Information Grids

Applications to Geographic Information Systems

By Ahmet Sayar

Research Committee:

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

Indiana University – June, 2007
Computer Science - Community Grids Lab (CGL)

Table of Contents

List of Figures

1. Introduction

We investigate the issues pertaining to the traditional Geographic Information Systems (GIS) [2, 9] approaches and solutions to these problems based on modern Service Oriented Grids approaches. As in general science domains, GIS requires decision making and situation assessment based on integrated data display. We generally focus on the information Grid issues in terms of GIS and geographic data, but findings and recommendations are relevant for any other science domains and data types.

GIS is a system of computer software, hardware, and data used to manipulate, analyze, and graphically present a potentially wide array of information associated with geographic locations. GIS's powerful ability to integrate different kinds of information about a physical location can lead to better informed decisions about public investments in infrastructure and services—including national security, law enforcement, health care, and the environment—as well as a more effective and timely response in emergency situations. However, long-standing challenges to data sharing and integration need to be addressed before the benefits of geographic information systems can be fully realized. Our focus regarding data integration is different from the Database or digital library communities. We deal with the integration at the higher level than they do and, we try to utilize their approaches at the bottom level by proposing generic mediator services (See Chapter 4).

Our work is about developing a Web Services architecture that provides coupling of scientific geophysical applications with archival data through the innovative interactive smart decision making tools. This work can be detailed in a couple of sub-research areas such as: accessing and querying heterogeneous data provided by heterogeneous storages with unified query structures, developing GIS data services, considering performance issues of transferring, parsing and rendering of large geographic data, and composition of GIS services.

In the light of the explanations above we categorize our work as below:

1. Coupling Geo-Science computational Grid with geo-data Grid
    - Integrating Web Map Services with Geo-Science Grid [11, 12]
    - Enabling decision making through integrated data display (3-layered display structure)
    - Creating view-level integration structure. Views are abstracted as layers.
    - Creating generic plotting Web Services (Sci-Plot) in order to couple Geo-Science Grid outputs with their inputs at the view level.
2. Handling the heterogeneity in geo-data Grid
    - Different data types
    - Different storage types
    - Proposing usage of: The OGC and ISO/TC211 standards, Web Services and, mediator services (to integrate third party computation and data sources).
3. Interactive and smart decision making tools
    - Coupling interface for browser based remote access
    - Data/information display
    - Interactive querying and mining the data
    - Visualization and analysis of the data and Science Grid simulation outputs

- Movies and animation tools for the time-series data

4. Performance
    - Accessing remote large data sets provided by geographically distributed data vendors.
    - Transferring, integrating, processing and interpreting data.
    - Proposing: High-performance streaming data services through messaging middleware.
    - Proposing: Advanced pre-fetching, caching and load balancing techniques.

The importance of providing access to "*computational resources"* has been central in many research efforts in Grid community. Another such important issue is distributed access to data stored in various types of "*data resources"*. GIS is especially affected by the developments in both of these areas since these systems are traditionally data-centric; they require access to data from many different sources for creating layers, and tend to use various types of data processing tools for analysis or visualization of the geographic data.

The proposed coupling framework (see Chapter 3) is targeted to a community with a broad and demanding range of functional requirements: the situation understanding and information management systems community. A major challenge in designing and building such a system is not only to develop basic system capabilities but to provide a framework such that the best available tools can be integrated into the system with minimal effort and that each of these components can communicate with each other to create new applications. These components must be able to bi-directionally interact with document management components by sending and receiving information (capability metadata transactions through *getCapabilities* Web Services of the components). With this in mind, we designed the framework as a component based system that will be able to support continuous increase of functionality as new and more sophisticated tools become available.

Distributed data access in GIS is traditionally regarded as dealing with distributed data archives, databases or files. The data storages and data itself can be heterogeneous. As an example of storage or service heterogeneity, we can give three major types of GIS servers used by different Indiana State counties. (1) ESRI [50] ArcIMS and ArcMap Servers are used for Marion, Vanderburgh, Hancock, Kosciusko, Huntington and Tippecanoe counties, (2) Autodesk MapGuide [51] is used for Hamilton, Hendricks, Monroe and Wayne counties and (3) WTH Mapserver Web Mapping Application [52] is for Fulton, Cass, Daviess and City of Huntingburg counties based on several Open Source projects. When a client needs to access these servers he needs separate code or application to access these data. Integrating them is almost impossible without advanced integration techniques by using some expensive ad-hoc solutions.

We also observe the same interoperability problem at the data level. There are numerous ways of describing geospatial data in various formats such as ESRI shape files, ASCII files, XML files, Geography Markup Language (GML) files etc. Even further, these different kinds of data are kept in different types of databases such as relational, object and XML databases. Depending on the user's choice of software, applications that digest geospatial data require input in different formats. Users spend significant amount of time converting data from one format to other to make it available for their purposes.

Problems with the data integration mostly come from:

1. Adoption of universal standards: Over the years organizations have produced geospatial data in proprietary formats and developed services by adhering to differing methodologies.
2. Distributed nature of geospatial data: Because the data sources are owned and operated by individual groups or organizations, geospatial data is in vastly distributed repositories.
3. Service interoperability: Computational resources used to analyze geospatial data are also distributed and require the ability to be integrated when necessary.

In order to solve data and service heterogeneities for the GIS computation and data services we use OGC standards in general and mediator services at the lower level of data integration hierarchy. Mediators provide an interface of the local data sources and enable interoperable service interface (Web Feature Services interface) to query and access the data. They have mapping rules that express the correspondence between the global schema (GML) and the data source ones. They change depending on the data type kept at the resource. The problem of answering queries is another point of the mediation integration – a user poses a query in terms of a mediated schema (such as *getFeature* to Web Feature Server), and the data integration system needs to reformulate the query to refer to the sources. The mediator services are explained in Chapter 4.1.

In order to utilize and interact with the coupling framework enabling access to computational and data resources in an ordered and synchronized manner, we develop innovative interactive smart decision making tools (see Chapter 5). Interactive decision making tools enable feeding the Geo-Science applications (projects) with the data and associate the input and output of the applications at the layer (view)-level. The decision making tools also enable the accessing and querying of the attributes of the data/information which are building these comprehensible representations in overlaid layers interactively. Resources are organized into projects and they are compliant with some resource schemas defined in Aggregator Web Map Server (WMS) [5, 99]. The client portal provides both map-based and project based user interfaces based on the capability metadata of initially connected Aggregator WMS. The former are suitable for geospatial resources with location information while the later caters for all kinds of resources with or without location information. The two interfaces co-exist and are synchronized. That is, resources selected using the map-based interface will also be highlighted in the project based interface, and vice versa.

We build services for supporting scientific GIS applications (mostly *ServoGrid* projects [53, 54 and 55]) that includes computation and data resource demanding high-performance and high-rate data transfers. 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. In most cases the amount of collected data reaches to an amount in the order of gigabytes or even terabytes, handling this data becomes a challenge for most users and organizations. Also, simulation and visualization software used in conjunction require high performance computing platforms which are unreachable for common users. We have developed Grid oriented Web Map Web Services in OGC standards and Web Services principles. Grid oriented Map Services enable data integration and layer-overlaid display for the coupling framework by using innovative pre-fetching, load-balancing and caching techniques. It also provides different running modes set before run-time such as streaming or non-streaming data transfer modes.

Non-streaming gives better results for applications using small amount of data (less than 20MB). In other cases for large scale applications streaming mode gives high performance results. We give the detailed performance test results in Chapter 8.

These issues are undeniably the crucial points of the numerous research and development efforts [45, 46]. Especially the problems related to the data formats and standards are being addressed by a number of groups and organizations some of which also offer solutions to the application level interoperability issues [47, 48 and 49]. As it is said before, we generally focus on the issues from the GIS and special data point of view, but findings and recommendations are relevant for any other science domains and data types.

The rest of the paper is organized in accordance with the listing presented above. Chapter 2 highlights the important terms and concepts in our research domain in order for the readers to better understand the discussions in the following chapters. Chapter 3 presents the innovative coupling framework for Geo-Science computational Grid with distributed and heterogeneous data sources. Coupling is done at the view level by introducing 3-layer structured display. Chapter 4 explains the challenges in the heterogeneous data and service integration in the framework and, introduces an innovative approach (mediators) to solve the problem. Chapter 5 presents proposed interactive and smart decision making tools enabling end-users to interact with the coupling framework in a synchronized manner to make decisions over the integrated data display. Section 6 discusses performance issues and proposes innovative pre-fetching, load-balancing and caching techniques for the un-evenly distributed geospatial data. Section 7 explains the key-service architecture in the framework (Grid oriented Aggregator WMS) enabling of coupling at an acceptable performance level. Section 8 gives the preliminary performance tests. Chapter 9 lists the expected contributions and Chapter 10 presents the future work.

2. Background and Definitions

Before starting the issues mentioned in the introduction, this chapter gives the definitions and explanations of some key terms and concepts in proposed GIS Information Grid domain. Among those are: GIS, Web Services, GIS Web Services, Web Map Service (WMS), Web Feature Service (WFS) geospatial data such as raster data and vector data, GML (structured standard common data format in GIS) and GIS standard bodies (OGC and ISO/TC211.

2.1. Geographical Information Systems (GIS)

GIS introduce methods and environments to visualize, manipulate, and analyze geospatial data. The nature of the geographical applications requires seamless integration and sharing of spatial data from a variety of providers. Interoperability of services across organizations and providers is a main goal for GIS and also Grid computing [32, 31].

GIS Grid services can be grouped into three different categories; these are data services, processing services and, registry (catalog) services. Data services are tightly coupled with specific data sets and offer access to customized portions of the data. Processing services provide operations for processing or transforming data in a manner determined by user-specified parameters. Registry or catalog services allow users and applications to classify, maintain, register, describe, search and access information about Web Services.

GIS provide the capability to manage and manipulate geospatial information, including rendering of maps and map-based information. Increasingly, GIS has become a critical tool for managing spatial data in a variety of disciplines. Indeed, creating, managing, analyzing and presenting spatial data is now a fundamental component of many scientific data analyses and decision-support systems, including in the Geo-Sciences. The main thrust of GIS is the integration of heterogeneous data based on co-location in space. Joining (or overlaying) data based on spatial relationships requires that data layers are converted into a common coordinate system [33].

The primary function of a GIS is to link multiple sets of geospatial data and graphically display that information as maps with potentially many different layers of information (see Figure 1). Each layer of a GIS map represents a particular "theme" or feature, and one layer could be derived from a data source completely different from the other layers. As long as standard processes and formats have been arranged to facilitate integration, each of these themes could be based on data originally collected and maintained by a separate organization. Analyzing this layered information as an integrated entity (map) can significantly help decision makers in considering complex choices.

In Figure 1, each layer composing map (or integrated data/information) is an abstraction of different type of vector and raster data (see Chapter 2.4 for the definitions). For example, street data is set of line strings, buildings are represented in points and vegetation data is represented in polygon or poly-lines.

2.2. GIS Web Services

Web Services: The World Wide Web consortium (W3C – http://www.w3c.org) describes Web Services as: "A software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards". Another definition is "A Web Service is an interface that describes a collection of operations that are network accessible through standardized XML messaging." [38].

Web Service standards [34] are a common implementation of Service Oriented Architectures (SOA) ideals, and Grid computing has converging requirements. By implementing Web Service versions of GIS services, we can integrate them directly with scientific application Grids [31, 37]

Web Services give us a means of interoperability between different software applications running on a variety of platforms. Web Services support interoperable machine-to-machine interaction over a network. Every Web Service has an interface described in a machine-readable format. Web Service

interfaces are described in a standardized way by using Web Service Description Language (WSDL) [40]. WSDL files define input and output properties of any service and services' protocol bindings. WSDL files are written as XML documents. WSDL is also used for describing and locating Web Services. Web Services are defined by the four major elements of WSDL, "*portType*", "*message*", "*types*" and "*binding*". Element *portType* defines the operations provided by the Web Services and the messages involved for these operations. Element message defines the data elements of the operations. Element types are data types used by the Web Service. Element binding defines the communication protocols. Other systems interact with the Web Service in a manner as described in WSDL using Simple Object Access Protocol (SOAP) [35] messages. WSDL enables Web Services to be located and invoked remotely through registry and catalog services. Universal Description, Discovery and Integration (UDDI) specification [39] can be used by the service providers to advertise the existence of their services.

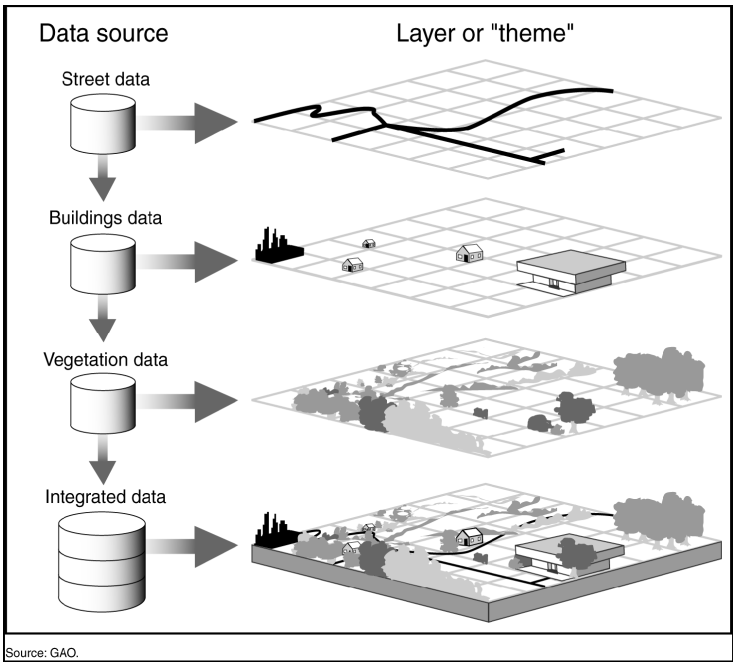


Figure 1: Layered display – a map is composed of distributed multiple set of layers. Figure is from [33]

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

The major difference between the Web Services and the other component technologies is that, the Web services are accessed via the ubiquitous Web protocols such as Hypertext Transfer Protocol (HTTP) and Extensible Markup Language (XML) instead of object-model-specific protocols such as Distributed Component Object Model (DCOM) [43] or Remote Method Invocation (RMI) [44] or Internet Inter-Orb Protocol (IIOP) [42].

GIS Web Services: The Geo-Sciences are a collection of primarily data and observation driven disciplines, yet a mechanism to share collected data and developed software tools has not been widely established. The data collected are stored in several different formats on different platforms. Software developed in the community employs a variety of mechanisms for accessing such data and conduct analysis on them, with little or no collaboration and standards. To make these data available to the larger scientific community and to stimulate integration of resources, earth scientists need to get involved in an infrastructure for integrating this information. In order for scientists to have a common mechanism of accessing and querying their data and tools, they need to have a uniform interface for retrieving information.

As GIS Web Services, we have developed OGC defined GIS services implemented in Web Service principles for our GIS applications. These are called Web Feature Services (WFS) and Web Map Services (WMS). WFS and WMS specify the interfaces for the access to geospatial data sources: WFS provides access to map features (vector data) which are encoded in GML (Geographic Markup Language); WMS produces maps which are encoded as pictures in rater formats like GIF, JPEG, and SVG. These services define the implementation and use of several basic operations.

WFS (Basic) provide 3 operations, *getCapabilities*, *describeFeatureType* and *getFeature*. In case of transactional WFS it provides 2 more service interfaces, *transaction* and *lockFeature*. We developed basic WFS. To access a WFS, a client usually first sends a *getCapabilities* request to the WFS server to learn which feature types it can service and what operations are supported on each feature type; then the client sends a *describeFeatureType* request to get the structure information of the interested feature type; and finally, a *getFeature* request is sent to get the feature data.

In case of WMS, there are again 3 operations provided, *getCapabilities*, *getMap* and *GetFeatureInfo*. WMS provide its data in the layer format. *GetCapabilities* request allows the server to advertise its capabilities such as available layers, supported output projections, supported output formats and general service information. Before a WMS Client requests a map from WMS, it should know what layers WMS provides in which bounding boxes. *GetCapabilities* request enables WMS Clients to obtain this type of information about the contacted WMS. The *getMap* service interface allows the retrieval of the map. *GetFeatureInfo* is an optional WMS service. It is used only when a user needs further information about any feature type on the map. GetFeatureInfo service returns type is user readable text or html formats.

Advantages of the Web Services in GIS area can be grouped into three categories [36]:

Distribution: It will be easier to distribute geospatial data and applications across platforms, operating systems, computer languages, etc. They are platform and language neutral. Web services can be used on different platforms than those on which they were implemented.

Integration: It will be easier for application developers to integrate geospatial functionality and data into their custom applications. It is easy to create client stubs from WSDL files and invoke the services. Web Services based frameworks are loosely coupled and component oriented. Because of the standard interfaces and messaging protocols the Web Services can easily be assembled to solve more complex problems.

Infrastructure: We can take advantage of the huge amount of infrastructure that is being built to enable the Web Services architecture – including development tools, application servers, messaging protocols, security infrastructure, workflow definitions, etc.

2.3. Geospatial Standards

The standard bodies aim is to make the geographic information and services neutral and available across any network, application, or platform. Currently the two major geospatial standards organizations are the Open Geospatial Consortium (OGC) and the Technical Committee tasked by the International Standards Organization (ISO/TC211).

The OGC is an international industry consortium of more than 300 companies, government agencies and universities participating in a consensus process to develop publicly available interface specifications. OGC Specifications support interoperable solutions that "geo-enable" the Web, wireless and location-based services, and mainstream IT. OGC has produced many specifications for web based GIS applications such as Web Feature Service (WFS) [8] and the Web Map Service (WMS) [5, 6]. Geography Markup Language (GML) [10] is widely accepted as the universal encoding for geo-referenced data (see Chapter 2.4 for more detail). On the other hand ISO Standards proposes a standard framework for the description and management of geographic information and geographic information services. OGC has introduced standards by publishing specifications for the GIS services.

Technical Committee 211 (Geographic Information/Geomatics) of the International Organization for Standardization (ISO) [see http://isotc211.org] develops the series of international standards for geospatial data, metadata and services. Overall scope of the series of standards is outlined in ISO 19101 (2002). This work aims to establish a structured set of standards for information concerning objects or phenomena that are directly or indirectly associated with a location relative to the Earth. These standards may specify, for geographic information, methods, tools and services for data management (including definition and description), acquiring, processing, analyzing, accessing, presenting and transferring such data in digital/electronic form between different users, systems and locations. ISO/TC 211 did not specify the actual implementation specifications for different platforms and the private software vendors. Instead, ISO/TC 211 defines a high-level data model for the public sector, such as governments, federal agencies, and professional organizations [41].

In summary, OGC is interested in developing both abstract definitions of OpenGIS frameworks and technical implementation details of data models and to a lesser extent services. On the other hand, ISO/TC 211 focuses on high-level definition of geospatial standards from an institutional perspective [41]. They have been working closely to align their work to produce compatible standards.

2.4. Geospatial Data and Structured Common Data Format-GML

Geospatial data, in general, refers to a class of data that has a geographic or spatial nature, e.g., the information that identifies the geographic location and characteristics of natural or constructed features and boundaries on the earth.

Geospatial data represents real world objects (roads, land use, elevation) with digital data. Real world objects can be divided into two abstractions: discrete objects (a house) and continuous fields (rain fall amount or elevation). There are two broad methods used to store data in a GIS for both abstractions: Raster and Vector.

Raster data is called coverage data by OGC. Raster data type consists of rows and columns of cells where in each cell is stored a single value. Most often, raster data are images (raster images), but besides just color, the value recorded for each cell may be a discrete value, such as land use, a continuous value, such as rainfall, or a null value if no data is available. Raster data is stored in various formats; from a standard file-based structure of TIF, JPEG, etc. to binary long object (BLOB) data stored directly in a relational database management system (RDBMS) similar to other vector-based feature classes.

Common data format for the raster data in our system: In our GIS system we use image formats such as JPEG or TIFF to represent the raster data provided by third party OGC compatible Web Map Services or Coverage Portrayal Services (CPS) [13].

Vector data type uses geometries such as points, lines (series of point coordinates), or polygons, also called areas (shapes bounded by lines), to represent objects. Examples include property boundaries for a housing subdivision represented as polygons and well locations represented as points. Vector features can be made to respect spatial integrity through the application of topology rules such as 'polygons must not overlap'. Vector data can also be used to represent continuously varying phenomena.

Common data format for the vector data in our system: The data model developed by OGC is the Geography Markup Language (GML) and it is currently widely accepted as the universal encoding for geo-referenced data. GML is an XML grammar written in XML Schema for the modeling, transport, and storage of geographic information including both the spatial and non-spatial properties of geographic features; it provides a variety of kinds of objects for describing geography including features, coordinate reference systems, geometry, topology, time, units of measure and generalized values. (See APPENDIX 6)

Just as XML helps the Web by separating content from presentation, GML does the same thing in the world of Geography. GML allows the data providers to deliver geographic information as distinct features. Using latest Web technologies, users can process these features without having to purchase proprietary GIS software. By leveraging related XML technologies such as XML Schema, XML Data Binding Frameworks, XSLT, XPath, XQuery etc. a GML dataset becomes easier to process in heterogeneous environments.

By incorporating GML in our systems as common data format we gain several advantages:

1. It allows us to unify different data formats. For instance, various organizations offer different formats for position information collected from GPS stations. GML provides suitable geospatial and temporal types for this information, and by using these types a common GML schema can be produced. See the APPENDIX 6 for a sample GML.
2. As more GIS vendors are releasing compatible products and more academic institutions use OGC standards in their research and implementations, OGC specifications are becoming de facto standards in GIS community and GML is rapidly emerging as the standard XML encoding for geographic information. By using GML we open the door of interoperability to this growing community.
3. GML and related technologies allow us to build general set of tools to access and manipulate data. Since GML is an XML dialect, any XML related technology can be utilized for application development purposes. Considering the fact that in most cases the technologies for collecting data and consecutively the nature of the collected data product would stay the same for a long period of time the interfaces we create for sharing data won't change either. This ensures having stable interfaces and libraries.
4. One approach to achieve machine to machine communications and autonomous computations.
5. It enables separating representation from the context.
6. Since it is XML based, it can be leveraged to other XML based systems and communication protocols such as XMLHttpProtocol (in other words AJAX) and Web Services [30]
7. It is an approach to achieving cross-language interoperability.
8. Using GML with the capability metadata as OGC defined is a kind of application of the semantic approaches to data and service integrations and coupling.

Due to the numerous advantages of using semi-structured data representation, other science domains also adapt using this kind of structured representation of data. For example, Chemistry domain uses CML (Chemistry Markup Language), Astronomy domain uses VOTable (Virtual Observatory Tables), Physics science domain uses PhysicsML (Physics Markup Language), Mathematic science domain uses MathML (Mathematic Markup Language) etc.

3. Coupling Computation and Data Sources in GIS

Coupling is done through the Aggregator WMS [99] which is the key service in GIS and in or proposed framework. The Aggregator WMS is actually an OGC compatible WMS implemented in Web Service principles. The coupling is done at the view level. Throughout the document we sometimes call it layer level. In order to achieve that kind of coupling or integration, the Geo-Science Grid's outputs and Map Server's output should be represent-able in layers and synchronized as well. Since Aggregator WMS is basically an OGC compatible WMS with some extensions, it already provides data/information in layers.

However, in order to be able to associate or couple it with Geo-Science Grids we have created Sci-Plotting Web Services [99].

