High Performance Federated Service-Oriented 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)

Abstract

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.

Geospatial information is critical to the effective and collaborative decision making in earth-related disaster planning, crisis management and early-warning systems. The decision making in GIS increasingly relies on analyses of spatial data in map-based formats. Maps are complex structures composed of layers created from distributed heterogeneous data and computation resources belonging to the separate virtual organizations from different disciplines and skill/expert levels.

We propose infrastructure for understanding and managing the production of knowledge from distributed observation, simulation and analysis through integrated data-views in the form of multi-layered map images. Infrastructure is based on common data model, standard GIS Web-Service components and a federator. Federator federates GIS services and enables unified data access/query, display and analysis over integrated data-views. .

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Geospatial information is critical to the effective and collaborative decision making in earth-related disaster planning, crisis management and early-warning systems. The decision making in Geographic Information Systems (GIS) increasingly rely on analyses of spatial data in map-based formats. Maps are complex structures composed of layers created from distributed heterogeneous data and computation resources belonging to the separate virtual organizations from various expert skill levels.

We propose a Service-oriented architecture for understanding and managing the production of knowledge from the distributed observation, simulation and analysis through integrated data-views in the form of multi-layered map images. Infrastructure is based on common data model, standard GIS Web-Service components and a federator. Federator federates GIS services and enables unified data access/query and display/analysis over integrated data-views through event-based interactive display tools.

Integrated data-views are defined in the federator's capability metadata as composition of layers provided by standard GIS Web-Services. Our grid approach is based on the WS-I+ interoperability standards.

1.1. Motivation

Geographic Information Systems (GIS) are systems for creating, storing, sharing, analyzing, manipulating and displaying spatial data and associated attributes.

The general purpose of GIS is extracting information/knowledge from the raw geo-data. The raw-data is collected from sensors, satellites or other sources and stored in databases or file systems. The data goes through the filtering and rendering services and presented to the end-users in human recognizable formats such as images, graphs, charts etc. GIS are used in a wide variety of tasks such as urban planning, resource management, emergency response planning in case of disasters, crisis management and rapid responses etc.

Over the past decade, GIS have evolved from the traditional centralized mainframe systems to desktop systems to modern collaborative distributed systems. Centralized systems provide an environment for stand-alone applications in which data sources, rendering and processing services are all tightly coupled and application specific. Therefore, they are not capable of allowing seamless interaction with the other data or processing/rendering services. On the other hand, the distributed systems are composed of geographically distributed and loosely coupled autonomous hosts that are connected through a computer network. They aim to share data and computation resources collaborating on large scale applications.

Modern collaborative GIS require data and computation resources from distributed virtual organizations to be composed based on application requirements, and accessed and queried from a single uniform access point over the refined data with interactive display tools. This requires seamless integration and interaction of data and computation resources. The resources span over organizational disciplinary and technical boundaries and use different client-server models, data archiving systems and heterogeneous message transfer protocols.

Furthermore, GIS particularly used in emergency early-warning systems like homeland security and natural disasters (earthquake, flood etc) and crisis management applications require quick responses. However, because of the characteristics of geo-data (large sized and un-evenly distributed such as distribution of human population and earthquake seismic data), time-consuming rendering processes and limited network bandwidth, the performance and responsiveness stand as the toughest challenges in distributed modern GIS [Peng].

These motivated us to research on a federated Service-oriented responsive Geographic Information System framework enabling sharing and integration of heterogeneous data and computation resources for the collaborative decision support applications requiring quick response times.

1.2. Why Federated Service Oriented Design

The proposed federated Service-oriented information system framework supports collaborative decision making over integrated data views, described in layer-structured hierarchical data provided by a federator. The users access the system as though all the data and functions come from one site. The data distribution and connection paths stay

hidden and formulated as hierarchical data defined in federator's capability metadata. The users access the system through integrated data-views (maps) with the event-based interactive mapping display tools. Tools create abstract queries from the users' actions through action listeners and communicate with the system through federator.

Federation is based on federating service-oriented standard GIS Web Services capabilities metadata and their standard service interfaces about data access/query and rendering. Creating such a federated design has some advantages in data sharing, reliability, and system growth (interoperability and extensibility).

Capability is a metadata about the data and services together. It includes information about the data and corresponding operations with the attribute-based constraints and acceptable request/response formats. Compared to Web Service Description Files (WSDL), Web Services provide key low level capability but do not define an information or data architecture. These are left to domain specific capabilities metadata and data description language (GML). Capability also provides machine and human readable information that makes integration and federation of data/information easy. It also enables developing application based standard interactive re-usable client tools for data access/query and display.

1.2.1. Architectural Design Features

Federated Service-oriented GIS framework is composed of two parts. One part is consists of interoperable Service-oriented GIS components compliant with Open Geographic Standards, and other part is federator composing those components according to the application requirements by providing integrated data-views in its capability metadata.

Regarding the first part, service descriptions, standards service APIs and capability definitions are defined in standard specifications published by Open Geographic Standards. According to standards we developed two types of services. These are map-data rendering services (Web Map Services (WMS)) and data providing services (Web Feature Services (WFS)). In the system there are two types of data, vector data provided by WFS in the form of XML-encoded common data model (GML) and binary map-images provided by WMS. GML is a data description language which is XML encoding of the heterogeneous data. It consists of two parts, content (core data) and presentation (attribute and geometry elements). The common data model's properties enable data to be displayed, queried and easily integrated.

In summary for the first part, we basically developed a GIS framework according to the standard specifications and then, enhanced it with Web Service-based Service Oriented Architecture (SOA) principles through WS-I standards. The standard GIS Web Services' (WMS and WFS) definitions are also extended with the streaming data transfer capability by introducing topic-based publish/subscribe message oriented middleware into the system. The system uses Web Service interface as negotiation protocol and data transfers are done through publishers and subscribers on the same topic agreed on by negotiation process.

The second part is the federator which federates the standard GIS Web Services' capabilities metadata and presents a single database image to the user through application-based hierarchical data defined in its federated capability metadata. This enables unified data access/query/display from a single access point through abstract queries from event-based interactive map tools (or even from the console or command-

lines). Event-based interactive map tools are generic tools enabling seamless interaction with the system through federator or any other compatible WMS.

Application-based hierarchical data is defined as integrated data-view in the federator's capability metadata. It actually defines a static workflow starting from the federator and ending at the original data sources (WFS serving GML or WMS serving map layers). The services are linked through the references defined in their capability metadata. The user's interaction with the system is carried over the integrated data views through event-based interactive map tools. Integrated data views are defined in the hierarchical data format as explained below:

Application ->Map -> Layer -> Data {GML and/or binary map images} ->Raw data (any type).

Map is application-based human recognizable integrated data display, and composed of layers. A layer is a data rendering of a single homogeneous data source. Layers are created from the structured XML-encoded common data model (GML) or binary map images (raster data). Heterogeneous data source are integrated to the system through the mediators in the form of GML (WFS-based mediation) or binary map images (WMS-based mediation). The mediators have resource specific adaptors for request and response conversions and appropriate capability metadata describing the data and resources.

Our experiences with GIS showed that federated Service-oriented GIS-style information model can be generalized to many application areas such Chemistry and Astronomy. We call this generalized framework Application Specific Information System (ASIS) and give blueprint architecture in terms of principles and requirements.

Developing such a framework requires first defining a core language (such as GML) expressing the primitives of the domain, second, key service components, service interfaces and message formats defining services interactions, and third, the capability file requirements (based on core-language) enabling inter-service communications to link the services for the federation.

1.2.2. High-performance Design Features

The high-performance design issues addressed in the proposed framework can be grouped into two. First group is regarding the extension of service descriptions and specifications of Open Geographic Standard specifications, and second group is regarding the federated design.

The first group of design issues is related to the extension and enhancements over Open Geographic Standards (OGC). We extended OGC'S online service descriptions with the streaming data transfer capabilities and called them streaming GIS Web Services. These services are the main building blocks of the federated Service-oriented GIS framework. At the service API level they provide standard functionalities and interfaces according to standards but the data payloads are transferred with the topic and publish/subscribe based messaging middleware. Each service has a publisher module and a subscriber module enabling the streaming transfer. Web Service interfaces are used for negotiation enabling client (subscriber) and server (publisher) to agree on the IP-port and topic of the broker through which data will be streamed.

The second group of design features is regarding the federator. The federation framework inspired us developing some performance-oriented designs such as *pre-fetching*, *caching* and *parallel processing enabling* quick responses. Since the geo-data is

in massive sizes, un-evenly (geo-location) distributed and variable sized, application of these techniques requires us to introduce novel approaches. These design approaches are addressed in the following paragraphs.

We introduced one-time session-based caching. In this approach, instead of caching whole data the federator caches only the recently fetched data and uses it for serving the successive requests coming from the same session. At the end of serving every-other request, federator replaces the cached data with the recently fetched data. The technique is based on sessionID transfer through the SOAP messages. This design enables application to run on any ordinary servers not having large storage capacity as in Google Servers or any other high-performance computing servers.

Parallel processing approach is developed on the proposed caching approach mentioned earlier. It is based on decomposition of un-cached data region and assigning the partitioned regions to the separate threads. Each thread runs in parallel and creates small maps correspond to their assigned partitions. The number of partitions is defined by locality information obtained by utilizing session-based caching.

Another performance-oriented design feature is the pre-fetching. It is mutually exclusive of the other design features caching and parallel processing which are applied together. Pre-fetching is explained as fetching the data before it is needed. It basically reduces drawback affects of the network bandwidth problem and repeated time/resource cosuming query/response conversions overheads spent at WFSs. The pre-fetching is done by federator for the performance critical data defined in the capability metadata. The pre-fetched data is kept in federator's local file system, and the requests are served from this intermediary storage.

We test the responsiveness of the system by applying these performance enhancing designs, and analyze the results by comparing the baseline tests results obtained by using naïve conventional GIS approaches (no-caching, single-threaded on demand data access/query). Our aim is turning compliance requirements into competitiveness and providing high-performance responsive geographic information systems under the interoperability and extensibility requirements.

1.3. Summary of Contributions

We developed a framework for federated Service-oriented Geographic Information Systems. Under this title, we addressed interoperability issues by integrating Web Services with Open Geographic Standards for supporting interoperability at both data, service and application levels. We also enhanced the standard GIS Web Service components with the streaming data-transfer capability by using publish/subscribe based messaging middleware architecture.

On top of the proposed service-oriented GIS data-grid, we introduced a federator enabling unified data access/query/display from a single access point through integrated data-views as a composition of distributed heterogeneous GIS data sources. Under this title, we introduced hierarchical data for integrated data-views defined in federator's capability metadata.

We have also investigated performance efficient designs regarding federator, data transfer and rendering, and done detailed benchmarking over real GIS application requiring quick response times. We compared and contrast the proposed streaming GIS Web Services with non-streaming counterparts. We have developed pre-fetching

algorithm for archived data access and display, and compared it with on-demand fetching.

We have introduced novel load balancing and parallel processing technique with attribute-based query partitioning for unevenly distributed variable-sized data processing/rendering. In order to do that we have proposed set of techniques such as mapping browser-based sessions to Web Services and introducing session based caching carried out between federator and event-based interactive mapping tools. We have also introduced the techniques for workload forecasting by using session-based caching. This enabled us to find out the best number of partitions for the parallel processing giving the best performance results.

Finally, we have defined the principles for generalizing federated GIS to the general science domains as Application Specific Information Systems (ASIS).

Regarding the system software contribution, we have developed streaming and non-streaming versions of Web-based GIS Map Server (WMS) with Open Geographic Standards and in Web Service principles. We also developed a federator supporting high-performance designs such as pre-fetching and parallel processing with client/session-based caching. We have also developed generic browser/event-based interactive map tools for data access, query and display.

1.4. Research Issues

In order to develop federated Service-oriented GIS framework supporting event-based unified data access/query/display from a single access point we address the following research issues.

Interoperability and extensibility: We first investigate the adoption of Open Geographic Standards to GIS to create an *interoperable geographic* information system with standard data models, service description and service API, and service capabilities metadata. Second, we apply Web-Service principles to develop Service-oriented Architecture for GIS data-grid.

Federation: Another research issue will be investigating a framework for capability file-based federation of GIS data-grid services enabling unified data access/query/display through event-based interactive tools over integrated data-views. Federation is done by capability metadata federation of proposed GIS Web Services by the federator.

We investigate how to make capability federation to develop application-based hierarchical data definitions in federated capability file. We first define GIS Web Services (extension to OGC standards) and their service API allowing inter-service communication through capability metadata exchange. We also investigated the standard event-based interactive query and display tools for the standard GIS data services enabling unified data access/query/display over integrated data-views.

We also investigate the principles for generalizing the proposed federated GIS system for general science domains such as Chemistry and Astronomy in terms of components and framework requirements..

Performance and Responsiveness: We analyzed/investigated the ways to turn the compliance requirements into competitiveness in Service-oriented federated Geographic Information Systems built on XML-encoded common data models. Interoperability requirements bring up some compliance costs. These are resulted from using OGC

defined XML-encoded common data model (GML) and GIS Web Services' XML-based communication protocol (Simple Object Access Protocol (SOAP) over HTTP for message exchange).

We first investigated the performance efficient designs for XML structured data transfer and processing (parsing and rendering). Second, we research on federator oriented design features to support high-performance for Geographic Information Systems requiring quick response times. Compos-ability nature of the standard GIS data services (WMS and WFS) inspired us developing a federated information system framework enabling enhancement with novel caching, load-balancing and parallel processing techniques.

We identify the following research questions in the scope of this thesis:

- How to develop federated GIS data Grid architecture enabling fine-grained dynamic information presentation?

- How can we incorporate widely accepted Open Geographic Standards (OGC) with Web Services standards?

- How to make streaming data transfer between Open Standards GIS data services through messaging middleware?

- Can we build GIS like Federated Service-oriented Information System for any other science domains? What are the requirements, constraints and limitations?

- How to merge Asynchronous Java Script and XML (AJAX) with Web Services client stubs for event and browser-based interactive map client tools?

- Can we build responsive Service-oriented Geographic Information System requiring data access and rendering in which data is provided by autonomous resources in XML-encoded common data model (GML)?

  - How to access stateless Open Standard's GIS data services in a state-full manner?
  - How to make load balancing in federated GIS system?
  - How do you apply locality principles on un-predictable workload sharing?

1.5. Organization of Dissertation

The first chapter consists of an overview of the Geographic Information Systems, architectural and high-performance design features of the federated service-oriented GIS, summary of the outstanding issues that relate to the research outlined in this thesis, and discussion on the contribution of the thesis.

The remaining of the thesis is organized as follows. Chapter 2 consists of two parts. First part gives background information about Geographic Information Systems and Web Services based Service-oriented architectures. Second part reviews some of the related projects.

Chapter 3 explains the design principles and components of the information Grid architecture. Our system is an example of the Grids of Grids paradigm [Fox04] which consists of three major architectural components. Components are developed in

accordance with Open Geographic Standards and integrated with Web Service principle at both data and application level.

Chapter 4 investigates a fine-grained Service-oriented federation architecture built over the proposed GIS components given in Chapter 3. It enables unified data access/query and display over integrated data views. Last part studies the design principles/requirements of the proposed framework for the general science domains and gives blue-print architecture.

The proposed information system framework, Web Map Services and event-based interactive decision making tools have been used in several large-scale GIS projects. Chapter 5 discusses three of them.

Chapter 6 first introduces common performance issues in interoperable Service-oriented Geographic Information Systems and then, presents high performance design features and their evaluations. The performance issues mostly stem from using structured/annotated common data model and XML-encoded Web Service protocols. Corresponding high-performance design features concentrate on data rendering, handling/transferring and federator-oriented novel load balancing and caching techniques.

In Chapter 7, we give answers to the research questions identified in Chapter 1, outline future research directions and conclude the dissertation.

Chapter 2

Literature Survey

2.1. Background

This chapter gives the definitions and explanations of some key terms and concepts in the proposed GIS Grid domain. These are mostly related to GIS, geo-data, required service functionalities and components, interoperability and standardization issues involving Open Geographic Standards, structured common data models and Web Services implementation of Service-oriented (SOA) distributed GIS.

2.1.1. Geographic 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 [Foster04, Berman03].

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 [Koontz].

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.

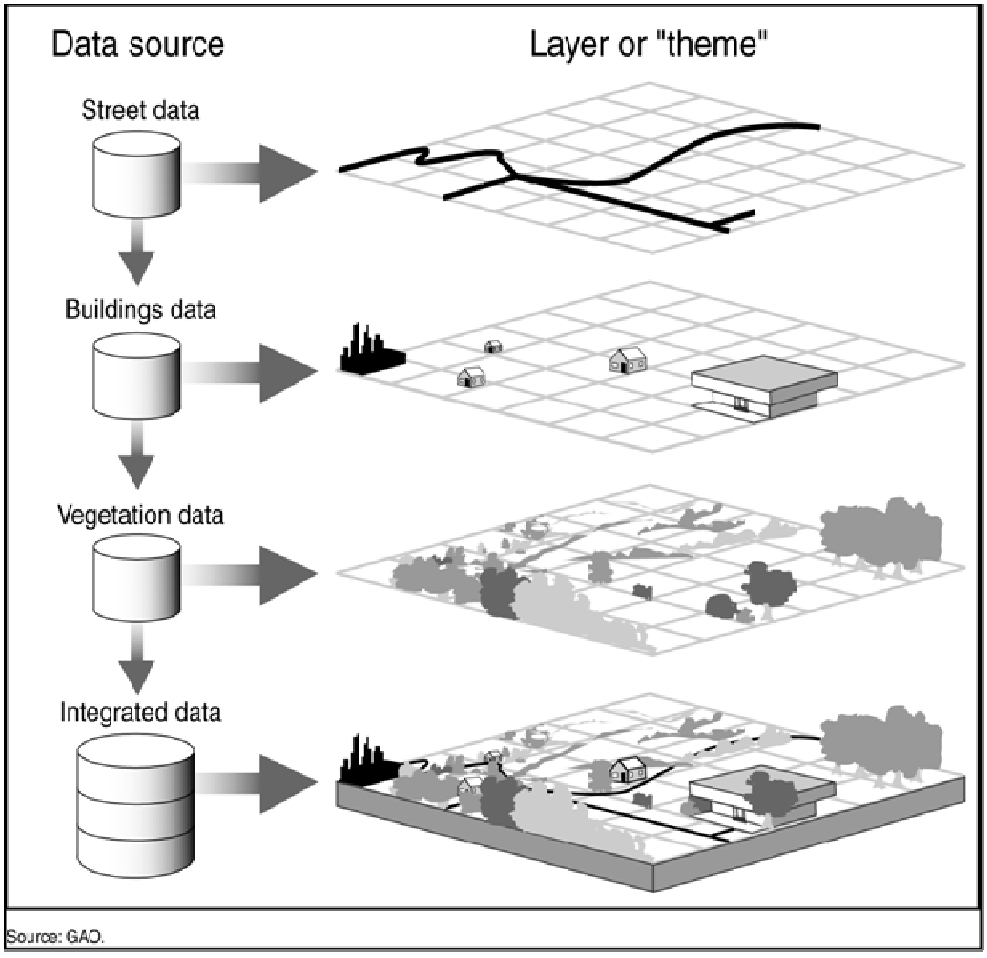

Figure 1: Layered display – a map is composed of distributed multiple set of layers. Figure is from [Koontz].

In Figure 1, each layer composing map (or integrated data/information) is an abstraction of different type of vector and raster data (see Chapter 3 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.

2.1.2. 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." [Kreger].

Web Service standards [Booth04] 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 [Foster04, Aydin].

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) [Christensen]. 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) [Donbox] messages. WSDL enables Web Services to be located and invoked remotely through registry and catalog services. Universal Description, Discovery and Integration (UDDI) specification [Belwood] can be used by the service providers to advertise the existence of their services.

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) [Redmond] or Remote Method Invocation (RMI) [Rmi] or Internet Inter-Orb Protocol (IIOP) [Kirtland].

2.1.3. Open 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) [WFS] and the Web Map Service (WMS) [WMS, Kris03]. Geography Markup Language (GML) [GML] is widely accepted as the universal encoding for geo-referenced data (see Chapter 3 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 [Peng03].

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 [Peng03]. They have been working closely to align their work to produce compatible standards.

2.2. Related Works

2.2.1. Linked Environments for Atmospheric Discovery (LEAD)

Linked Environments for Atmospheric Discovery (LEAD) is a large scale project funded by NSF Large Information Technology Research grant for addressing fundamental IT and meteorology research challenges to create an integrated framework for analyzing and predicting the atmosphere. The proposed framework helps researchers to identify and access, prepare, manage, analyze or visualize a broad array of meteorological data and model output independent of format and physical location [Kelvin04].

For adaptive utilization of distributed resources, sensors and workflows LEAD is developing the middleware. The LEAD system is constructed as a service-oriented architecture and decomposes into services which communicate via well-defined interfaces and protocols [Plale06].

LEAD provides the scientists with necessary tools to build forecast models using available observations or model generated data and manages necessary resources for executing the model. The tools include supercomputer resources, automated search, selection and transfer of required data products between computing resources [Beth06]. One major feature of LEAD is support for adaptive analysis and prediction of mesoscale meteorological events. To provide such features LEAD data subsystem supports three important capabilities: 1 - automated data discovery by replacing the manual data management tasks with automated ones, 2 - a highly scalable data archiving system which allows transfer of large scale data products between resources, metadata descriptions of the available information and protected storage facilities, 3 – easy search and access interfaces for the data via a search GUI and underlying ontology [Beth06].

2.2.2. Geosciences Network (GEON)

The Geosciences Network (GEON) [Zaslavsky04] is a multi-university project funded by the National Science Foundation to develop cyber infrastructure to enable sharing of data sets and services in a distributed environment for the Earth Sciences. The GEON Grid is a distributed network of GEON nodes, each of which runs a GEON software stack that includes Web and Grid services to enable users to register data sets; register services; issue queries across multiple information sources, using spatiotemporal search conditions and ontologies; download data into personal spaces; invoke analysis

services; and visualize output of queries and/or analysis. The architecture includes data mediation services, workflow services, and a portal. Much of the data is geospatial and spatiotemporal in nature and provides appropriate search interfaces, and efficient mapping interfaces for such data is an important requirement. The GEON Grid software stack will include ArcIMS [EsriArcIms] as one of its components to provide GIS and mapping functionality.

