated

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

B and E parts are extensions to the standard map tools



Select Layers for: Pattern_Informatics

- Nasa Satellite
- Google Map
- Google Satellite
- California States
- World Seismic

Set bbox (minx,miny,maxx,maxy): (, , ,)

Select Area of Interest: Zoom to ...

Resize Map: 400 x 400

Customized Dim: x

Time Interval for Seismic Data (Month / Day / Year)

From (t0) 01 / 01 / 1987

To (t2) 12 / 31 / 1992

Min Magnitude: 4

PI Specific parameters: (Not affect the current MAP)

Estimate(t1) 06 / 01 / 1990

Bin Size: 0.2

Time Steps: 30

COORDINATES (according to SRS): LON: -120.62 LAT: 40.38

Movie created based on the data from: 01/01/1987 to: 12/31/1992

Select Time Periods: per year

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.