Scientific plotting Web Services are based on Dislin [24] plotting libraries developed originally in C language. After having wrapped them as Web Services [4], we enable these libraries to be used by Science community. To create a complete framework for coupling we also have developed intermediary service enabling Google Map servers to be used as Web Map Services (see Chapter 5.2). This is also explained in a separate document [30].

As an output of this coupling we introduced 3-layer structured display. Layer order is important to create reasonable and human interpretable maps. Please see the Figure 2. In summary, layer sets 1 and 2 come from geo-data Grid and, layer set 3 come from Geo-Science Grid through Sci-plotting Web Services. *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 the 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.

Figure 2: General structure of the outputs of the coupling framework.

Related to the layer structures displayed above, there are three major components in the architecture (See Figure 3). These are Aggregator Map Services (see also Chapter 7), Sci-Plotting Services and User Portal (services in grey colors in Figure 3). Throughout the document, the terms “user portal” and “interactive decision making tools” are used interchangeably.

Layer-1 and *Layer-2* come from Aggregator 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 (see Chapter 5).

We have created our proposed framework in Web Service principles. 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*.

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. GML is an XML based data representation. 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 20*). Using XML based data representation decreases the performance severely. Geo-Science applications use large data in size. There is a trade off here that we propose a solution approach in Chapter 6.

Coupling framework is illustrated in *Figure 3*. Data Grid is represented with triangle. Computational Grid is the remaining set of services. Geo-Science Grid is represented as blue ellipse integrated to the data Grid through Job Manager, WS-Context and Sci-Plotting services.

3.1. Major Components in the Framework (Figure 3)

Web Feature Service (WFS) [8] (by G. Aydin) provide geo-data 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 [83] 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 the layers. The 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 3. 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).

Aggregator WMS [5, 99] 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. Moreover, Aggregator WMS also provides Google Map layers in the image format such as JPEG which can be archived and manipulated depending on the application purposes. Aggregator WMS is expected to improve the performance compared to the 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 7) and, provides different running modes set before run-time such as streaming or non-streaming data transfer modes.

User Science Portal (Interactive decision making tools): Resources are organized into projects (see Chapter 3.4) and they are compliant with some resource schemas. The client portal is capable of

supporting capabilities metadata of Web Map and Web Feature Services. The client portal provides both map-based and project based user interfaces. The former are suitable for geospatial resources with location information while the later caters for all kinds of resources with or without location information. The two interfaces co-exist and are synchronized. That is, resources selected using the map-based interface will also be highlighted in the project based interface, and vice versa. When it is deployed on a portal as a portlet, varying levels of access and authorization can be set.

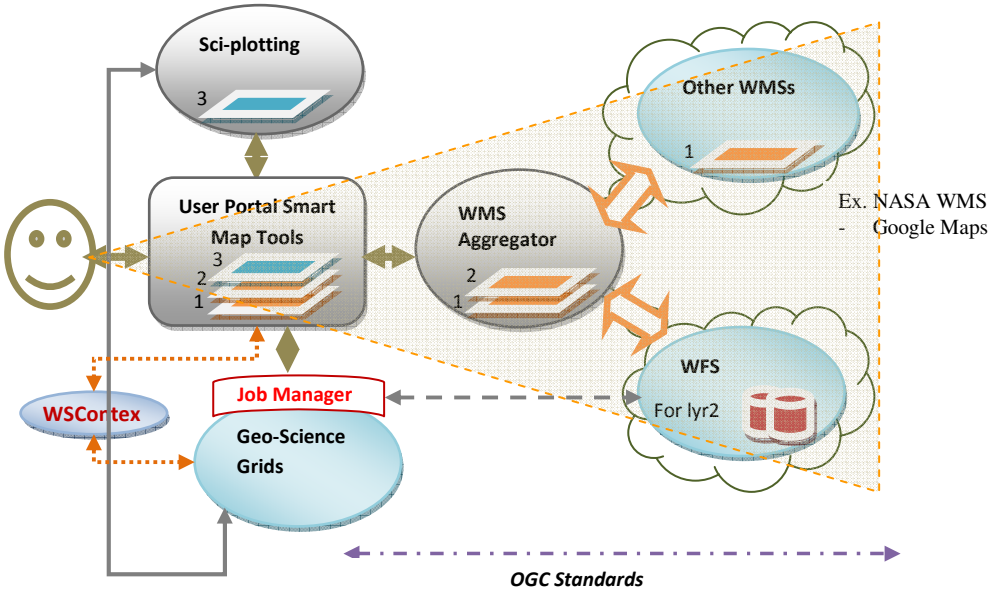


Figure 3: Illustration of coupling framework.

Sci-Plotting Services [98]: For the core functionality we use Dislin scientific-data plotting libraries [24]. 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 3. 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 2).

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) [84]. 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 [53, 54 and 55]: Geo-Science Grids (or applications) are based on processing and analyzing of the data (called as features, spatial data or temporal data) related to the earth. They are composed of processing and analyzing services, intermediary services for transformations of data (spatial reference systems, projections, geometry elements etc) and data services. In our framework we take them as a black box. All the communication with this back box is handled by the job manager. As Geo-Science Grids we have been using ServoGrid projects [22, 21]. See the Chapter 3.4 for motivating use cases and sample Geo-Science Grids.

3.2. How the Components Work Together

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

The coupling framework enables scientific users to analyze the remote and distributed heterogeneous data over 3-laye structured display (*Figure 2*) through browser based interactive decision making tools. (See APPENDIX 10 and 11). The 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 (vector data), and analyzing the results interactively. The process is diagrammatically illustrated in Figure 3.

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 2 and Figure 3).

User interface provided by Coupling framework client service 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 (see sample request at APPENDIX 17). 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 (See APPENDIX 1) 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.

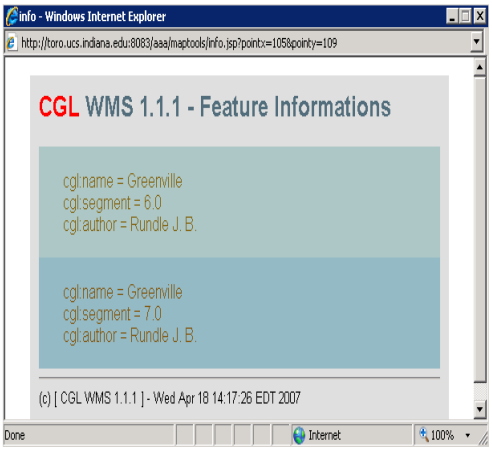


Figure 4:A getFeatureInfo query-response

Data used for creating layer set 2 is queried interactively through WMS's "*getFeatureInfo*" (See APPENDIX 3) Web Service. When any WMS client sends a *getFeatureInfo* request to WMS, WMS creates a *getFeature* (See APPENDIX 5) 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 6.1. Returned feature data is converted to comprehensible format by using our creation of generic XSL file used for converting GML data into HTML pages. After creating human readable results of the query, WMS send it to EMS client. See the sample window popping up at client's browser (Figure 4) and, see the generic XSL file used to convert GML to HTML at APPENDIX 12.

Nearest Neighbor query for *getFeatureInfo*: User science portal (GUI) allows users to access attribute information about any map feature displayed in the GUI. Users can access the attribute information for any thematic layer, by clicking on map features displayed in the GUI browser. Considering that most objects displayed on the map are either thin lines or small points, it would be relatively rare that a user would click on the actual object that they are interested in. it is assumed that generally users tend to click in the immediate vicinity of an object that they are interested in, rather than on object itself. For this reason, a nearest neighbor query was used for retrieving information about a map feature.

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.

In order to invoke Geo-Science Grid, 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 3). 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.

The whole system is implemented using the client-server architecture paradigm. The client provides the GUI interface for users to access all the available services. 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 2).

3.3. Chaining of the Components

Components: OGC Web Services in triangle shown in Figure 3

Service chaining is a mechanism for assembling of services i.e., the process of combining or pipelining results from several complementary services to create customized data or knowledge. The service composition techniques provide mechanisms to combine the services to allow composite services to be defined and executed. It allows repetitive tasks to be automated and specialized services to be defined to target specific application areas in situation understanding and decision making. The service chaining-composition techniques can be grouped into three. These are client-coordinated service chaining, static chaining using aggregate services and workflow-manager service chaining [85]. In our proposed coupling framework we use workflow-managed service chaining (for Geo-Science grids) and static-chaining using by using aggregator service.

Regarding the remote service discovery and binding for the chaining, there are two general approaches. These approaches can be grouped as dynamic approaches and static approaches. Dynamic approaches are based on catalog-registry services (such as UDDI) and static approaches are based on pre-defined path of chained services described in semi-structured XML files (suchs as OGC's capability metadata definitions for WMS and WFS).

In order to find the services to chain we use static chaining at the beginning by defining their address manually before run-time. After system start running, due to the inter-service communication capabilities of the system, chaining becomes partially dynamic. WMS and WFS exchange their capabilities and keep their own local capabilities file up-to-date. We use static service chaining approaches by using the capability metadata defined by OGC standard specifications.

Workflow managed service chaining is a balance between opaque static aggregate-service chaining and transparent client-coordinated chaining. It replaces aggregate services with smarter mediating services that act as gateways. These services offer access to data and processes, but they do not necessarily serve that data themselves. They retrieve it from other services. This type of service chaining is used for chaining Geo-Science Grid applications. This is out of scope for this paper. We use HPSearch as Job Manager and it provides scripting based workflow managing. Job manager shown in *Figure 3* is actually a workflow-managed scripting based chaining and orchestration service.

This chapter focuses on GIS Web Service chaining. As we mentioned before, we use static-chaining technique using Aggregator WMS orchestrating data Grid shown as triangle in *Figure 3*. We focus here pre-defined and capability based linking-chaining of the services without using UDDI [82] registry catalog services. Aggregate services, which third-party providers usually supply, bundle static (predefined) chains of services and present them to the client as one. The chaining becomes totally opaque to the client, which sees only a single aggregate service that coordinates the individual services in the chain.

Why service chaining is needed: We can summarize the necessities and advantages of the service chaining in Information Grids as: (1) load balancing, (2) fault tolerant which is mostly related to the first one, (3) creating more specific and complex data/information, (4) filtering and re-naming of data/information and (5) enabling integration of mediator-adaptor services into the system. These are all needs to be explained in separate chapters but, in this chapter we focus on only the chaining techniques, i.e. how to do the OGC Web Services' chaining.

3.3.1. OGC's Considerations on Chaining of Services

OGC's WMS and WFS services are inherently capable of being cascaded and chained in order to create more complex data and information according to their specifications. In order to standardize these issues OGC introduced Web Map Context (WMC) [70] standard specifications. Before that, OGC was recommending application developers to extend their services' capabilities for the cascading. WMC is actually a companion specification to WMS, but for WFS there is not any definitions and standardization for the cascading. Therefore, for WFS to WFS cascading, developers still need to add cascading specific extension to their WFSs' capabilities files.

WMC is needed when a map comprising layers from several distinct servers being built up in one viewer client, the creation of a platform-independent description of that map, the retrieval of that description by an entirely different Client, and the display of the map in the second Client. The present Context specification states how a specific grouping of one or more maps from one or more map servers can be described in a portable, platform-independent format for storage in a repository or for transmission

between clients. This description is known as a "Web Map Context Document," or simply a "Context." Presently, context documents are primarily designed for WMS bindings. However, extensibility is envisioned for binding to other services [70].

A Context document is structured using XML and its standard schema is defined in the WMC specifications. A Context document includes information about the server(s) providing layer(s) in the overall map, the bounding box and map projection shared by all the maps, sufficient operational metadata for client software to reproduce the map, and additional metadata used to annotate or describe the maps and their provenance for the benefit of end-users.

There are several possible uses for context documents besides providing chaining and binding of services. The context document can provide default startup views for particular classes of user. For example specific applications require specific list of layers. The context document can store not only the current settings but also additional information about each layer (e.g., available styles, formats, SRS, etc.) to avoid having to query the map server again once the user has selected a layer. Finally, the Context document could be saved from one client session and transferred to a different client application to start up with the same context. In this document, we just focus on its binding functionalities.

As you see from the OGC definitions, currently they do not have mature system dn standards for chaining the services. They propose Web Map Context standards but in the future they plan to extend the capabity file and embed these definitions and elements of Context into WMS capability file. So, currently there are two possible ways one is extending the WMS capability file other is using Web Map Context's standards defining chaining in a context document.

Cascading concept in OGC: The term "cascading WMS" is widely used but not standardized clearly by OGC. If a WMS provides a layer whose attribute "*cascade*" has a value different from 0 then that WMS is called cascading WMS. A cascading WMS is a WMS which can read other WMS and display layers from them.

A Layer is said to have been "cascaded" if it was obtained from an originating server and then included in the service metadata of a different server. The second server may simply offer an additional access point for the Layer, or may add value by offering additional output formats or re-projection to other coordinate reference systems. In the same way a feature data is said to be cascaded if it was obtained from an originating server and then included in the service metadata of a different server. Since WFS to WFS cascading is not standardized yet we do that by extending WFS capabilities as it is shown in Chapter 3.3.2.2.

If a WMS cascades the content of another WMS, then it shall increment the value of the *cascaded* attribute for the affected layers by 1. If that attribute is missing from the originating server's service metadata, then the Cascading WMS shall insert the attribute and set it to 1 [5, 6].

3.3.2. Proposed Chaining Techniques

Chains are composed of sets of WMSs and WFSs supporting *getCapability* Web Service interface to exchange their capability document. Both types of services have their own standard capability schema (See APPENDIX 13 and 14). Chaining of the services is done basically through capability based inter-service communication. Communication is done between the chained services for the cascaded data and other interface level upgrades. A WMS can cascade multiple WMSs and WFSs. Contrary, a WFS can cascade only WFSs.

A snapshot of the chaining scenario is given in Figure 5. In the figure, possible chains are A, A+B, A+C, A+B+C. Chains B and C are similar to A with different combinations of WMS, WFS and adapter/mediator services. The chains are based on abstraction of data as layer and definition in the capability metadata. As we mentioned before our chaining approach is a static approach and based on aggregate service (AWMS in Figure 5).

The layer-data abstractions are organized into applications. The application in this respect means the set of layer-data which are the chains of WMSs and WFSs. Applications are defined in Aggregator WMS's capabilities file. For example, see the layer list at the browser in Figure 5 for the Pattern Informatics application. Each one of these layers represents a different chain and defined in AWMS capabilities file. For the current demo in the below figure, user selects layers "NASA Satellite" and "World Seismic".

Here is the rough definition of applications and layers as chains in Aggregator WMS. We give more details in the following sub-sections.

<!—This is shown as project names list at the top as shown in Figure above as dropdown list "Select Layers for ". Please see the below figure browser -->

<layer Type="Applications" Name= "Pattern Informatics">

<!—This is shown as a layer name listed below selected project name -->

<SubLayer type="Chain" name="Nasa Satellite">

<!-- Some other tags –see WMS schema definition file -->

<Sub Layer type="Chain" name="Google Map">

<!-- Some other tags –see WMS schema definition file -->

</SubLayer>

……..

….

<Sub Layer type="Chain" name="World Seismic">

<!-- Some other tags –see WMS schema definition file -->

</SubLayer>

</Layer>

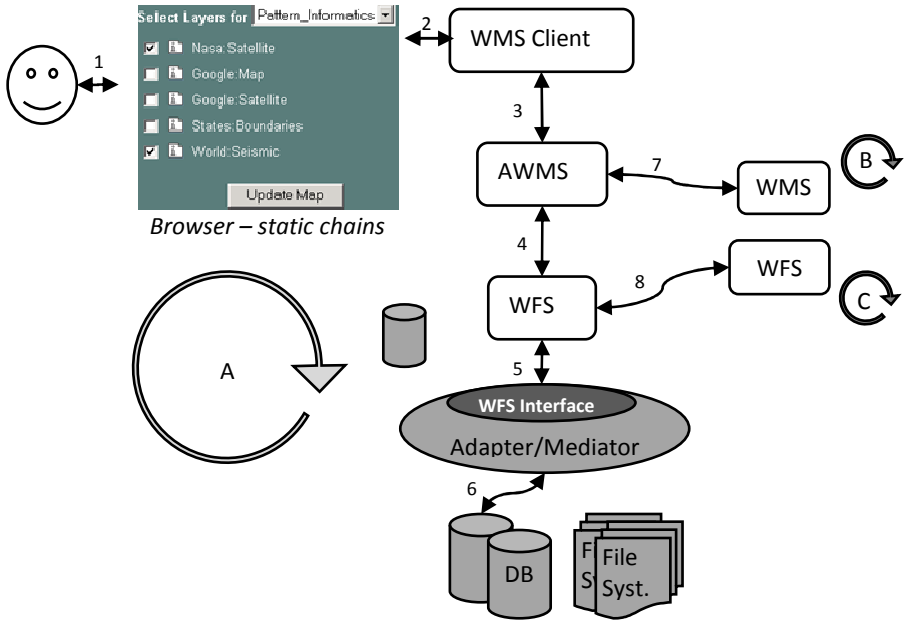


Figure 5: Illustration of chaining of OGC Web Services

Regarding querying through the chain, each service (WMS or WFS) has standard service interfaces and a capability file and, request and responses are in OGC defined redefine formats. Therefore, query handling is done with the help of capabilities documents and query re-writing techniques. For more information about the querying see the Chapter 4.

There are two types of services in our proposed GIS systems composing our framework, WMS and WFS. Therefore, all the possible alternative bindings of the services are grouped as:

- WMS to WMS (Chapter 3.3.2.1),
- WMS to WFS (Chapter 3.3.2.1),
- WFS to WFS (Chapter 3.3.2.2).

The first two are WMS initiated chaining and, chaining is achieved by using the same technique explained in chapter 3.3.2.1. Whereas, the last one is WFS initiated and its techniques is explained in chapter 3.3.2.2.

3.3.2.1. WMS Cascades other WMS and/or WFS

Links 7 and 4 in Figure 5

There are two possible ways to define binding of services in this category. First one is using context document created according to Web Map Context specifications. Second one is extending WMS

capabilities file for the cascaded layers. Here we will not explain the structures of the capabilities metadata and context document. We illustrate these options based on real application scenarios.

Let's assume end-user selects the layers "Nasa Satellite" and "World Seismic". "Nasa Satellite" is a good example of WMS to WMS cascading and, "World Seismic" is a good example of WMS to WFS cascading. Furthermore, chaining in the form of cascading and bindings are defined in Aggregator WMS. According to our architecture, Aggregator WMS doesn't care and even doesn't know whether the cascaded layers are re-cascaded at the following servers in the chain. It just looks at the cascaded server's binding information and attributes of the cascaded data.For example, WMS at CGL (Community Grids Labs) cascade satellite data from WMS at NASA JPL (Jet Propulsions Labs). WMS at CGL lab does not case if the layers are re-cascaded at the NASA JPL.

This chapter presents two options to set up cascading from WMS to WMS and/or WFS. First option proposes a solution by using context document. Second option proposes a solution by extending WMS's capabilities file via adding DataURL element to the cascaded data.

Option-1 Context document:

Context document idea is based on defining the chain in machine readable format. Below is the sample context document defining the chaining of the "Nasa Satellite" and "World Seismic" layers. We just give the definition of the cascading in machine readable format. We do not explain the bottom level implementation details.

```
<ViewContext version="1.0.0" id="OGCContext" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <General>
  <Window width="500" height="400" />
  <BoundingBox srs="EPSG:4326" minx="-180.00" miny="-90.00" maxx="180.00" maxy="83.62" />
  <Title>Maps for Pattern Informatics Application</Title>
  <Abstract />
 </General>
  ….
 <LayerList>
  <Layer queryable="1" hidden="0">                    ——————————→   WMS to WFS cascading
   <Extension infoFormat="text/xml" ID="4e4b-83e" editable="0" local="1" />
   <Server service="WFS" version="1.1.0" title="CGL_WFS">
    <OnlineResource xlink:href="http://cgl/wfs/services" />
   </Server>
   <Name>World Seismic</Name>
   <Title>Earthquake Seismic Data</Title>
   <Abstract>Sample WMS to WFS layer cascading</Abstract>
   <DataURL format="text/xml">
    <OnlineResource xlink:href="http://cgl/wfs/services" />
   </DataURL>
   <SRS>EPSG:4326</SRS>
   <FormatList>
    <Format current="1">image/png</Format>
   </FormatList>
```

```
    …..
   </Layer>
   <Layer hidden="0">                                    WMS to WMS cascading
    <Extension infoFormat="text/html" ID="1fc-4e4b-83e" editable="0" local="1" />
    <Server service="WMS" version="1.1.1" title="CGL_WMS">
     <OnlineResource xlink:href="http://nasawmsserver/wms/services " />
    </Server>
    <Name>Nasa Satellite</Name>
    <Title>Nasa Satellite Data</Title>
    <Abstract>Sample WMS to WMS layer cascading</Abstract>
    <DataURL format="text/xml">
     <OnlineResource xlink:href="http://nasawmsserver/wms/services" />
    </DataURL>
    <SRS>EPSG:4326</SRS>
   </Layer>
   …..
  </LayerList>
  …
</ViewContext>
```

The unnecessary details at the above context file are truncated. We just use related elements and tags for the data cascading and service binding.

Option-2 capability extension:

This specific part of WMS 1 capability metadata (for the google map data) shows that Google map is going to be obtained from "*href=http:// WebService_Address/"* in the form of "image/gif" and in the bounding box defined in "*EX_GeographicBoundingBox*" sub element.

```
<Layer cascaded="1">
        <Name>NASA Satellite</Name>
        <Title> NASA Satellite </Title>
        <EX_GeographicBoundingBox>
                <westBoundLongitude>-180</westBoundLongitude>
                <eastBoundLongitude>180</eastBoundLongitude>
                <southBoundLatitude>-90</southBoundLatitude>
                <northBoundLatitude>90</northBoundLatitude>
        </EX_GeographicBoundingBox>
        ……
        …
        <DataURL>
                <Format>image/gif</Format>
                <OnlineResource xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="simple"
                        xlink:href=" http://toro.ucs.indiana.edu/cgi-bin/wms0.cgi?" />
        </DataURL>
</Layer>
```

A Map Server may use DataURL to offer more information about the data underneath a particular layer. While the semantics are not well-defined, as long as the results of an HTTP GET request against the DataURL are properly MIME-typed, viewer clients and cascading Map Servers can make use of this.

3.3.2.2. WFS Cascades Other WFS

Links 5 and 8 in Figure 5

This type of cascading is used more importantly for proposed WFS-based mediating services (Chapter 4) to solve heterogeneity problem based on common data model and service interfaces. Here we define the cascading of feature data (encoded in GML) and WFS services based on OGC's standard specifications. In order to cascade the WFS services we need to extend our WFS services as XLink-WFS providing their data in common data format through *GetGMLObject* Web Service interface. Link 5 in Figure 5 illustrates these issues. Xlink communication between two WFS is actually called re-direction instead of calling cascading as in WMS case.

There are three classes of WFS defined in the OGC specification. These are Basic-WFS, XLink-WFS and Transaction-WFS. We have implemented Basic-WFS for our GIS Grid framework. Basic-WFS is a read-only WFS that supports the *GetCapabilities*, *DescribeFeatureType* and *GetFeature* requests. In order to enable WFS services (and indirectly data) cascading we need to extend our basic WFS to XLink-WFS. XLink-WFS add the *GetGmlObject* operation, including local and/or remote XLinks capabilities to the Basic-WFS. *GetGmlObject* operation enables retrieving GML instances by following XLink references.

GetGMLObject operation allows clients to retrieve features and element instances by their XML id. The requested element could be any identified element such as a feature, geometry, or a complex attribute. The XML Linking language (XLink) allows elements to be inserted into XML documents in order to create and describe links between resources.