Geopsciences Network (GEON) provides ontology enabled applications mostly based on data registration, discovery, manipulation and display in the GIS domain [Bhatia00]. They also have myGEON concept functioning similarly as in the LEAD, and they have data display tools in a portal implemented by GridSphere [Novotny04].

GEON is based on a "service-oriented architecture (SOA)". Advanced information technologies are being developed in the project to support "intelligent" searching, semantic integration, and visualization of multidisciplinary information spaces as well as 4D scientific datasets and geospatial data, and to provide access to high performance computing platforms, for data analysis and model execution. The GEON Portal also provides a Web-based interface to access the various resources.

2.2.3. Laboratory for Advanced Information Technology and Standards *(LAITS):*

The LAITS [Laits] is a project of Center for Spatial Information Science and Systems (CSISS) in George Mason University. The LAITS project is primarily working on integrating OGC Web Services with Globus-based Grid technology [Foster97] for geospatial modeling and applications. The objectives of the project are enabling the management of geospatial data by Grids, providing OGC standard compliant access to

Grid-managed geospatial data, and enabling geospatial modeling and the production of virtual geospatial products in the Grid environment [Liping03]. For the test and demonstration of their architecture, they use NASA EOS data environment and coverages data provided by OGC WCS (Web Coverage Service) [Doyle]. Their aim for the complete architecture is using OGC WCS, WMS and WFS services in the Grid environment. Currently they have WCS services to demonstrate their work.

They also have a demo to access GIS data kept in the form of coverages in different DBs connected to different WCS. These OGC compatible WCS are implemented and wrapped as Grid services and called as GWCS (Grid Web Coverage Services). LAITS enhanced the WCS to process 4-D HDF-EOS data which is from LLNL (Lawrence Livermore National Laboratory) netCDF (network Common Data Format) [Rew1990] modeling data. In their proposed illustrated architecture data providers are deployed as WCS in NASA Ames, in LLNL and in LAITS servers. In their GCSW (Grid Catalog Services for Web) they store and serve information about the active coverage servers. They use OGC CSW (Catalog Services for Web) services to search for specified data server (in their applications data is coverage and provided by WCS). Data transfer is achieved by using GridFTP [Allock03].

The brain of the system is iGSM (Intelligent Grid Service mediator). iGSM dispatches user requests from WCS/WMS portal to the most appropriate GWCS/GWMS in the Virtual Organization. Portals tasks are implemented at iGSM [Chu06]. Portals instances and data-service providers meet at the iGSM. iGSM also does request conversion. Geospatial-data access requests from OGC WCS portal are transferred to an

appropriate format for the Grid enabled WCS (GWCS). Catalog Service search is also done in iGSM. It is basically the brain of the system.

Regarding workflow or process pipelining, LAITS use its management and execution engine called BPELPower. It supports BPEL-based web service chain completely.

LAITS's grid approach is based on Globus toolkit. On the other hand, our Grid approach is based on WS-I+ interoperability standards and Web Service principles. The implementation of SOA in the web environment is called Web services and in the Grid environment the open Grid Services. Currently the web service and grid service are converged with the introduction of Web Service Resource Framework (WSRF) [Wsrf].

Chapter 3

GIS Web Service Data-Grid Components

A Geographic Information System (GIS) is 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.

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.

Proposed Information System Grid framework is based on Common data models, GIS Web Service components and Service-oriented architecture implemented with WS-I

Web Service principles. In this chapter we first present the requirements for the common data models and their advantages of usage in such a framework (Chapter 3.1). Next, we present needs and advantages of extending/enhancing components as Web Services to develop a SOA framework for GIS (Chapter 3.2). Finally, we present system's general architectural features in terms of its components, interactions and data-flow from the archived data stores to the end users (Chapter 3.3)

3.1. Geo-data and Common Data Models

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) [Lansing02].

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 H).

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 (see Appendix H) 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 [Sayar06]
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) [Cml], Astronomy domain uses VOTable (Virtual Observatory Tables) [Votable] and Mathematic science domain uses MathML (Mathematic Markup Language) [Mathml].

3.2. Web Service Extensions to Standard Service definitions

The proposed GIS framework is Service-oriented and has components as Web Services providing standard service interfaces and communicating with common messages formats defined in standard specifications. By integrating Web Services with Open Geographic Standards, we support interoperability at both data and application level and have the common advantages of SOA architectures listed below:

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.

OGC Web Feature Service implementation specification [Vretanos] defines HTTP as the only explicitly supported distributed computing platform which requires use of one of the two request methods: GET and POST. Although SOAP messages are also supported, they are also required to be transported using HTTP POST method. However employing HTTP protocol and GET or POST introduces significant limitations for both producers and consumers of a service. As discussed above Web Services provide us with valuable capabilities such as providing standard interfaces to access various databases or remote resources, ability to launch and manage applications remotely, or control collaborative sessions etc. Developments in the Web Services and Grid areas provide us with significant technologies for exposing our resources to the outer world using relatively simple yet powerful interfaces and message formats. Furthermore sometimes we need to access several data sources and run several services and for solving complex problems. This is extremely difficult in HTTP services but rapidly developing workflow technologies for Web and Grid Services may help us orchestrate several services. For

these reasons we have based our implementation of standard GIS services on Web Services principals.

Moreover, complex scientific applications require access to various data sources and run several services consecutively or at the same time. Since this is not in the scope of HTTP but can be supported using rapidly developing workflow technologies for Web and Grid Services, we have based our implementations on Web Services principals. Our goal is to make seamless coupling of GIS Data sources with other applications possible in a Grid environment.

GIS systems are supposed to provide data access tools to the users as well as manipulation tools to the administrators. In principle the process of serving data in a particular format is pretty simple when it is made accessible as files on an HTTP or FTP server. But additional features like query capabilities on data or real-time access in a streaming fashion require more complicated services. As the complexity of the services grows, the client's chance of easily accessing data products decreases, because every proprietary application developed for some type of data require its own specialized clients. Web Services help us overcome this difficulty by providing standard interfaces to the tools or applications we develop.

No matter how complex the application itself, its WSDL interface will have standard elements and attributes, and the clients using this interface can easily generate methods for invoking the service and receiving the results. This method allows providers to make their applications available to others in a standard way.

Most scientific applications that couple high performance computing, simulation or visualization codes with databases or real-time data sources require more than mere remote procedure call message patterns. These applications are sometimes composite systems where some of the components require output from others and they are asynchronous, it may take hours or days to complete. Such properties require additional layers of control and capabilities from Web Services which introduces the necessity for a messaging substrate that can provide these extra features.

3.3. System Framework and Web-Service Components

The proposed geographic information system is based on common data models provided by common standard service components and their service interfaces. Service interactions start with a discovery step which involves retrieving the capabilities document. Capability document is an XML encoded metadata file about both the service and data. Its formats and schema are defined by Open Geographic Standards (OGC specifications). Sample capabilities documents are given in Appendix-D for WFS and Appendix-C for WMS. All the interactions and service bindings are done through capability exchange. So, each service keeps its own capability defining its data providing and available operations on these data. For the sample interaction steps between WMS and WFS to get feature data from WFS, see Chapter 3.3.1.

The proposed Service-oriented GIS is illustrated in Figure 2. It is composed of two major types of GIS Web Services (see Chapter 3.3.1). These are Web Map Services and Web Feature Services. Optionally, in order to find and bind services in Service-oriented architecture, system can also be extended with Catalog/registry services.

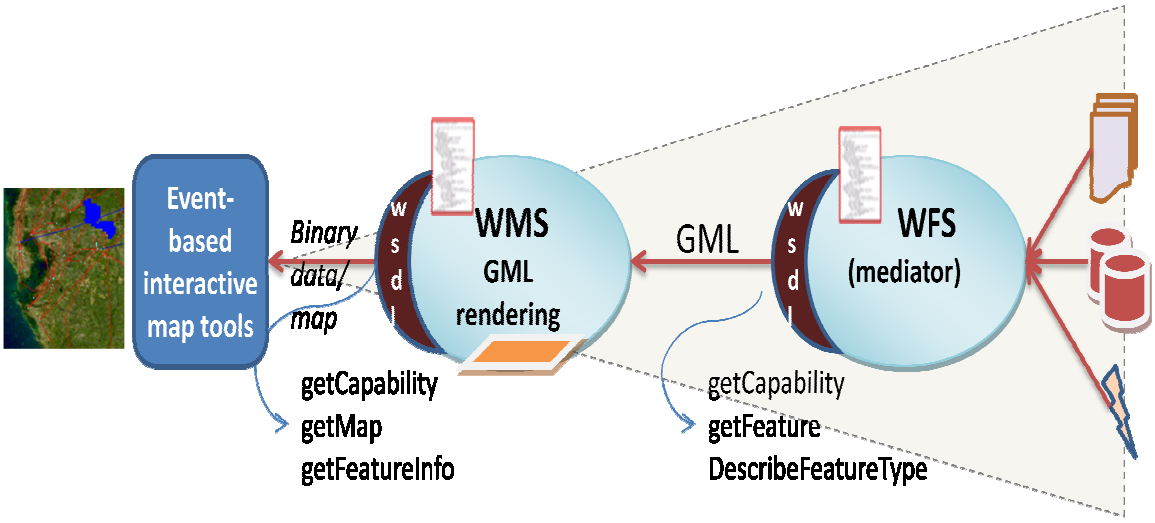

Figure 2: GIS framework with the proposed Web Service components and data flow. See also Figure 3.

In the system there are also two types of common data model. First one is provided by WFS in XML encoded GML data format and second one is provided by WMS in binary map images. For more detail about the common data models and their usage advantages see Chapter 3.

3.3.1. Web Feature Service

Web Feature Service is one of the major service standards defined by Open Geographic Standards (OGC) for creating a GIS framework. Web Feature Service implementation specification defines interfaces for data access and manipulation operations on geographic features using HTTP as the distributed computing platform. Via these interfaces, a web user or service can combine, use and manage geo-data from different sources by invoking several standard operations [Vretanos].

OGC specifications describe the state of a geographic feature by a set of properties where each property can be thought of as a [name, type, value] tuple.

Geographic features are those that may have at least one property that is geometry-valued. This also implies that features can be defined with no geometric properties at all. According to the Open Geographic Standard's definition WFS provide 3 operations, *getCapabilities*, *describeFeatureType* and *getFeature*. In case of transactional WFS it provides 2 more service interfaces, *transaction* and *lockFeature*. For the sake of our purposes, we just mention about the basic WFS and describe their standard operations as below [Vretanos]:

- *GetCapabilities*: A Web Feature Service 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*: A Web Feature Service must be able, upon request, to describe the structure of any feature type it can service.

- *GetFeature*: A Web Feature Service must be able to service a request to retrieve feature instances. In addition, the client should be able to specify which feature properties to fetch and should be able to constrain the query spatially and non-spatially.

Illustration of client-server interaction: WFS services' clients are mostly Web Map Services. Client's interaction with WFS usually starts with a discovery step which involves retrieving the capabilities document. A client usually first sends a *getCapabilities* request to the WFS server to learn which feature types are serviced and what operations are supported on each feature type in what constraints. Upon receiving the list of available feature data available with their specific properties (given in capability file of WFS), client sends a *describeFeatureType* request to get the structure information of the interested feature type. Finally, client makes getFeature request with

appropriate request created based on client's purpose and WFS server's capability metadata. However the most common queries used are GetFeature requests to retrieve particular features.

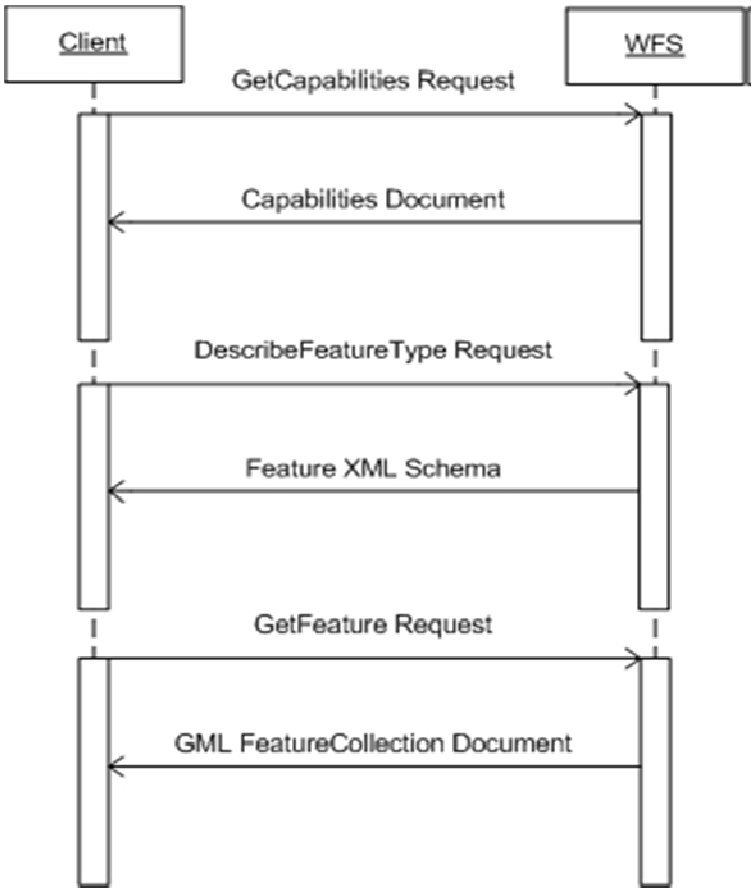

Figure 3: Illustration of client (WMS)-WFS interaction steps to get feature data.

Figure 3 illustrates three groups of coupled bars representing client and WFS interactions.

The first group of request/response at the top explains capability exchange between client and server. This is done with WFS's GetCapabilities service interface. The clients (Web Map Server or users) start with requesting a capabilities document from

WFS. When a GetCapabilities request arrives, the server may choose to dynamically create a capabilities document and returns this, or simply return a previously created XML document.

The second group of request/response in the middle explains requesting structured information (schema) about the interested feature data listed in capability metadata of WFS. This is achieved by using WFS's DescribeFeatureType service interface. After the client receives the capabilities document he/she can request a more detailed description for any of the features listed in the WFS capabilities document. Upon invocation of this service interface, WFS returns an XML Schema that describes the requested feature as the response.

The third group of request/response at the bottom of Figure 3 explains request for feature data based on user defined constraints in an appropriate request format. This is done through WFS's GetFeature service interface. After it is done with first two steps, the client may then ask WFS to return a particular portion of any feature data. GetFeature requests contain some property names of the feature and a Filter element to describe the query. The WFS extracts the query and bounding box from the filter and queries the related database(s) that holds the actual features. The results obtained from the DB query are converted to that particular feature's GML format and returned to the client as a FeatureCollection object.

WFS allow clients to access and manipulate the geographic features without having to consider the underlying data stores. Clients' only view of the data is through the WFS interface which allows the data providers to integrate various types of data stores with one WFS instance. Figure 2 displays this instances where the WFS server is

accessed by different types of clients and has access to various types of spatial databases, file systems and any-type of storages. Clients interact with WFS by submitting database queries encoded in OGC Filter Encoding Implementation [Vretanos01] and in compliance with the Common Query Language [Rao00]. The query results are returned as GML FeatureCollection documents. In this context, WFS also behave as mediator services to provide feature data in common data model (Geographic Markup Language) through standard service interfaces. For the technical details about implementing Web Service based WFS see Galip's thesis [Galipthesis].

3.3.2. Web Map Service

Web Map Service (WMS) [WMS, Kris03] is the key service to the information visualization in GIS domain. WMS produces maps from the geographic data in GML from WFS or binary data mostly from Coverage Portrayal Services (CPS) [Lansing02].

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

Web Map Service (WMS) enables visualizing, manipulating and analyzing geospatial data through maps displayed on browser based interactive GUI (see Chapter 3.3.3). 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 [OnEarth]. WMSs can be federated and cascaded to create more detailed and comprehensible map images. We explain this in Chapter 4.

WMS provides three main services (Appendix A); these are getCapabilities (Chapter 3.3.2.1), *getMap* (Chapter 3.3.2.2) and *GetFeatureInfo* (Chapter 3.3.2.3). GetCapabilities and *getMap* are required services to produce a map but *GetFeatureInfo* is an optional service. These are explained in the following chapters.

3.3.2.1. GetCapabilities Services

The purpose of the GetCapabilities operation is to obtain service metadata, which is a machine and human readable description of the server's information content and acceptable request parameter values. Figure 5 presets getCapabilities request schema.

WMS provide its data in the layer format. *GetCapabilities* request and corresponding service interface 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. The capability file is kept in the local file system and sent to clients upon getCapabilities request (see Figure 4). For the sample capabilities file instances see APENDICES C and D.

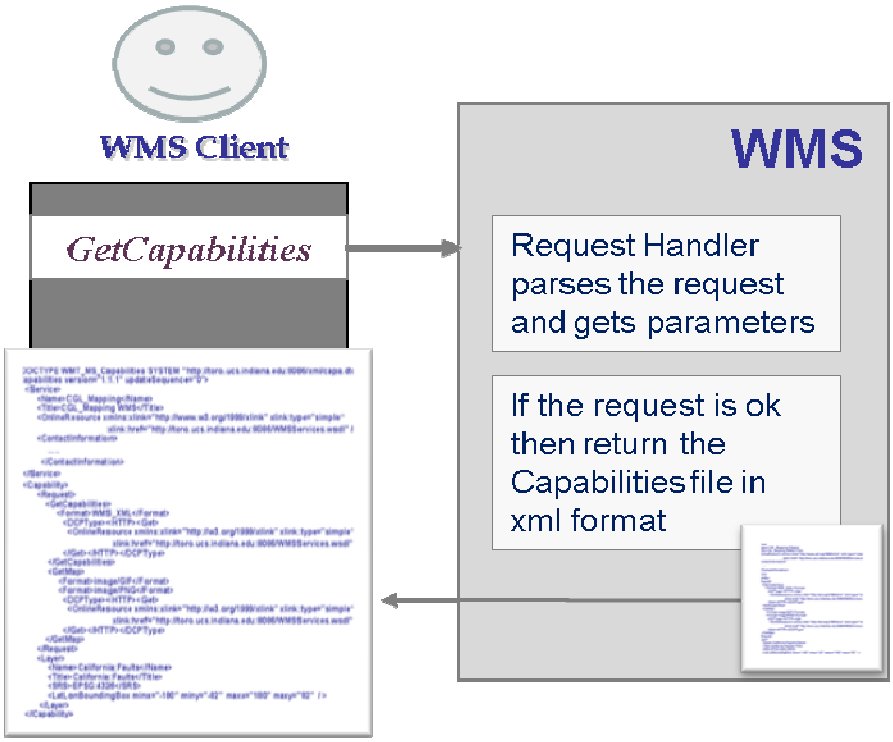

Figure 4: GetCapabilities operation steps. See Appendix C for a sample WMS capabilities file instance

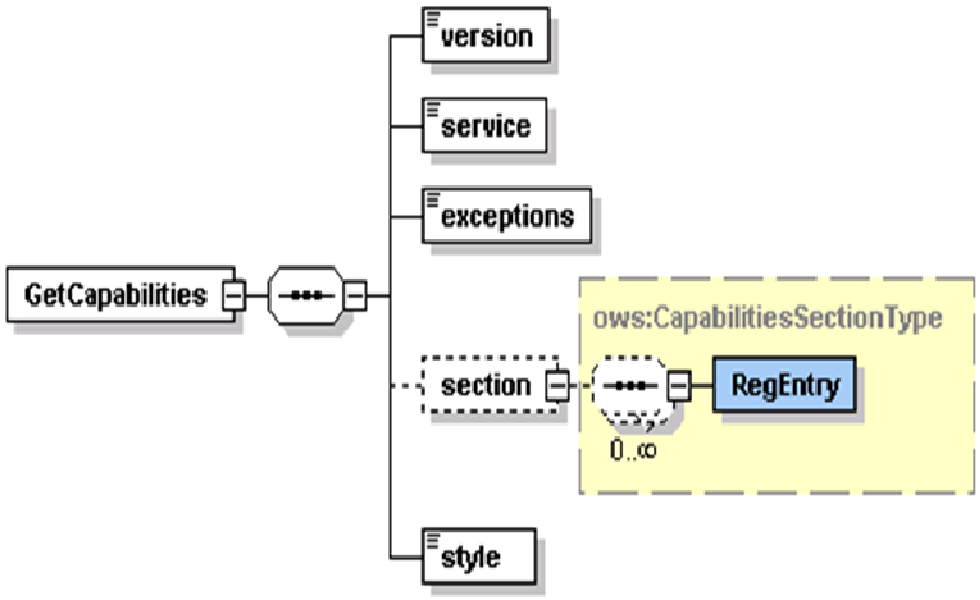

Figure 5: GetCapabilities Request Schema. See Appendix A for an instance of this request schema.

3.3.2.2. GetMap Services

The *getMap* service interface allows the retrieval of maps. Maps are provided in different various formats based on user-defined parameters and layer attributes. All the supported formats for map-image layers and corresponding layer specific attributes and constraints are defined in WMS Capabilities document. Before invoking *getMap* service interface, clients first obtain capabilities document by invoking getCapabilities service interfaces (see Chapter 3.3.2.1). The image is returned back to the WMS Client as an attachment to SOAP message. If the WMS encounters any problem during handling of the request, it sends an exception message in SOAP back to the WMS Client.

Major operation steps to produce maps are illustrated in Figure 6. *GetMap* request schema to create valid requests is given in Figure 7.