Defining re-directed data (cascaded) in the capability file: If the GetGmlObject operation is supported, for local resources, remote resources, or both, then this fact must be advertised on the capabilities document as described below

```
<ows:OperationsMetadata>
…..
<ows:Operation name="GetGMLObject">
    <ows:DCP>
        <ows:HTTP>
                <ows:Post xlink:href="http://www.BlueOx.org/wfs"/>
        </ows:HTTP>
    </ows:DCP>
    <ows:Parameter name="outputFormat">
        <ows:Value>text/xml; subtype=gml/3.1.1</ows:Value>
        <ows:Value>text/xhtml</ows:Value>
    </ows:Parameter>
    <ows:Parameter name="LocalTraverseXLinkScope">
        <ows:Value>0</ows:Value>
        <ows:Value>*</ows:Value>
    </ows:Parameter>
```

```
      <ows:Parameter name="RemoteTraverseXLinkScope">
          <ows:Value>0</ows:Value>
          <ows:Value>*</ows:Value>
      </ows:Parameter>
   </ows:Operation>
…..
</ows:OperationsMetadata>
```

This should be added to XLink-WFS providing WFS interface to heterogeneous resources displayed in Figure 4.

In addition provided (cascaded) data at the XLink-WFS side should be defined in its capabilities file as below:

```
<wfs:ServesGMLObjectTypeList>
      <wfs:GMLObjectType xmlns:bo="http://www.BlueOx.org/BlueOx">
          <wfs:Name>bo:OxType</wfs:Name>
          <wfs:Title>Babe's Lineage</wfs:Title>
          <wfs:OutputFormats>
               <wfs:Format>text/xml; subtype=gml/3.1.1</wfs:Format>
               <wfs:Format>text/xhmtl</wfs:Format>
          </wfs:OutputFormats>
      </wfs:GMLObjectType>
</wfs:ServesGMLObjectTypeList>
```

This section defines the list of GML Object types, not derived from gml:AbstractFeatureType, that are available from a web feature service that supports the GetGMLObject operation. These types may be defined in a base GML schema, or in an application schema using its own namespace

Requesting cascaded data: The XML encoding of a GetGmlObject request is defined by the following XML Schema fragment:

Semi-structured XML-encoded for the Web Service based invocations:

```
<xsd:element name="GetGmlObject" type="wfs:GetGmlObjectType"/>
<xsd:complexType name="GetGmlObjectType">
    <xsd:complexContent>
        <xsd:extension base="wfs:BaseRequestType">
          <xsd:sequence>
            <xsd:element ref="ogc:GmlObjectId"/>
          </xsd:sequence>
          <xsd:attribute name="outputFormat"
                    type="xsd:string" use="optional" default="GML3"/>
          <xsd:attribute name="traverseXlinkDepth"
                              type="xsd:string" use="required"/>
          <xsd:attribute name="traverseXlinkExpiry"
                              type="xsd:positiveInteger" use="optional"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

Or HTTP GET:

http://www.someserver.com/wfs.cgi?
SERVICE=WFS&VERSION=1.1.0& REQUEST=GetGmlObject& TRAVERSEXLINKDEPTH=2&
EXPIRY=1& GMLOBJECTID=t1

And the response:

```
<Town gm:id="t1">
        <gml:name>Bedford</gml:name>
        <gml:directedNode orientation="+"> <!-- xlink:href="#n1"
                <gml:Node gml:id="n1">
                        <gml:pointProperty
                        xlink:href="http://www.bedford.town.uk/civilworks/gps.gml#townHall"/>
                </gml:Node>
        </gml:directedNode>
</Town>
```

3.4. Motivating Use Cases

We have been applying our proposed information grid architecture over some Geo-Science applications. We group these applications (projects) into two. These are iSERVO (The Solid Earth Virtual Observatory) GIS Grid Research Project and IEISS (The Interdependent Energy Infrastructure Simulation System).

This Chapter presents some of these projects as proof of concepts and make our proposed Information Grid more clear.

3.4.1. 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) [54] 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 2*) 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.

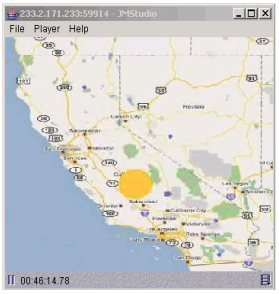
Layer set 3 is created from the PI simulation outputs. Sci-Plotting server gets the output through HTTP protocol and plot the image as layer. The result is a map which shows the fluctuations in seismic activity which are found to be related to the preparation steps for large earthquakes. This map layer 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.

Figure 6: A snapshot from Pattern Informatics Information Grid application

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 2). We get earthquake seismic data (Layer set 2 in Figure 2) 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.

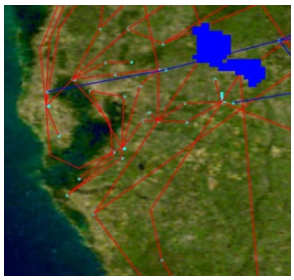
Figure 7: A snapshot from Virtual California Information Grid application

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.

3.4.2. 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.

Figure 8: A snapshot from IEISS Information Grid application

In the following chapter (Chapter 4), we explain our approach to possible data and service heterogeneity problems encountered during the development of the above projects. Moreover, In Chapter 5, we present the ways to interact and utilize the proposed Information Grid via interactive and smart mapping tools.

4. Data and Service Heterogeneities in GIS

Expansion of World Wide Web has brought better accessibility to information sources. However, in the same time, the big amount of different formats, data heterogeneity, and machine un-readability of this data have caused many problems. Data heterogeneity is related to both the data types and storage formats. As geospatial data is stored in a variety of systems and formats, one important issue for geospatial Information Grid applications is service interoperability and data heterogeneity. In other words, the seamless integration and sharing of geospatial data from distributed heterogeneous data sources has been the major challenge of the Information system communities.

We are dealing with the data heterogeneity and interoperability problems in GIS domain. Heterogeneity of geographic resources may arise for a number of reasons, including differences in projections, precision, data quality, data structures and indexing schemes, topological organization (or lack of it), set of transformation and analysis services implemented in the source. The geo-data is accessed via the chain of WFS and WMS services. Upper levels are mostly occupied by WMSs and lower levels are occupied by WFSs. At the bottom level mediators are located. WFSs play the role of wrappers in the mediation system (Chapter 4.1.2). Our integration system uses GML as global data model to represent and manipulate geographic data in vector format. GML is the geographic XML based language.

We face the service interoperability and data heterogeneity issues in our propose Information Grid due to the requirements of uniform data access and integration. By the data integration we mean providing one global view over several data sources and let them processed as one source. There are two general issues here. The first is the data modeling (how to integrate different source schemas); the second is their querying (how to answer to the queries posed on the global schema). The integration process is not easy. Yet, there is no universal tool or method that could be used every time when needed. Nevertheless, there are some partial solutions in many research areas.

The proposed information Grid (In *Figure 3*) is interoperable in itself because, it uses common data format defined by OGC standards (GML and layers in image formats) and, common data and computation services whose standards defined by OGC. Since it is OGC compatible and implemented in SOA principles through Web Services, it is easy to extend and enhance the system with some other computation and data sources. But, when we do that, we will have interoperability and heterogeneity problems. In order to handle these issues we propose two types mediator/adapter services (Chapter 4.1).

In this chapter we explain the service interoperability and data heterogeneity challenges and our approaches to their solution in Information Grid applications to GIS.

4.1. Mediator Systems

We solve the data integration and uniform access and querying problems by developing our architecture in Web Service principles and OGC standards. However, in order to be able to integrate third party data sources and computation resource to our system we propose mediator-wrapper components.

Mediators provide an interface of the local data sources and, play the roles of connectors between local source backgrounds and the global one. The principle of integration is to create non-materialized view in each mediator. These views are then used in the query evaluation. Essential are mapping rules that express the correspondence between the global schema (GML) and the data source ones. The problem of answering queries is another point of the mediation integration – a user poses a query in terms of a mediated schema (such as *getFeature* to WFS), and the data integration system needs to reformulate the query to refer to the sources. Therefore, information integration architecture emerges based on a common intermediate data model (GML) providing an abstraction layer between legacy storage structures and exposed interfaces. In our system we use OGC to enable these interfaces. GML provides a semantic abstraction layer for data files, and is exposed through higher level data delivery service called WFS.

Mediator-based systems support homogeneous views in a common data model over heterogeneous data sources. The proposed mediator systems are based on a 3-level architecture, which include a foundation layer (databases with wrappers), a mediation layer (which supports exchange of queries and results between wrapped legacy data sources and applications), and an application/user interface layer [75]. The advantage of this architecture is its modularity and scalability. These systems support combining query results from individual sources rather than combining the data. In addition, the use of semi-structured data model at the mediator enables the modeling of sources with no structure or implicit structure. Examples of such semi-structured mediator-based systems include TSIMMIS [65, 66], DISCO [67], and Information Manifold [68].

Figure 9 illustrates 3-level federation architecture as a means of extension to OGC + Web Service based Information Grid proposed in *Figure 3*. Levels 1 and 2 in Figure 9 are actually triangle displayed in *Figure 3*. This chapter proposes Layer 3 as a solution to integrate third party heterogeneous data and services to our proposed Information Grid framework.

The mediators-wrappers enable data sources integrated to the system conform to the global data model (such as GML in GIS), but enable the data sources to maintain their internal structure and, at the end, this whole mediator system provides a large degree of autonomy. With this regards, mediator-based federations differ from loose-federations that have no explicit common policy. They also differ from distributed database solutions that mandate a uniform internal structure [76]. Sometimes, successful federal models may be adopted by federation members also as their own internal models. This avoids the need for mediation and imposes a unique model which overcomes heterogeneity.

One more difference with the federate systems is that mediator-based information systems are targeted at read-only applications. On the other hand, federated models tend to be very detailed, to encompass different approaches avoiding oversimplification. Mathematically, given n different systems to interconnect, n^2-n different adapters are necessary [78].

Mediator systems include adapters. They change based on the data they provide and global data model. Some example of adapters are Table to GML, formatted file to GML, HTML to GML, XML to GML and relation tables to GML.

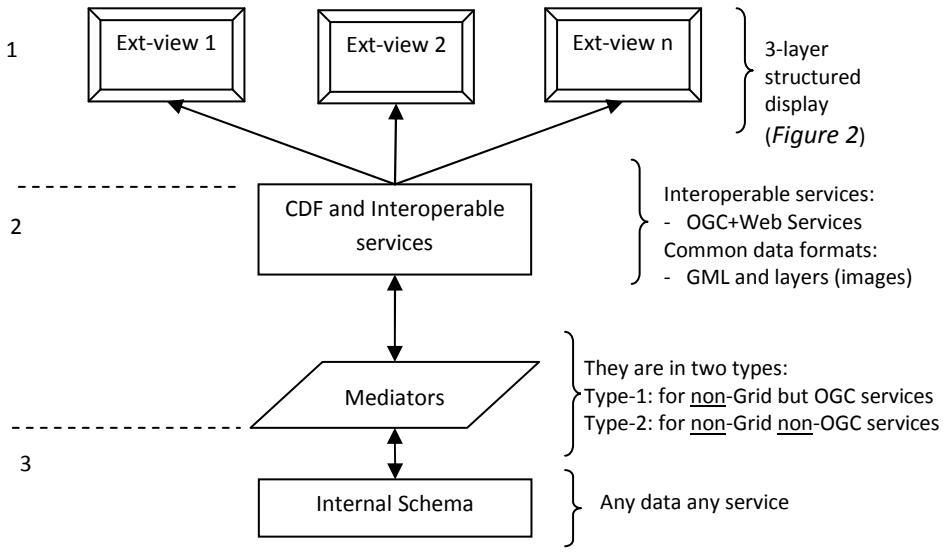


Figure 9: 3-layer general mediation architecture

In Figure 9, Layer 1 consists of layer level data integration and layer-overlaying. Layer 2 consists of GML data and layer images integration and, Layer 3 consists of heterogeneous data provided by any standalone (database and file systems) or federated data sources (SRB or FDBS). There are several advantages in adopting the approach shown in Figure 9. First of all, the integration process does not affect the individual data sources functionality. These nodes can continue working independently to satisfy the requests of their local users. Local administration maintains the control over their systems and yet provides access to their data by the global users at the federation level. Here, we not only handle the data heterogeneity but also any operating system, hardware and, service and communication platform heterogeneities by developing mediators as WFS-based Web Service.

We group the mediators into two and call them mediators type-1 and type-2. Type-1 mediators (Chapter 4.1.1) deal with the horizontal heterogeneity and type-2 mediators (Chapter 4.1.2) deal with the vertical heterogeneity. Horizontal heterogeneity results from the service interoperability and transport-communication protocols problems. It happens when the third party source to be integrated-communicated is OGC compatible but not in Web Service. In contrast, vertical heterogeneity results from the data heterogeneity. It happens when the provided data are in different format from GML.

4.1.1. Mediator Type-1

This section discusses how Grid-enabled OGC web services can be integrated with non-Grid-enabled ones. The OGC web services can be cascaded in some way (see Chapter 3.3). For example, WFS can use other WFS as its feature source; and a WMS can act as a WMS client and combine contents from several WFSs. The benefit from such cascading is that the contents and capabilities of several services can be aggregated, thus facilitating the access. To integrate with non-Grid-enabled OGC services, we make the

cascaded OGC Web Services Grid-enabled, and use these Grid-enabled OGC Web Services as gateways to non-Grid-enabled ones. This is current common approach mentioned at [69].

Mediator type-1 is used for the cases when the services are OGC compatible and the only heterogeneity is in the transport protocol. Mediator type-1 is proposed to solve the heterogeneity between OGC Web services (using SOAP) and OGC services (using HTTP GET/POST).

Each OGC service will have its own type of mediator such as WMS-mediator or WFS-mediator. Mediators are not bidirectional. They are just like read-only. They enable Grid-enabled (GES) and non-grid enabled (Non-GES) services to communicate seamlessly. As Grid-enabled services are invoked through the SOAP over HTTP, we need to parse requests to these services, extract all the parameters, and then repackage them in a new request and send over HTTP (i.e., HTTP GET/POST, as defined in OGC specifications) to non-Grid-enabled services.

Figure 4 illustrates 2-way request-response mediation. In the figure, a Grid-enabled- WMS acts as the client to Grid-enabled WFS, and meanwhile, acts as the client to a non-Grid-enabled WMS and non-Grid-enabled WFS. When both services are Grid-enabled, they communicate via SOAP, otherwise, HTTP is used.

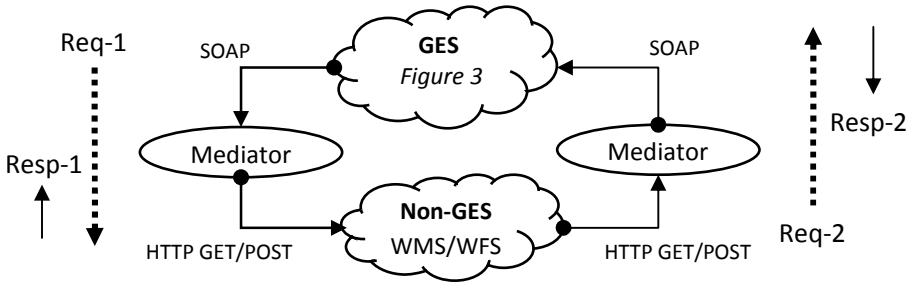


Figure 10: Mediator system, GES: Grid-enabled Service, Non-GES: Non-Grid-enabled

Let's take WMS as a case and explain the details about mappings and request/response re-writing in case of GES and Non-GES WMS inter-service communications.

4.1.1.1. Mapping and Query Re-writing:

As it is illustrated in Figure 10, there are two ways in inter-service communication, one is from GES to Non-GES and, the other is from Non-GES to GES. In this chapter we group the mediation services into two. These are request mediations and response mediations.

2-way Request mediation:

Non-GES GIS services use HTTP-GET/POST and GES GIS services use SOAP as communication protocols. Over these protocols they both use different request formats as displayed below. These are illustrated in Figure 7 as Req. arrows.

Sample HTTP-based *getMap* request :
http://gf8.ucs.indiana.edu:8092/wmsstream/WMSServlet?request=GetMap&width=400&height=400&layers=Google:Map,California:Faults&styles=&srs=EPSG:4326&format=image/jpeg&transparent=true&bgcolor=0xFFFFFF&bbox=-120,32,-110,40

Parameters are added at the end of service address as name value pairs. See the example below for WMS *getMap* request.

Sample Web Service based *getMap* request:

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

This request actually embedded in SOAP envelope. Before run-time client stubs for the web service invocation have to be created. Through client stubs semi-structured XML based service requests are wrapped in SOAP envelope and remote job is invoked with the similar code briefly listed below. “service_address” is actually Web Service address obtained from the properties file or from the catalog service depending on the application development.

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

We predefined service requests and instances are created according to these schemas. For example above *getMap* request is created according to its schema. In case of the HTTP based requests OGC defines the standards in its specifications. Mediators convert these two types of requests to each other to mediate these heterogeneous communication protocols. Both type of requests are based on key value pairs. So it is not hard to convert one to another.

2-way Response Mediation:

In case of *getMap* request WMS returns image MIME type such as image/jpeg as DataHandler object attached to SOAP message. OGC has different return types defined for the different types of requests. Our implementation for the return types will be improved and changed in accordance with OGC Web Services Specifications.

Response-1: From HTTP to SOAP attachments:

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

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

BufferedImage im = ImageIO.read(url);   //returned image or map.
ImageIO.write(im, "jpeg", mapImageCreated);
DataHandler dhSource = new DataHandler(new FileDataSource(mapImageCreated));

dhSource is returned as an attachment to the client.
```

HTTP IO-read returns *BufferedImage* object. After creating buffered image map, it is written to a file as jpeg or any other MIME type defined in the request. Map image created as a file is converted to a DataHandler object and casted to any Object and returned to the client as SOAP attachment.

Response-2: From SOAP attachments to byte streams:

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

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

WMSServicesSoapBindingStub binding = (WMSServicesSoapBindingStub)
                new WMSServicesServiceLocator().
                getWMSServices(new URL(service_address));

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

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

Then map byte array is written to HTTP writer in order to return the map image to the client.

4.1.2. Mediator Type-2

In the proposed Information Grid, mediator type-2 actually behaves as wrapper more than mediator. Throughout the document we use mediator and mediation-wrapper terms interchangeable. The task of wrapper is to translate a request from the Grid-enabled OGC compatible GIS system's (triangle in *Figure 3*) language to that of the information source and transform the results provided by the information source back to the Grid enabled OGC compatible GIS's language. Acting as a proxy of information source, the wrapper communicates with an information source in its native language and API, and communicates with the Grid-enabled OGC GIS system in a commonly agreed language (GML) and WFS Web Service API (*getCapabilities*, *GetFeature* and *DescribeFeatureType*). In this way, "wrapping" each information/data source into the translation software makes the particular sources manageable.

The proposed type-2 mediator-wrapper system uses GML for the encoding and the transport of geographic information and the WFS interfaces to perform communications with clients and data sources. Type-2 mediator service can be either for WMS or WFS but not both. We focus here on mediator-wrappers for WFS, but general architecture is applicable to WMS with its specific extensions. Currently we only integrate relational and file-based data formats.

In data integration literature, well-known data integration approaches can be characterized as either global-as-view (GAV) or local-as-view (LAV) [71], or global-local GLAV [72] or, more recently, as both-as-view BAV [73]. Whatever approach adopted, one needs to provide a framework for schema transformations, i.e., a language for schema mappings. The type-2 mediator-wrapper system is based on local-as-view approach.

The type-2 mediator is composed of three layers: (1) WFS service layer, (2) GIS mediator-wrapper and (3) data sources. Please see the *Figure 11* and *Figure 5* (dark grey parts).

(1) WFS is at the top of the stack providing interoperable OGC service API. The WFS layer is in charge of manipulating the data sources: it receives request from Grid-enabled OGC Web Services, executes them, and returns the results. It hides all the heterogeneity and complexity of the data and services. The request is issued by the client and posted to a mediator via HTTP (through mediator type-1) or SOAP-standard WFS Web Services client stubs. A request combines the operations defined in the WFS specification.

Among those operations are:

• *GetCapabilities*: a WFS must be able to describe its capabilities. Specifically, it must indicate which feature types it can service and what operations are supported on each feature type.

• *DescribeFeatureType*: WFS must be able, upon request, to describe the structure of any feature it can service.

• *GetFeature*: WFS must be able to service a request, and to retrieve feature instances. *GetFeature* requests allow the retrieval of feature instances and properties. The WFS reads and executes the request and returns an answer to the user as a GML or XML (such as capabilities document or HTML for *getFeatureInfo* requests) document depending on the request.

The WFS accesses various geospatial databases to retrieve and present the data to the users in a standard format. However, because of that the specification describes a HTTP GET/POST based services, we have extended them by implementing a Web Service version which allowed us to integrate several installations of this service and other Web Services to create Grids for particular purposes.

(2) The GIS mediator is composed of a mapping module, a decomposition/rewrite module, an execution/invocation module and a composition module. Since we do not deal with the semantic heterogeneities we create one-to-one wrappers and data sources are tightly coupled with their mediators. The GIS mediator is in charge of analyzing the query, including the various transformations that involve schema information.

• The request handler module gets the requests in SOAP over HTTP and extracts the key value pairs. The request handler module does it with the help of OGC Filter encoding Implementation specifications [95].

• The mapping module uses integrated schema information in order to express user queries in terms of local source schemas. Each mapping rule expresses a correspondence between global schema features and local ones. For the global schema definition, a Local As View (LAV) [71] approach is applied. This approach consists in defining the local sources as a set of views made on the global schema. This module turns the mediator-wrapper system into a WFS-enabled server.

• The *Source execution/invocation module* is in charge of collecting information from the WFS sources. It has knowledge of source's capabilities and cost information. It processes queries it

receives and sends them to the appropriate data/ information sources. Currently we just use database systems as data sources and use database connection manager developed by using JDBC.

• The composition module treats the final answer to delete duplicates and produce GML document, which is returned to the user. In other words, it creates an answer to be returned to the client. It has GML handler, and GML is created from the result set based on schema mapping file pre-defined based on the database's schema.

(3) Any legitimate type of data sources such as file stems, relational databases, XML databases, object databases, SRB (Storage Resource Brokers) etc. We have been working only with the relational databases so far, but in the future we will be testing our mediator propositions over other types of data sources. In current applications we have been implementing and using MySQL [81] relational databases and JDBC connections.

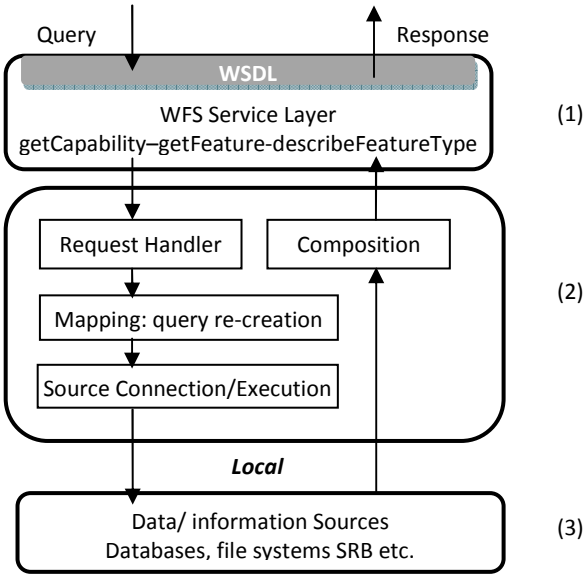


Figure 11: Mediator Type-2 for WFS (3-layer)

We use GML for the encoding and the transport of geographic information, and Web Feature Service (WFS) to access data sources. Each geographic data source is an abstraction of the real world according to a particular point of view. The semantic conflict between data may arise when the same concept is represented differently in local data sources. Those conflicts can be associated to concept names, their data type representations (a building can be a polygon in a data source and a point in another) and their properties (some attributes may not be present at some data sources, some differences may occur in their presentations). These issues will be touched in the future (see Chapter 10). We do not deal with the semantic heterogeneity here we take all the heterogeneity together and propose application based ad-hoc solution as one-to-one mediator system. To be more concrete, we can say that we develop source dependent mediator system for MySQL relational databases for Geographic Information

Systems. General architectural principles explained can be applied to any other resource types with the resource specific extensions.

We initially propose one-to-one mappings without taking the semantic heterogeneity issues into considerations. A typical WFS request-response cycle starts with the client request. The Client communicates with the service via the WSDL interface. A request may include several types of queries such as SELECT, UPDATE, DELETE etc. Since we have implemented the basic WFS currently we only support SELECT queries. After the request is received the WFS creates the SQL query from the request using the OGC Filter Encoding implementation [83] classes. Then the query is executed and the results are received. At this point the WFS uses the configuration files explained above to create the GML FeatureCollection object. Depending of the type of the WFS (Streaming or Non-Streaming) the results are returned to the user via appropriate channel.

Here we explain the request-response cycle with example documents for mediating and wrapping a relational database source. A sample query encoded according to OGC Filter Encoding Implementation is as follows:

```
<?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="ca_faults">
                <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>
```

This query simply means that the user request the name, segment, author and coordinates properties of the fault features which are inside the -150,30 -100,50 bounding box. The bounding box is defined as a rectangular region with minx, minY and maxX, MaxY coordinate.

After the WFS receives this request it parses the xml and extracts the required properties, query type and decodes the Filter to create a SQL query like following:

```
SELECT name, segment, author, coordinates FROM ca_faults WHERE
(LatStart>-150 and LonStart>30 and LatEnd<-100 and LonEnd>50);
```

Afterwards WFS uses the configuration file of this feature type to find the database that holds this feature and associated username and password information. Using such a configuration file allows us to integrate several databases through one WFS. Also the configuration file holds the location of the other required files such as the database mapping file. WFS uses the mapping file to create the SQL query. As soon as the results are returned the WFS uses the sample xml file and the mapping file to populate the GML features and create a GML feature collection.

Following is a segment from the feature collection generated for the above request:

```
<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/wfs/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>1.0</segment>
      <author>Rundle J. B.</author>
      <gml:lineStringProperty>
        <gml:LineString srsName="null">
          <gml:coordinates>-122.98,39.49 -122.91,39.41</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>
```

4.1.2.1. Related Works

There are some other research groups related to mediator type-2 for GIS services, you can have a look at VirGIS [79] developed in France and MIX (Mediation of Information Using XML) [80] at USDC (San Diego Super Computing Center).

MIX is collaboration between the UCSD Database Laboratory and the Data-intensive Computing Environments (DICE) group at SDSC. MIX group at SDSC focuses on the wrapper-mediator systems which employ XML as a means for information modeling, as well as interchange, across heterogeneous information sources. The wrapper associated with each source exports an XML view of the information at that source. The mediator is responsible for selecting, restructuring, and merging information from autonomous sources and for providing an integrated XML view of the information. MIX project propose XMAS as its standard query language and global schema definition. On the other hand, we used GML as data model and XML based semi-structured *getFeature* querying and OGC Filter Encoding standards to query and access the data.

In order to provide effective spatial database integration, VirGIS works on flexible GIS data integration solutions. VirGIS is using the techniques similar to multidatabases [105]. They create mediators mostly for partitioning the queries into sub-queries, re-writing the queries and sending to the appropriate worker WFSs and, upon receiving the responses reforming the final response and returning to the client. VirGIS is trying to build a mediator system providing an integrated view of the data supplied by all sources, and a geographical query language to access and manipulate integrated data.

4.1.2.1.1. Relation to Federated Database Systems (FDBS):

To the best of my knowledge, the best sample of FDBS for Geo-Sciences is Chronos [106, 107]. It is a collaborative work among GEON and SDSC. They integrate 7 independently developed geosciences databases.

Chronos use geographically distributed and heterogeneous data sources such as relational databases and flat files. Federate database can create global views that define data in a uniform way across the data sources. Applications then access the data through the global view using the standardized SQL schema. In our case, cascading XLink-WFS does this bu defined a flobal view in GML encoded featire collections and a standard service interfaces such as getFeature.

Here is the top level view of FDBS that Chronos is based on (Figure 12). Federated database uses wrappers in order to connect to the remote database and federate the data. Wrapping is a mechanism that the federated server uses to communicate with a data source. Each wrapper has a corresponding "driver" code internally in the federator database side.

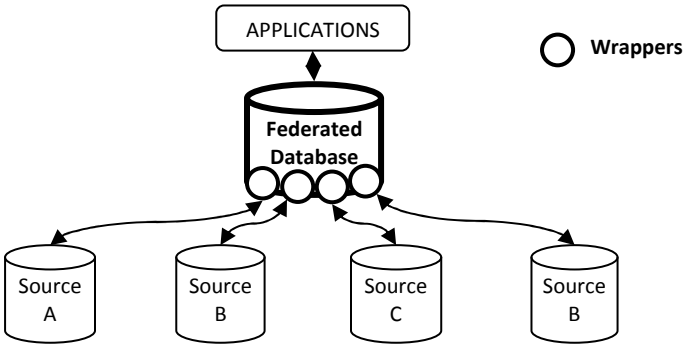


Figure 12: FDBS general architecture

When the application invokes the federated service, federator database identifies which sources request covers and specifies which wrappers to use.

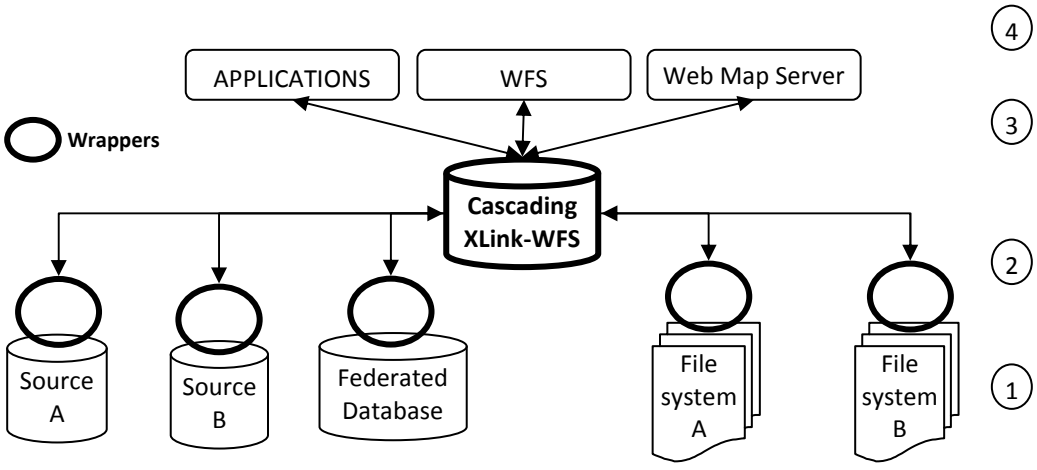


Figure 13: illustration of the usage of the proposed Mediator type-2 and leveraging the FDBS

FDBS federates only database systems (some applications of it federate file systems as well). On the other hand, our mediator system proposes federating any data source as long as it has source specific mediator as proposed in Figure 11.

Mediator type-2 in Figure 13 is the mediator type explained in Chapter 4.1.2 and Figure 11. Currently, for each data source we will have separate mediator including source specific wrappers etc. However, in the future we will be working on more general solutions to integrate heterogeneous data by taking semantic issues into considerations (see chapter 10).

Our approach is actually an implementation of the FDBS and enables leveraging of other FDBS through WFS-based mediator services (Figure 13). The mediators enable 4-level data/information abstractions as listed below. You can also see these classes of abstractions in Figure 13:

4- Map (1 an/or 2)
3- Layer
2- GML
1- Any Data (local data)

FDBS's main goal is providing a uniform and transparent interface to distributed and heterogeneous databases. The main server doing this is federator database. Federator in our proposed architecture is XLink-WFS. Cascading XLink-WFS is a standard defined by OGC. Here we introduce innovative mediation servers to be able to use XLink-WFS as federated databases. This will help us to solve heterogeneity and semantic problem in data and services. Since mediator services are WFS-based wrappers and provide standard WFS service interfaces we can chain them to cascading XLink-WFS. See chapter 3.3.2.2 for the architecture of XLink-WFS as an extension to Basic-WFS. That chapter also illustrates re-directing (or cascading) the mediator services to enable XLink-WFS to behave as federated data providers.

Internally our system, we use standard service interfaces and semi-structured human and machine readable request formats such as getMap, getFeature etc. since we use common data model provided by mediators and WFSs, we solve the data semantic problems internally. Unlike the federator database in FDBS which federates databases, "cascading XLink-WFS" (see Figure 13) federate WFSs. These WFSs are actually mediator type-2 explained in Figure 11. Here, we don't touch the general issues regarding semantic issues such as data-schema matching and mapping to convert local data into global data model (GML in our case). This might be our future work but *currently*, we use source-specific mediators providing common service interfaces and data model and, each one uses source specific ad-hoc solutions to the heterogeneity problems.

In addition, because of the characteristics of the GIS applications we don't need the transaction management capabilities in the proposed system as FDBS deal with. Our architecture only supports querying which is called "pull data approach" and on-demand data access by going through the data abstraction steps. The path from the comprehensible data/information to the raw data stored in data sources are composed of OGC GIS services WMS and WFS.

5. Interactive Decision Making Tools for GIS Information Grids

In order to utilize and interact with the coupling framework enabling access to computational resources and data resources in a seamless, ordered and synchronized manner, we develop innovative interactive smart decision making tools. Tools enable feeding the Geo-Science applications (projects) with the data and associate the applications' input and output data at the layer (view)-level in a 3-layer structured display. The attributes of the data/information which are building these comprehensible representations (layers) should be interactively accessed and queried. Resources are organized into projects and they are compliant with some resource schemas. Client portal provides both map-based and project based user interfaces. The former are suitable for geospatial resources with location information while the later caters for all kinds of resources with or without location information. The two interfaces co-exist and are synchronized. That is, resources selected using the map-based interface will also be highlighted in the project based interface, and vice versa. In addition, when it is deployed on a portal as a portlet, varying levels of access and authorization can be set.

Proposed interactive system hides the underlying complexities of the coupling framework. End-users want to get work done and they do not want to deal with the complexity of building workflows that expose details of the underlying Grid services or other middleware. They should be able to express their problem by composing application specific components in an easy to use form.

The rest of this chapter is organized as follows. Chapter 5.1 explains the basic generic mapping tools to display and query the maps interactively. In this case interaction is done directly with the Aggregator WMS which is an entry point of OGC compatible geospatial data Grid. Chapter 5.2 explains integrations of the AJAX approach to the interactive decision making interface. This chapter covers coupling architecture of the Google Maps with WMS and WFS and, creating generic mediator service to integrate Google maps to the proposed framework at the layer level. Chapter 5.3 presents the animation tools to interpret the time series geospatial data. We propose two types of techniques to animate time-series data. One is map movies created by using Java Media Framework (JMF) [97] libraries and the other one is GIF animations.

5.1. Interactive Decision Making Tools over Map Images

Initially, we have developed a portal and browser based display client for our Web Service based standard GIS visualization system for testing and the demonstration purposes and, called it basic WMS client. A sample WMS Client is shown in Figure 14 (please also see the APPENDIX 10 and 11). It has several capabilities related to visualization of the geo-spatial data such as zooming to different level of resolutions, measuring the distance between two points on the map for different coordinate reference systems, querying the layer data for the attributes of the features on the map by making *getFeatureInfo* requests, drag-drop the map to display different bounding boxes. Interface also enables users to request maps for the area of interest by selecting predefined options clicking the drop-down lists. The user interface also allows the user to change the map sizes from the drop-down lists (predefined) or enable them to give specific dimensions.

WMS Client is not just for displaying and manipulating maps anymore. Rather, it provides some additional capabilities as a GIS portal for the Geo-Science applications. It combines different kinds of layers from different sources, enables superimposing data plotting layers crated from the output of the Geoscience applications, interacting and invoking some geophysics applications through the intelligent workflow management tools and software such as HPSearch Engine [57] from CGL. In some contexts, using the term WMS Client is better such as interacting with WMS and displaying the GIS map after receiving the response from the WMS. In all other contexts, using the term GIS portal makes much more sense such as integrating and orchestrating all the components involving in the GIS systems, integrating third party geo-applications through the workflow engines, interacting and directing the workflow engine, providing signing in or out capabilities, session handling etc.

GIS portal enables the end-users to request layers from two different kinds of layer providers, WMSs and Sci-Plotting servers. Plotting is done over the outputs of the geophysics applications. If these layers, even if they are coming from different servers, are in the same bbox then they can be overlaid together. Please see the Figure 14 for the sample output showing two layers overlaid. In this figure, one layer is coming from NASA [56] OneEarth WMS through our implementation of proxy-master WMS and the other is coming from the Sci-Plotting Server from CGL (Community Grids Lab.). By overlaying different layers from different kinds of servers, decision makers and end users will be obtaining more informatory and explanatory maps. In that sense, WMS Clients can also be considered as GIS portals that enable user and data-map interactions giving control capabilities to users via some mapping tools which are mentioned earlier.

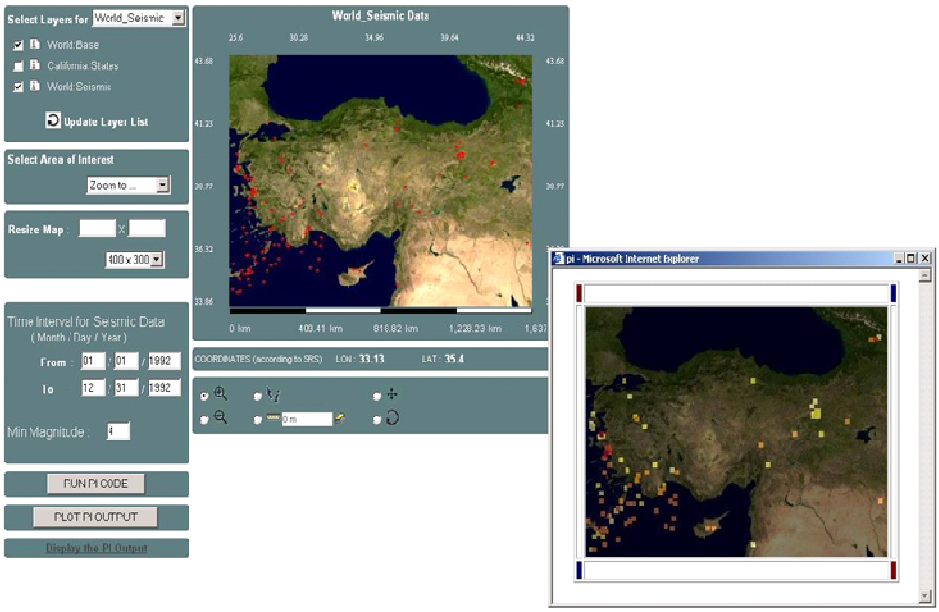


Figure 14: Overlaid layer created by PI module in WMS Client after running PI geophysics application over the map.

GIS portal interconnects the GIS visualization services (WMS and WFS) with the GIS scientific applications and Sci-Plotting services in accordance with the specific purposes of the geo-applications. GIS portal achieves this task through the user oriented orchestration of the layer composition and mapping tools. We created generic and application independent WMS Client. It can support more than one geophysics application at the same time. Each geophysics application is bound to a set of layers. These bindings are defined in structured XML properties file. Users navigate over the applications by selecting set of layers from the dropdown list. A set of layers in the dropdown list is created according to communicated WMS. Binding properties are updated based on the set of supported layers of the that WMS.

GIS portal basically serves for the GIS end users and decision makers. It has several capabilities for the decision makers to access and interpret geo-data seamlessly. GIS portal is built up with the various technologies; among these are Java, Java Servlet and Java Server Pages (JSP), Java-Script, CSS and Web Services.

5.2. Integrating AJAX Technologies with GIS Visualization Web Services

- Coupling Google Maps with WMS and WFS

This section explains the AJAX integration framework which is designed for browser based web applications using Web Services. Proposed framework enables users to utilize AJAX and Web Services advantages together. For the detailed information about the architectural implementation and advantages of the system please see Figure 15 for more details.

As the Web platform continues to mature, we see an increasing number of amazing technologies that take the GIS visualization applications to new levels of power and usability. By integrating new powerful technologies into GIS systems, we get higher performance results with additional functionalities. The most recent development capturing the attention of the browser based application developers is AJAX (Asynchronous JavaScript and XML). In this chapter we present a generic and performance efficient framework for integrating AJAX models into the browser based GIS visualization Web Services systems.

AJAX [63] is an important web development model for the browser based web applications. It uses several technologies which come together and incorporate to create a powerful new model. Technologies forming AJAX model such as XML JavaScript, HTTP and XHTML are widely-used and well-known. High performance Google mapping uses this new powerful browser based application model. Web Services are self-contained, self-describing, and modular. Unlike earlier, more tightly coupled distributed object approaches such as Common Objects Request Brokers (CORBA), Web Service systems support an XML message-centric approach, allowing us to build loosely coupled, highly distributed systems that span organizations. Web Services also generalize many of the desirable characteristics of GIS systems, such as standards for providing general purpose specifications for publishing, locating, and invoking services across the Web. Web Services also use widely-used and well-known technologies such

as XML and HTTP as AJAX does. Since AJAX and Web Services are XML based structures they are able to leverage each other's strength.

There are some GIS projects adapting only Web Services or only AJAX approaches into their GIS systems but not both. That is because of the idea that they are totally different technologies using different communication protocol and it is impossible to use them in the same framework. To give examples, ESRI, Cubewerx, Demis and Intergraph are adapting Web Servcie technologies and, Googe Maps and KA-Map [102] are adapting AJAX to their GIS systems.

ECMAScript [103, 104] for XML E4X is the only related work involving AJAX and Web Services together. E4X is a simple extension to JavaScript that makes XML scripting very simple. It is actually the official name for JavaScript. The European Computer Manufacturers Association (ECMA) is the standards body where JavaScript is standardized E4X uses all other incorporated AJAX technologies without extension.

Via the E4X, you don't have to use XML APIs such as DOM or SAX; XML documents become one of the native types that JavaScript understands. You can update XML documents from the JavaScript very easily. These properties of E4X enable creating calls to Web Services from the browser, but the only browser that supports E4X so far is the developer release of Mozilla 1.8. E4X helps to interact with Web Services but again it is just an extended version of JavaScript. Some issues regarding how to put request in SOAP messages and how to manipulate returned SOAP messages are still complicated. If you use E4X for a web applications based on AJAX model, you cannot use the application on every browser. This is another drawback of the system.

In our approach, you don't have to extend any technology involved in the AJAX model. We use all the technologies in AJAX with their original forms. This gives the developers and users the ability to integrate and customize their applications easily.

The remaining of this chapter is organized as follows. Chapter 5.2.1 presents the main generic framework for solving heterogeneous transport protocols problem. This is called integration framework for AJAX and Web Services (see (C) in Figure 15). Chapter 5.2.2 presents the usage scenarios of the generic framework. As usage scenarios we give Google Map Server's integration to OGC's WMS (Chapter 5.2.2.1) and WFS servers (Chapter 5.2.2.2).

5.2.1. General Framework to Solve Transport Heterogeneity

- XMLHTTPRequest+HTTP vs. SOAP+HTTP

AJAX uses HTTP GET/POST requests (through JavaScript's XMLHttpRequest) for the message transfers (see (A) in Figure 15). Web Services use Simple Object Access Protocol (SOAP) as a communications protocol (see (B) in Figure 15) In order to be able to integrate these two different message protocols, we must convert the message formats into a common format or make them interoperable. Since there is no ready to use common protocol to handle messages communications between AJAX and Web Services, we implemented a simple message conversion technique (see (C) in Figure 15). This essentially works by having the XMLHttpRequest communicate with a Servlet, which in turn acts as a client to a remote Web service. This allows us to easily convert between SOAP invocations and HTTP POSTS. It also has the

benefit of avoiding JavaScript sandbox limitations: normally the XMLHttpRequest object in the browser can only interact with its originating Web server.

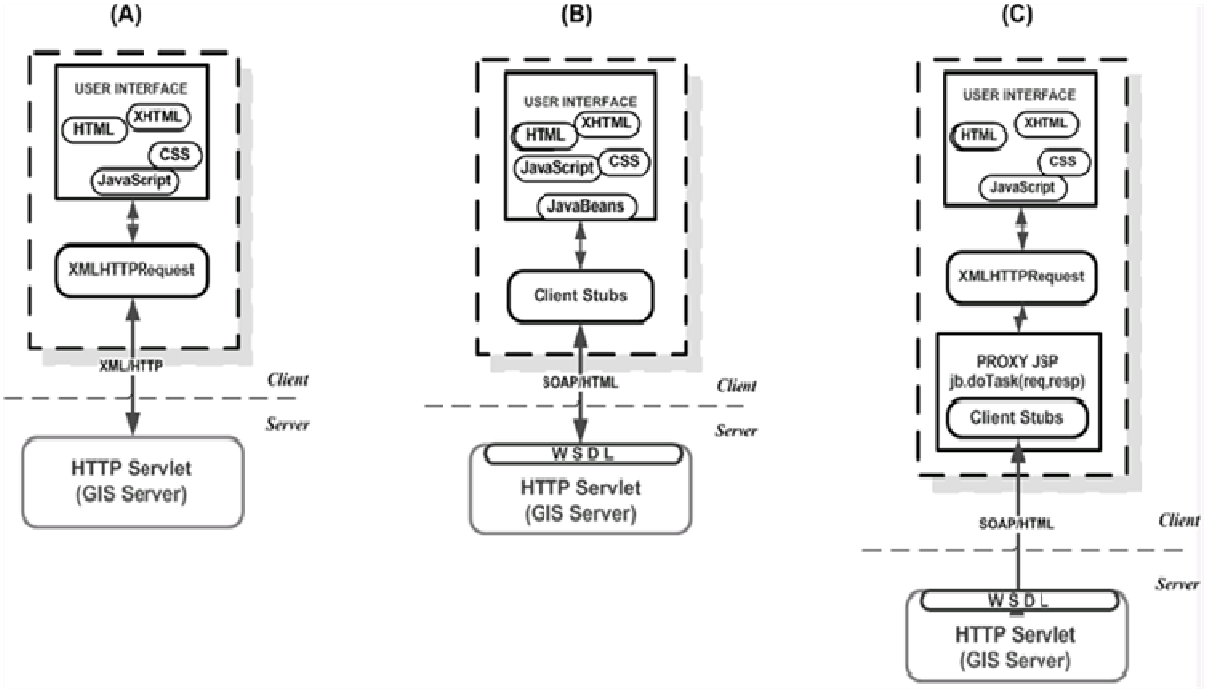


Figure 15: (A) Pure AJAX Approach, (B) Web Services Approach, and (C) Hybrid (AJAX + Web Services) Approach

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

In more detail:

Client first creates an XMLHttpRequest object to make a remote scripting call.