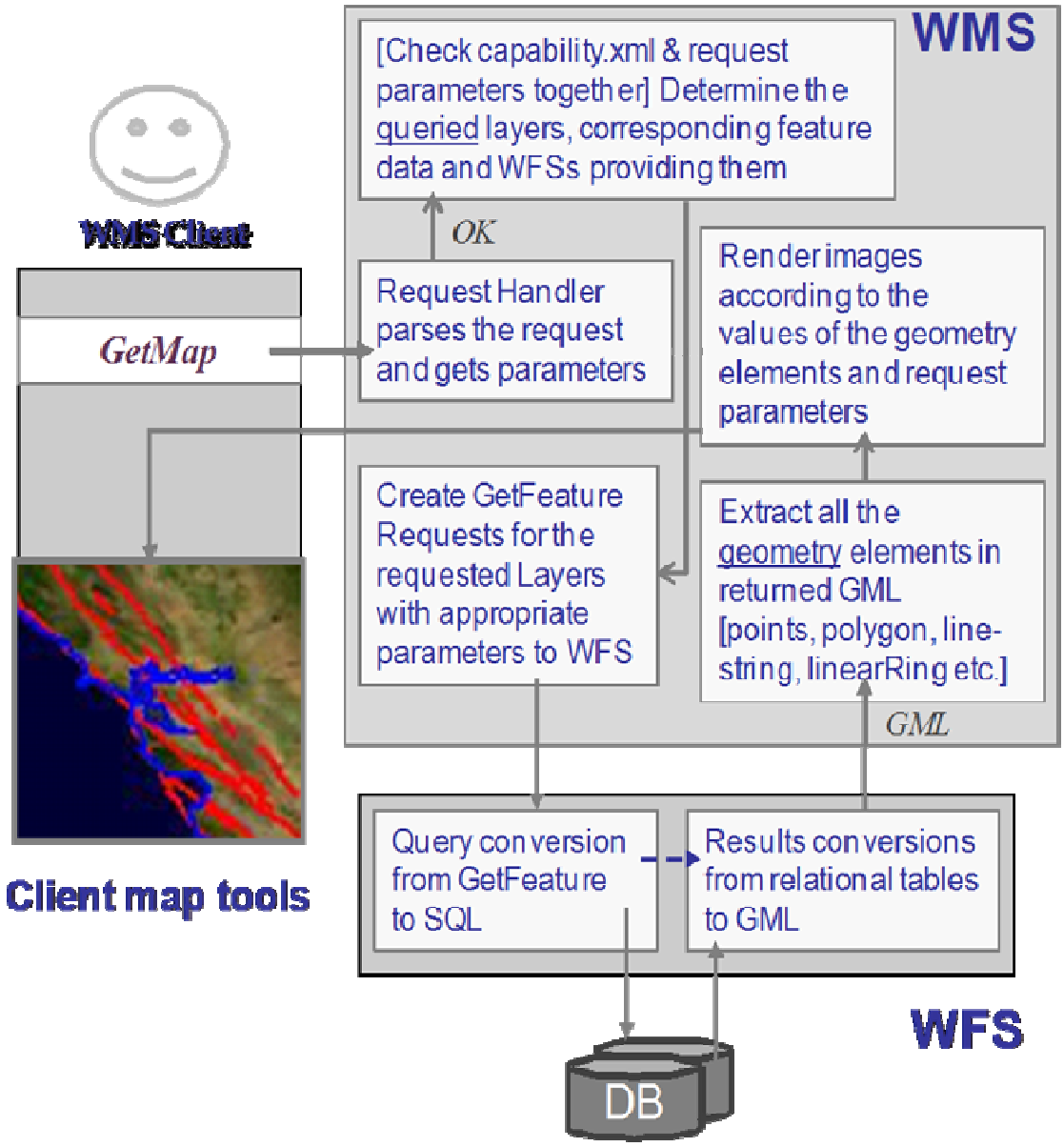

Figure 6: *GetMap* operation steps.

WMS first parses the request and gets the parameter values. WMS first determines what layers are requested, in which bounding box, in which form, and so forth. After determining all the request parameters, it communicates with WFS services providing requested feature data by using their getFeature service interfaces and requested feature data in GML format. If the parameter defining returned image format in

getMap request is Scalable Vector Graphics (SVG), then WMS creates SVG from returned feature data by using its geometry elements. If the requested image is not in SVG format, we first create the SVG image and then convert it into the desired image formats (such as PNG, GIF, or JPEG). Apache Batik provides libraries for this conversion. Batik is a Java(tm) technology based toolkit for applications or applets that use images in the SVG format for various purposes, such as viewing, generation or manipulation. By using these schema files we derive geometry elements from the GML file to visualize the feature data. These geometry elements in GML are basically Point, Polygon, LineString, LinearRing, MultiPoint, MultiPolygon, MultiGeometry, etc.

To create the images from the features returned from the WFS, we have used Java Graphics2D and Java AWT libraries. For each layer we create a different graphics object. If you assign each layer to different graphics object than Java libraries allow you to overlay these graphic objects in various combinations.

Alternatively, WMS uses SVG conversion to create map-image layers. When this way is used, WMS uses its internally defined XSL file to convert standard GML files into SVG by using XSLT machine. We developed standard XSL (see Figure 13) file to convert XML coded GML feature collections into SVG files. After having SVG, these image objects then converted into any image format such as JPEG, TIFF PNG etc. [Sayartech].

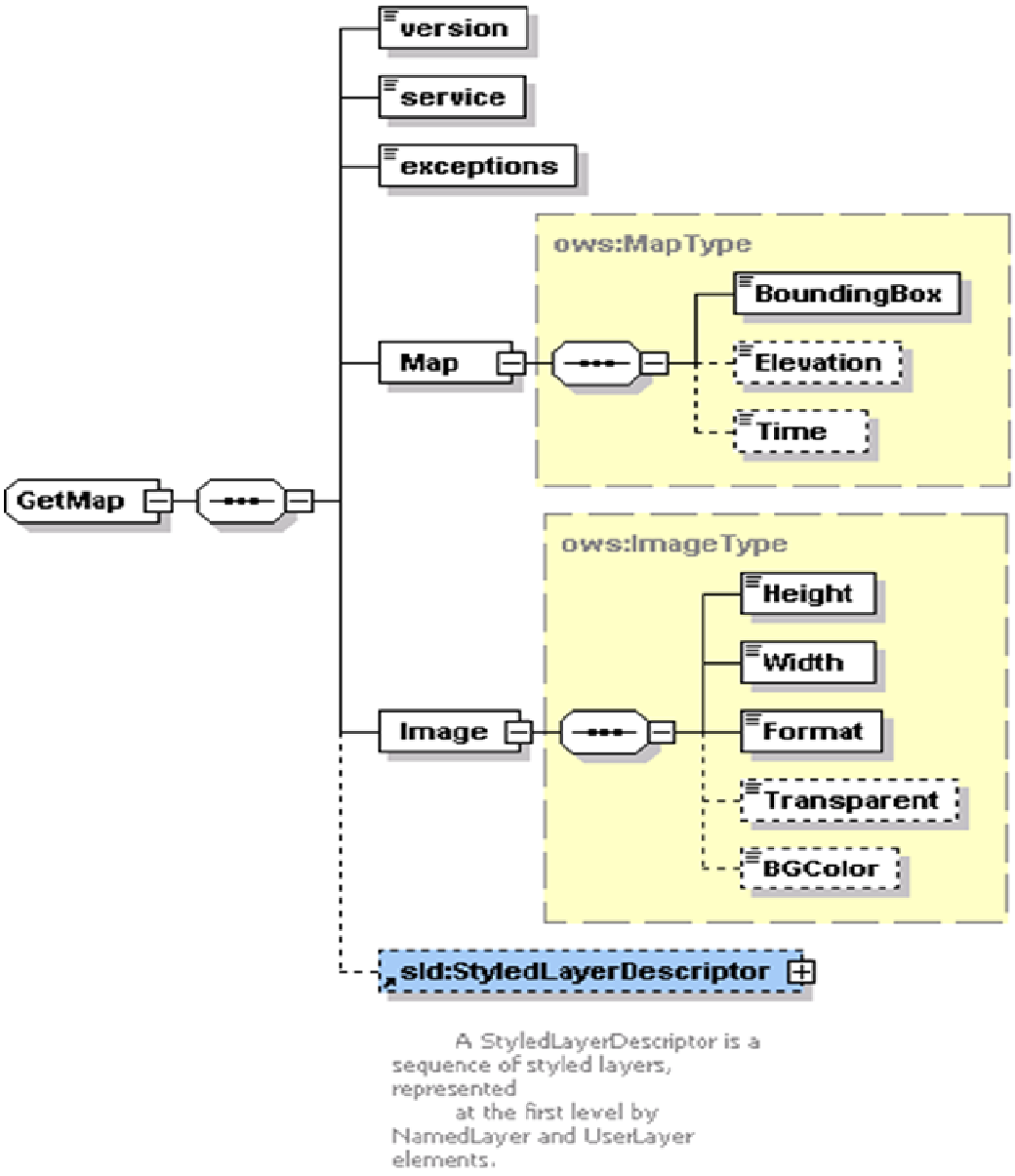

Figure 7 : ***GetMap*** **Request Schema. See Appendix A for an instance of this request schema**

Below you see the sample code fragment showing how to extract geometry elements from GML and overlay it on a raster map image as a separate layer. In this simple pseudo-like code, the 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 WFS. Using a layer from some other WMS is defined in OGC specifications and called as cascading. WMS behaving as a client to another WMS is called cascading WMS and layer used is called cascaded layer.

```
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);
 }
}
…
```

Check all the geometry elements in GML for a queried region of the map .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.

```
g.dispose();
```

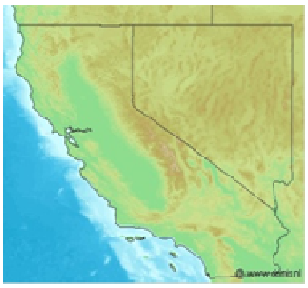

Figure 8: Sample output of the above map images generating code

How to send binary map images with SOAP messages:

1. *Server side:*

Below sample code shows how to attach a map image to SOAP message in response to *getMap* request. We assume map image name is maimage.jpeg. WMS server first creates a data handler from the image and cast it as an object, and return.

```
Object map = file2DataHandlerObject (APPLPATH+"/mapimage.jpeg");
public Object file2DataHandlerObject(String filePath) {
      try {
         DataHandler dhSource = new DataHandler(new
               FileDataSource(filePath));
         return (Object) dhSource;
      } catch (Exception ex) {
         ex.printStackTrace();
         return null;
      }
```

2. *Client side:*

Client has client stubs for WMS services created earlier from WMS's Web Service Description File (WSDL). It uses its client stubs and get the map as an attachment to SOAP message. It first extracts the attachment and then data handler from the attachment. It created map images as byte array through data handler.

```
java.lang.Object value = null;
value = binding.getMap(request);

byte[] bs = null;
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());

   bs = new byte[bis.available()];
   bis.read(bs, 0, bs.length);

   bis.close();
```

3.3.2.3. GetFeatureInfo Services

The *GetFeatureInfo* operation is designed to provide clients of a WMS with more information about features over the map images that were returned by previous Map requests. *GetFeatureInfo* is used when a user needs further information about any feature data on the map. Its return type is user readable text or HTML which is defined as request parameter. See Figure 7 for general schema for creation of *getFeatureInfo* query instances.

The GetFeatureInfo works as follows (see also Figure 11*):*

The user supplies an (x, y) Cartesian coordinates and the layers of interest and gets the information back in the form of HTML, GML or ASCII format.

The basic operation provides the ability for a client to specify which pixel is being asked about, which layer(s) should be investigated, and what format the information

should be returned in. Because the WMS protocol is stateless, the *GetFeatureInfo* request indicates to the WMS what map the user is viewing by including most of the original *GetMap* request parameters (all but VERSION and REQUEST). From the spatial context information (BBOX, CRS, WIDTH, HEIGHT) in that *GetMap* request, along with the x, y position the user chose, the WMS can (possibly) return additional information about that position. The actual semantics of how a WMS decides what to return more information about, or what exactly to return, are left up to the WMS provider.

Figure 11 illustrates the successive process steps done by the WMS to respond to *getFeatureInfo* requests. After checking the request parameters with the capability metadata, WMS creates appropriate getFeature queries to fetch the GML data from WFSs. After getting the feature collections data from the WFS, WMS extracts all the non-geometry elements and attributes in the returned GML files and create another text or HTML file based on request parameter and create the response to *getFeatureInfo* query in accordance with the return parameter defined by the client in the query. The parameter called “INFO_FORMAT”defines the return format whose possible values are plain text files, HTML and GML.

For the *getMap* request WMS extracts geometry elements from the returned GML file but for the *getFeatureInfo* it extracts non-geometry elements. From the list of non-geospatial elements, WMS creates a new XML file to be able to transform non-geometry elements into HTML. This XML file is simply another form of GML which includes just non-geometry elements, properties and attributes. After creating new XML file from the non-geo elements, WMS creates HTML file from newly created XML file by using generic XSL [xslurl] file and XSLT transformation machine. Figure 10 explains the

general architecture of creating a response from the GML file through generic XSL stylesheet file given in Figure 13.

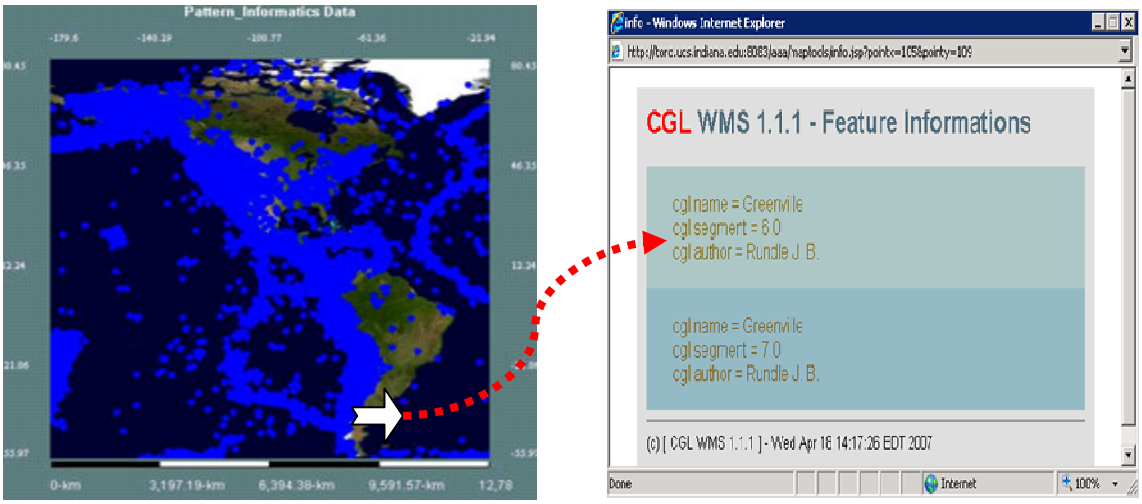

Figure 9: A snapshot of response to *getFeatureInfo*. It is actually an attribute querying of earthquake seismic data layer shown on the map image.

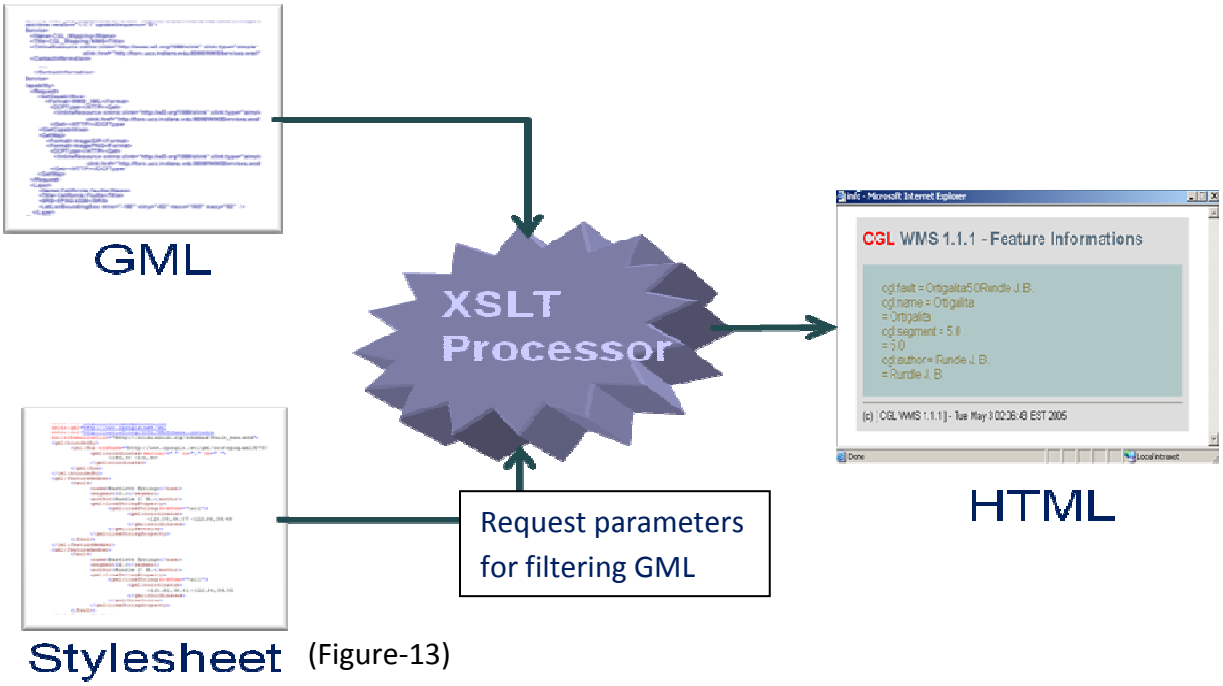

Figure 10: Creating *getFeatureInfo* reponse by using a stylesheet and XSLT processor. See Figure 10 for generic stylesheet for GML.

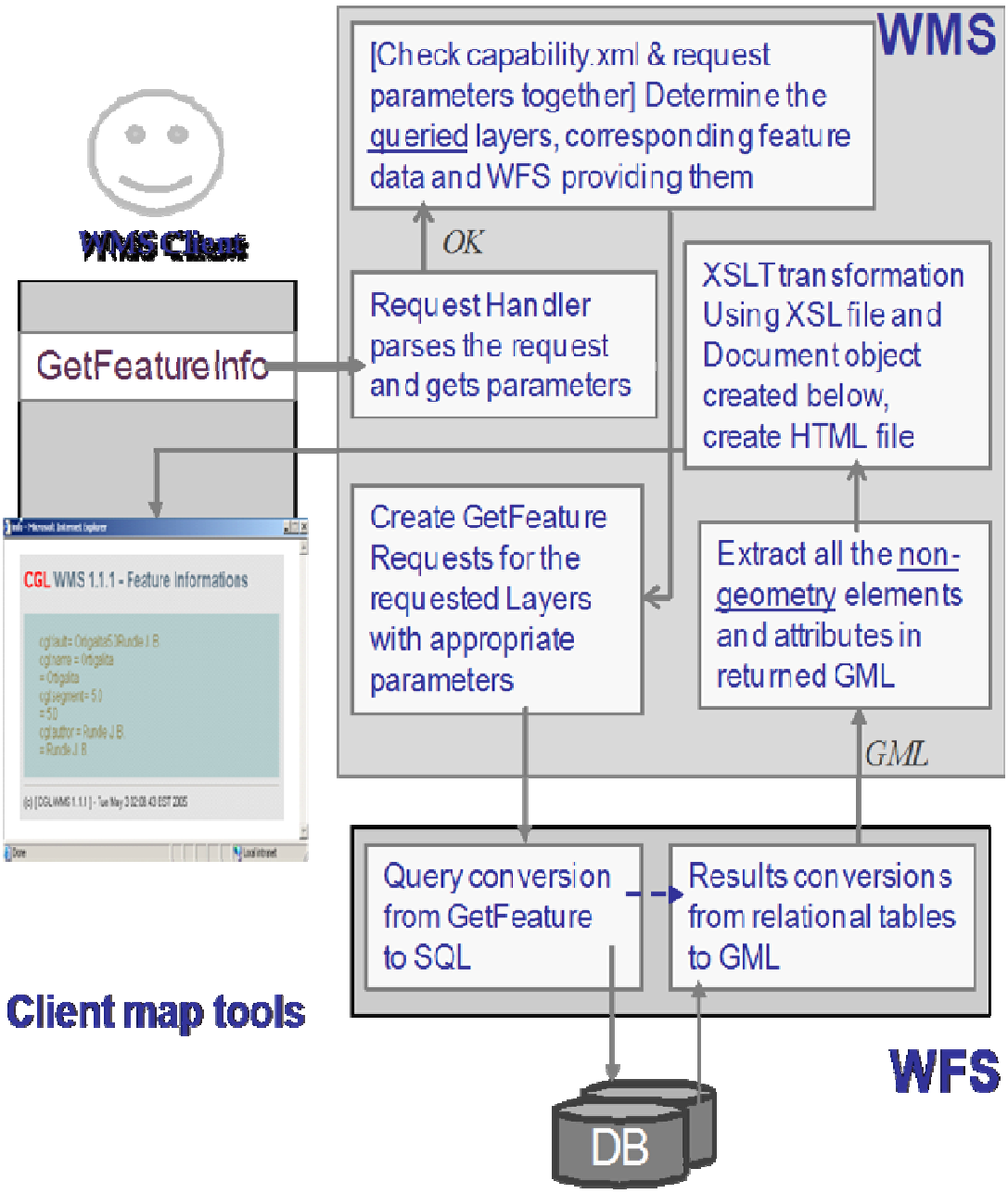

Figure 11: ***GetFeatureInfo*** **operation steps**

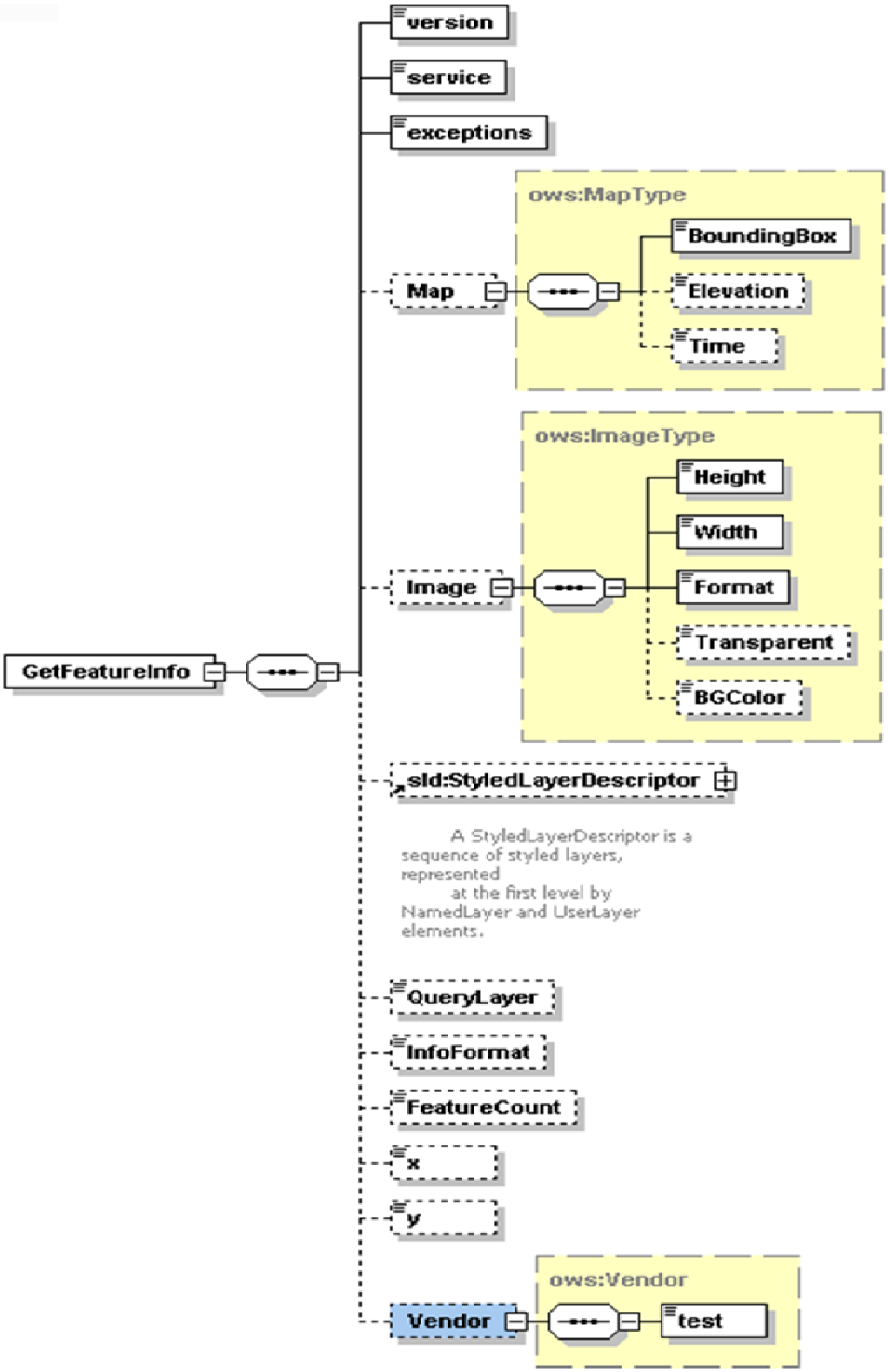

Figure 12: *GetFeatureInfo* Request Schema. See Appendix-A for an instance of this request schema.

```
<?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>
```

Figure 13: Generic XSL file for HTML creation from the GML in order to create responses for the *getFeatureInfo*.