- var http = new XMLHttpRequest();

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

- var url = “proxy.jsp”;

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

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

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

- jb.doTask(request,response)

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

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

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

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

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

- http.onreadystatechange = handleHttpResponse

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

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

5.2.2. Usage Scenarios

Integration is basically coupling AJAX actions with the Web Services invocations, and synchronizing the actions and returned objects from the point of end users. The usage scenarios explained below use the generic integration architecture illustrated in Figure 15. In the usage scenarios there will be minor difference in the form of extensions. Differences come from the service specific requests created according to the service provider’s service API (published as WSDL), or handling returned data to display on the screen. But these are all implementation differences.

5.2.2.1. Google Maps and WFS Integration

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

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

Even though Google Mapping API supports just two of them, Points and LineStrings, the other geometry elements can also be converted to these two types with minor updates. Having extracted and obtained geometry elements, these elements are plotted over the Google Map by using “GPoints” and “GPolylines” objects and the “mapOverlay” function of the Google Map API [100].

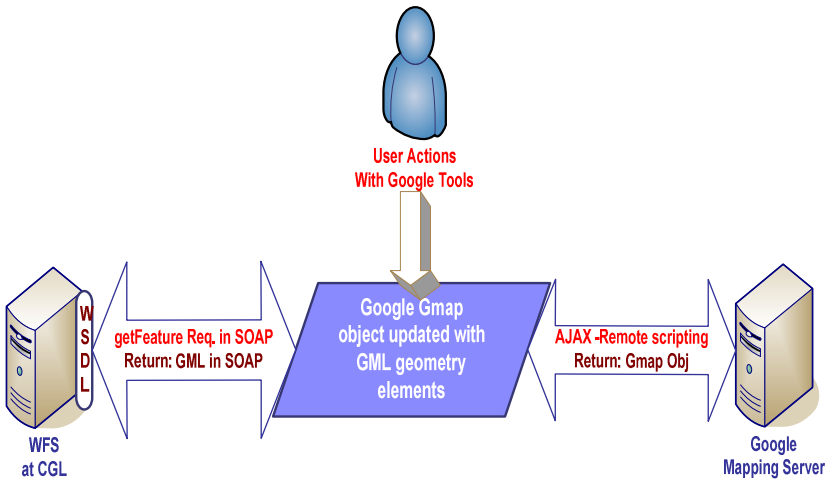


Figure 16 **:** Integration of Google Maps with OGC WFS by using architecture defined in **Figure 15**.

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

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

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

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

5.2.2.2. Google Maps and WMS Integration

There are two different path working in parallel by the given user parameters created by the client actions. Actions are interpreted by the browser through the Google Mapping tools. JavaScript captures these actions by ActionListeners and Google Binding APIs and gives to Layer-2 object (see the Figure 17).

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

Interconnection for creating Layer-2 is done in accordance with the proposed architecture defined above in Figure 15. For Layer-1, a classic Google mapping application is used through the AJAX web application module and XMLHttpRequest protocol. Google handles creating the map by using XMLHttpRequest and given remote JavaScript file in the browser.

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

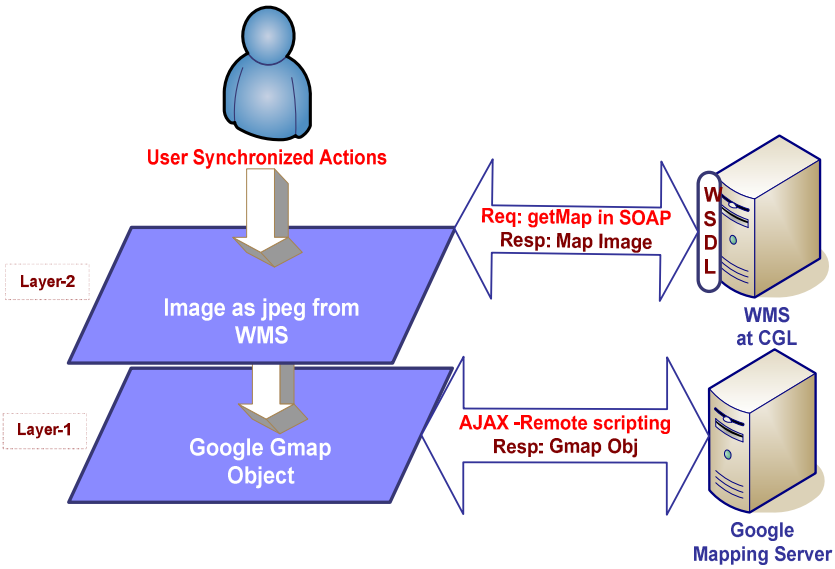


Figure 17: Integration of Google Maps with OGC WMS by using architecture defined in **Figure 15**.

In addition to all of the approach illustrated here, we utilize from the Goole maps in OGC compatible GIS through developing intermediary Google Mapping Server (See APPENDIX 10 and 11for sample outputs). Web Map Service returns maps in the form of images such as JPEG, GIF and PNG. Web Map Service clients get the maps in image formats and overlays them. Ordinary Web Map Service clients cannot use

maps coming from Google Map Servers. To solve this problem and use high performance Google maps in our Web Map Service applications and overlay different map layers coming from the common Web Map Service with the Google Maps, we created an intermediary Google Mapping Server. It takes Web Map Service compatible requests from the Web Map Service clients, converts these requests into a new form that real Google Map Server can understand. In contrast to Open Geospatial Consortium compatible *getMap* requests, Google Map server uses requests with different parameters such as zoom level, tile numbers and tile width.

5.2.3. Evaluation of Integration Approach

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

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

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

5.3. Streaming Map Movies and Animation Tools

- Map movies and animations

Maps are traditional means of presentation and tools for analysis of Geographic and spatial information. They are easy to create and there are lots of tools to help creating them online. Compared to static maps, animated maps have always been difficult to make, distribute, and access. Even with today's powerful software and computers, a developer will want to be reasonably confident that their efforts will pay off with a map that is both attractive and informative. In other words, developers should make

sure if the animation lend something to the representation that would be difficult or impossible to convey in static form.

Map animation makes sense for the time series data. In the literature it is also called spatio-temporal data. This type of data is about phenomena that change with time. Exploration of such data requires highly interactive and dynamic maps [62]. We implement such kind of mapping in streaming map movies and GIF animation tools designed for various types of spatially referenced time-series data: occurrences of events, movement of objects in space, and statistical data referring to parts of territory. As a theoretical background, we develop a classification of analytical tasks depending on what aspect of a spatial phenomenon varies with the time (existence, spatial location, size and shape, or thematic properties) and what kind of view is required, with respect to time (instant, interval, or overall). On the basis of this classification we select appropriate presentation techniques and devise interactive map manipulation tools supporting various kinds of tasks.

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

5.3.1. Related Works

Traditional GIS environments frequently have not incorporated tools for building time series animations. As a result, other tools like Quicktime and Flash are often used to create the animations from the GIS imagery after having frames created as static map images locally. For example, using ArcGIS 3D Analyst, each frame of a 24 time step sequence can be stored as a JPEG image. Using the Layout editor, metadata elements like notations of each individual time step, a map legend and titles for the animation sequence can be added to each JPEG image. All 24 frames are then be assembled into a Quicktime or Flash movie for viewing on the Web across the Internet [88]. More complex “rich media” presentations that include audio narrations can be developed using streaming media technologies like the Synchronized Multimedia Integration Language (SMIL) [87].

One of the leading GIS initiatives, ESRI also has been working on developing efficient architecture for creating on-the-fly map animations. Their projects about map animation and movies are called TimeMap. They take a simple shape-based approach to map animation of cultural data, based on tweening of vector shapes recorded as "snapshots" in a TimeMap-compatible GIS format to generate intermediate forms for each frame of the animation. This results in smoother animation, although it does not necessarily respect the actual system mapped [86].

Another work is done by Cartographer Mark Harrower’s group at University of Wisconsin-Madison. They build a series of animated maps that incorporate innovative temporal controls called *visual benchmarks*, a technique that allow users to display multiple time periods simultaneously in animated maps. Visual benchmarks were designed to help map readers cope with long, complex animations by shifting some of the cognitive burden of understanding patterns of geographic change from short-term memory to the display itself [91, 92].. They basically use ESRI tool to create static maps and animate them using Quicktime tools.

We see that ESRI and any other GIS projects (educational or commercial) related to map animations mostly go through the same processes. That is, creating frames as static map images by using any map tools and, assembling them with Quicktime, Flash or GIFF animations. They don't embed the map animation framework into the information grid as a whole architecture. Furthermore, they are not taking interoperability and heterogeneity issues together with the map animations. In our proposed architectural framework, we merge GIS tools with movie streaming techniques by using Java Media Framework (JMF) and tried to create a complete system for map animations and movie streams.

5.3.2. Introduction to Proposed Time Series Data Animations

The history of creating map animations for time series data fits into three (overlapping) eras: (1) manual, (2) computer-assisted, and (3) computer-based production. With manual production, each frame of the animation (e.g., map) is drawn by hand. In computer-assisted production (from the late 1960s through the mid-1980s), computers are typically used to create the individual frames of the animation, but the process of assembling and filming these map frames is done using traditional (i.e., mechanical) cinematography. By contrast, in computer-based production the entire process, from creation to distribution, occurs digitally. Completely digital animation first appeared in the mid-1980s as both the software and hardware permitted [93]. In summary, our approach fits into the computer-based map animations category.

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

The map animation and movie streaming architecture we have been developing is based on the proposed Information Grid architecture explained so far. So WMS and WFS are our main components of the map movies system. Because the whole system's characteristics, we propose on-demand maps, map-movies and animation architecture.

Standard Web Map Servers produce static images, but a lot of geographic data are time dependent. In order to understand geographic phenomena and characteristics of temporal data it is necessary to examine how these patterns change over time for these types of data. We are therefore investigating the problems of creating streaming video map servers based upon appropriate standard collaboration technologies [61].

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

Regarding the streaming map movies unlike the other research colleagues we propose whole system prom the beginning (definition of time-series data and metadata descriptions for movies) to end (playing the static map images as movie streams for the end-users). In order for that we use some other projects developed in CGL such as WFS, NaradaBrokering and GlobalMMCS [59, 60] and, JMF [97] for creating streaming map movies from the chains of static map images. WFS is used for accessing and querying archived vector data, NaradaBrokering is used for message based streaming data transfers and Global MMCS is used basically for archiving the movie streams and playing them in collaboration sessions.

We categorize our work regarding time series data animations into two. These are Streaming map movies and browser-based map animations. Streaming Map movies are explained in Chapter 5.3.3 and browser-based map animation tools are explained in Chapter 5.3.4.

5.3.3. Architecture of the Streaming Map Movies for GIS Information Grid

In GIS area there is a huge demand of using multimedia technology over internet scale to support groups collaborative work in which members are distributed at different geological locations, such as distance learning, virtual classroom, video conferencing etc. Because of the characteristics of the geospatial data, one organization or entity cannot have all the geo-data and geo-services available locally. So they need some other data and services from other entities. They require sharing of data. Sometimes organizations need to make a decision about the geographic data by looking at the map or animated movies. They can achieve this by getting together physically or using collaborative GIS services. This growing need and demand for large scale interactive and collaborative GIS systems present several interesting research challenges in computer science such as: design of good collaboration framework with advantages of high scalability, extensibility, reliability and security, design of synchronization and composition mechanisms to synchronize multiple video/audio streams. In order to achieve these aims, partially or fully, we have been using our Lab's GlobalMMCS collaboration services for the GIS collaborative visualization applications. GlobalMMCS is an integrated video conferencing solution which enables heterogeneous clients to join the same real-time multimedia sessions. It provides a flexible architecture to support even more standards and applications. We also inherit many additional features such as replay and collaborative annotation and whiteboard systems.

All the services participating in the proposed architecture are Web Services. System utilizes all the advantages of the Web Services such as easy integration, using widely acceptable technologies, cross-platform, cross-language etc. However, because of the characteristics of the geographic data and Web Services message exchange protocol, SOAP, there are performance limitations if we use the current SOAP approaches to integrate Geographic Information Systems (GIS) applications with Web Service based collaboration systems, especially for the multimedia GIS applications such as displaying streaming map movies. To overcome this type of performance issues we created streaming version of Open Geospatial Consortiums (OGC)'s Web Map Service (WMS) for creating both static maps and map movie streams composed of more than one static map. Streaming data transfer is enabled by using Community Grids Lab's NaradaBrokering messaging middleware. The NaradaBrokering messaging substrate enables

scalable, fault-tolerant, distributed interactions between entities, and is based on the publish/subscribe paradigm. The NaradaBrokering substrate provides support for transport protocols such as TCP, Parallel TCP, UDP, Multicast, HTTP and SSL; it also facilitates communications across NAT and firewall/proxy boundaries. For the overall architecture and sample request for map movies, please see the APPENDIX 7 and 9 respectively.

WMS is not able to create movie for all of its supported layers listed in its capabilities file. If WMS supports movie functionality for a layer it adds some attributes under this layer definition in capabilities file. Before making the request, the client first makes a "get capabilities" request and after getting information about the WMS, it makes requests according to capabilities of the WMS. If a client makes a request to get a movie for a specific layer, to succeed, this layer should have a time dimension defined under this layer element in the capabilities file. Clients should make the "get map" standard request to WMS to get the movie for a specific layer. WMS does not provide any other request types for the movie creation functionalities. Clients set the "format" variable to string "movie/<movietype>" and "time" variables to a value in an appropriate format to make a request to get movie from the WMS.

Streaming Map movies are created in a few separate but connected orderly stages. These are listed and explained below. We also illustrated the whole architecture in detail in APPENDIX 9.

According to our proposed architecture of streaming map movies, there are a couple of stages including set of WMSs and WFSs to create streaming map movies and display them in our proposed architecture. These are: (1) Invocation Stage, (2) Data Retrieval Stage, (3) Frames (Map Images) Creation Stage, (4) Stream Creation Stage and (5) Stream Publishing Stage. For the general picture of the stages and their communications to create map movies see APPENDIX 9.

5.3.3.1. Invocation Stage:

This is the first step. The system for map movie streaming is triggered by the end-user remotely through interactive decision making tools (Chapter 5). You can also see Figure 19 for the sample user interface for map movies' creation.

So far, we kept saying that movies can be created only for the time-series data and layers having time-series data as sub-layers. OGC call this kind of data as multi-dimensional data objects. Multi-dimensional data objects are described with "Dimension" and "Element" terms and XML tag elements. If a layer is available for creating map movies <Dimension> tag element is paced under that layer as sub element according to the pre-defined OGC's capabilities schema for WMS [101]. This is a way to notify the WMS clients of map movies creation availability and required parameters to invoke for the appropriate movies' creation. We explain how to use "Dimension" tag element in order to create map movies below.

Time is one of the dimension name defined in WMS capabilities file. If time dimension is defined for a layer then clients can make requests for a specific time intervals and periodicity value to create movies. Time dimension is defined in WMS capabilities metadata for the time-series data. Time dimension basically covers the required information and parameters to be able to create successive frames for the map movies. The most important information provided by the time dimension is the data's available date ranges and periodicity (or duration).

You can have a look at the example Dimension tag element placed in WMS's capabilities file below. If WMS provides a time series data in a layer listed in its capabilities file, it puts a time dimension element as displayed below under the specific layer. For this sample layer WMS provides data from 01/01/1999 to 08/22/2000 in daily values. Here P1D means "per one day". There are also other standard definitions for the data durations.

<Dimension name="time" units="ISO8601" default="2000-08-22">1999-01-01/2000-08-22/P1D</Dimension>

For the different time intervals there might be different periodicities. At that time WMS adds additional lines to time dimension element as displayed below as sample.

<Dimension name="time" units="ISO8601" default="2000-08-22">
1999-01-01/2000-08-22/P1D
1990-01-01/1998-08-22/P1Y
</Dimension>

P1D and P1Y define the duration (The length of time nothing changes at the display). The term duration is sometimes called frequency. For P1D, duration is one day and, for P1Y duration is one year. There are some other alternative definitions. For more information see WMS standard specifications [53].

Successive dates separated with slash '/' is called display rate. An example is shown in above Dimension tag element "1999-01-01/2000-08-22". Display rate is defined as the time intervals at which some display change is initiated. In other words static map images as movie frames will be created between these dates depending on the frequency (or duration parameter) of the frames.

After having examined the capabilities file of the interacted WMS, client creates a request to invoke the service for creating map movies. Invocations are done to WMS. The request given below is for a map movie which is going to be created in *mpeg* format. Key elements are Time and Format as highlighted.

In case of that WMS provides its functionality as Servlet (HTTP GET/POST):
http:\\servletname\VERSION=x.y.z WIDTH=600&
REQUEST=map HEIGHT=300&LAYERS=ozone ***TIME=1987-01-01/1992-07-31/P1Y****&*
SRS=EPSG:4326 ELEVATION=1000&BBOX=-180,-90,180,90 ***FORMAT=video/mpeg***

In case of that WMS provides its functionality as Web Service:

```
<?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>
        ……..
        ……
</GetMap>
```

5.3.3.2. Data Retrieval Stage

After having parsed the *getMap* request and created linked list of *getFeature* requests, WMS invoke the WFS's *getFeature* Web Service interface to get the feature data in GML format. Sample *GetFeature* request is shown below. This stage is summarized as: (1) creating chain of *getFeature* requests depending on the parameter "Time" in *getMap* request (2) creating chain of time series feature data in GML format. The number of requests to WFS to get feature data for the specific time intervals and number of frames for the movie change according to the parameter values given in time parameter in *getMap* request. For example the sample request has this time tag:

<Time>01-01-1987/12-31-1992/P1Y</Time>

Since the data duration is defined as P1Y, we cut the data range into pieces in years. So, for our sample time tag defined in getMap request we will have 5 getFeature requests to WFS. Each one of these requests' filter parameter "date" will be set as below. Other parameters such as bbox etc will have to be the same.

Req1: from 1987 to 1988, *Req2: from 1988 to 1989,* *Req3: from 1989 to 1990,*

Req4: from 1990 to 1991, *Req5: from 1991 to 1992*

The original WFS specification is based on HTTP Get/Post methods, but this type of service has 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. We have developed a Web Service version of WFS and are testing in several scenarios where scientific data analysis tools such as Pattern Informatics [21] require fast access to large amount of data.

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.

Our streaming WFS and WMS use standard SOAP messages for receiving queries from the clients; however, the query results are published (streamed) to a NaradaBrokering topic as they become available. Our initial implementation uses MySQL database for keeping geographic feature data, and we employ a capability in MySQL that streams the results row by row, allowing us to receive individual results and publish them to the messaging substrate instead of waiting for whole result set to be returned. The initial performance results show that (especially for smaller data sets) streaming removes a lot of overhead introduced by object initializations.

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

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

```
<?xml version="1.0" encoding="iso-8859-1"?>
      <wfs:GetFeature outputFormat="GML2" xmlns:gml="http://www.opengis.net/gml"
xmlns:wfs="http://www.opengis.net/wfs" xmlns:ogc="http://www.opengis.net/ogc">
     <wfs:Query typeName="scedc">
      <wfs:PropertyName>YEAR</wfs:PropertyName>
      <wfs:PropertyName>MONTH</wfs:PropertyName>
      <wfs:PropertyName>DAY</wfs:PropertyName>
      <wfs:PropertyName>HOUR</wfs:PropertyName>
      <wfs:PropertyName>MINUTE</wfs:PropertyName>
      <wfs:PropertyName>SECOND</wfs:PropertyName>
      <wfs:PropertyName>MAGNITUDE</wfs:PropertyName>
      <wfs:PropertyName>LATITUDE</wfs:PropertyName>
      <wfs:PropertyName>LONGITUDE</wfs:PropertyName>
      <wfs:PropertyName>QUALITY</wfs:PropertyName>
      <wfs:PropertyName>DEPTH</wfs:PropertyName>
      <ogc:Filter>
       <ogc:BBOX>
        <ogc:PropertyName>coordinates</ogc:PropertyName>
        <gml:Box>
         <gml:coordinates>-117,32 -114,37</gml:coordinates>
        </gml:Box>
       </ogc:BBOX>
      </ogc:Filter>
     </wfs:Query>
     <wfs:Query typeName="scedc">
           <ogc:Filter>
             <ogc:PropertyIsBetween>
```

```
            <ogc:Literal>MAGNITUDE</ogc:Literal>
            <ogc:LowerBoundary>
                  <ogc:Literal>2</ogc:Literal>
            </ogc:LowerBoundary>
            <ogc:UpperBoundary>
                  <ogc:Literal>6</ogc:Literal>
            </ogc:UpperBoundary>
          </ogc:PropertyIsBetween>
        </ogc:Filter>
  </wfs:Query>
  <wfs:Query typeName="scedc">
    <ogc:Filter>
      <ogc:PropertyIsBetween>
        <ogc:Literal>DATE</ogc:Literal>
        <ogc:LowerBoundary>
          <ogc:Literal>575448425980</ogc:Literal>
        </ogc:LowerBoundary>
        <ogc:UpperBoundary>
          <ogc:Literal>575534835293</ogc:Literal>
        </ogc:UpperBoundary>
      </ogc:PropertyIsBetween>
    </ogc:Filter>
  </wfs:Query>
</wfs:GetFeature>
```

5.3.3.3. Frames (Map Images) Creation Stage

There are basically two main ways to create an image from the simple features. First one is to create Scalable Vector Graphics (SVG) file and convert it into any image format. Second is using Java Graphics2D libraries. Whatever way WMS uses, it needs to use the feature collections obtained at the previous stage to create map images.

When the first way is used, WMS uses its internally defined XSL file to convert standard GML files into SVG by using XSLT machine. We also developed standard XSL (APPENDIX 12) file to convert XML coded GML feature collections into SVG files. The second way to create map images from GML data is explained in the sample code fragment as below.

WMS, in this stage, create chains of the graphics object from the chains of GML objects coming from the previous stage as outputs. These image subjects then converted into any image format. In this stage we obtain chains of static map images created from the time-series vector data from WFS overlaid on raster map images. Below you see the sample code fragment showing how to do that. Here, raster data is coming from HTTP Servlet based WMS server (defined in URL) and the other data represented as features are coming from our implementation of Web Service based WFS.

```
URL url = new URL(
                Wmsaddress+”?request=GetMap&width=" +
                width + "&height=" + heigth +
                    "&layers="+layername+ ”&styles=&srs=EPSG:4326&format=”+format+”&bbox=" +
                bbox);
BufferedImage im = ImageIO.read(url);
Graphics2D g = im.createGraphics();
…
 if(istherePoint)
   String[] points   = getPointsFromFeatureData();
if(isthereLineString)
   String [] LineStrings = getLineStringFromFeatureData();
if(isthereLineRing)
   String [] LineRings = getLineRingFromFeatureData();
if(istherePolygon)
   String [] polygons = getPolygonsFromFeatureData();
…
…
if(polygons!=NULL){
for(int i=0; i<polygons. length; i++){
    int [][] xypoints = wm.getXYpoints(polygons[i]);
    g.setColor(Color.darkGray);
    g.drawPolygon(xypoints[0], xypoints[1], xypoints[0].length);
}
}
if(LineRings!=NULL){
for(int i=0; i< LineStrings. length; i++){
    int [][] xypoints = wm.getLinesInStr(LineStrings[i]);
    g.setColor(Color.darkGray);
    g.drawPolyline(xypoints[0], xypoints[1], xypoints[0].length);
 }
}
…
g.dispose();
```

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

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

We have been using both ways in different places but in this code we illustrate using JAVA graphics library. We saw that images drawn with graphics2d are of higher quality then the images drawn by SVG conversion.

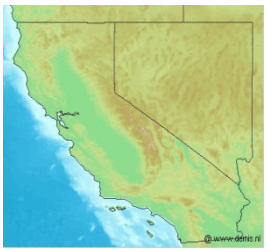


Figure 18: ***Sample frame – static map image***

5.3.3.4. Map Streams' Creation Stage

- "Input stage", "processing stage" and "output stage" in APPENDIX 9.

This stage is mostly related to Java Media Frameworks (JMF). Through JMF [97] libraries, WMS creates available formats from the chains of the static map images for feeding JMF processor. JMF processor publishes the streams map images to Real-time Transport Protocol (RTP) [96] sessions after some required parameter settings.

Map movies are created from array of images by using JMF libraries. Movie streams are sent to RTP sessions. If anyone wants to see the movies needs to have JMF client or GlobalMMCS client or AccessGrid client.

The Java Media Framework (JMF) is a recent API for Java dealing with real-time multimedia presentation and effects processing. JMF handles time-based media, media which changes with respect to time. Examples of this are video from a television source, audio from a raw-audio format file and animations. JMF is built around component architecture. The components are organized into a number of main categories: Media handlers, Data sources, Codecs/Effects, Renderers and Mux/Demuxes.

Input Stage: Map image objects are transcoded into video streams. The data is read from a source and passed in buffers to the processing stage. The input stage consists of reading data from a local capture device which is a local file system keeping static map images.

Processing Stage: The processing stage consists of a number of codecs and effects designed to modify the data stream to one suitable for output. These codecs may perform functions such as compressing or decompressing the audio to a different format, adding a watermark of some kind, cleaning up noise or applying an effect to the stream (such as echo to the audio).

Map video stream has several parameters that can be adjusted. These parameters affect the quality of the produced map video stream. Among these configurable parameters are frame rate and video format of the stream, update rate of the map images in the video stream. In our experiments we updated map images for every 0.5 seconds while we kept the video frame rate at 10 frames per second (fps). This provides a high quality of the video stream at the receiving side. This is necessary because some clients might not be capable of visualizing video streams with low frame rate or can visualize them with very low quality.

Map video streams produced are published to RTP sessions, whether unicast or multicast sessions. The RTP defines a standardized packet format for delivering audio and video over the Internet. AccessGrid clients use multicast sessions to send/receive video. Once a video is published to a multicast session, it

can be received by any client listening that multicast session as long as the underlying network lets client receive multicast packets. GlobalMMCS can also provide this map video stream to its clients as unicast video stream.

Output Stage (Publishing): Once the processing stage has applied its transformations to the stream, it passes the information to the output stage. The output stage may take the stream and pass it to a file on disk, output it to the local video display or transmit it over the network.

As soon as the movie frames are created at WMS side for each time slice for the same data layer, WMS publishes them as streams to specific RTP sessions. RTP sessions are represented as <IP Address, Port Number> pairs. A video stream published to a RTP session can be visualized by any video client connecting to the same RTP session by using JMF client locally installed. You can see the sample movie streams displayed by the JMF Client software in Figure 19. RTP session can be configured at properties file. The supported video stream formats are H.261 and H.263, which are mostly used formats in AccessGrid [95] sessions. Map video stream can be played in collaborative environments such as AccessGrid and GlobalMMCS sessions

outputURL = "rtp://" + ipAddress + ":" + portNumber + "/video";

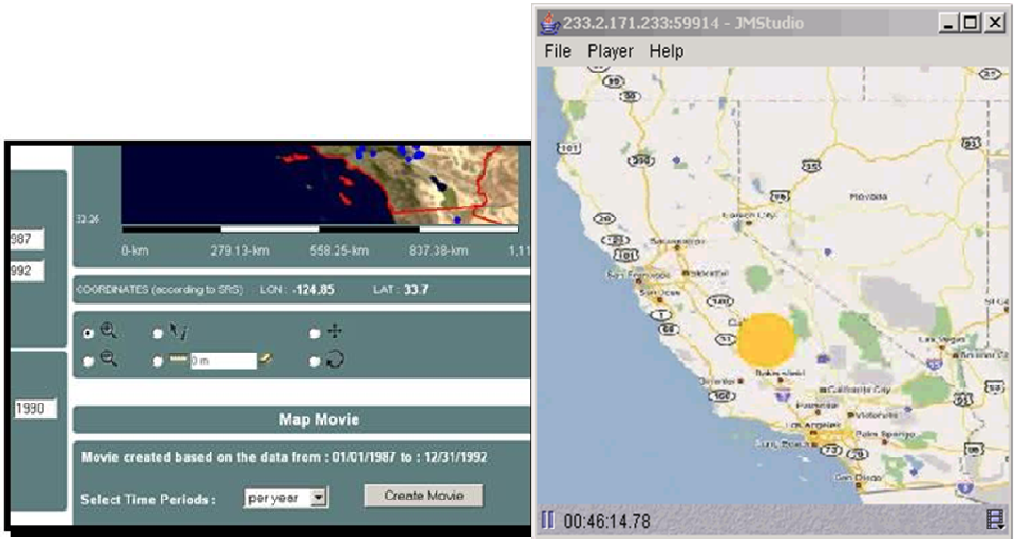


Figure 19: Interactive streaming map movies interface and sample map movie snapshot displayed through the JMF client

5.3.4. Browser-based Map Animation Techniques

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

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

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

5.3.5. Future Directions on Map Movies

There is no much work done on this area. This is also an ongoing work in our Lab. We have implemented a couple of good sample applications of streaming map movies for ServoGrid projects such as Pattern Informatics and Virtual California. We use Java Media Framework (JMF) libraries and JMF Studio to display the movie streams published to a multicast IP address and port number. Broadcast and Unicast addresses are also supported by the JMF Studio. Server creating movie streams sends streams to a multicast address, clients who wants to see the map movie streams should subscribe to this address. In our basic proof of concept application, clients use JMF studio to subscribe to the Real Time Protocol (RTP) session that they are interested in.

We will continue to integrate the streaming map server with Global-MMC's [59] archiving capabilities. This will enable useful functionality, such as allowing users to select a movie from the archive. To make the WMS available for the collaborative conferences or online education, admin user will be able to update map streams on the fly while it is playing. This is called annotation feature for the collaboration. For this purpose, we plan to integrate e-Sports [58] whiteboard drawing tools.

Currently, we support on-demand map movies creation and publishing at WMS side. In the future, we are planning to ceate movies at the client side even in case of collaborative movie creating environment. There will be significant performance gain when we use this approach. Clients can archive both previously created frames and movies. If a client needs same type of frames for the same matching time intervals then it does not need to go back to WMS and spend time for getting the movie frames

Right now for publishing collaborative map videos streams, we are using multicast addresses but in the future we will be using publish/subscribe properties of NB. Approaches to collaboration have tended to use IP Multicast to deal with the content distribution problem. Multicast provides a powerful, elegant and flexible framework for implementing collaborative systems. In this scenario, participants agree upon a multicast group and collaborate by exchanging data over this group; the system relies on MBONE to manage this interchange. Far more powerful framework for collaboration is the publish/subscribe paradigm. In publish/subscribe system the routing of messages from the publisher to subscriber is within the purview of the message oriented middleware (MOM), which is responsible for routing the right content from the producer to right consumers [64].

We are also planning to improve the user interface and propose user-oriented quality of services such as re-watching the animation multiple times in case of disappearances of some frames, stopping the animation and proceeding frame-by-frame, and adjusting the frame-rate or speed of the animation.

From our experience, the main issue in creating map movies and animations is the performance. Making animated maps sometimes take substantial amount of time. For example creating 10 minutes of map might require many hours to create depending on the application and data. Moreover, since our proposed Information Grid architecture and map movies creation are based on distributed systems, the performance of the system is limited by the network bandwidth. Distributed map services and data sources are connected with the internet protocols. Since we could not do anything to increase network bandwidth and fasten the internet connection, we work on improving the system performance through software solutions such as streaming data transfer addressing the bandwidth problems. We use NaradaBrokering publish-subscribe based messaging middleware system to implement streaming data transfer (See Chapter 6).

For the performance challenges faced during the development of GIS Information Grid and streaming map movies and our solution approaches see Chapter 6.

6. Performance Issues for Geo-Data handling

- Handling: Querying, transferring, rendering and displaying in Information Grid

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 network-bandwidth, 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.

In order to provide interoperability and extensibility we use common data format 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 and WFS are implemented by using NaradaBrokering publish-subscribe based messaging middleware [14].

The bottleneck of the proposed integration framework is “triangle” illustrated in *Figure 3*. From now on, our focus will be on improving this triangle’s performance. Therefore, we introduce innovative

techniques to increase the performance of the system to meet Geo-Science Grids' performance requirements.

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 6.1.

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 6.2.

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.

6.1. 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 (20MB see *Figure 20*-a). 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 OGC 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.

6.2. 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 20 shows current system's performance result improved with streaming data transfer and pull parsing techniques mentioned so far. This figure 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.

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 20* (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 3 while you are reading through the steps.

(a) (b)

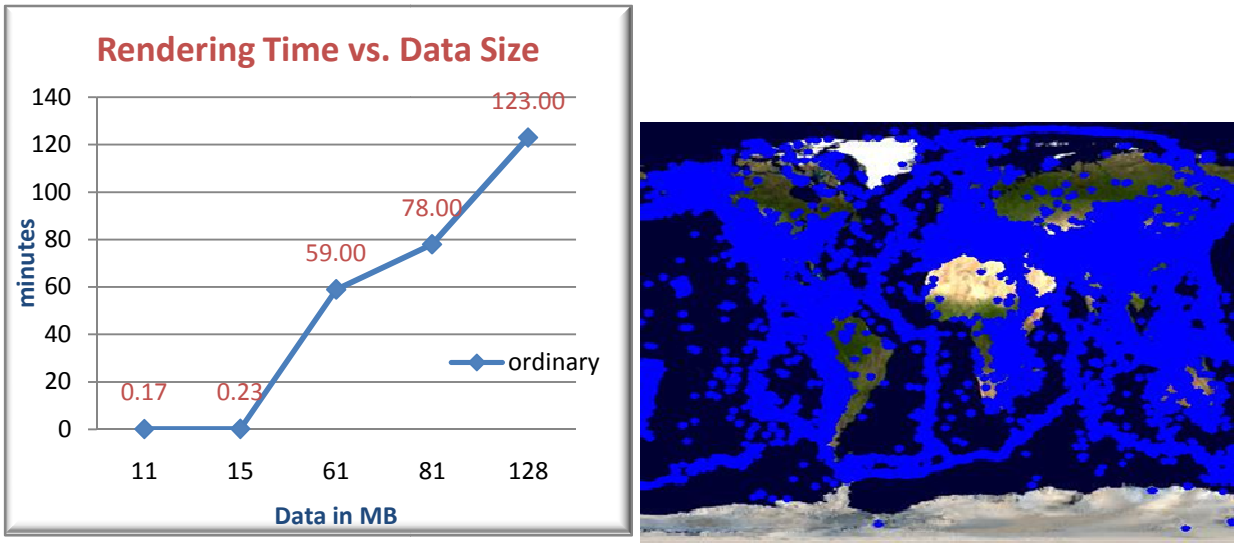


Figure 20: (a) Performance result of the current system. (b) Sample output consisting of Layer-1 and 2. See Figure 2.

- **[*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 the triangle (which is bottleneck of the system). Almost %90 of the *time*$_{(measured)}$ comes from so called "transferring GML data (feature collections) and cascaded map layers from WFS and WMS". This explains why the triangle 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 time still stays that much high as shown in the figure.

In the next chapter, we explain our efforts to improve the system's performance summarized in *Figure 20* (a). As we see from the figure, streaming data transfer and pull-parsing techniques are not good enough. Therefore, we develop innovative performance increasing techniques applied on geo-data transfer and processing.

6.3. 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 21 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 21 (a) client needs a map of the earth defined by bounding box (bbox) value of (a,b,c,d).

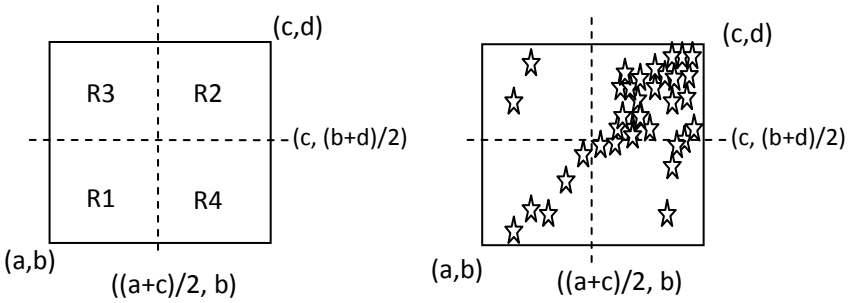


Figure 21: Classic partitioning cannot share the work equally among worker servers. Server assigned the partition of "((a+b)/2, (b+d)/2), (c, d)" gets the most of the work.

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 22) 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 21. We therefore propose architecture (Section 7) composed of combination of these three approaches in order to overcome performance and scalability problems in rendering of large size distributed spatial data.

7. 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. The innovative Web Map Service architecture is developed under the considerations of the General GIS problems mentioned in chapter 6 and Grid integration framework mentioned in chapter 3.

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.

7.1. Brief Architecture

Grid oriented WMS introduces three major concepts over regular WMS architecture [99] improving overall performance of the system. These are pre-fetching, load-balancing and caching. In summary, pre-fetching is purely for overcoming the natural bandwidth problem, caching helps us to prevent redoing the jobs of querying and rendering for the data requested before and 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 22*) 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 22.

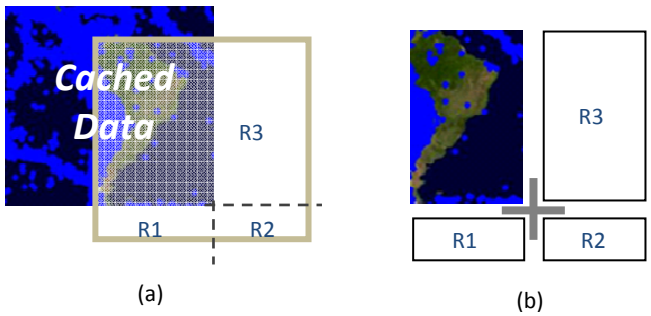


Figure 22: Illustration of the (a) cached data extraction and (b) rectangulation.

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).

The rectangles (ex. R1, R2 and R3 in *Figure 22*) 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 22 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.

The load balancing technique is indirectly applied. Each rectangle obtained through *rectangulation* represents a worker Map Service. The sub images formed by rectangulation processes 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.

7.1.1. 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.

7.1.2. 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 22). 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 22* 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 22 for the sample rectangles obtained through *rectangulation* process.

7.1.3. 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 6.1. A performance result of the pre-fetching is displayed at Figure 21. 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

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.

8. Preliminary Performance Tests

The performance tests here are for the bottleneck of the system which we called it as triangle illustrated in *Figure 3*. Later, we might give more performance results involving map movies and, coupling framework tested on one of information Grids applications such as Pattern Informatics.

Let's go back to *Figure 20*. Since we build our framework on XML and Web Services technologies, the figure showing current performance of the map rendering is not surprising (*Figure 20*). 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 triangle) regarding rendering and displaying maps consist of layer set 1 and 2 (see Figure 2), we mostly worked on these issues and proposed an approach in Chapter 7.

We have tested the proposed architecture over Pattern Informatics [21] Geo-Science Grids (see Chapter 3.4.1). 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 number 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: As we mentioned earlier, we have developed interactive decision making tools in order to utilize the proposed GIS Information Grid accessible remotely 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 time to a level measured in seconds. See the *Figure 20* 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 to 0 by using SOAP libraries, it will never throw timeout exception.

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. Briefly, we solve this problem by using pull parsing techniques mentioned earlier.

Categorizing proposed system's development 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 + Web Service:
   We have extended the services with Web Service principles and started using SOAP over HTTP protocols for data transfer and service invocations
3. OGC + Pull Parsing + 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 + Streaming:
   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 7 for more information)
5. OGC + Pull Parsing + Web Service + Streaming + Grid Oriented:

   - Browser time out issue is resolved

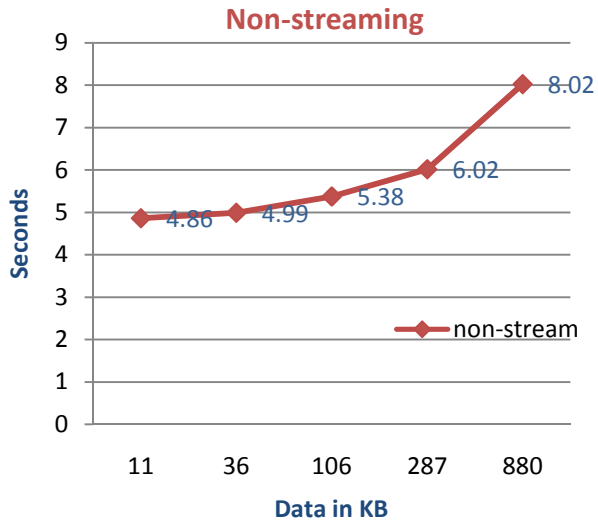
   Since we propose an interactive system and because of the geo-science applications' characteristics, response times should be restraint in a reasonable range. Till the step 4 we always had timeout problem when the user invokes the application with the large parameters. We solved this problem by using proposed techniques explained in Chapter 7.1 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:

We take last three categories listed above and show the improvements through the development life-cycle of the proposed Information Grid system. For later versions of this document, we are going to add

the performance results showing first two categories to dramatize the performance improvements over the regular GIS systems.

Performance graphs are all created from the end-user points of view. In other words graphs shows time values passed from the time client makes request to the time final result is obtained. The detailed steps building the time values in the graphs are given before in Chapter 6.2.

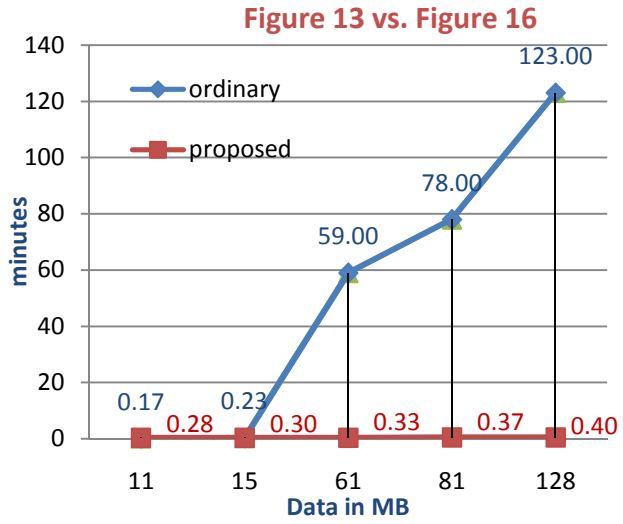


Data in **KB**	Non-stream Time in **seconds**
11	4.86
36	4.99
106	5.38
287	6.02
880	8.02

Figure 23: 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.**

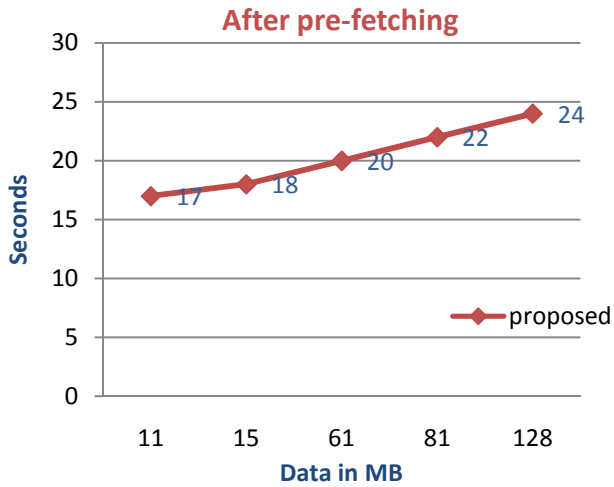


Data in **MB**	-Usual- **minutes**	-proposed- **minutes**
11	0.17	0.28
15	0.23	0.30
61	59.00	0.33
81	78.00	0.37
128	123.00	0.40

Figure 24: Performance graph of the system for Category-4

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 Chapters 6.1, 6.2, 7.1.1 and 7.1.2.
Note, the data sizes are in **MB** and timings are **MINUTES.**



Data in **MB**	-Proposed- seconds
11	17
15	18
61	20
81	22
128	24

Figure 25: 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 7.1.3 in addition to the techniques used in the previous performance test.

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

9. Expected Contributions

GIS is our motivating domain to define information Grids' capabilities, requirements and challenges. 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.

Information Grid is composed of data/information and computational sources. In order to create GIS Information Grid we have introduced a coupling framework for Geo-Science computational Grids with distributed heterogeneous data sources. We have introduced extensions to the OGC standards in order for OGC GIS Web Services (WMS and WFS) to be chained through capability federation and inter-service communication (see Chapter 3.3).

In order to build such a coupling 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 introduce two types of mediators called type-1 and type-2. Type-1 enables Grid-enabled OGC web services and non-Grid-enabled ones to interact for data access and querying. Type-2 mediator translates a request from the Grid-enabled OGC compatible GIS system's language to that of the information source and transform the results provided by the information source back to the Grid-enabled OGC compatible GIS's language.

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. In this context, we also introduced innovative architectural framework to integrate AJAX and Web Services technologies for WMS and WFS OGC Web Services.

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 also developed innovative animations and streaming map movies architecture for time-series geospatial data. In order to overcome the drawbacks of these time and CPU consuming jobs we introduced innovative data transfer (streaming through *Naradabrokering* messaging middleware), parsing (pull parse) and rendering techniques. Over all, we use high performance load balancing, caching and pre-fetching techniques to solve performance issues to keep pace with the GIS time constraints and performance requirements for decision support.

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.

10. Future Work

In this thesis we have outlined our initial research and implementations to build a geophysical information Grid architecture and, focused on the issues in terms of GIS and geographic data. We

addressed several issues such as coupling of data and computational Grids, data and service heterogeneity and chaining of OGC GIS Web Services related to archival data access and processing. At the end of this dissertation we discuss possible future directions for this research.

Although our motivating domain was GIS and, we have used domain dependent data models and online services defined by OGC standard specifications, findings and recommendations are relevant for any other science domains and data types. 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. We think that it is important to explore how the common data standards such as GML and service standards such as WFS or WMS are being used in different sub-domains in the GIS community, and if any improvements are needed in such standards. Semantic conflicts among information systems occur whenever information systems do not use the same interpretation of the information.

In order to chain GIS Web Services we have proposed some techniques. These are explained in Chapter 3.3. In case of chaining Web Feature Services or creating WFS-oriented mediators as proposed in Chapter 4.1.2 as mediator type-2, we need to develop XLink-WFSs. Currently, we have developed Basic WFS and we need to upgrade them to XLink-WFS according to OGC standard specifications and definitions.

Regarding the data heterogeneity, we mostly focused on structured (different data models and schematic heterogeneities), syntactical (different languages and data representations or basically database heterogeneity) and systemic (different hardware and operating systems) heterogeneity issues in the integration framework and, the feasibility of such an integration framework [74]. We have used Local as View (LAV) data integration approach to solve these problems. The best know approach to solve this problem is using mediator systems. Since we applied our proposed integration framework in OGC compatible GIS domain we did not take the semantic heterogeneity of the data into consideration. Semantics of the data and services are already defined by OGC as long as we stay in GIS domain. Capabilities metadata for the services and Geographic Markup Language (GML) for the data model can be considered as semantic concepts and applications.

Integration has been an acknowledged data processing problem for a long time. However, there is no universal tool for general data integration. Because various data descriptions, data heterogeneity, and machine un-readability, it is not easy way. Improvement in this situation could bring the Semantic Web. Its idea is based on machine understandable web data, which bring us an opportunity of better automated processing. The semantic Web is still future vision, but there are already some features we can use. Semantic Web considers data to go along with their meanings. An addition of semantics would make data machine readable and understandable. Automated processing also would be easier.