3.3.3. Browser/event-based Interactive Map Client Tools

Interactive information visualization tools provide researchers with remarkable capabilities to support discovery. These tools were developed for interacting with standard Web Map Servers developed in Open Geographic Standards providing OGC compatible online services such as getMap, getFeatureInfor and getCapabilities. The tools provide structured multi-layered map images display (Figure 14 and Figure 15). Structured data display is composed of multiple layers and each layer is defined in the corresponding WMS service's capabilities file. As you remember, capabilities files are metadata defining service + data together. In case of WFS, the data is defined as feature

collections (see Appendix D), and in cased of WMS, the data is defined as layers (see Appendix C). Client tools enable users and decision makers to interact with the system through interactive event-based maps seamlessly and easily by hiding the system complexity. It also enables querying of the vector data in the multi-layered structured map images shown on the screen (see Figure 9). It does so by using WMS's standard *getFeatureInfo* service interface.

Several capabilities are implemented for the user to access and display geospatial data. The client tools enable the user to zoom in, zoom out, measure distance between two points on the map for different coordinate reference systems, to get further information by making *getFeatureInfo* requests for the attributes of the features on the map, and drag and drop the map to display different bounding boxes. Users can also request maps for the area of interest by selecting predefined options clicking the drop-down list. The user interface also allows the user to change the map sizes from the drop-down lists or enable them to give specific dimensions. Zoom-in and zoom-out features let the user change the bounding box values to display the map in more or less details. Each time user change the bounding box values, user interface shows the updated bounding box values at the each side of the map.

The proposed client tools are generic and capable of interacting with any other WMS and WFS developed in Open Geographic Standards. 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.

Figure 14 shows generic interactive map tools and user interface enabling interactive data access/query and display over integrated data views which is called map images. The sample map in the figure shows California earthquake seiemic data superimposed over Google Map.

Figure 15 is application based decision making tools extended based on generic map tools. System is developed as modular and can be updated according to the application requirements in terms of parameters and output results. Sample project in the figure super impose earthquake forecasting outputs of Pattern Informatics project over the Google maps.

Map layers (their orders, numbers, attributes etc.) are manipulated through the parts A, C and D (Figure 15). Application output is manipulated through part B/E and utilizes the parameters given in part A. Part C is the output screen and enables interactive manipulation of the layers and interactive query of the feature data on the map. Part E is used for animating successive static map images to create map movies from time series feature data. 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.

Here are the listings of the major generic action listeners for the user-map interactions (see Figure 14).

```
<event_controller>
        <event name="init" class="Path.InitListener" next="map.jsp"/>
        <event name="REFRESH" class=" Path.InitListener " next="map.jsp"/>
        <event name="ZOOMIN" class=" Path.InitListener " next="map.jsp"/>
        <event name="ZOOMOUT" class="Path.InitListener" next="map.jsp"/>
        <event name="RECENTER" class="Path.InitListener“next="map.jsp"/>
        <event name="RESET" class=" Path.InitListener " next="map.jsp"/>
        <event name="PAN" class=" Path.InitListener " next="map.jsp"/>
        <event name="INFO" class=" Path.InitListener " next="map.jsp"/>
</event_controller>
```

Event “init” sets all to initial opening settings. Events "REFRESH", "ZOOMIN", "ZOOMOUT", "RECENTER", "RESET" and "PAN" causes *getMap* request to WMS to get layers in map images. Event “INFO” causes *getFeatureInfo* request to get further information about feature data displayed on map images.

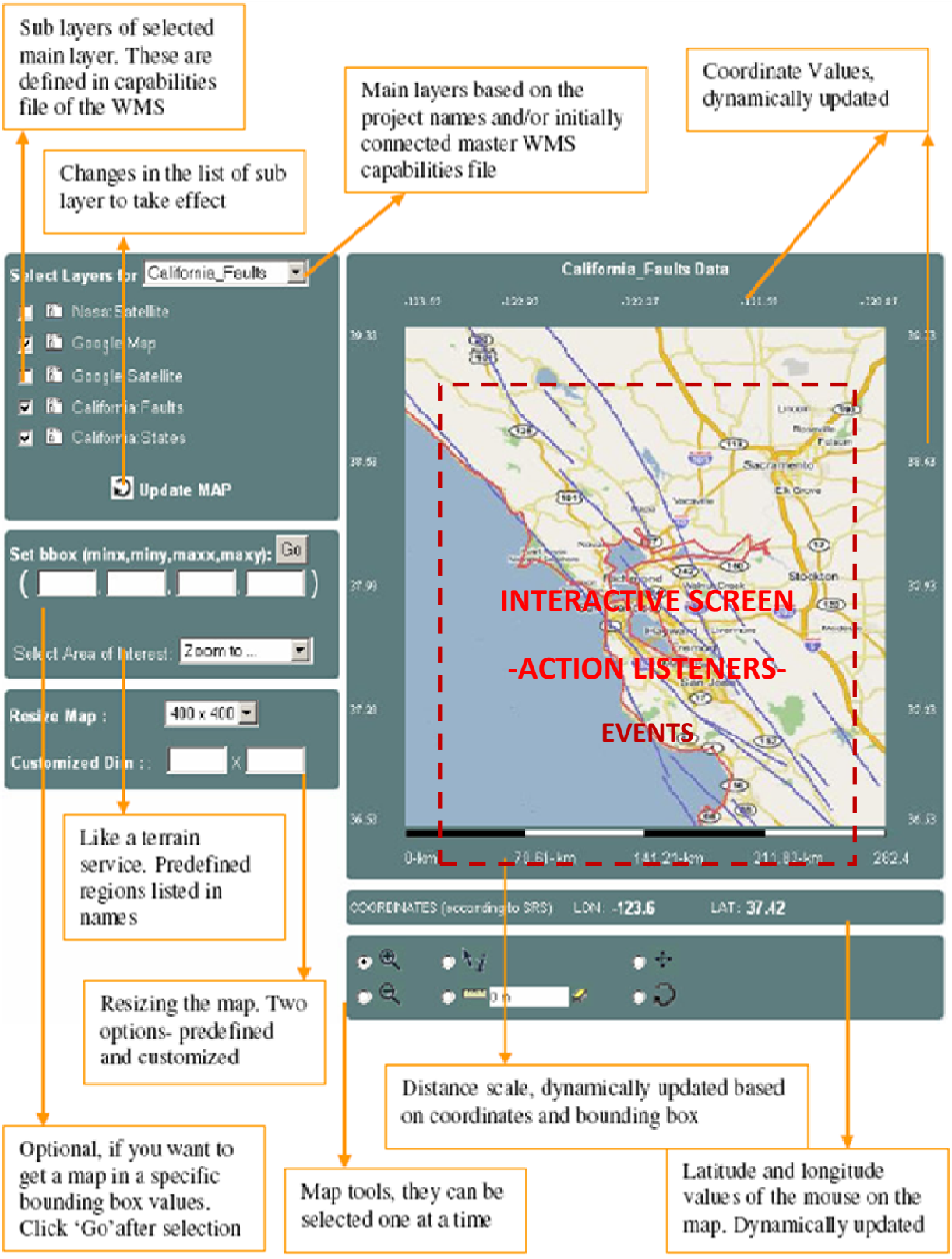

Figure 14: Event-based interactive map tools capable of interacting with any map server developed in open geographic standards.

B and E parts in Figure 15 are application based extensions to the standard map tools given in Figure 14. The figure illustrates Pattern Informatics application. Color bar and colored squares plotted over the map shows earthquake probability values sent out by PI application.

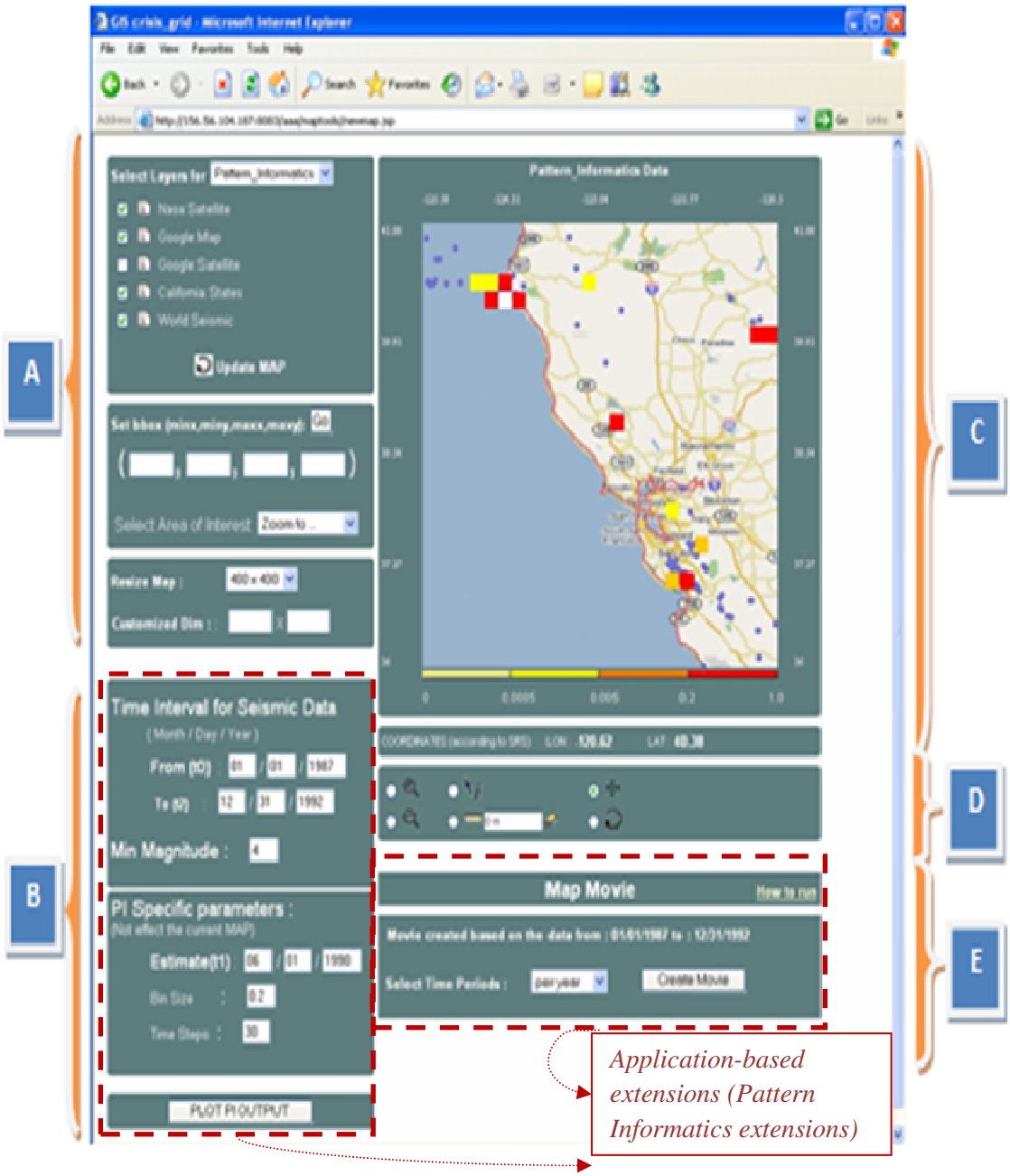

Figure 15: Standard interactive map tools extended with capabilities of integrating map images with outputs of geo-science grid applications.

There are many related works in developing such a framework for interacting GIS systems and enabling end-users to use system seamlessly. Our contribution is developing a framework capable of interacting with Service-oriented GIS systems with AJAX [AjaxSerrano] technologies. The following chapter gives more details about this intermediary framework to synchronize Web Service and AJAX transport protocols (SOAP over HTTP vs. XMLHttpRequest) and corresponding request/response formats.

3.3.3.1. Integration of AJAX approach to GIS Web Service Invocations

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. Our major focus on developing such a framework was GIS domain, but it can be applied to any browser/event-based interactive user interfaces communicating with Web Service components remotely.

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 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 [Mitchell] are adapting AJAX to their GIS systems.

ECMAScript [Ecma1, Ecma2] 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.

We first present the intermediary component to synchronize AJAX and Web Service protocols in terms of request and responses. Later, we give a sample scenario.

3.3.3.1.1. Architecture: Intermediary Synchronization Framework.

AJAX uses HTTP GET/POST requests (through JavaScript's XMLHttpRequest) for the message transfers (see (A) in Figure 16). Web Services use Simple Object Access Protocol (SOAP) as a communications protocol (see (B) in Figure 16) 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 16).

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.

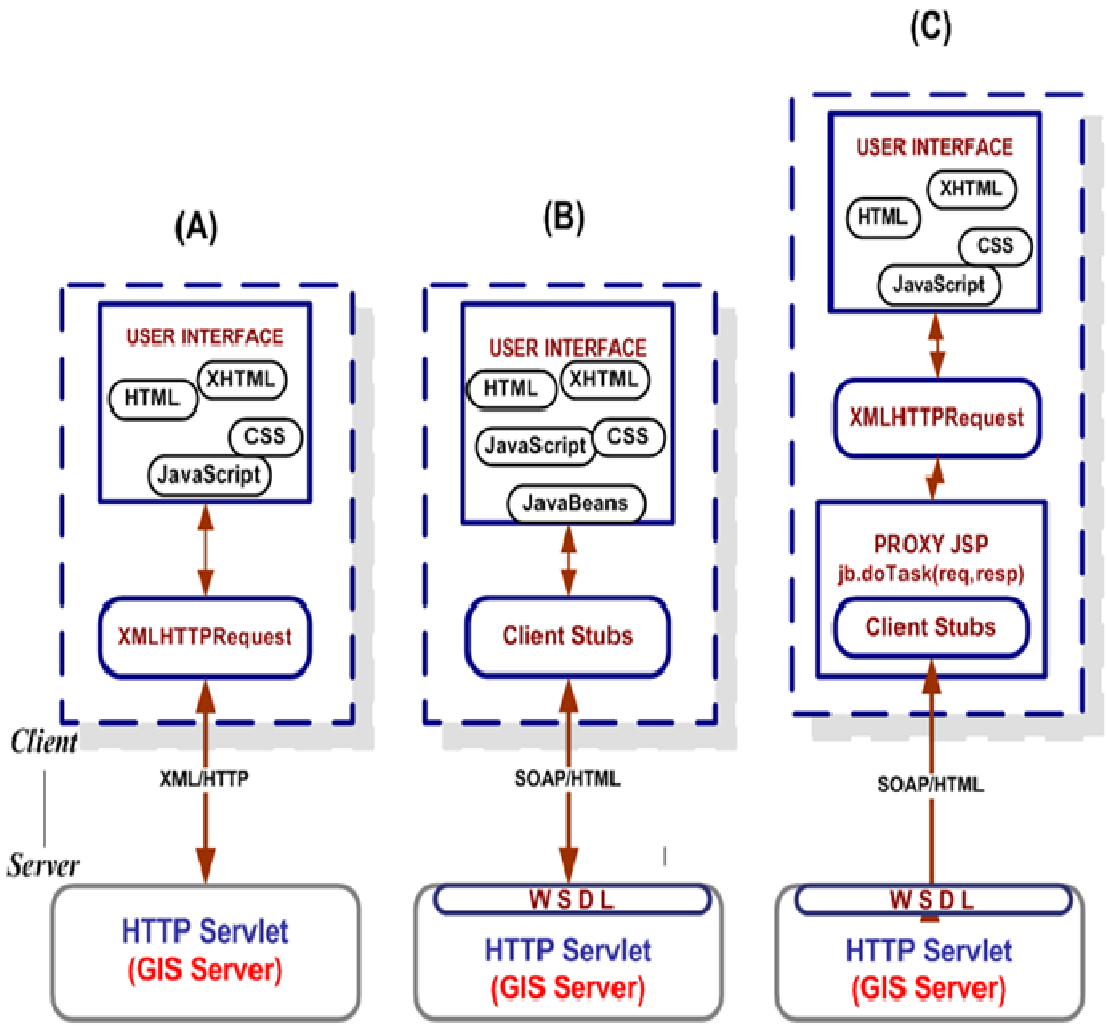

Figure 16: (A) Pure AJAX Approach, (B) Web Services Approach, and (C) Hybrid (AJAX + Web Services) Approach.

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 +…[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 the following section, we prove the applicability and efficiency of the proposed integration framework by giving a usage scenario.

3.3.3.1.2. A Case Scenario: Overlays of Google Maps and OGC's WMS

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 16-C. 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.

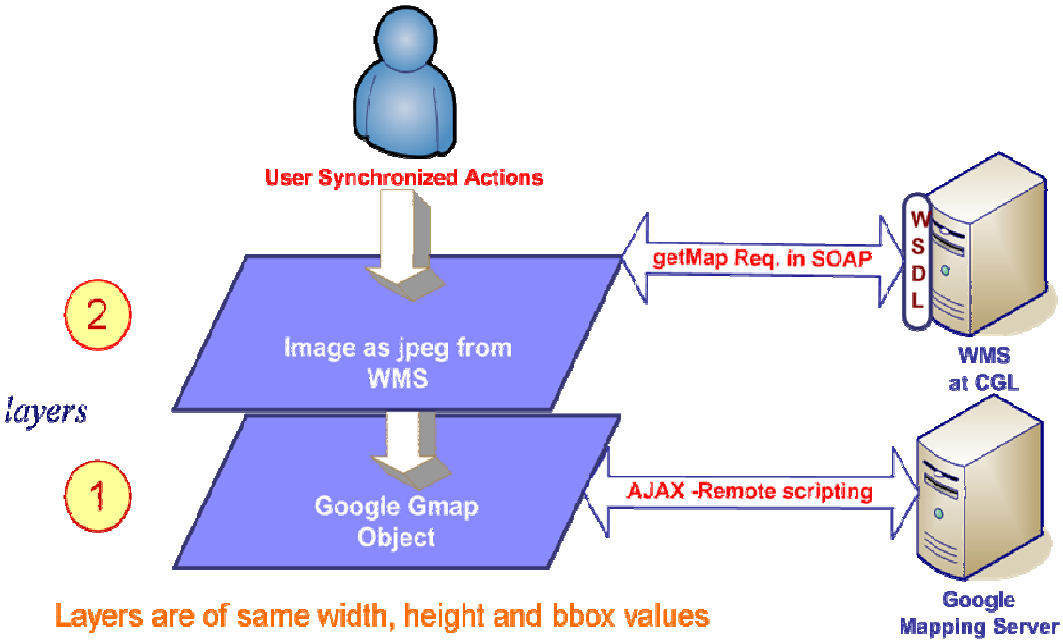

Figure 17: Integration of Google Maps with OGC WMS by using architecture defined in Figure 16.

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 Chapter 3.3.3 for sample GUIs). 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.

Evaluation of the 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 framework enables application developers to easily integrate AJAX based browser applications into Web Services.

AJAX and Web Services are XML based structures and this property allows 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 also 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, but the gains is much higher than the overhead times coming from the proposed intermediary service.

Chapter 4

Fine-grained federation of GIS Web-Service Components

The framework proposes infrastructure for understanding and managing the production of knowledge from distributed observation, simulation and analysis through integrated data-views in the form of multi-layered map images. Infrastructure is based on common data model, standard GIS Web-Service components (presented in Chapter 3) and a federator. Federator is actually an extended WMS federating GIS services and enabling unified data access/query and display over integrated data-views.

By the federation, we mean providing one global view over several data sources and let them processed as one source. There are three 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); and the third is the common presentation model of data sources, i.e. mapping of common data model to a display

model enabling integration/overlaying with other data sets (integrated data-view). The first two groups of research issues are related to lower level (database and files) data format/query/access heterogeneities summarized as semantic heterogeneity. In the proposed framework we take them as granted by applying Open Geographic Standards specifications for data models (GML) and online services (WMS and WFS). For the sake of responsiveness and quality of services we extended standard service components as Web Services with streaming data transfer and handling capabilities. We present high-performance design supports for the responsiveness in Chapter 6.