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] Cox, S., Daisey, P., Lake, R., Portele, C., and Whiteside, A. (eds) (2003), OpenGIS Geography Markup Language (GML) Implementation Specification. OpenGIS project document reference number OGC 02-023r4, Version 3.0.

[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

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

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

[33] Koontz, L. D. Geographic Information Systems: Challenges to Effective Data Sharing, Washington, D.C.: General Accounting Office, Report GAO-03-874T. 2003.

[34] 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.

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

[36] Ahmet Sayar, Marlon Pierce and Geoffrey Fox, Developing GIS Visualization Web Services for Geophysical Applications, ISPRS 2005 Spatial Data Mining Workshop, Ankara, Turkey.

[37] Galip Aydin, Ahmet Sayar, Harshawardhan Gadgil, Mehmet S. Aktas, Geoffrey C. Fox, Sunghoon Ko, Hasan Bulut, and Marlon E. Pierce Building and Applying Geographical Information Systems Grids. (To appear) in journal of Concurrency and Computation: Practice and Experience

[38] Kreger, H., Web Services Conceptual Architecture (WSCA 1.0). 2001. p. 6-7.

[39] Belwood, T., L. Clement, and C. von Riegen, UDDI Version 3.0.1: UDDI Spec Technical Committee Specification. Available from http://uddi.org/pubs/uddiv3.0.1-20031014.htm. 2003

[40] Christensen, E., et al., Web Services Description Language (WSDL) 1.1. 2001, March.

[41] Peng, Z.R. and M. Tsou, Internet GIS: Distributed Geographic Information Services for the Internet and Wireless Networks. 2003: Wiley

[42] Kirtland, M., A Platform for Web Services. 2001, Jan.

[43] Redmond, F.E., Dcom: Microsoft Distributed Component Object Model with Cdrom. 1997: IDG Books Worldwide, Inc. Foster City, CA, USA

[44] Redmond, F.E., Dcom: Microsoft Distributed Component Object Model with Cdrom. 1997: IDG Books Worldwide, Inc. Foster City, CA, USA. 90. Microsystems, S., Java Remote Method Invocation Specification. 2002.

[45] Di, L., et al., The Integration of Grid Technology with OGC Web Services (OWS) in NWGISS for NASA EOS Data, in GGF8 & HPDC12 2003: Seattle, USA. . p. 24-27.

[46] Fox, G. and M. Pierce. Web Service Grids for iSERVO. in International Workshop http://www.eps.s.u-tokyo.ac.jp/jp/COE21/events/20041014.pdf on Geodynamics: Observation, Modeling and Computer Simulation University of Tokyo Japan October 14 2004. 2004.

[47] de La Beaujardiere, J., Web Map Service, OGC project document reference number OGC 04-024. 2004.

[48] Vretanos, P. (2002) Web Feature Service Implementation Specification, OpenGIS project document: OGC 02-058, version 1.0.0. Volume,

[49] Cox, S. (2003) Observations and Measurements. Volume, DOI: OGC 03-022r3.

[50] ESRI, ArcIMS, 9 Architecture and Functionality, J-8694. ESRI White Paper, http://downloads.esri.com/support/whitepapers/ims_/arcims9-architecture.pdf. 2004.

[51] Autodesk. MapGuide http://usa.autodesk.com. [cited.

[52] MapServer, W. http://www.wthengineering.com/GIS/web_gis.htm. [cited.

[53] Fox, G. and M. Pierce. Web Service Grids for iSERVO. in International Workshop http://www.eps.s.u-tokyo.ac.jp/jp/COE21/events/20041014.pdf on Geodynamics: Observation, Modeling and Computer Simulation University of Tokyo Japan October 14 2004. 2004.

[54] Aydin, G., et al. SERVOGrid Complexity Computational Environments (CCE) Integrated Performance Analysis. in Grid Computing, 2005. The 6th IEEE/ACM International Workshop on. 2005: IEEE.

[55] Fox, G. and M. Pierce, SERVO Earthquake Science Grid, in summary of iSERVO technology October 2004 in January 2005 report High Performance Computing Requirements for the Computational Solid Earth Sciences edited by Ron Cohen and started at May 2004 workshop on Computational Geoinformatics.

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

[57] Harshawardhan Gadgil, Geoffrey Fox, Shrideep Pallickara, Marlon Pierce, Robert Granat A Scripting based Architecture for Management of Streams and Services in Real-time Grid Applications Proceedings of the IEEE/ACM Cluster Computing and Grid 2005 Conference (CCGrid 2005). Cardiff, UK May 2005 Pages 710-717.

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

[59] Geoffrey Fox, Wenjun Wu, Ahmet Uyar, Hasan Bulut, Shrideep Pallickara: Global multimedia collaboration system. Concurrency - Practice and Experience 16(5): 441-447 (2004).

[60] Hasan Bulut, Wenjun Wu, Geoffrey Fox, Ahmet Uyar, Shrideep Pallickara, Harun Altay: A Web Services Based Streaming Gateway for Heterogeneous A/V Collaboration. International Conference on Internet Computing 2004: 493-502.

[61] Wenjun Wu, Geoffrey Fox, Hasan Bulut, Ahmet Uyar, Harun Altay “Design and Implementation of A Collaboration Web-services system”, Journal of Neural, Parallel & Scientific Computations (NPSC), Volume 12, 2004.

[62] Blok, C., Kobben, B., Cheng, T., and Kuterema, A., 1999, Visualization of relationships between spatial patterns in time by cartographic animation, Cartography and Geographic Information Science, 26 (2), 139-151.

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

[64] Shrideep Pallickara, Hasan Bulut, Pete Burnap, Geoffrey Fox, Ahmet Uyar, David Walker Support for High Performance Real-time Collaboration within the NaradaBrokering Substrate Technical Report May 2005

[65] Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, J. D. Ullman: A Query Translation Scheme for Rapid Implementation of Wrappers. DOOD 1995: 161-186. http://www.cse.ucsd.edu/~yannis/papers/querytran.ps

[66] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object Fusion in Mediator Systems. In Intl. Conf. on Very Large Data Bases (VLDB), 1996.

[67] A. Tomasic, L. Raschid, P. Valduriez: Scaling Access to Heterogeneous Data Sources with DISCO. IEEE Transactions on Knowledge and Data Engineering, Vol. 10, No. 5, 1998:808-823.

[68] A. Levy, A. Rajaraman and J. Ordille: Querying Heterogeneous Information Sources Using Sources Descriptions. Proceedings of VLDB, 1996: 251-262.

[69] Y. Shu, J. F. Zhang and X. Zhou, "A Grid-enabled Architecture for Geospatial Data Sharing", to appear in Proceedings of the 2006 IEEE Asia-Pacific Conference on Services Computing (APSCC 2006), 12-15 December 2006, Guangzhou, Guangdong, China. IEEE Computer Society Press.

[70] S. Jerome, Web Map Context Service (WMC), OGC project document reference number OGC 05-005. Version 1.1.0. 2005.

[71] Michael Boyd, Charalambos Lazanitis, Sasivimol Kittivoravitkula, Peter Mc. Brien, and Nikolaos Rizopoulos. AutoMed: A BAV Data Integration Sys-tem for Heterogeneous Data Sources. In Advanced Information Systems Engineering 16th International Conference, CAiSE 2004, Riga, Latvia, June 7-11, 2004, Pro-ceedings. Springer-Verlag, 2004.

[72] Mark Friedman, L. Y. Alon and M. D. Todd. Navigational Plans For Data Integration. In AAAA/IAAI, pages 67-73, 1999.

[73] Maurizio Lenzerini, Data Integration: A Theoretical Perspective. In PODS Proceedings, PAGES 233-246, 2002. Invited Tutorial.

[74] Y. A. BISHR: Overcoming the Semantic and Other barriers to GIS Interoperability. International Journal of Geograohic Information Science 4 (1998), 299-314.

[75] Wiederhold G (1992) Mediators in the Architecture of Future Information Systems. IEEE Computer, 25, 3, 38-49.

[76] Busse, S., Kutsche, R.-D., Leser, U., and Weber, H.: Federated Information Systems: concepts, terminology and architectures, Technical Report Nr. 99-9, TU Berlin, 1999.

[77] Zaslavsky, I., Ludäscher, B., Gupta, A., and Tran, J. (2002). "Feature interpretation in vector data: reconciling spatial and semantic integrity constraints". Extended abstracts of the 2nd International Geographic Information Science Conference, Boulder, Colorado, November 2002, pp. 215-218

[78] L. Bigagli, S. Nativi, P. Mazzett, Mediation to deal with information heterogeneity – application to Earth System Science, Advances in Geosciences, Vol. 8, pp 3-9, 6-6-2006.

[79] Boucelma,O., Essid, m., Lacroix, Z., Vinel, J., Garinet, J., Bétari, A.: VirGIS: Mediation for Geographical Information Systems. IEEE, ICDE 2004: 855-856.

[80] Baru, C., Gupta, A., Ludäscher, B., Marciano, R., Papakonstantinou, Y., Velikhov, P., Yannakopulos, A. "XML-Based Information Mediation with MIX", Proceedings of the SIGMOD'99, 1999.

[81] MySql, A.B., MySQL Database Server, WWW page, at URL: http://www. mysql.com, last accessed on 12/17/2006.

[82] Belwood, T., L. Clement, and C. von Riegen, UDDI Version 3.0.1: UDDI Spec Technical Committee Specification. Available from http://uddi.org/pubs/uddiv3.0.1-20031014.htm. 2003.

[83] OGC Filter Encoding Implementation Specification, OGC document number 04-095 Volume,

[84] Organization for the Advancement of Structured Information Systems (OASIS), Service Oriented Architecture Reference Model Technical Committee (SOA-RM TC), www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm

[85] Alameh N., Chaining geographic information web services, IEEE Internet Computing, Sept-Oct 2003, 22- 29.

[86] Ian Johnson and Andres Wilson, "The TimeMap Project: Developing Time- Based GIS Display for Cultural Data", Journal of GIS in Archaeology, Vol. 1, ESRI Inc., Redlands, California, 2002.

[87] Hoschka, P. (1998).An Introduction to the Synchronized Multimedia Integration Language (SMIL), IEEE Multimedia , Vol. 5, No. 4, Oct.-Dec (pp. 84 -88).

[88] Theresa- Marie Rhyne, Web Horizons for Geographic Visualization, Geoinformatics, December 2000, pp. 35-37

[89] Bederson, B.B. and Boltman, A.,Does Animation Help Users Build Mental Maps of Spatial Information. CS-TR-3964, UMIACS-TR-98-73,(1998). In IEEE Info Vis 99, 28-35.

[90] Kehoe, C., Stasko, J., and Taylor, A. Rethinking the Evaluation of Algorithm Animations as Learning Aids: An Observational Study. Intl. Journal of Human Computer Studies 54(2), (2001), 265-284.

[91] Harrower, M., "Visual Benchmarks: A new method for enhancing animated maps." Paper presented at North American Cartographic Information Society (NACIS) XXII, Columbus, Ohio, October 2002.

[92] Harrower, M., Tips for Designing Effective Animated Maps. Cartographic Perspectives 2003, (44), 63-65.

[93] M. Harrower, A look at the history and future of animated maps. Cartographica 2004, 39(3): 33-42

[94] Gersmehl, P. J. 1990. "Choosing tools: Nine metaphors for four-dimensional cartography." Cartographic Perspectives, Spring: 3-17.

[95] Access Grid side is available at http://www.accessgrid.org

[96] RTP: A Transport Protocol for Real-Time Applications (IETF RFC 1889) http://www.ietf.org/rf c/rfc1889.txt.

[97] Sun Microsystems Inc. Java Media Framework API Guide, September 3, 1999, v. 0.8.

[98] Sayar A., Pierce M., Fox G. C., Grid Map Tools and GIS Portal, Technical Report, March 2005. Available at http://complexity.ucs.indiana.edu/~asayar/gisgrids/docs/sci-data-plottingdoc

[99] Sayar A., Pierce M., Fox G. C., Web Services Based Web Mapping Service, Technical Report, June 2004. Available at http://complexity.ucs.indiana.edu/~asayar/proposal/asayar_servo3.pdf

[100] Google Map API, available at http://www.google.com/apis/maps/

[101] OGC 's standard schema files are available at http://schemas.opengis.net/

[102] Tyler Mitchell, "Build AJAX-Based Web Maps Using ka-Map" August 2005. Available at http://www.xml.com/pub/a/2005/08/10/ka-map.html.

[103] EcmaScript project web site is available at http://www.ecmainternational.org/

[104] ECMAScript Language Specification. ECMA International, third edition, 1999.

[105] LITWIN, W. 1985. An overview of the multidatabase system, Multi Relational Database Systems Management (MRDSM). In Proceedings of the ACM National Conference (Denver, Oct.), pp. 495-504

[106] Reed, Joshua A., Cervato, Cinzia, Fils, Doug, CHRONOS's PSICAT: a graphical core-logging tool for the 21st century, p 33-34 GeoSpectrum vol 5, no 2, February 2006

[107] Cervato, C., Goldstein, S.L., Grossman, E.L., Lehnert, K., and McArthur, J.M., Joint Discussion of Sedimentary Geochemistry Data Management Systems That Cross the Waterline, EOS, vol. 85, 44, 2 November 2004.

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.**

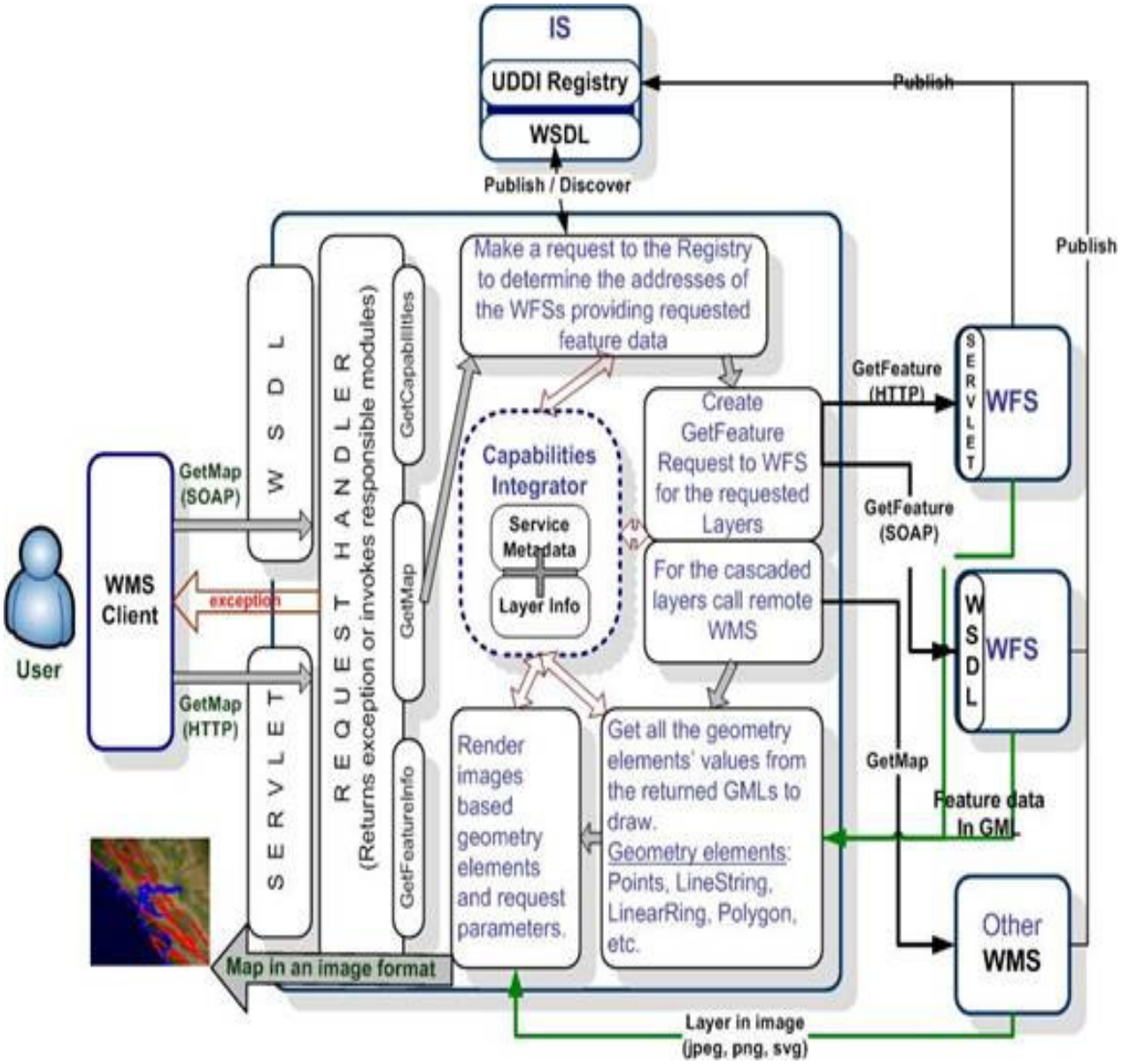
```
<wfs:FeatureCollection
      xmlns:wfs=http://www.opengis.net/wfs
      xmlns:gml=http://www.opengis.net/gml
      xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
      xsi:schemaLocation="http://crisisgrid.org/schemas/fault_new.xsd">
      <gml:boundedBy>
            <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#27354">
                  <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**

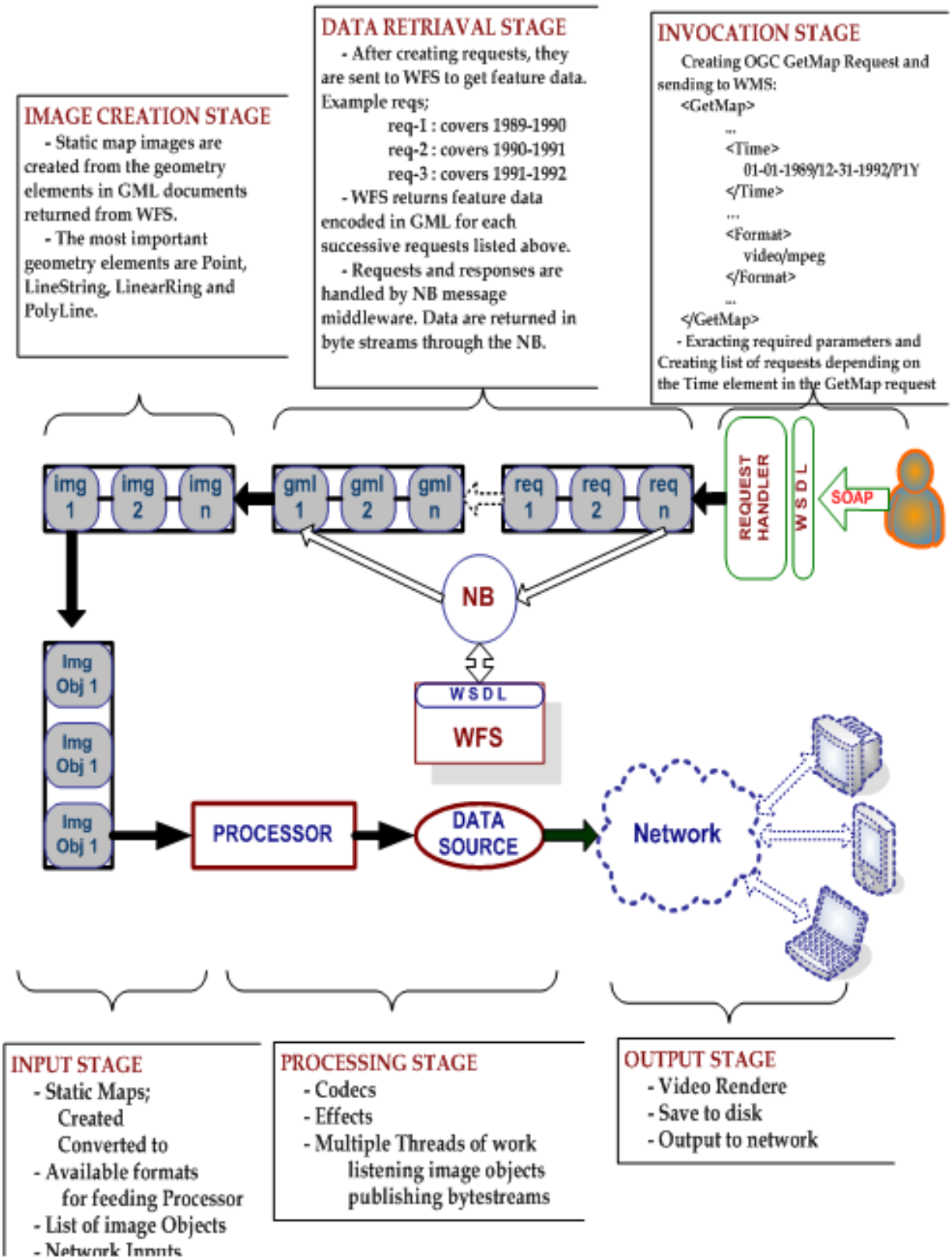
```
<?xml version="1.0" encoding="UTF-8"?>
<GetMap xmlns="http://www.opengis.net/ows">
    <version>1.1.1</version>
    <service>wms</service>
    <exceptions>application_vnd_ogc_se_xml</exceptions>
    <Map>
        <BoundingBox decimal="." cs="," ts="">
            -124.85,32.26,-113.56,42.75
        </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, you can also see the reference 99 for more details.