The proposed extended standard GIS Web Service components are integrated to the system through a federator which is actually a WMS extended with capability aggregating and state-full service capabilities to enable high performance support for responsive GIS applications. Actual federation is done through capability federation of standard GIS components at federator to create a global view over several data sources and let them processed as one source.

In this thesis, we concentrate on fine-grained view-level information presentation through federation of standard GIS Web Service components enabling uniform data access/query and analysis over integrated data views in the form of multi-layered map images. Although the framework is fine-grained for GIS domain we present the generalization architecture in terms of principles and requirements at the end of the chapter (Chapter 4.4).

4.1. Geo-Data and integrated data-view

The geo-data is provided by geographically distributed various kinds of different vendors in different formats, stored in various different storage systems and served through heterogeneous service API and transport protocols. Moreover, 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.

These issues have been touched in Chapter 3, and an interoperable service oriented GIS framework in accordance with Open geographic standards has been proposed. Online service descriptions and data models are described by Open standards but extended with our recommended functionalities and quality of services such as Streaming data transfer and processing and Web Services standards. In brief, according to the standard specifications there are two general groups of data services. One is Web Map Services and other is Web Feature Services. WMS provides rendered data in map images, and WFS provides annotated feature data in GML. Since both has standard service API and capability metadata about their services and data, they can be composed/chained by inter-service communications accomplished through their standard *getCapability* service API. This idea inspired us of developing infrastructure for creating and managing the production of knowledge from distributed observation, simulation and analysis through integrated data-views in the form of multi-layered map images (see Figure 18).

The geo-data is accessed through federator and the chain of WFS and WMS services. Upper levels are mostly occupied by WMSs (render/display services) and lower levels are occupied by WFSs (feature data level- annotation). At the bottom level mediators are located. WFSs play the role of wrappers in the mediation system. They provide data in common data model and with standard service interfaces. Our integration system uses GML as global data model to represent and manipulate geographic data in vector format.

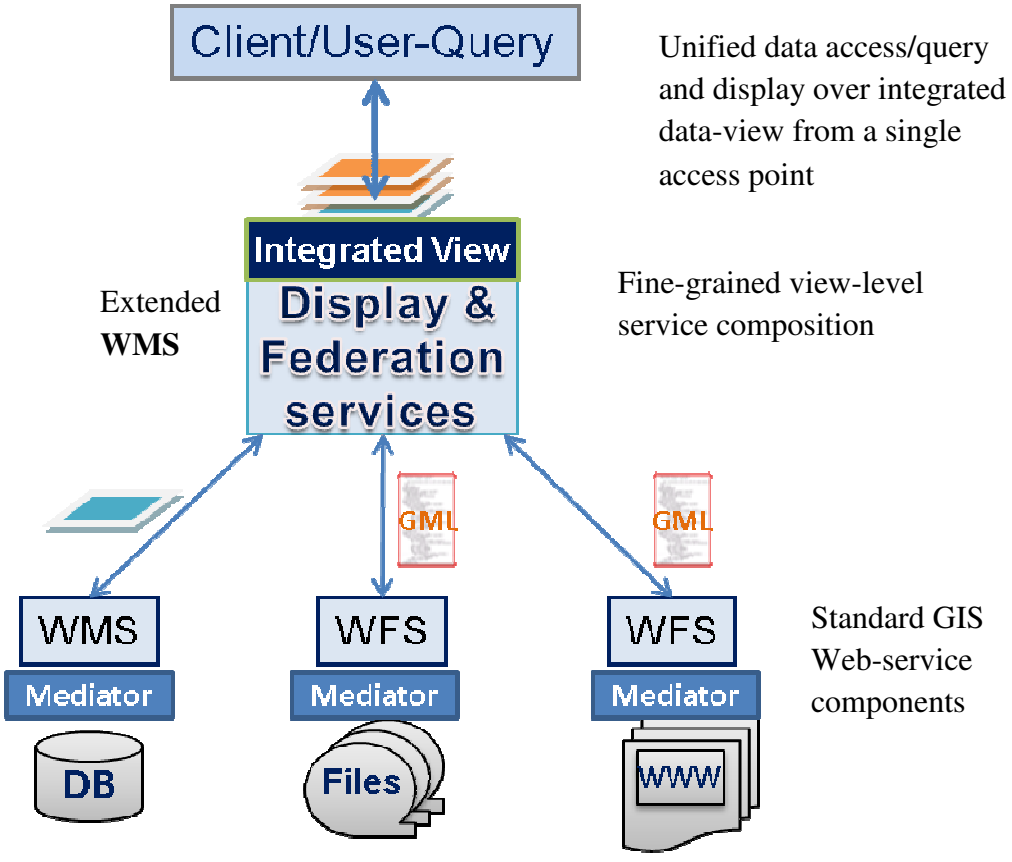

Figure 18: Data life-cycle and integrated data-view creation

There is three-level hierarchy of data in such a federated system enabling integrated data-view creation. At the top level of the hierarchy there is an application specific data called integrated data-view which is actually a map composed of multi-layers (Figure 18). Layers composing integrated data-view come from the standard GIS Web Service components which form the middle layer of the hierarchy. The middle part consists of GIS Web Services providing data in common data models with interoperable standard service interfaces.(see Chapter 3).

Heterogeneous data sources which form the bottom layer of the hierarchy are integrated to the system through mediators. 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.

There are several advantages in adopting the approach shown in Figure 18. First of all, 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. 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 Services.

We focus on upper levels of data flow and query refinements. In other words, we focus on the definition of service compositions and integrated data views. We give the architectural details in the following chapter.

4.2. Federation Framework

The proposed federation framework is built over the Service-oriented GIS framework and its components presented in Chapter 3. Federation is based on federating service-oriented standard GIS Web Services capabilities metadata and their standard service interfaces for accessing, querying, and rendering data. Creating such a federated design has some advantages in data sharing, reliability, and system growth (interoperability and extensibility).

We don't define common data models, online standard service components and their capability metadata definitions in GIS. These are already defined by Open Geographic Standards (OGC). We developed the components according to the open

standard specifications for online services and data model, and applied them to our proposed information system framework by defining required extensions at implementation and application levels in compliance with WS-I+ Web Service standards (Chapter 3).

This chapter presents a federation framework based on common data models (GML), standard Web Service components (see Chapter 3) federator and event-based interactive decision making tools over integrated data views in the form of multi-layered map images. The general architecture is illustrated in Figure 19.

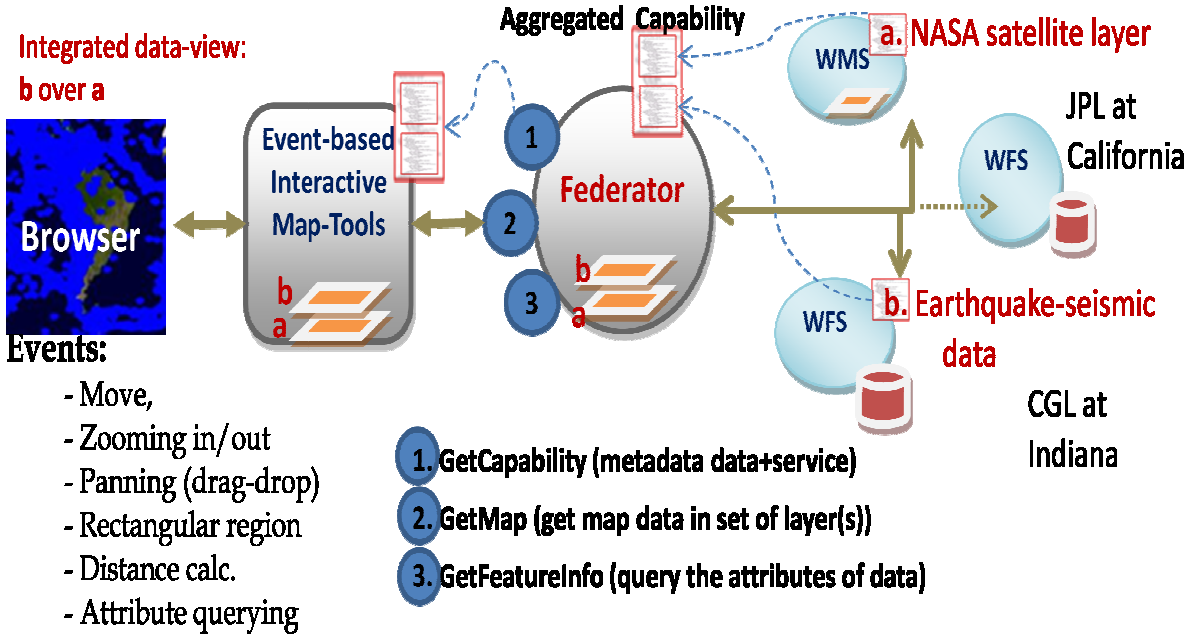

Figure 19: Fine-grained federated GIS framework

Federation is based on defining an application based hierarchical abstract data as multi-layered map images. The abstract data is defined by federator in its capability metadata. The hierarchical data is created by composition of layers (from WMS) and

feature data (from WFS). The layer composition is achieved by capability metadata federation (see Chapter 4.3.2) by using components' standard getCapability service interfaces.

The framework enables users/decision-makers access the system as though all the data and functions come from one site. The data distribution and connection paths stay hidden and formulated as hierarchical data defined in federator's capability metadata. The users access the system through integrated data-views (maps) with the event-based interactive mapping display tools (Chapter 3.3.3). Tools create abstract queries from the users' actions through action listeners and communicate with the system through federator.

As it is shown in Figure 19, the federator is actually a WMS with extended capabilities and functionalities but still provides standard WMS service interface and functionalities. These are getCapabilities, *getMap* and *getFeatureInfo* standard Web Services explained earlier in Chapter 3.3.2.

How Federation runs:

There are two stages. In the first stage integrated data-view is defined in federator's aggregated capability metadata (set-up stage), and in the second stage user/client interacts with the system and display/query integrated data-view (run-time stage).

1. Set-up stage –dotted lines, there is no client/user interaction yet
    a. Creation of application specific hierarchical data definitions

i. Service compositions in federator's aggregated capability metadata through *getCapability* standard service interfaces.

ii. Federator searches for standard GIS Web Service components (WMS or WFS) providing required data layers and organize them in one aggregated capability file.

iii. Aggregated capability is basically a WMS capability created by utilizing cascading definition of OGC standards (see Chapter 4.3.2.2).

b. Federator provides that aggregated capability metadata to its clients through its *getCapability* service interface.

2. Application Run-time (green lines, actual user interactions with the system): Users access/query and display data sources from a single access point (federator) over integrated data-views (multi-layered map images) defined in federator's aggregated capability metadata.

   a. Clients/user interacts with the system through event-based interactive map tools associated with the federator with the help of its aggregated capability metadata.

   b. Since federator is also a WMS, clients still use *getMap* service interface to display multi-layered map images and/or query it through *getFeatureInfo* service interface.

   c. On Demand Data Access: There is no copying of the data at any intermediary places. Data are kept at their originating sources. Consistency and autonomy.

The issues regarding creation of aggregated capability metadata and multi-layered map images definitions are presented in Chapter 4.3.

Why Capability metadata:

Capability is a metadata about the data and services together (see APPENDICES B and C). It includes information about the data and corresponding operations with the attribute-based constraints and acceptable request/response formats.

It supplements the Web Service Description Language (WSDL) [wsdl], which specifies key low-level message formats but do not define an information or data architecture. These are left to domain specific capabilities metadata and data description language (GML) [gml]. Capabilities also provide machine and human readable information that enables integration and federation of data/information. It also aids the development of interactive, re-usable client tools for data access/query and display.

As we mentioned before we don't define their structure and schema, we just use the open standard specifications definitions and present the required extensions for the federation through hierarchical data creation by service chaining.

4.3. Capability Aggregation for integrated data-views

The integrated data-view in multi-layered map images is defined in federator's aggregated capability metadata. There are two major issues here. These are definition of aggregated capability metadata and definition of multi-layered map images.

As we mentioned before, federation framework is built over the standard GIS Web Service components and federator concept is inspired from OGC's cascading WMS definition. In this respect, federator is actually a WMS with cascading and some other

related and extended capabilities not deteriorating the Open Standards requirements. In the following chapters we present OGC's ideas related to the service chaining and aggregation, and define multi-layered map images in the aggregated capability metadata.

4.3.1. Integrated data-view in multi-layered map images

Application-based hierarchical data is defined as an integrated data-view in the federator's capability metadata. It actually defines a static workflow starting from the federator and ending at the original data sources (WFS serving GML or WMS serving map layers). The services are linked through the references defined in their capability metadata. Decision makers' interactions with the system are carried over the integrated data views through event-based interactive map tools. Integrated data-views are defined in the hierarchical data format as explained below:

Application ->Map -> Layer -> Data {GML / binary images} ->Raw data (any type).

Applications have different interests of maps composed of different data-layers in different numbers and combinations (Figure 20). Maps are multi-layered complex structures whose layers might come from distributed heterogeneous resources and rendered from any type of data.

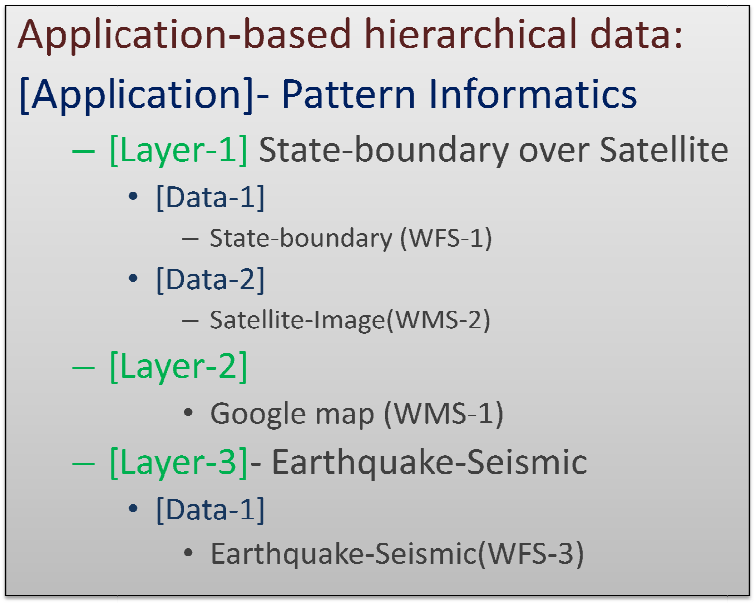

Figure 20: Pseudo hierarchical data definition in multi-layered map image for Pattern Informatics Application.

A map is application-based, human-recognizable, integrated data display and is composed of layers. A layer is a data rendering of a single homogeneous data source. Layers are created from the structured XML-encoded common data model (GML) or binary map images (raster data). Heterogeneous data source are integrated to the system through the mediators in the form of GML (WFS-based mediation) or binary map images (WMS-based mediation). The mediators have resource specific adaptors for request and response conversions and appropriate capability metadata describing the data and resources (see also Chapter 4.1).

This type of multi-layered map image is created in the federator with utilization of cascading WMS properties and inter-service communication between the components through exchanging capabilities via *getCapability* standard Web Service interfaces. The structure shown in Figure 20 for displaying composition and hierarchy of layers provided

from standard service components are formulated in two-ways according to the Open Geographic Standard specifications (OGC). First option proposes a solution by using context document. Second option proposes a solution by extending WMS's capabilities file. For more detail see the following Chapter.

4.3.2. OGC's proposals for Service Chaining –Cascading WMS

A "Cascading Map Server" is a WMS that behaves like a client of other WMSs and behaves like a WMS to other clients. Cascading WMS is a WMS which can read other WMS and display layers from them. For example, a Cascading Map Server can aggregate the contents of several distinct map servers into one service. Furthermore, a Cascading Map Server can perform additional functions such as output format conversion or coordinate transformation on behalf of other servers.

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) [Jerome05] 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.

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 [Jerome05].

A Context document is structured using XML and its standard schema is defined in the WMC specifications [Jerome05]. 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/complete standard specifications for chaining the services. They propose Web Map Context standards but in the future they plan to extend the capability file and embed these definitions and elements of Context into WMS capability file.

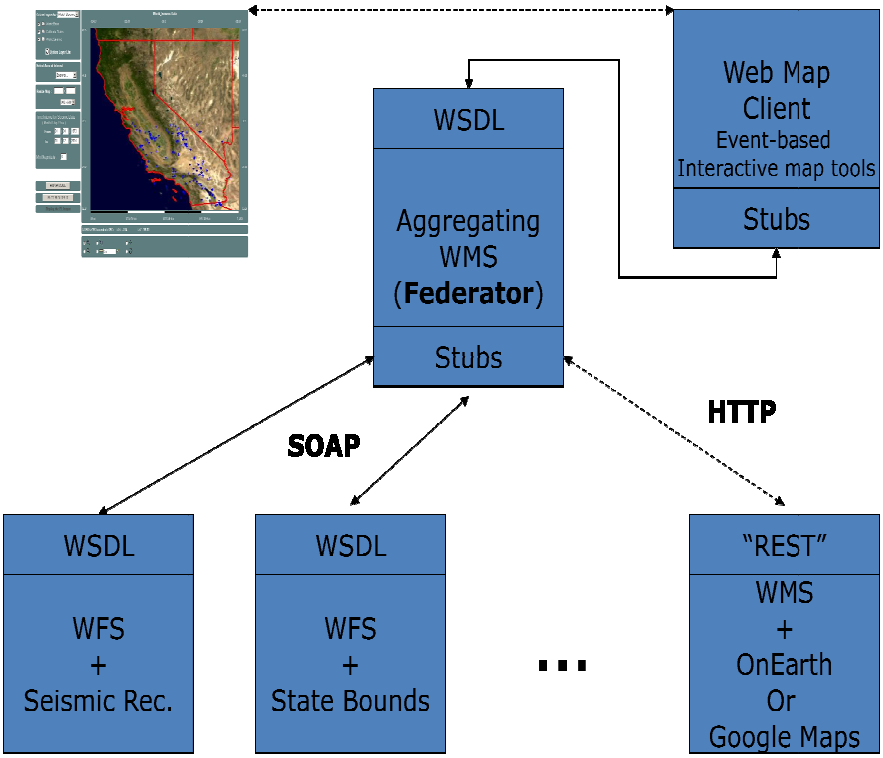

Figure 21: A sample scenario of the application specific federation of GIS Web Service components, Cascading/aggregating WMS as a federator.

In summary, currently there are two possible ways to chain the services to be able to create a federator framework and application specific hierarchical data in integrated data-view. One is extending the WMS capability file by giving the reference to the service access points providing the required layer (WMS) and/or feature data (WFS) (Chapter 4.3.2.2). Another way is using Web Map Context's standards defining chaining in a context document (Chapter 4.3.2.1). In any case, we utilize the cascading WMS definitions to develop a federator providing information/knowledge in multi-layered map images.

4.3.2.1. Federating Through Context Document

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. Here we will not explain the structures of the capabilities metadata and context document. We illustrate these options based on real application scenarios (See Figure 21).

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.

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.

4.3.2.2. Federating through aggregated WMS capability

Federator is actually a WMS. So, in order to define hierarchy of federation we utilize the WMS's capability document and describe the links to cascaded (or chained) services in capability's layer tag. Each layer defines data and corresponding standard GIS

Web Service components with their limitations, restrictions and attributes. Second one is extending WMS capabilities file for the cascaded layers.

Data providing in WMS is called “layer” and defined in layer tags in capability metadata with attributes and features according to the standard WMS capability schema [WMS. Service chaining is accomplished through cascaded layer definition. A Layer is said to have been "cascaded" if it was obtained from an originating server and then included in the Capabilities XML 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 spatial reference systems.

If a WMS cascades the content of another WMS then it shall increment by 1 the value of the cascaded attribute for the affected layers. If that attribute is missing from the originating WMS's Capabilities XML (means layer has not been cascaded before), then the Cascading WMS shall insert the “cascade” attribute to layer tag and set it to 1 [WMS]. Default value of cascading is 0.

Please also see [APPENDIX C] in order to better understand the layer attribute settings mentioned above for chaining/cascading of services and their descriptions in cascading WMS’s capability metadata.

4.4. Abstraction of the Framework for the General Domains

Our experiences with GIS have shown that federated, service-oriented, GIS-style information model can be generalized to many application areas such Chemistry and Astronomy. We call this generalized framework Application Specific Information System (ASIS) and give blueprint architecture in terms of principles and requirements.

When we tried to develop GIS like framework proposed in this thesis we need basically two types of standard service definitions in terms of service interfaces and their capability metadata defining service+data providing. Moreover, we need a domain specific common data model schema as in GIS domain called GML. Standard service characteristics should match to WMS and WFS. In other words, one group of service should provide application specific common data model (like GML) and others should provide comprehensible data representation corresponding to or created from the common data model or their combinations.

Here we give a course-grained architecture consists of abstract components and explain their data flow and components interactions. We mostly focus on principles and requirements to generalize GIS-like architecture to any other information system domains.

4.4.1. Generalization Framework

Based on our analysis of OGC Open Geographic Standard specifications in GIS domain, we have identified a number of generic system components enabling federated data and information filtering to support distributed access, querying and transformation (see Figure 14). These components are basically Web Map Service and Web Feature services which are filter-like services and provide standard Web Service interfaces. After having such a complete system for GIS, we investigate the possible principles and architectural requirements to the system to make it applicable to all domains such as Astronomy and Chemistry.

We called such generalization architecture as Application Specific Information System (ASIS) and illustrated in Figure 22.

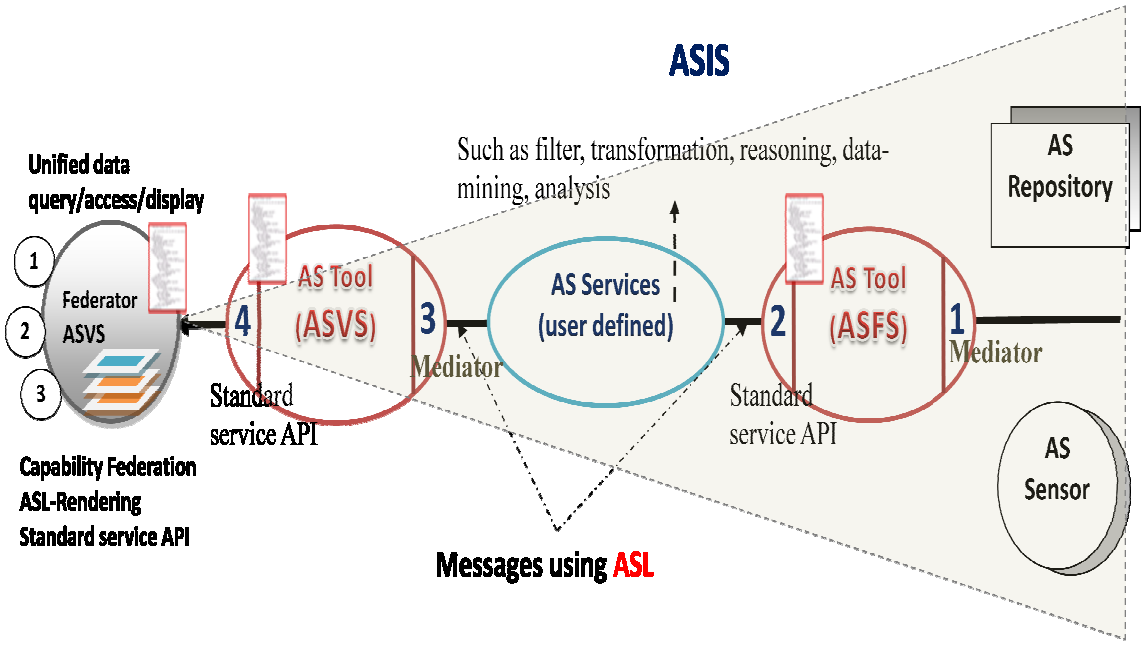

Figure 22: Application Specific Information System (ASIS)

ASIS propose an alternative solution to heterogeneous data integration. Solution enables inter-service communication through well-defined service interfaces, message formats and capabilities metadata. Data and service integration is done through "capability" federation of these services which are implemented in Web Services. In ASIS approach, there are two general group of services (ASFS and ASVS), and each service is described by corresponding generic metadata descriptions that can be queried through Web Service invocations. Going beyond the enablement of service discovery, this approach enables at least three important things. First, services of the same type that provide a subset of the request can be combined into a "super-service" that spans the query space and has the aggregate functionality of its member services. Second, the capability metadata can be used to determine how to combine services into filter chains

with interconnected input-output ports. Third (and building on the previous two), capabilities of "super-services" can be broken into smaller, self-contained capabilities that can be associated with specific services. This enables performance gains through load-balancing.

ASIS consists of filter Web Services (ASFS and ASVS) having common interfaces and communicating with each-other through capability exchange interface. Being a Web Service enables filter services to publish their interfaces, locate each other and chain together easily. Filters have inter-service capabilities and chainable. If the Filter is capable of communicating and obtaining data from other Filters, and updates (or aggregates) its capability metadata with these data (after capability files exchange), then it can claim that it serves those data. Filter Services are information/data services that enable distributed data /information access, querying and transformation through their predictable input/output interfaces defined by capability document. Filter located in the same community network can update their capability metadata dynamically through "getCapabilities" service interface of the filters. Dynamically updating capabilities of filters enable removal of obsolete data or down filters.

4.4.2. Components abstraction – ASFS and ASVS

In ASIS, Filter Services are grouped into two: Application Specific Feature Service (ASFS) and Application Specific Visualization Service (ASVS). ASVS is abstraction of WMS and ASFS is abstraction component of WFS in GIS domain (see Table 1). These standard Web Service components have capability metadata defining the data provided and operations available on the data with domain specific constraints and attributes. The data related constraints and attributes are very closely related to the

Application Specific Language (ASL) defining the common data model. ASL corresponds to Geographic Markup Language (GML) in GIS domain. As we mentioned before, ASIS is based on standard Web Service components, common data model and capability metadata (see Table 1).

ASL is a common data model structured in XML with query capability. It is basically an annotated data representation containing content and presentation tags. It is developed and recommended by the authorities in a specific domain, for example Open Geospatial Consortium (OGC) in GIS domain. If there is no available ASL defined as in Chemistry domain (see Table 2) then ASIS developer must define it by himself. XML based hierarchical data model enables common language and communication across operation system and platforms to exchange and federate information. In order to make the proposed architecture feasible, all the data should be converted to ASL through an adapter deployed database end Filter Service such as Web Feature Service (WFS) in GIS domain. Filter Services provide data in consistent formats and define these formats and the ways to access them in their capability documents.

Table 1: From GIS to ASIS components' mapping

GIS Component		**ASIS Component**	**Descriptions**
GML	→	ASL	Common Data Model
WFS	→	ASFS	Feature data mediation
WMS	→	ASVS	Display data mediation
Capability.xml	→	Metadata	For ASFS and ASVS

ASVS both visualize information and provide a way of navigating ASFS and their underlying DB. ASVS provide human readable information such as text, graphs (scalable vector (SVG) or portable (PNG)) and images. ASFS are like annotation services providing heterogeneous data in common data model with attribute based query capability. ASFS basically serve data in ASL (domain specific XML-encoded common data model) containing content and representation tags. Heterogeneity in queries and data formats is handled through resource-specific mediators.

Application Specific (AS) Services in ASIS (Figure 22) are user defined services providing application specific data and services. These are like transformations, reasoning and data-mining tools for extraction knowledge from the feature data provided by ASFS in ASL format. To be more specific we can give Pattern Informatics Geo-physics applications as an example. In PI, ASIS needs to overlay multi-layered map images with earthquake forecast values as hot-spot plots in colored boxes showing magnitudes of expected earthquake seismicity. See sample output layer at Figure 26.

System can enable dynamic metadata update via catalog-registry services or P2P metadata exchange. In that case, whenever server is down or data is corrupted or event in any other change, other services will be notified and filter chain will be automatically fixed. We have not implemented that feature even in our completed GIS framework. This will be our future research work.

Investigating correspondents of ASIS components in different domains:

GIS is a mature domain in terms of information system studies and experiences. It has standard bodies defining interoperable online service interfaces and data models such as OGC ISO/TC210, but many others still do not have.

For example in Chemistry, specifications for distributed scientific applications are very immature, moreover, they have just data model specification Chemistry Markup Language (CML) which is not widely accepted and used.

Regarding Astronomy, they have standard body called International Virtual Observatory Allience (IVOA) for data formats and online services. FITS (Flexible Image Transfer), Images and VOTable [Williams02] are the data models. SkyNodes is a Database server with an ADQL (Astronomy Distributed Query Language) based SOAP interface returning VOTable based results. VOPlot and TopCat are the services to visualize the astronomy data in the format of VOTable, FITS and images. VOResource and UCD are the metadata definition and standards for the service descriptions [Yasuda04].

Table 2: Components and common data model matching for generalization of GIS to ASIS. Two selected domains are Astronomy and Chemistry.

ASIS / Science Domains	Common data Model ASL	Components ASFS	Components ASVS	Metadata
GIS	GML	WFS	WMS	capability.xml schema
Astronomy	VOTable, FITS	SkyNode	VOPlot TopCat	VOResource

Chemistry	CML	None	NO standard JChemPaint	None

4.4.3. Standard Service Interfaces and Mediators

- **1, 2, 3 and 4 in Figure 22**

The standard Service interfaces can be grouped into three. One is for capability metadata exchange; one is for query of data itself and one is for getting further information about the data attributes.

As mentioned before, capability helps client make valid requests for its successive queries. Capability basically provides information about the data sets and operations available on them with communication protocols, return types, attribute based constraints etc. Each domain has different set of attributes for the data and it is defined in ASL common data model. For example in GIS domain sample attributes might be bounding box values (defining a range query for data sets falling in a rectangular region) and coordinate reference system.

Standard requests/query instances for the standard service interfaces are created according to standard agreed-on request schemas. For the Web Map Service in GIS domain see Figure 5, Figure 7 and Figure 12 as samples of standard request schemas. Request instances contain some format and attribute constraints related to the ASL common data model. For example in GIS domain, *getMap* request define map images' return format (JPEG, PNG, SVG etc), height, width, bounding box values etc. Format, height and width are related to display, but bounding box values are related to the attributes of the data defined in its ASL representation provided by ASFS (or WFS in GIS). In that specific example of *getMap* request, ASVF both visualize information

through *getMap* service interface and provide a way of navigating ASFS and their underlying DB. ASVS make successive queries (with user-defined bounding box values in *getMap* query) to the related ASVSs to get the ASL data and render it to create final display for its clients.

In ASIS, the task of mediators is to translate requests to the standard service interfaces to those of the information/data sources', and transform the results provided by the information source back to the ASIS's standard formats. In ASFS case it is ASL and in ASVS case it is any kind of display format such as images.

Acting as a proxy of information source, the mediators communicate with an information source in its native language and API, and communicate with the ASIS in a commonly agreed language (ASL) and Web Service API (such as *getCapabilities*, *GetFeature* and *DescribeFeatureType in GIS*). In this way, "wrapping" each information/data source into the translation software makes the particular sources manageable.

The mediators-wrappers enable data sources integrated to the system conform to the global data model (AS), but enable the data sources to maintain their internal structure and, at the end, this whole mediator system provides a large degree of autonomy. Instead of actual physical data federation, system makes distributed querying and response composition on the fly.

Chapter 5

Applications of the Federation Framework

Our proposed service-oriented federated GIS framework architecture and its components WMS Web Services, browser/event-based interactive decision making tools have been used in several GIS projects. This chapter discusses three of them. One is Los Alamos National Laboratory (LANL) project (Chapter 5) and other two are Solid Earth Virtual Observatory Grid (SERVOGrid) projects [Servo, Cce] (Chapter 5.3 and Chapter 5.4).

5.2. Los Alamos National Laboratory, NISAC SOA Architecture

The National Infrastructure Simulation and Analysis Center (NISAC) at Los Alamos National Laboratory (LANL) develop advanced modeling and simulation tools

for analysis of the critical infrastructure. These tools allow authorities to understand interdependencies, vulnerabilities, and complexities of the infrastructure and help develop policies, investment plans, education and training etc for crisis situations [Meyer03].

The Interdependent Energy Infrastructure Simulation System (IEISS) [Bush03], 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.

Traditionally IEISS runs as a desktop application with local input data supplied as XML files collected from various sources, and the result is locally generated. The data are either being kept in databases such as Environmental System Research Institute (ESRI) [Esri] spatial database, or in proprietary XML files. The person who runs the application collects the data to local machine and runs the simulation. The results are usually shared with e-mails. However this approach has several limitations; every time the simulation is to be run the data have to be copied to the local file system, there is no way of running the simulations remotely and getting the results instantly.

We have worked with IEISS people at LANL and applied our GIS Grids ideas to create a Service-oriented Architecture for Los Alamos National Laboratory, National Infrastructure Simulation and Analysis Center. We have integrated several Web Services including Web Map Service and interactive event-based decision making and map-data

display tools with IEISS (Interdependent Energy Infrastructure Simulation System) [Bush04]. In our sample SOA demonstration we were able to invoke IEISS to simulate interdependencies between electrical and natural gas infrastructure components using a provided sample data set. The data do not actually correspond to real-world infrastructure maps however it allowed us to demonstrate that the normally desktop based simulation applications could be integrated into a Grid architecture using Web Services approach.

In summary, we have created an architecture consisting of several Web Services which exposes IEISS as a Web Service and shows the analysis results on an interactive online mapping application.

The major data flow in IEISS is in accordance with the general flow as expressed in Figure 2. Figure 24 shows a snapshot of system client interaction GUI and a sample output. Output image shows overlays of feature data layers on a satellite picture provided by the NASA OnEarth WMS Server [OnEarth]. Feature data in that application are electric and natural gas infrastructure components provided by WFS in GML common data model in XML files.

The components of this architecture are as follows:

Feature Database: This is our MySQL spatial database which holds various geospatial features such as California faults and earthquake data, US state borders, global seismic hotspots etc. For the NISAC SOA demonstration we have acquired a sample XML file which contains natural gas and electric power components for the State of Florida. This sample data is inserted into feature database as two distinct feature types. This allows us to make geospatial queries on the feature data and obtain the desired components as GML documents.

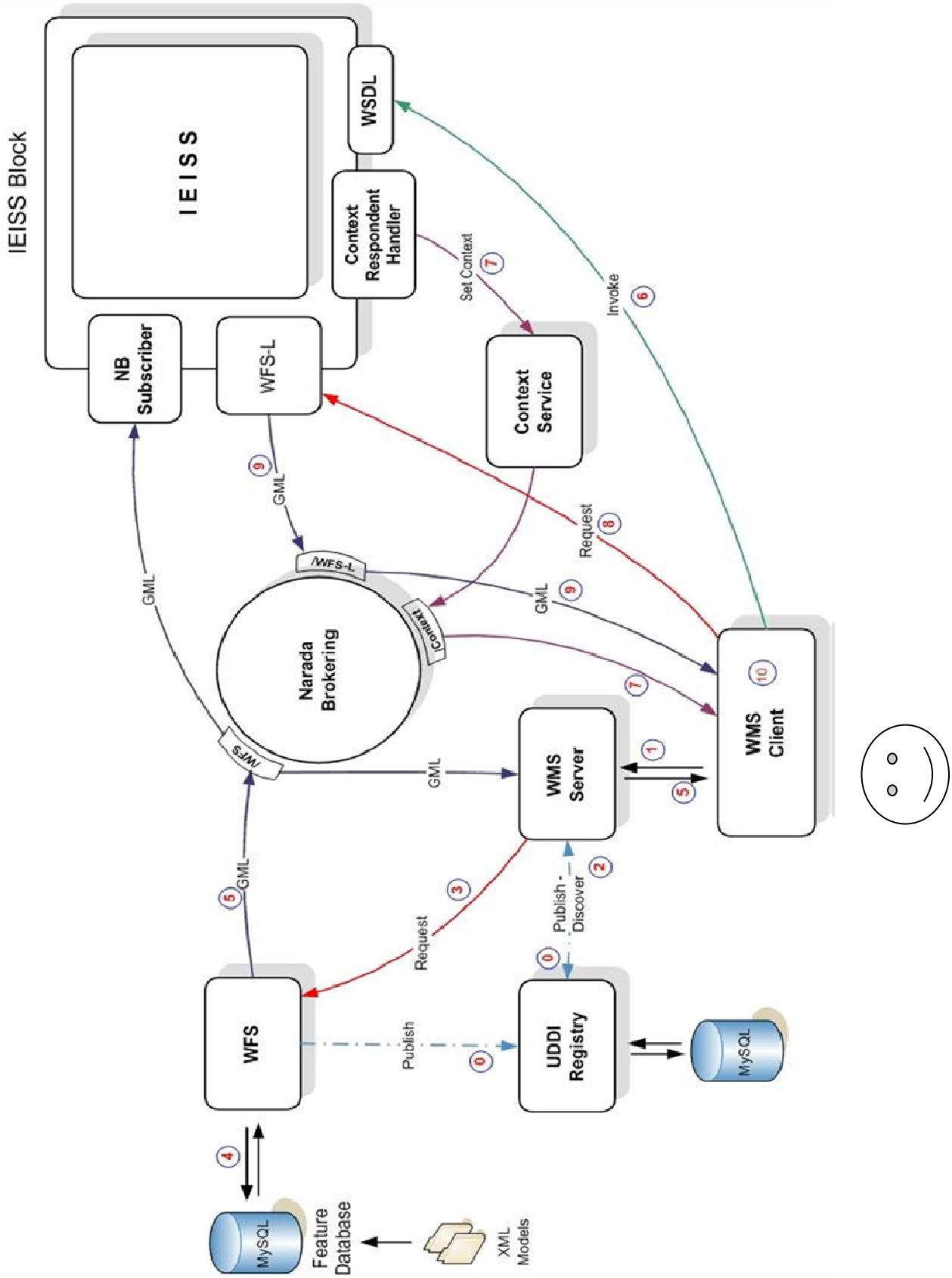

Figure 23: NISAC SOA Demonstration Architectural Diagram and Data Flow.

Web Feature Service: Provides interfaces to access and query the Feature Database and receive the geospatial features. The features are provided as GML Feature Collections which then can be used as map overlays or for geo-processing etc. We have created lightweight WFS in this project (WFS-L) which receives the new model XML created by IEISS, converts to GML and publishes to NB.

UDDI Registry: This service provides an API for publishing and discovery of geospatial and visualization services. It extends existing Universal Description, Discovery and Integration (UDDI) [Belwood] Information Model to provide GIS domain specific Information Services.

Web Map Client: It provides a user interface that displays the map overlays and allows client interaction with the maps. It also synchronizes and control all the user interactions with the system.

Web Map Server: Relays the client requests to the WFS, and receives the response as GML documents. WMS then converts GML to map images (JPG, TIFF, SVG etc.) and forwards these to the Web Map Client.

NaradaBrokering: This is a standalone publish/subscribe service. Allows providers to publish their data products to topics and forwards this data to the subscribers of a particular topic. We use NaradaBrokering as the messaging substrate of the system. All GML and XML data transport is done through this service.

Context Service [Bunting03]: The Context Service provides a dynamic, fault tolerant metadata hosting environment to enable services to share information within a workflow session to correlate their activities.

Context Respondent Handler: The Context Response Handler is used to communicate with the Context Service. It allows Context Service to inform its consumers about results of the operations.

gml2model Tool: Geospatial data exchange format for the system is GML. According to the user's selection WFS encodes requested geospatial feature data in GML and publishes to a certain NaradaBrokering topic. A NaradaBrokering Subscriber tool is used to save GML FeatureCollection published by WFS into a file. IEISS requires input data to be in a certain format called XML Model. We wrote a tool called gml2model to convert GML FeatureCollection documents to IEISS XML Model format.

shp2gml Tool: One type of the IEISS outputs is ESRI Shape files which show calculated outage areas etc. We use an open source tool called shp2gml by open source *deegree* project [deegree] to convert these shape files to GML, which are sent to WMS Client by the lightweight WFS. Data

The data flow in this architecture is explained here:

0. WFS and WMS publish their WSDL URL to the UDDI Registry
1. User starts the WMS Client on a web browser; the WMS Client displays the available features. User submits a request to the WMS Server by selecting desired features and an area on the map. WMS Client is actually event-based interactive map tools.
2. WMS Server dynamically discovers available WFS that provide requested features through UDDI Registry and obtains their physical locations (WSDL address).

3. WMS Server forwards user's request to the WFS.
4. WFS decodes the request, queries the database for the features and receives the response.
5. WFS creates a GML FeatureCollection document from the database response and publishes this document to NaradaBrokering topic *'/NISAC/WFS'*; WMS Server and IEISS receive this GML document.

   WMS Server creates a map overlay from the received GML document and sends it to WMS Client which in turn displays it to the user. After receiving the GML document IEISS NB Subscriber invokes *gml2model* tool; this tool converts GML to XML Model format to be processed by IEISS.
6. User invokes IEISS through WMS Client interface for the obtained geospatial features, and WMS Client starts a workflow session in the Context Service. On receiving invocation message, IEISS updates the shared state data for the workflow session to be "*IEISS_IS_IN_PROGRES*" on the Context Service. Both IEISS and WMS Client communicate with Context Service via asynchronous function calls by utilizing Context Respond Handler Service. IEISS runs and produces an ESRI Shape file that has the outage areas for the given region.
7. IEISS invokes ***shp2gml*** tool to convert produced Shape file to GML format. After the conversion IEISS updates shared session state to be "*IEISS_COMPLETED"*. As the state changes, the Context Service notifies all interested workflow entities such as WMS Client. To notify WMS-Client, the Context Service publishes the updates to a NB topic (***/NISAC/Context://IEISS/SessionStatus***) from which the WMS-Client receives notifications.

8. WMS makes a request to the WFS-L for the IEISS output
9. WFS-L publishes the IEISS output as a GML FeatureCollection document to NB topic *'NISAC/WFS-L'*. WMS Server is subscribed to this topic and receives the GML file then converts it to map overlay,
10. WMS Client displays the new model on the map

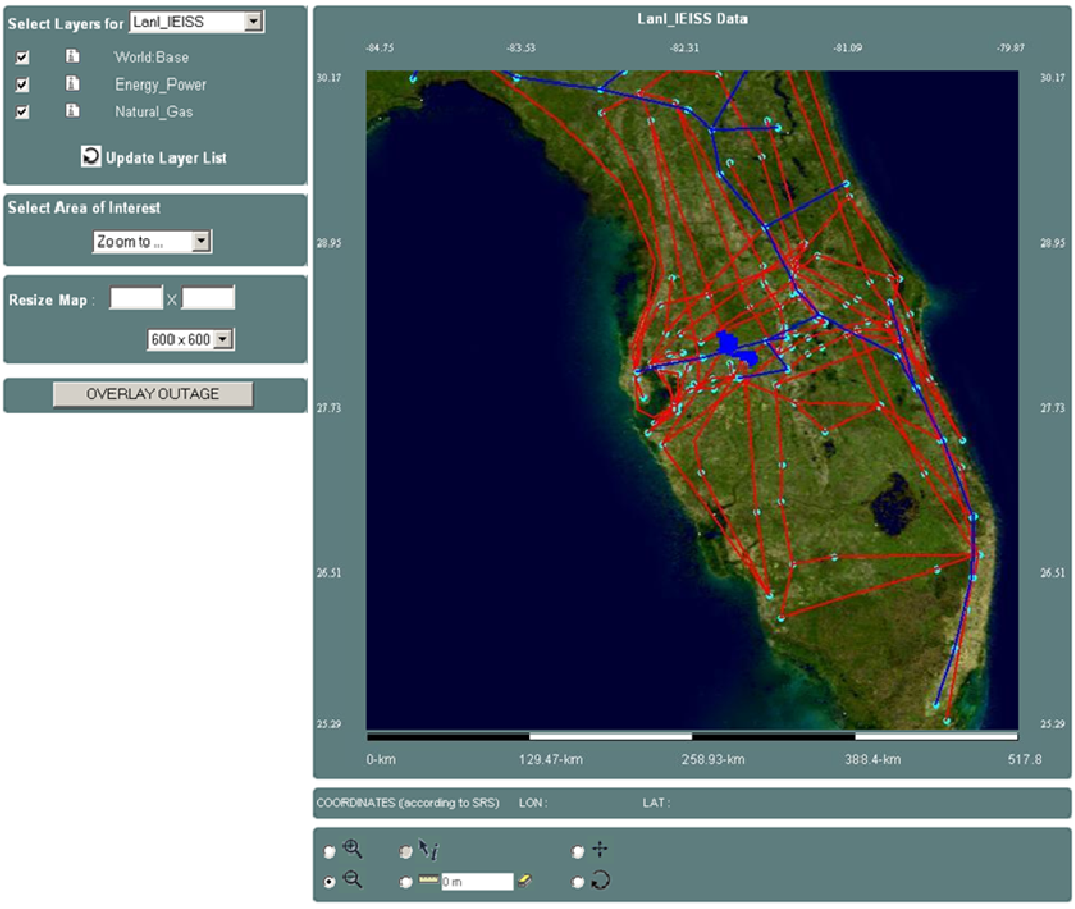

Figure 24: Sample Florida State Electric Power and Natural Gas Components as overlays on a Satellite Picture provided by NASA OnEarth WMS Server. Electric power components are connected with red, natural gas components are connected with blue lines.