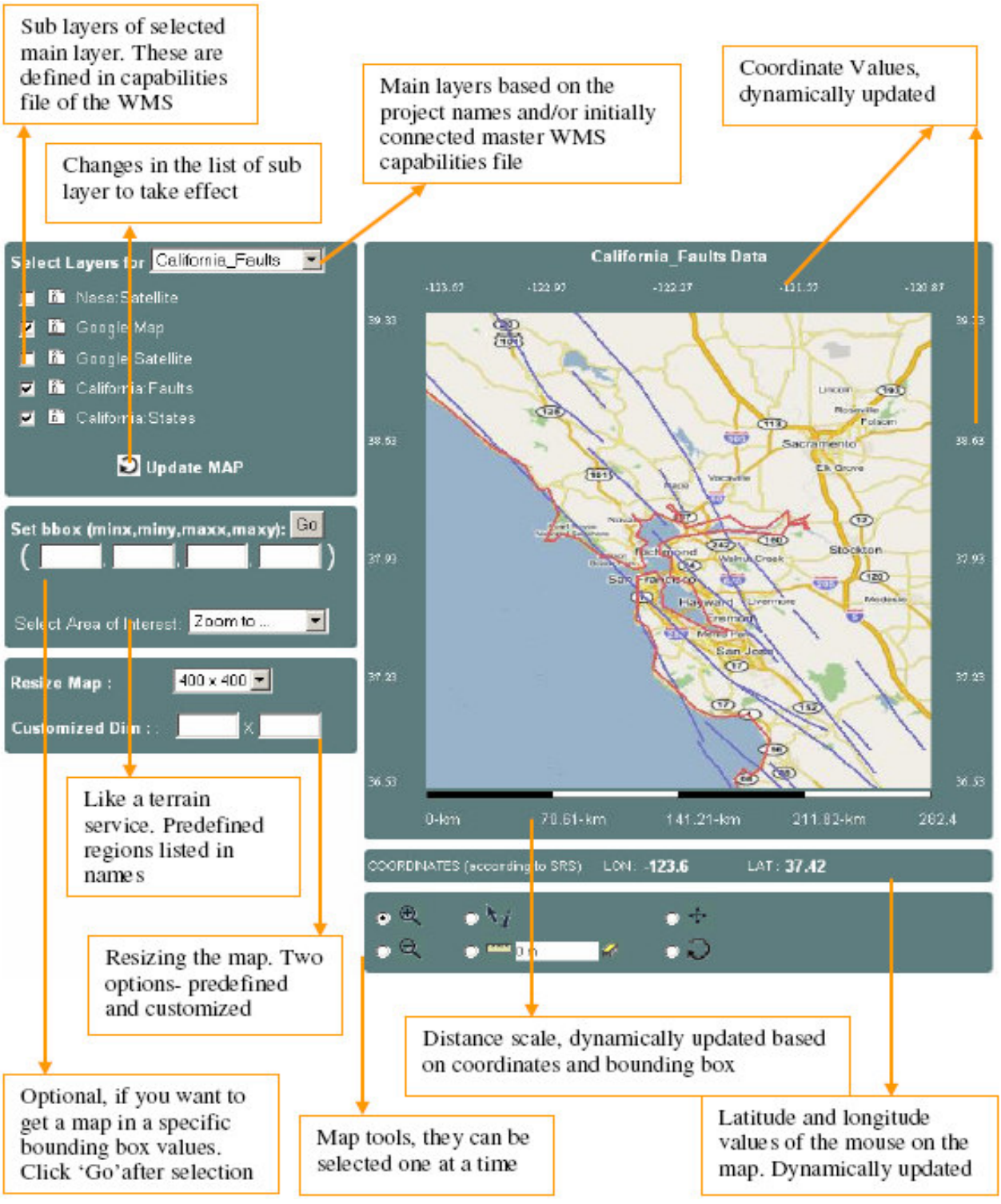
9. Detailed Movie creation architecture from the static map images:



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

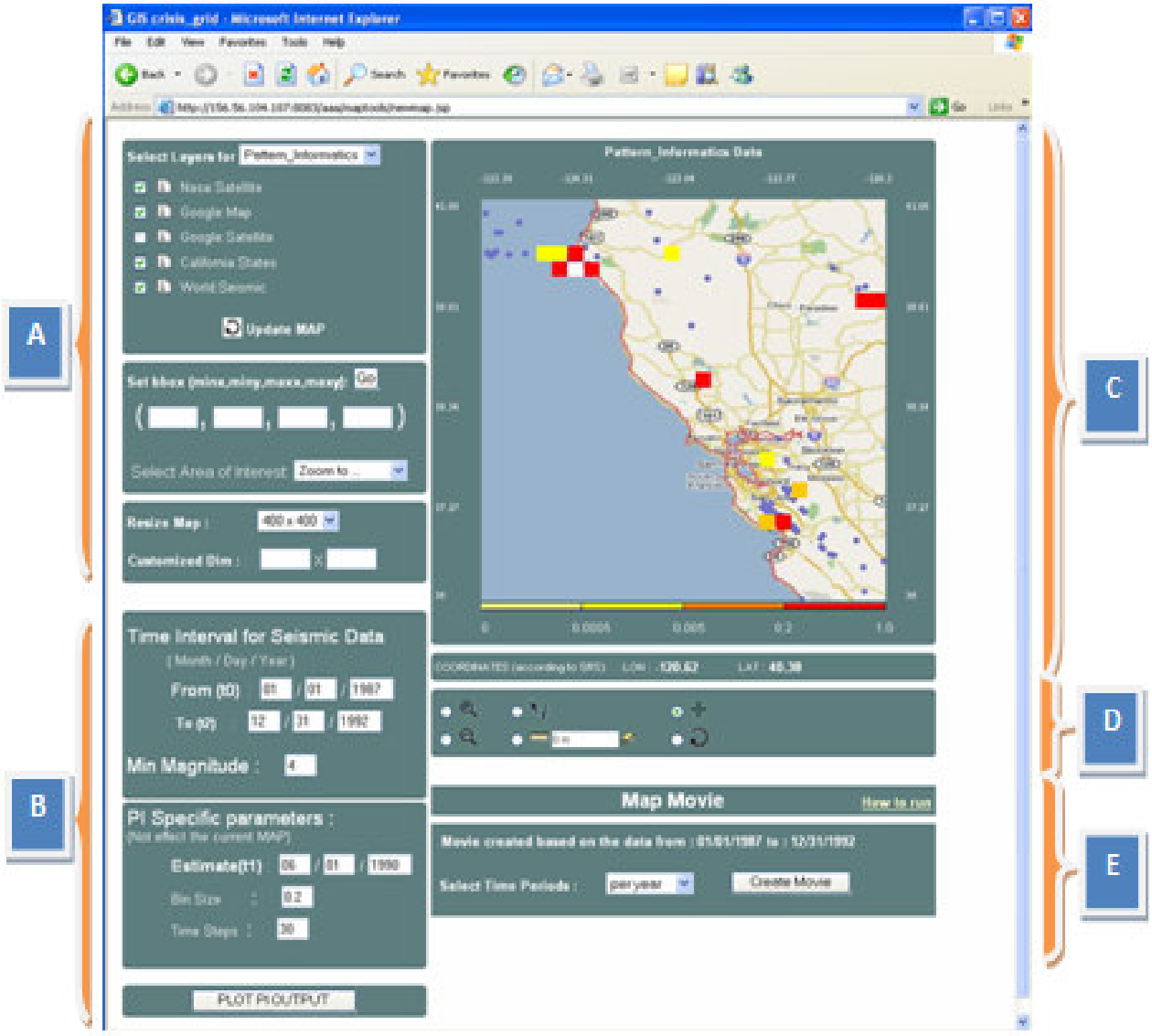
California earthquake fault data is overlaid Google Map.

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



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.

12. Generic innovative XSL file for HTML creation from the GML in order to create responses for the getFeatureInfo

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:cgl="http://www.commu
  xmlns:gml="http://www.opengis.net/gml">
  - <xsl:template match="cgl:FeatureCollection">
    - <html>
        <meta content="" />
      - <body bgcolor="#FFFFFF">
        - <table align="center" bgcolor="#DDDDDD" border="0" cellspacing="10">
          - <tr>
            - <td>
              - <h2>
                  <font face="Helvetica, Arial, sans-serif" color="#FF0000">CGL</font>
                  <font face="Helvetica, Arial, sans-serif" color="#566D7E">WMS 1.1.1 - Feature Informations</font>
                </h2>
              - <table border="0" width="500" cellpadding="0" cellspacing="0">
                  <xsl:apply-templates select="gml:featureMember" />
                </table>
                <hr />
              - <font face="Helvetica, Arial, sans-serif" size="-1">
                  (c) [ CGL WMS 1.1.1 ] -
                  <script LANGUAGE="JavaScript">var now = new Date(); document.write( now );</script>
                </font>
              </td>
            </tr>
          </table>
        </body>
      </html>
    </xsl:template>
  - <xsl:template match="gml:featureMember">
    - <tr>
      - <xsl:if test="position() mod 2 > 0">
          <td bgcolor="#AFC7C7" width="30" />
        - <td bgcolor="#AFC7C7">
            <br />
          - <xsl:for-each select="./child::*">
            - <font face="Helvetica, Arial, sans-serif" color="#806517">
              - <xsl:for-each select="./*">
                  <xsl:value-of select="name( . )" />
                  =
                  <xsl:value-of select="." />
                  <br />
                </xsl:for-each>
              </font>
            </xsl:for-each>
            <br />
          </td>
        </xsl:if>
      - <xsl:if test="position() mod 2 = 0">
```

```
        <td bgcolor="#95B9C7" width="30" />
      - <td bgcolor="#95B9C7">
          <br />
        - <xsl:for-each select="./child::*">
          - <font face="Helvetica, Arial, sans-serif" color="#805817">
            - <xsl:for-each select="./*">
                <xsl:value-of select="name( . )" />
                =
                <xsl:value-of select="." />
                <br />
              </xsl:for-each>
            </font>
          </xsl:for-each>
          <br />
        </td>
      </xsl:if>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```

13. Sample WMS Capabilities File

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSPY v2004 rel. 4 U (http://www.xmlspy.com)-->
<WMS_Capabilities xmlns="http://www.opengis.net/wms" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wms
C:\capabilities_1_3_0.xsd" version="1.3.0" updateSequence="String">
        <Service>
                <Name>WMS</Name>
                <Title>Pervasive WMS</Title>
                <Abstract>wms reference implementation</Abstract>
                <KeywordList>
                        <Keyword >pervasive</Keyword>
                        <Keyword >wms</Keyword>
                </KeywordList>
                <OnlineResource xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="simple"
                xlink:href="http://toro.ucs.indiana.edu:8086/WMSServices.wsdl"/>
                <!-- the following service information is optional -->
                <ContactInformation>
                        <ContactPersonPrimary>
                                <ContactPerson>Ahmet Sayar</ContactPerson>
                                <ContactOrganization>Pervasive Tech Lab</ContactOrganization>
                        </ContactPersonPrimary>
                        <ContactPosition>Research Assistant</ContactPosition>
                        <ContactAddress>
                                <AddressType>XXXX</AddressType>
                                <Address>501 N. Morton St. Rm 222</Address>
                                <City>Bloomington</City>
                                <StateOrProvince>IN</StateOrProvince>
                                <PostCode>47404</PostCode>
                                <Country>USA</Country>
                        </ContactAddress>
                        <ContactVoiceTelephone>1(812)8560752</ContactVoiceTelephone>
                        <ContactFacsimileTelephone>1(812)8567972</ContactFacsimileTelephone>

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

```
                </DCPType>
        </GetCapabilities>
        <GetMap>
                <Format>image/gif</Format>
                <Format>image/png</Format>
                <Format>image/jpg</Format>
                <Format>image/tif</Format>
                <Format>image/bmp</Format>
                <Format>image/svg+xml</Format>
                <DCPType>
                        <!-- Currently there is just one DCPT environment supported HTTP.
                                        In the near future there will be web services
                                        support by the Open-GIS.
                                        Whenever they update their standard schemas, I
                                        will update my capabilities document.
                                Since I am going to use web mep services I do not need
                                these informations -->
                        <HTTP><Get><OnlineResource /></Get>
                                <Post>   <OnlineResource /></Post>
                        </HTTP>
                </DCPType>
        </GetMap>
</Request>
<Exception>
        <Format>application/vnd.ogc.se_xml</Format>
        <Format>application/vnd.ogc.se_inimage</Format>
        <Format>application/vnd.ogc.se_blank</Format>
</Exception>
<Layer queryable="0" cascaded="1" opaque="0" noSubsets="0" fixedWidth="1"
                                                                fixedHeight="1">
        <Name>pervasive WMS-demo Layers</Name>
        <Title>pervasive WMS-demo Layers</Title>
        <Abstract>pervasive WMS-demo Layers</Abstract>
        <KeywordList>
                <Keyword>pervasive</Keyword>
                <Keyword>WMS</Keyword>
                <Keyword>layer</Keyword>
        </KeywordList>
        <CRS>EPSG:4326</CRS>
        <EX_GeographicBoundingBox>
                <westBoundLongitude>-150</westBoundLongitude>
                <eastBoundLongitude>100</eastBoundLongitude>
                <southBoundLatitude>30</southBoundLatitude>
                <northBoundLatitude>50</northBoundLatitude>
        </EX_GeographicBoundingBox>
        <MinScaleDenominator>0</MinScaleDenominator>
        <MaxScaleDenominator>100000000</MaxScaleDenominator>

        <!-- WORLD SEISMIC -->
        <Layer queryable="0" cascaded="1" noSubsets="0">
                <Title>World_Seismic</Title>
                <Abstract>Seismic data for the world</Abstract>
                <CRS>EPSG:4326</CRS>
```

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

14. **Sample WFS Capabilities File**

```
<?xml version="1.0" encoding="UTF-8"?>
<WFS_Capabilities xmlns="http://www.opengis.net/wfs" version="1.0.0">
 <Service>
   <Name>Web Feature Service</Name>
   <Title>WFS@gf1:7474</Title>
   <Abstract></Abstract>
   <Keywords>WFS, OGC, Web Services</Keywords>
   <OnlineResource xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="java:java.lang.String">http://gf1.ucs.indiana.edu:7474/axis/services/
wfs?wsdl</OnlineResource>
   <Fees>None</Fees>
   <AccessConstraints>None</AccessConstraints>
 </Service>
 <Capability>
   <Request>
     <GetCapabilities>
       <DCPType>
         <HTTP>
           <Get
onlineResource="http://gf1.ucs.indiana.edu:7474/axis/services/wfs?wsdl"/>
           <Post
onlineResource="http://gf1.ucs.indiana.edu:7474/axis/services/wfs?wsdl"/>
         </HTTP>
       </DCPType>
     </GetCapabilities>
     <DescribeFeatureType>
       <SchemaDescriptionLanguage>
         <XMLSCHEMA/>
       </SchemaDescriptionLanguage>
       <DCPType>
         <HTTP>
           <Get
onlineResource="http://gf1.ucs.indiana.edu:7474/axis/services/wfs?wsdl"/>
           <Post
onlineResource="http://gf1.ucs.indiana.edu:7474/axis/services/wfs?wsdl"/>
         </HTTP>
       </DCPType>
     </DescribeFeatureType>
     <GetFeature>
       <ResultFormat>
         <GML2/>
       </ResultFormat>
       <DCPType>
         <HTTP>
           <Get
onlineResource="http://gf1.ucs.indiana.edu:7474/axis/services/wfs?wsdl"/>
           <Post
onlineResource="http://gf1.ucs.indiana.edu:7474/axis/services/wfs?wsdl"/>
         </HTTP>
       </DCPType>
     </GetFeature>
   </Request>
   <VendorSpecificCapabilities>WSDL-SOAPE</VendorSpecificCapabilities>
 </Capability>
 <FeatureTypeList>
   <FeatureType>
     <Name>rivers</Name>
     <Title>California Rivers Feature Type</Title>
     <Abstract>A Feature that has coordinate information of california
rivers</Abstract>
     <Keywords>California,River,Rivers,WFS</Keywords>
     <SRS>EPSG:4326</SRS>
     <Operations>
       <Query/>
     </Operations>
```

```
      <LatLongBoundingBox minx="-124.275833" miny="35.389717" maxx="-118.075287"
maxy="41.472763"/>
    </FeatureType>
    <FeatureType>
      <Name>fault</Name>
      <Title>California Fault data</Title>
      <Abstract>California Fault data provided by USC</Abstract>
      <Keywords>California,Fault,Segment,WFS</Keywords>
      <SRS>NULL</SRS>
      <Operations>
        <Query/>
      </Operations>
      <LatLongBoundingBox minx="-124.41" miny="31.89" maxx="-114.64"
maxy="40.2"/>
    </FeatureType>
    <FeatureType>
      <Name>europe</Name>
      <Title>europe borders</Title>
      <Abstract/>
      <Keywords>europe,wfs</Keywords>
      <SRS>EPSG:4326</SRS>
      <Operations>
        <Query/>
      </Operations>
      <LatLongBoundingBox minx="-31.291612" miny="-31.291612" maxx="44.834987"
maxy="71.181357"/>
    </FeatureType>
    <FeatureType>
      <Name>states</Name>
      <Title>US States Boundaries</Title>
      <Abstract>Borders for states</Abstract>
      <Keywords>borders,states</Keywords>
      <SRS>null</SRS>
      <Operations>
        <Query/>
      </Operations>
      <LatLongBoundingBox minx="-178.21759836237" miny="18.921786345087" maxx="-
67.007718759568" maxy="71.406235353271"/>
    </FeatureType>
    <FeatureType>
      <Name>scsn</Name>
      <Title>California Earthquake Data in SCSN Format</Title>
      <Abstract>Earthquake data</Abstract>
      <Keywords>California,Earthquake,WFS</Keywords>
      <SRS>EPSG:4326</SRS>
      <Operations>
        <Query/>
      </Operations>
      <LatLongBoundingBox minx="32" miny="-122" maxx="37" maxy="-114"/>
    </FeatureType>
    <FeatureType>
      <Name>scedc</Name>
      <Title>California Earthquake Data in SCEDC Format</Title>
      <Abstract>Earthquake data</Abstract>
      <Keywords>California,Earthquake,WFS</Keywords>
      <SRS>EPSG:4326</SRS>
      <Operations>
        <Query/>
      </Operations>
      <LatLongBoundingBox minx="-122.000" miny="-122.000" maxx="78.600"
maxy="37.000"/>
    </FeatureType>
   <FeatureType>
     <Name>sopac</Name>
     <Title>SOPAC GPS Stations</Title>
     <Abstract>Metadata About the SCIGN GPS station</Abstract>
     <Keywords>California,Earthquake,WFS</Keywords>
```

```
      <SRS>WGS84</SRS>
      <Operations>
        <Query/>
      </Operations>
      <LatLongBoundingBox minx="32.84073385" miny="-118.33381483"
maxx="33.9347574" maxy="-115.52137107"/>
    </FeatureType>
  </FeatureTypeList>
  <ogc:Filter_Capabilities xmlns:ogc="http://www.opengis.net/ogc">
    <ogc:Spatial_Capabilities>
      <ogc:Spatial_Operators>
        <ogc:BBOX/>
      </ogc:Spatial_Operators>
    </ogc:Spatial_Capabilities>
    <ogc:Scalar_Capabilities>
      <ogc:Arithmetic_Operators>
        <ogc:Simple_Arithmetic/>
      </ogc:Arithmetic_Operators>
    </ogc:Scalar_Capabilities>
  </ogc:Filter_Capabilities>
</WFS_Capabilities>
```

15. A Sample -easy- WMS Web Services Service Definition file (WSDL)

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://services.wms.ogc.cgl"
xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://services.wms.ogc.cgl"
xmlns:intf="http://services.wms.ogc.cgl" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.2RC2
Built on Dec 08, 2004 (12:13:10 PST)-->
  <wsdl:message name="getFeatureInfoResponse">
    <wsdl:part name="getFeatureInfoReturn" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="getMapResponse">
    <wsdl:part name="getMapReturn" type="xsd:anyType"/>
  </wsdl:message>
  <wsdl:message name="getCapabilityResponse">
    <wsdl:part name="getCapabilityReturn" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="getMapRequest">
    <wsdl:part name="request" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="getFeatureInfoRequest">
    <wsdl:part name="request" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="getCapabilityRequest">
    <wsdl:part name="request" type="xsd:string"/>
  </wsdl:message>
  <wsdl:portType name="WMSServices">
    <wsdl:operation name="getMap" parameterOrder="request">
      <wsdl:input message="impl:getMapRequest" name="getMapRequest"/>
      <wsdl:output message="impl:getMapResponse" name="getMapResponse"/>
    </wsdl:operation>
    <wsdl:operation name="getCapability" parameterOrder="request">
      <wsdl:input message="impl:getCapabilityRequest" name="getCapabilityRequest"/>
      <wsdl:output message="impl:getCapabilityResponse" name="getCapabilityResponse"/>
    </wsdl:operation>
    <wsdl:operation name="getFeatureInfo" parameterOrder="request">
      <wsdl:input message="impl:getFeatureInfoRequest" name="getFeatureInfoRequest"/>
      <wsdl:output message="impl:getFeatureInfoResponse" name="getFeatureInfoResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="WMSServicesSoapBinding" type="impl:WMSServices">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
```

```
    <wsdl:operation name="getMap">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getMapRequest">

        <wsdlsoap:body                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://services.wms.ogc.cgl" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="getMapResponse">
        <wsdlsoap:body                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://services.wms.ogc.cgl" use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getCapability">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getCapabilityRequest">
        <wsdlsoap:body                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://services.wms.ogc.cgl" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="getCapabilityResponse">
        <wsdlsoap:body                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://services.wms.ogc.cgl" use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getFeatureInfo">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getFeatureInfoRequest">
        <wsdlsoap:body                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://services.wms.ogc.cgl" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="getFeatureInfoResponse">
        <wsdlsoap:body                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://services.wms.ogc.cgl" use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="WMSServicesService">
    <wsdl:port binding="impl:WMSServicesSoapBinding" name="WMSServices">
      <wsdlsoap:address location="http://localhost:8080/wmsstream/services/WMSServices"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

16. Sample GetFeature Response from WFS to WMS

```
<?xml version="1.0" encoding="UTF-8"?>
<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://complexity.ucs.indiana.edu/~gaydin/wfs
C:/Projects/WFS/xml/schemas/fault_new.xsd
http://complexity.ucs.indiana.edu/~gaydin/ogc/original/wfs/1.0.0/WFS-
basic.xsd">
 <gml:boundedBy>
   <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#27354">
     <gml:coordinates decimal="." cs="," ts=" ">>-119.31,35 -
118,38</gml:coordinates>
   </gml:Box>
 </gml:boundedBy>
 <gml:featureMember>
   <fault>
     <name>White Wolf</name>
     <segment>5.0</segment>
     <author>Rundle J. B.</author>
     <gml:lineStringProperty>
       <gml:LineString srsName="null">
         <gml:coordinates>-118.65,35.26 -118.56,35.31</gml:coordinates>

       </gml:LineString>
     </gml:lineStringProperty>
   </fault>
</gml:featureMember>
<gml:featureMember>
  <fault>
    <name>White Wolf</name>
    <segment>4.0</segment>
    <author>Rundle J. B.</author>
    <gml:lineStringProperty>
      <gml:LineString srsName="null">
        <gml:coordinates>-118.73,35.21 -118.65,35.26</gml:coordinates>
      </gml:LineString>
    </gml:lineStringProperty>
  </fault>
</gml:featureMember>
<gml:featureMember>
  <fault>
    <name>White Wolf</name>
    <segment>3.0</segment>
    <author>Rundle J. B.</author>
    <gml:lineStringProperty>
      <gml:LineString srsName="null">
        <gml:coordinates>-118.82,35.15 -118.73,35.21</gml:coordinates>
      </gml:LineString>
    </gml:lineStringProperty>
  </fault>
</gml:featureMember>
<gml:featureMember>
  <fault>
    <name>White Wolf</name>
    <segment>2.0</segment>
    <author>Rundle J. B.</author>
    <gml:lineStringProperty>
      <gml:LineString srsName="null">
        <gml:coordinates>-118.9,35.1 -118.82,35.15</gml:coordinates>
      </gml:LineString>
    </gml:lineStringProperty>
  </fault>
</gml:featureMember>
```

```
      <gml:lineStringProperty>
        <gml:LineString srsName="null">
          <gml:coordinates>-118.19,37.14 -118.17,37.05</gml:coordinates>
        </gml:LineString>
      </gml:lineStringProperty>
    </fault>
  </gml:featureMember>
  <gml:featureMember>
    <fault>
      <name>White Mountains</name>
      <segment>9.0</segment>
      <author>Rundle J. B.</author>
      <gml:lineStringProperty>
        <gml:LineString srsName="null">
          <gml:coordinates>-118.2,37.23 -118.19,37.14</gml:coordinates>
        </gml:LineString>
      </gml:lineStringProperty>
    </fault>
  </gml:featureMember>
  <gml:featureMember>
    <fault>
      <name>White Mountains</name>
      <segment>8.0</segment>
      <author>Rundle J. B.</author>
      <gml:lineStringProperty>
        <gml:LineString srsName="null">
          <gml:coordinates>-118.22,37.32 -118.2,37.23</gml:coordinates>
        </gml:LineString>
      </gml:lineStringProperty>
    </fault>
  </gml:featureMember>
  <gml:featureMember>
    <fault>
      <name>White Mountains</name>
      <segment>7.0</segment>
      <author>Rundle J. B.</author>
      <gml:lineStringProperty>
        <gml:LineString srsName="null">
          <gml:coordinates>-118.24,37.41 -118.22,37.32</gml:coordinates>
        </gml:LineString>
      </gml:lineStringProperty>
    </fault>
  </gml:featureMember>
</wfs:FeatureCollection>
```