Figure shows a sample IEISS output; here the blue region depicts the affected outage area. This image is generated by the Web Map Service. The blue region is the

affected area calculated by IEISS because of a possible problem with the energy infrastructure.

5.3. Pattern Informatics (PI) Application

The Pattern Informatics (PI) [Tiampo, Patterninfo] method uses observational data to identify the existence of correlated regions of seismicity. The method does not predict earthquakes, rather forecasts the regions or so-called hotspots where earthquakes are most likely to occur in the relatively near future.

PI algorithms Geo-science applications developed at the University of California-Davis by SERVOGrid team member Prof. John Rundle and his group. 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.

There have been two major types of approaches for forecasting earthquakes. The first approach is based on empirical observation of precursory changes such as seismic activity, ground motions and others. The second approach is statistical patterns of Seismicity [Holliday05]. The hypothesis behind these approaches is that the earthquakes will occur in regions where typically large earthquakes have occurred in the past. The Pattern Informatics (PI) approach suggests that a more promising approach to this hypothesis is that the rate of the occurrence of small earthquakes in a particular region can be analyzed to assess the probability of much larger earthquakes [Rundle03].

PI tries to discover patterns given past data to predict probability of future events. The process of analysis involves data mining which is made using results obtained from a Web Feature Service. The Web Map Service is responsible for collecting parameters for

invoking the PI code. These parameters are then sent to an HPSearch [Hpsearch, Gadgil05] engine which invokes the various services to start the flow.

Additional components of the architecture:

In addition to the components mentioned for IEISS in Chapter 5, there is one more component called HPSearch. It is simply a scripting technique for managing distributed workflows. Different Geo-Science applications require different set of parameters for the users to utilize the system. 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. After the application finish the task, job manager send the output link to the user.

Figure 25's steps are summarized below. This is the basic scenario that we use for integrating Pattern Informatics, Regularized Deterministic Annealing Hidden Markov Model (RDAHMM) [Rabiner, Granat], and other applications.

*Flow in this architecture is explained here (*Figure 25*):*

0. WFS and WMS publish their WSDL URLs to the UDDI Registry.
1. User starts the WMS Client on a web browser; the WMS Client displays the available features. User submits a request to the WMS Server by selecting desired features and an area on the map. WMS Client is actually event-based interactive map tools.
2. WMS Server dynamically discovers available WFSs that provide requested features through UDDI and obtains their physical locations (WSDL address).
3. WMS Server forwards user's request to the WFS.
4. WFS decode request, query the database for features and receives the response.

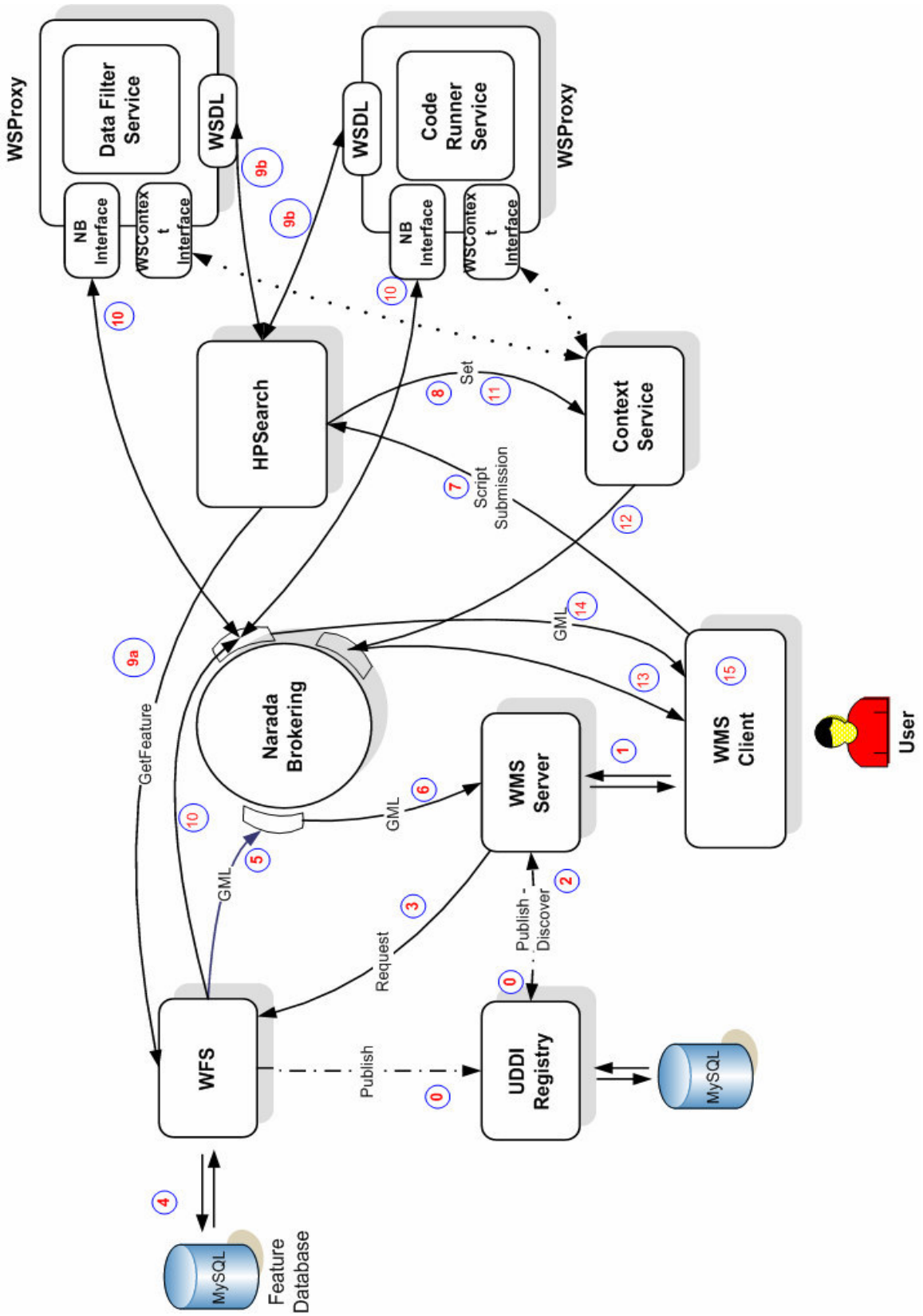

Figure 25: A general GIS Grid orchestration scenario involves the coordination of GIS services, data filters, and code execution services. These are coordinated by HPSearch

5. WFS creates a GML FeatureCollection document from the database response and publishes this document to a specific NaradaBrokering topic.
6. WMS receives the streaming feature data through NaradaBrokering's agreed upon topic. WMS Server creates a map overlay from the received GML document and sends it to WMS Client which in turn displays it to the user.
7. WMS submits flows for execution by invoking the HPSearch. This request also includes all parameters required for execution of the script. The HPSearch system works in tandem with a context service for communicating with WMS.
8. Initially, the context corresponding to the script execution is marked "Executing".
9. Once submitted, the HPSearch engine invokes and initializes (a) the various services, namely the Data Filter service, that filters incoming data and reformats it to the proper input format as required by the data analysis code, and the Code Runner service that actually runs the analysis program on the mined data. After these services are ready, the HPSearch engine then proceeds to execute (b) the WFS Web Service with the appropriate GML query as input.
10. The WFS then outputs the result of the query onto a predefined topic. This stream of data is filtered as it passes through the Data Filter service and the result is accumulated by the code runner service.
11. The code runner service then executes the analysis code on the data and the resulting output can either be streamed onto a topic, or stored on a publicly accessible Web server. The URL of the output is then written to the context service by HPSearch [Hpsearch1].
12. The WMS constantly polls the context service to see if the execution has finished.

13. The execution completes and the context is updated.

14. The WMS downloads the result file from the web server and displays the output.

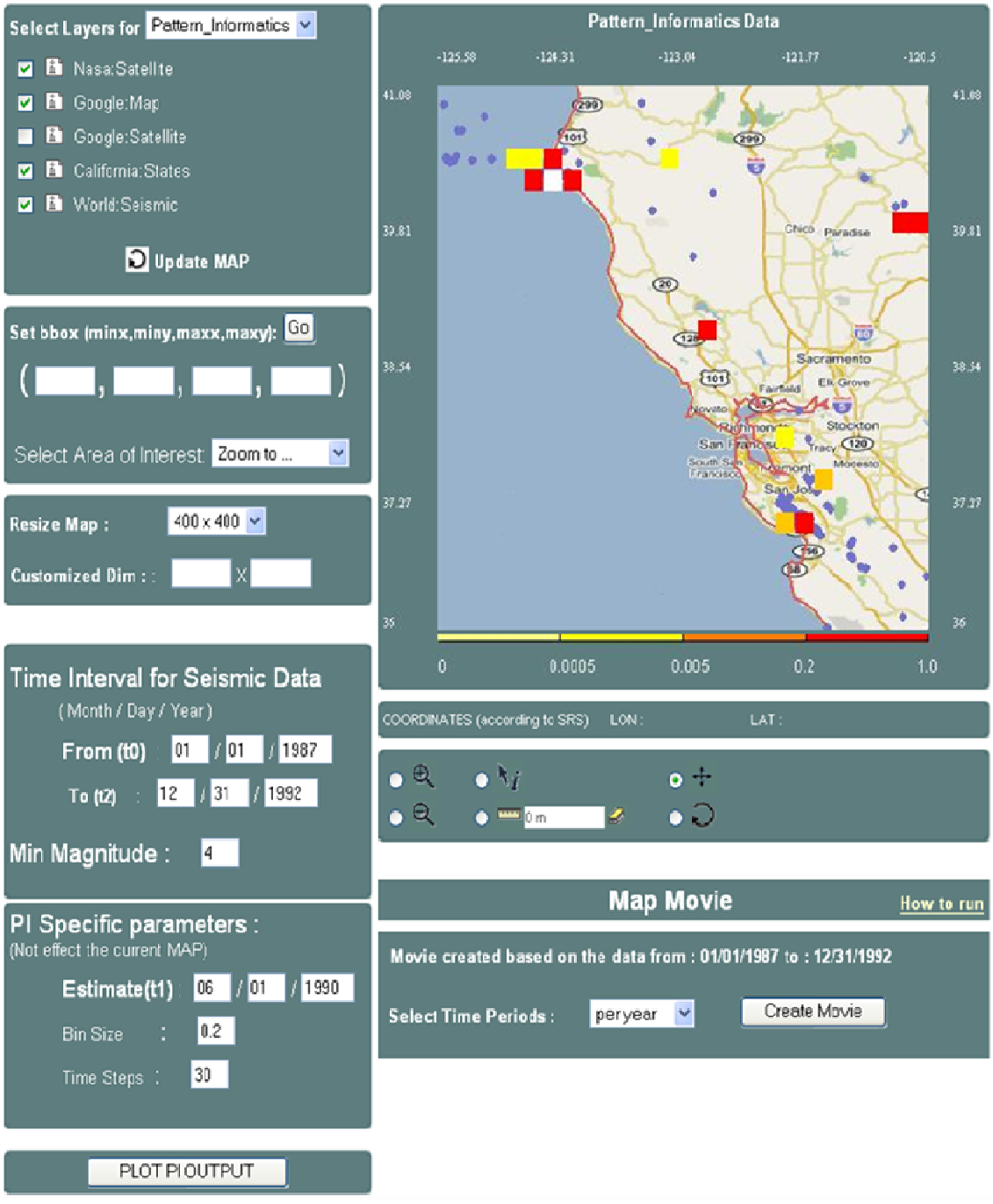

Figure 26: WMS Client or so called event-based interactive map tools. Google Map layer is superimposed by the plotting of the PI outputs. It shows probability of earthquake happenings. Red ones show high probabilities.

We used NASA *OnEarth* Map server as cascaded WMS and get earth satellite image.

In short, we run PI code through the proposed browser/event-based interactive user interface and plot the possibilities of the earthquake happenings in color-coded grid over the previously created seismic and earth map (see Figure 26). Seismic data are kept in WFS and accessed/queried based on the user provided attribute based search criteria.

5.4. Virtual California (VC) Application

VC [Rundle2002] 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 [Quaketables].

VC is a program to simulate interactions between vertical strike slip faults using an elastic layer over a viscoelastic half-space. It relies on fault and fault friction models.

At the application/simulation level, 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. 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 [Donnellan]. 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.

There is no additional component needed besides the components explained before.

*Flow in this architecture is explained here (*Figure 27*):*

a. GIS users interact with the system through the user interface provided by WMS Client and/or GIS Portal. GIS user enters the parameters to get specific region of the world as a map from the WMS server.

b. WMS Client makes a request to the WMS on behalf of the user. It submits a request to the WMS Server by selecting desired features and an area on the map. WMS returns a map in the form of an image or an exception in case of an error.

c. In order to create user specific maps, WMS Server forwards user's request to the WFS to get requested feature data. WFS decodes the request, queries the database for the features and receives the response. Feature data is returned to the WMS server as a set of feature collections.

1. After receiving and displaying the maps returned from the WMS server, the user starts running VC simulation code through GIS Portal. The GIS Portal provides the user with the ability to setup the experiment and the parameters associated with each set of run.

2. The user sets application specific parameters such as bounding box and the time frame of the experiment's data. These values are bundled as script execution parameters and sent to the HPSearch engine.

3. The HPSearch engine then runs the script with the specified parameters. For each run, the service selects an instance of the VC runner service and initializes it.

4. Once all initialization is done, the HPsearch engine invokes the streaming WFS service.

5. The WFS sends the requested seismic records to the VC Runner service. The VC Runner service filters the input data. This step also converts date to float format. Once all the data has been accumulated, the VC Runner service runs the VC code on the input data using the input parameters. Usually each instance of the VC Runner service will work with different set of parameters.

6. The output of the VC runs is stored in output files.

7. On completion the VC runner stores the best cost that was computed per run in the context service. The best cost is the smallest value and will be used for determining the set of input parameters that needs investigated further.

8. The services then notify the HPSearch engine of the completion

9. HPSearch engine queries the context service to retrieve the best cost and then again writes to the context service the location of the output file that corresponds to the best cost.

10. The WMS constantly monitors the context service to see if the computation was completed. Once the computation is complete, it retrieves the location of the output file that corresponds to the best cost.

11. Finally the output file is retrieved (via FTP) and the output is used for visualization purposes.

12. Depending on the data and the geophysics application GIS Portal superimpose returned data as a new layer or makes some animated map or movie streams. In

case of VC application, returned output data is multi-casted to a specific IP and port as movie streams.

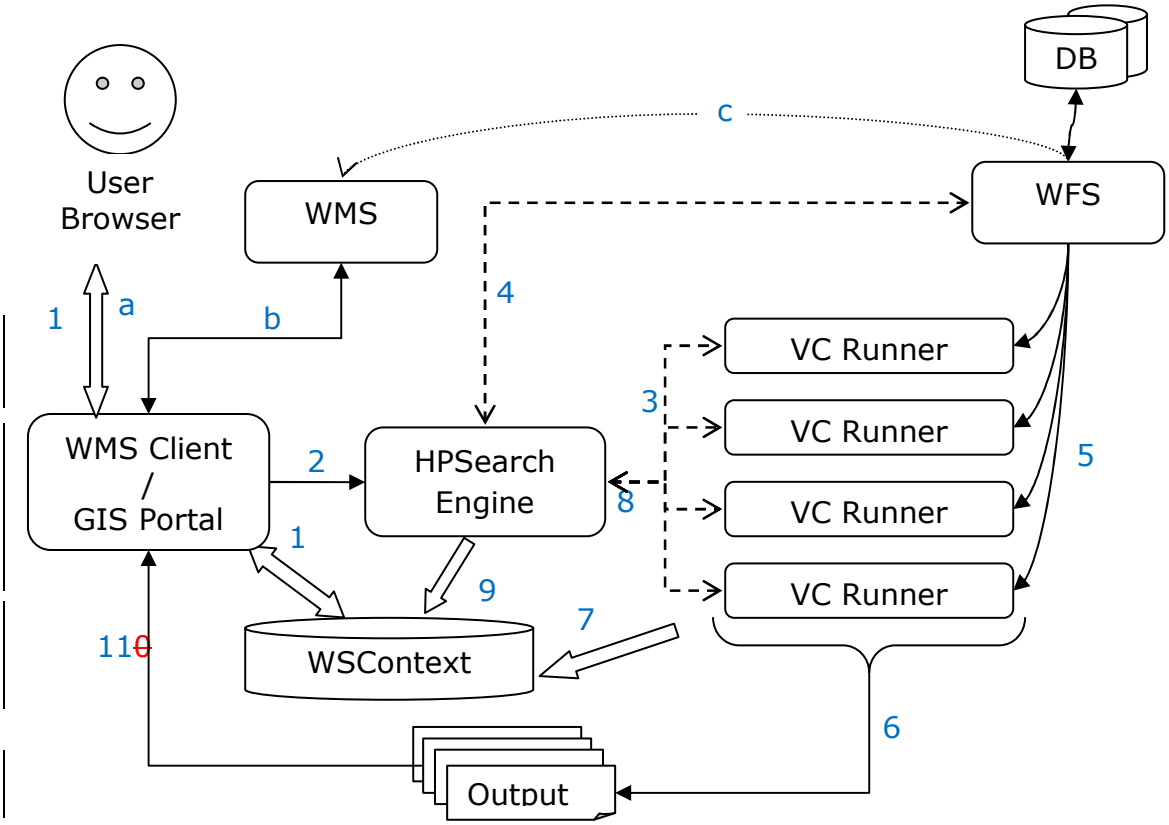

Figure 27: Virtual California Operation steps founded over proposed Service-oriented GIS framework

Outcomes from the VC demo are map movies like animations. Links to a sample movie for Virtual California is listed below.

For this sample case, there are 1144 records in the output file returned by VC Runner Service shown in Figure 27.

http://complexity.ucs.indiana.edu/~asayar/gisgrids/docs/VCDemo_03.swf (Flash version)

http://complexity.ucs.indiana.edu/~asayar/gisgrids/html/work/VC_01.avi (Avi format)

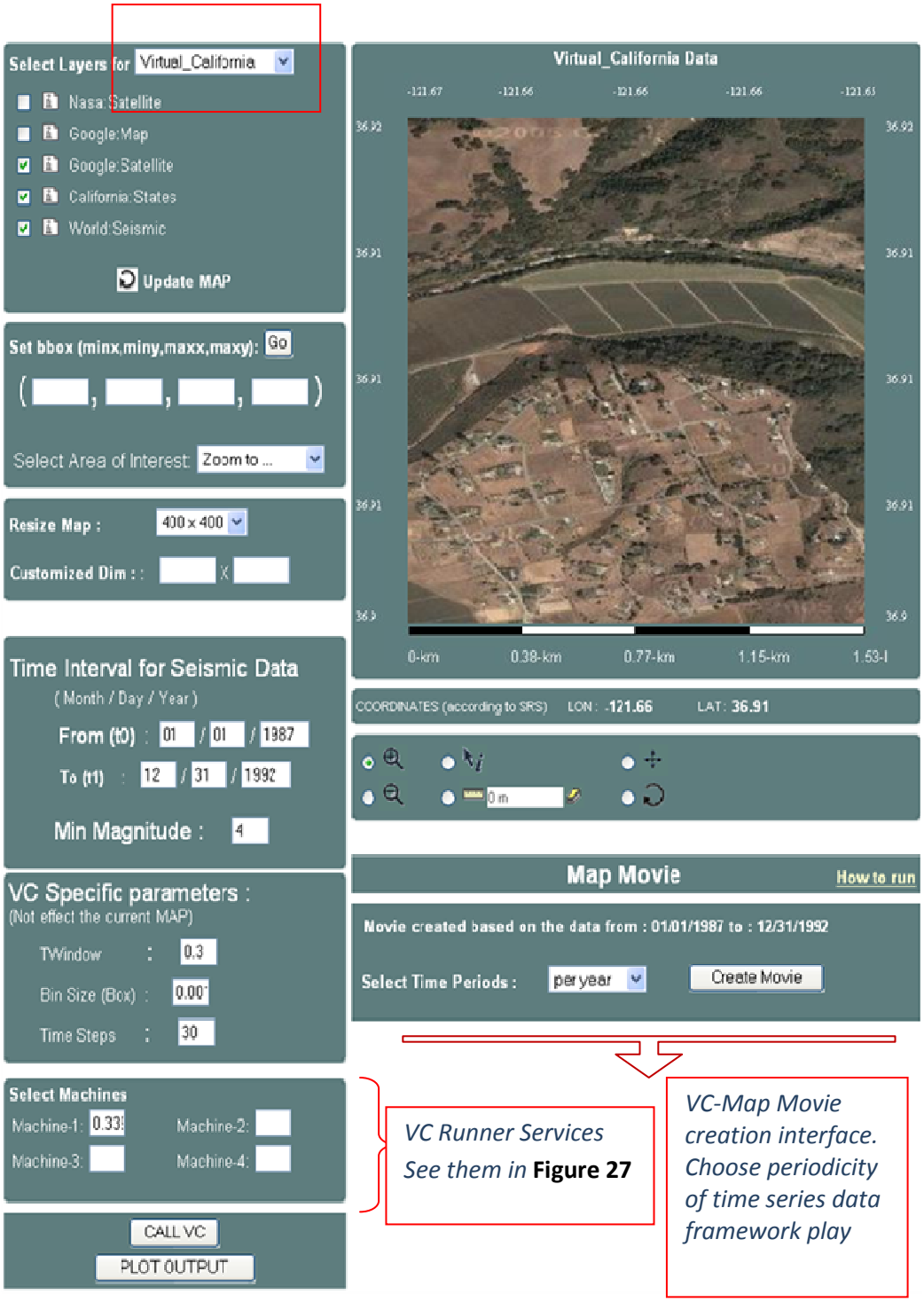

Figure 28: Event-based interactive user interface extended for Virtual California needs. It enables creating map movies by playing framework (created from time-series data) successively. Each framework is actually a map image.

Chapter 6

High-performance Design Features, Measurements and Analysis

This chapter presents the common performance issues and high-performance design features in service-oriented, federated and interoperable Geographic Information Systems (GIS) in which the interoperability is granted by standard Web Service interfaces and structured common data model provided by autonomous and heterogeneous resources. In order to be compliant with open standards to provide interoperable data and services, the system adopted OGC [ogc]'s Geographic Markup Language (GML) [GML] as the common data model, and WMS and WFS as the online standard Web Services.

Interoperability requirements bring up some compliance costs. Open Standards' XML-encoded GML definition and Web Services' XML-based SOAP (Simple Object

Access Protocol) protocol can be given as examples. Our aim is turning compliance requirements into competitiveness.

High performance enhancement approaches are grouped into two. The first group (Chapter 6.3) is related to (1) data transfer between OGC compatible Web Service components and (2) rendering of map images from the large XML-encoded GML data. In this group, we present our extension to OGC's standard service definition for the sake of performance issues. The second group (Chapter 6.4) is regarding the performance enhancement techniques result from the architectural features of the proposed Service-oriented federation framework. Compos-ability nature of the standard GIS data services (WMS and WFS) inspired us developing a federated information system framework enabling enhancement with novel caching and parallel processing techniques.

The proposed high performance design features/techniques will be evaluated by running the system on Pattern Informatics (PI) Geo-science earthquake application developed at University of California at Davis. At the end of each chapter explaining these techniques, performance tests and analysis are provided.

The organization of the rest of the chapter is as follows. Chapter 6.2 summarizes and reviews the general performance issues of interoperable service-oriented GIS systems in which interoperability is granted by using XML-structured common data model and standard GIS Web Services. Chapter 6.3 and Chapter 6.4 present the enhancement approaches, high-performance design features, measurements and analysis.

6.2. General Performance Issues in Interoperable Service-oriented GIS

Performance issues in interoperable service-oriented GIS can be generalized into three groups:

1. Issues regarding distributed nature of federated system.
2. Issues regarding the semi-structured XML-encoded data model (GML).
3. Issues regarding domain specific data characteristics. In GIS, the data is described with location attribute defined in (x, y) coordinates which are called latitude and longitude. Based on the location value, the data is characterized as un-evenly distributed and variable sized. See *Figure 29*-b.

6.2.1. Distributed nature of data sources

Geographic Information Systems are large scale data intensive scientific applications requiring creation of knowledge from distributed data sources provided by autonomous heterogeneous data and computation resources.

Since GIS is mostly used in early warning system and crisis management, it is very important to get the information just in time. On the other hand, data is large and provided by autonomous resources and geographically distributed virtual organizations from different disciplines and skill/expert levels. That makes it very hard to fetch and render the distributed data in a responsive manner.

These general issues will be touched throughout the rest of the chapter. (Chapters 6.3 and 6.4)

6.2.2. Using Semi-structured Data Model

Using semi-structured data model enables interoperability and inter-service communication. XML's emergence as the de facto standard for encoding tree-oriented, semi-structured data has brought significant interoperability and standardization benefits to distributed computing. On the other hand, performance has been still a persistent concern for large scale applications, because of the size issues and processing overheads [Lu2006]. The processing is detailed as parsing and differentiating (separating) the core-data from the attributes and other tags to create required application specific data formats.

Structured data representations enable adding some attributes and additional information (annotations) to the data. These attributes and additions are mostly due to the interoperability and security reasons. Those results in XML representations of data tend to be significantly larger than binary representations of the same data. The larger document size means that the greater bandwidth is required to transfer of data, as compared to the equivalent binary representations. The larger size often implies greater processing costs as well, since much of the overhead involved in communication processing is going to be based on the data volume. Advantages of using structured/annotated data come with its costs.

To be more specific, geographic data is encoded in GML. GML carries content and the presentation tags together with the core data. That enables the data sources to be queried and displayed together (ex. Map images interactively query-able through interactive map tools). Querying and displaying data in GML format needs parsing and rendering tools to extract requested tag elements such as geometry elements to draw map or non-geometry elements to answer content related queries.

There are two well-known and commonly-used paradigms for processing XML data, the Document Object Model (DOM) and the Simple API for XML (SAX). DOM builds a complete object representation of the XML document in memory. This can be memory intensive for large documents, and entails making at least two passes through the data. SAX operates at one level lower. Rather than actually constructing a model in memory, it informs the application of elements through callbacks. This also requires at least two passes through the data. These are all expensive and resource (such as CPU and memory) consuming processes and they don't provide enough performance for the large scale applications.

In the document these issues are called data-oriented performance issues, and the proposed solution approaches are presented in Chapter 6.3

6.2.3. Though Data Characteristics

The different domains have different data types having different characteristic. As an example, in motivating GIS domain, science applications need to manipulate geo-data. Geo-data is described with its location ((x, y) coordinates) on the earth. Based on the location attribute, geo-data is said un-evenly distributed (such as human-population and temperature distributions) and variable sized. In addition, geo-data collected from sensors are dynamically changed and/or updated over time.

These characteristics of geo-data (dynamic nature and un-evenly distributed) make it hard to implement some well-known performance enhancing techniques as applied in other science domains. Since it is not possible to know the work-load earlier, 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 sized. This issue is illustrated in Figure 29 for the case of using one-step-binary query partitioning based on the location attribute of the data. As it is illustrated in the figure, there are four worker nodes, and the worker node assigned to R2 gets the heaviest part of the total work (b), and therefore the expected performance gain from using classic load balancing will not be obtained.

The geo-data is queried based on their attributes. Since all the data is described by their locations, in order to get the data sets falling in a specific region, the bounding box (bbox) values are used. The regions are defined in bboxes. A bbox defines a rectangular shape in a two dimensional coordinate plane, and it is formulated as (minx, miny, maxx, maxy). For example, Figure 29 shows a region formulated in bbox value (a, b, c, d).

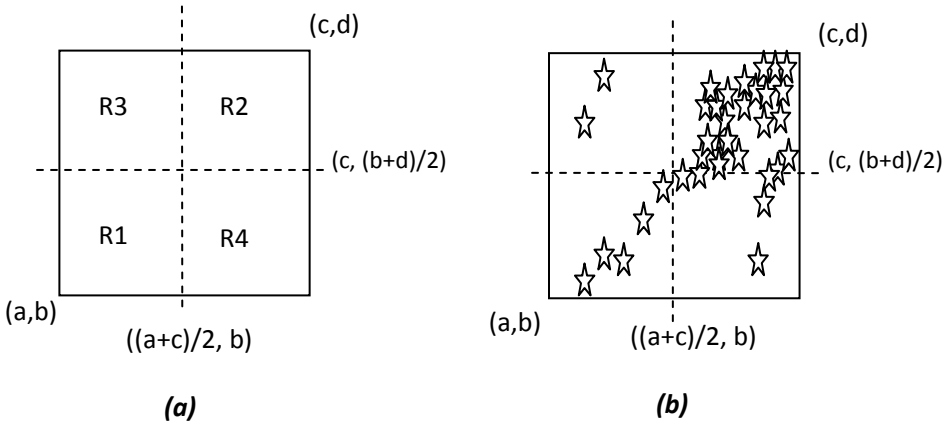

Figure 29: Unbalanced load sharing. Server assigned R2:"((a+b)/2, (b+d)/2), (c, d)" gets the most of the work.

The performance issues mentioned here are dealt with in Chapter 6.4.3.2.

6.3. Data Transfer and Rendering Enhancements

Distributed GIS systems typically handle a large volume of datasets. Therefore the transmission, processing and visualization/rendering techniques need to be responsive to provide quick, interactive feedback. There are some characteristics of GIS services and data that make it difficult to design distributed GIS with satisfactory performance. One of them is that GIS services often transmit large resulting datasets such as structured data, images, or large files in tabular-matrix formats.

In order to provide interoperability and extensibility we use common data format represented and formulated in XML. This degrades the performance even worse for large scale applications. The major hurdle of the proposed federated GIS framework is encoding, transferring and rendering the data in common data model. In the following two sub-sections we present our approaches to these issues. One is regarding large scale structured data transfer (Chapter 6.3.1.1) and other is regarding the large scale data parsing (Chapter 6.3.1.2).

6.3.1.1. Extensions to Open Standards, and Streaming Data Transfer/Rendering

The OGC's standard WMS and WFS specifications are based on HTTP Get/Post methods, but this type of services have several limitations such as the amount of 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 and applications we develop.

Our experience shows that although we can easily integrate several GIS 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 [Donbox] 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 [Pallickara2003] 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.

Naradabrokering is a message oriented middleware (MoM) [Tran] system which facilitates communications between entities through the exchange of messages. This also allows us to receive individual results and publish them to the messaging substrate instead of waiting for whole result set to be returned.

In case of transferring the GML data in the form of string causes some problems related to the performance when the GML is larger than some amount of size. Since the WFS returns the resulting XML document as an <xsd:string>, this has to be constructed in memory and the size will depend on several parameters such as the system configuration and memory allocated to the Java Virtual Machine etc. Consequently there will be a limit on the size of the returned XML documents. For these reasons we have investigated alternative ways for data transport and, researched the use of topic based publish-subscribe messaging systems for streaming the data. Our research on NaradaBrokering shows that it can be used to stream large amount of data between nodes without significant overhead. Additional capabilities such as reliable messaging and

support for different transport protocols already inherent in NaradaBrokering show that it is a powerful yet easy to integrate messaging infrastructure. For these reasons we have developed a novel Web Map Service and Web Feature Service that integrate OGC specifications with Web Service-SOAP [Donbox] calls and NaradaBrokering messaging system. Architecture is shown in Figure 30.

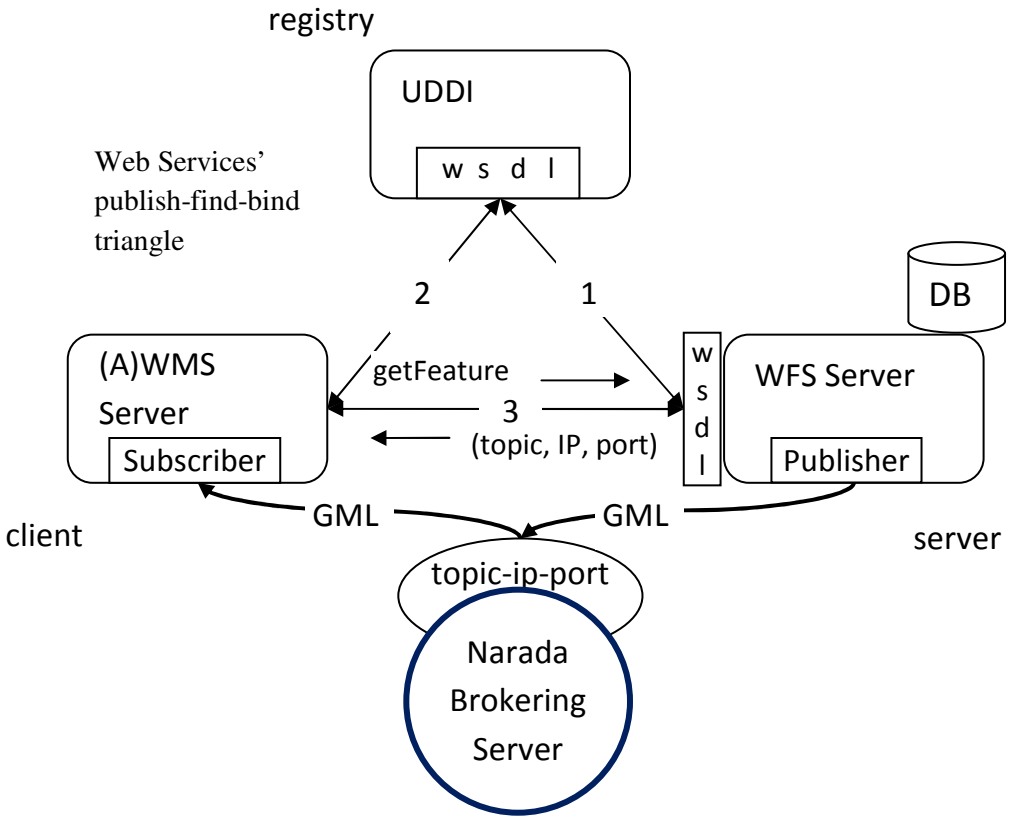

Figure 30: Streaming data transfer using Naradabrokering publish-subscribe topic based messaging middleware.

Connection lines 1 and 2, and UDDI (Universal Description, Discovery and Integration) [Belwood] service are displayed in the figure for showing classic publish-find-bind triangle of the Web Service based Service Oriented architecture. We don't go into details of these interactions and UDDI registry service in this document but these can be summarized as following. WFS services publish their existence and service providing

with their WSDL service description files (line-1). Clients (such as WMS) find appropriate WFS by searching UDDI registries (line-2). After finding appropriate service, clients are bind to that service by creating their client stubs. In case of that client knows what WFS provides the requested data, client can directly communicate with the services without need for UDDI registry service.

After finding WFS providing the requested data, WMS (as a client) make the getFeature request (wrapped in SOAP envelope) to WFS's standard service interface (line-3). As response WMS gets the topic (publish-subscribe for a specific data), IP and port to which WFS streams requested data. The standard Web Service interface is used for handshaking actual data transfer is done between subscriber and publisher deployed in WMS and WFS respectively.

Streaming data transfer through publish-subscribe based messaging middleware enable map rendering even in the case of partially returned data. This depends on the WMS's internal implementation.

Table 3 gives a comparison of the streaming and non-streaming data access approaches for the different data sizes. These values are obtained by applying the proposed framework on Pattern Informatics (PI) [Patterninfo] geo-science application using earthquake seismic data records. These are GML data access times including query conversion at WFS, result set conversion from database to GML and transfer times from WFS to federator or WMS.

As test setup we use Figure 30. Performance response time shown in Table 3Figure 31 are measured end-to-end times in which one end is DB relational tables and the other end is WMS. Naradabrokering agent, WMS and WFS are deployed in Local Area

Network (LAN) in Indiana University Community Grids Labs. In local area network we have used so-called gridfarm machines from gf12 to 19.ucs.indiana.edu. These machines have 2 Quad-core Intel Xeon processors running at 2.33 GHz with 8 GB of memory and operating Red Hat Enterprise Linux ES release.

Table 3: Data access times (from federator or WMS) while using (1) streaming and (2)non-streaming data transfer techniques.

	Streaming			Non-Streaming		
Data Size (KB)	Average Time for Streaming Transfer	Average Response Time	Standard deviation	Average Time Non-Streaming	Average Response Time	Standard deviation
10	31.3	2425	38	1518.8	3912.5	77
30	100	2661	27	1356.1	3917.1	38
100	320.1	2945	50	1473.8	4098.7	71
300	826.7	3405	48	1835.7	4414	39
1000	2414.2	4570	360	3506.8	5662.6	31

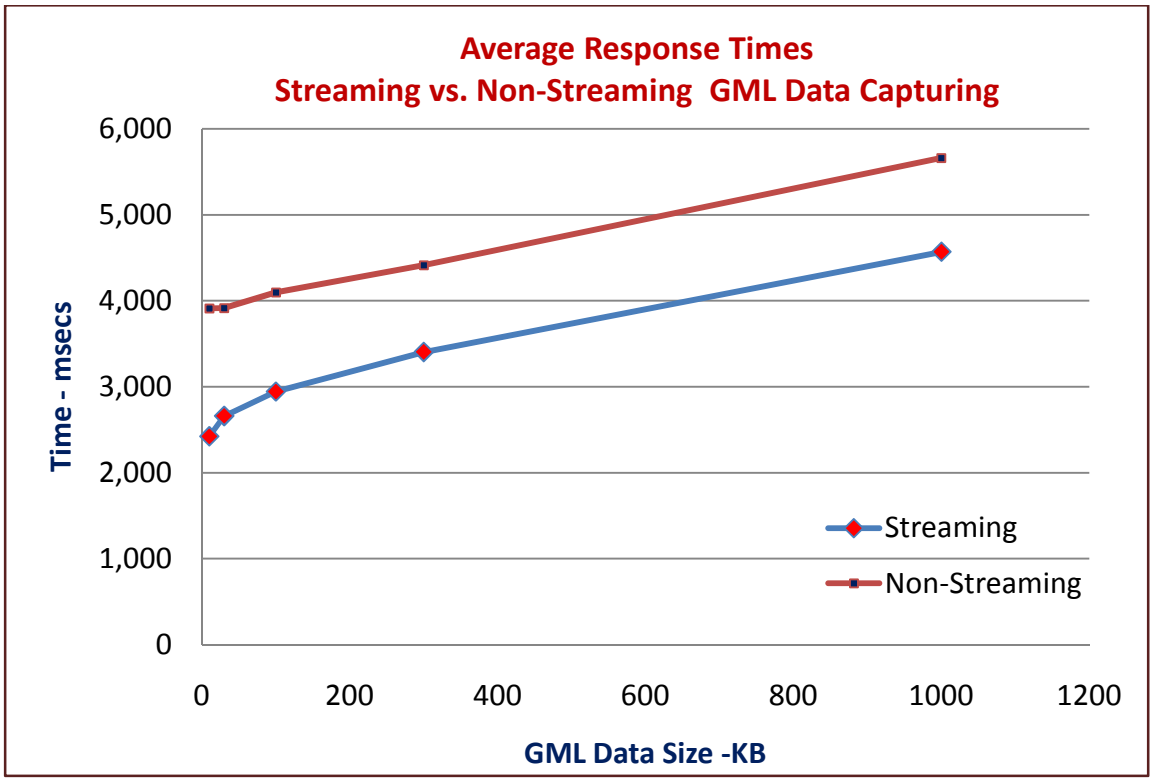

Figure 31: Comparisons of Streaming vs. Non-Streaming data response timings from source to federator or WMS.

We can deduce from the table that for the larger data sets when using streaming our gain is about 25%. But for the smaller data sets this gain becomes about 40% which is mainly because in the traditional Web Services the SOAP message has to be created, transported and decoded the same way for all message sizes which introduces significant overhead.

Besides giving better performance in general, streaming data transfer technique enables data rendering and processing even on partially returned data. It can even be applied to the real-time data rendering.

6.3.1.2. Data Rendering through Pull Parsing Technique

Proposed system includes data rendering/filtering tasks assigned to Web-based Map Services to create comprehensible data representations derived from the semi-structured common data (GML). These comprehensible representations are called maps. Regarding the rendering of large GML data and creating map images we use parsers.

There are three general parsing techniques proposed for processing XML structured data. These are document model, push model and pull model. There are also other hybrid alternatives built on these main approaches. In order to process data in XML structured common data model we use pull parsing technique.

Pull parsing, as exemplified by the XML Pull Parser [Alexander], is an efficient paradigm similar to SAX in that it does not build a complete object model in memory. It differs in that the tags and content are returned directly to the application from calls to the parser, rather than indirectly in the form of callbacks. 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 parser only parses what is asked for by the application rather than passing all events up to the client application as SAX parsing does. You can see the article where pull parsing is compared with other leading Java based XML parsing implementations [Sosnoski].

Pull parsing does not provide any support for validation. This is the main reason that it is faster than its competitors. Since all the services are OGC compatible and

created in Web Service principles, validation is not necessarily needed. 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 the 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 it is known that WFS provides feature data in OGC's GML format [GML], it is very advantageous skipping validation and using pull parsing.

For the application specific comparison of Pull parsing and DOM see Table 4 and Figure 32. The performance values are measured in milliseconds and data sizes are in MBs. Performance test is done with 1GB allocated JAVA Virtual Machine. The Figure 32 illustrates the timing values for the data size less than 100MB of GML data. Above this threshold value for the Virtual Machine allocated 1GB memory, DOM become useless.

Test case: For the XML data we use earthquake seismic data records encoded in GML. Each earthquake seismic record has some attributes and some geometry elements. In our tests we will parse the GML data in XML documents and extract the geometry elements. In case of DOM, parsing and extraction are done separate as it is shown in two columns in Table 4. In case of pull parsing, geometry data is extracted from GML with parsing and extraction applied all together.

Results for the DOM and pull approach are obtained by using dom4j and xpp respectively. Xpp is developed in Indiana University Extreme Labs. The experiment

performed in a single computer, utilizing Pentium 4 CPU operating at 3.4GHz with 1.00 GB of memory.

Table 4: The performance values of DOM and Pull parsing (Xpp) over GML data. Dashed-line values imply memory exception.

Average Timings

Data (KB)	DOM (dom4j)			Pull (Xpp)		
	Parsing + **Validation**	Data Plotting	**Total Rendering**	Data Extraction	Data Plotting	**Total Rendering**
1	469.22	0.00	**469.22**	15.59	0.00	**15.59**
10	494.06	3.00	**497.06**	72.81	3.00	**75.81**
100	625.54	15.33	**640.87**	183.06	15.33	**198.39**
1,000	760.20	83.11	**843.31**	270.47	83.11	**353.58**
5,000	1,422.91	153.67	**1,576.58**	671.74	153.67	**825.41**
10,000	3,557.44	828.50	**4,385.94**	1,025.67	828.50	**1,854.17**
100,000	----	----	----	7,059.72	3738.25	**10,797.97**

The dashed lines in Table 4 represent not-enough memory exceptions. It means the system does not have enough memory for completing its work with 1GB of allocated virtual memory in JAVA virtual machine. Since there is extreme performance difference between using DOM and pull parsing techniques, we plot their values in Figure 32 for less than 10MB of GML data.

Table 5: Standard deviations of average timings for total rendering

Data Size (KB)	Total Rendering	
	DOM-dom4j	Pull-(Xpp)
1	21.32	0.87
10	20.87	7.41
100	28.04	23.25
1,000	41.58	65.09
5,000	72.66	121.05
10,000	126.51	116.49

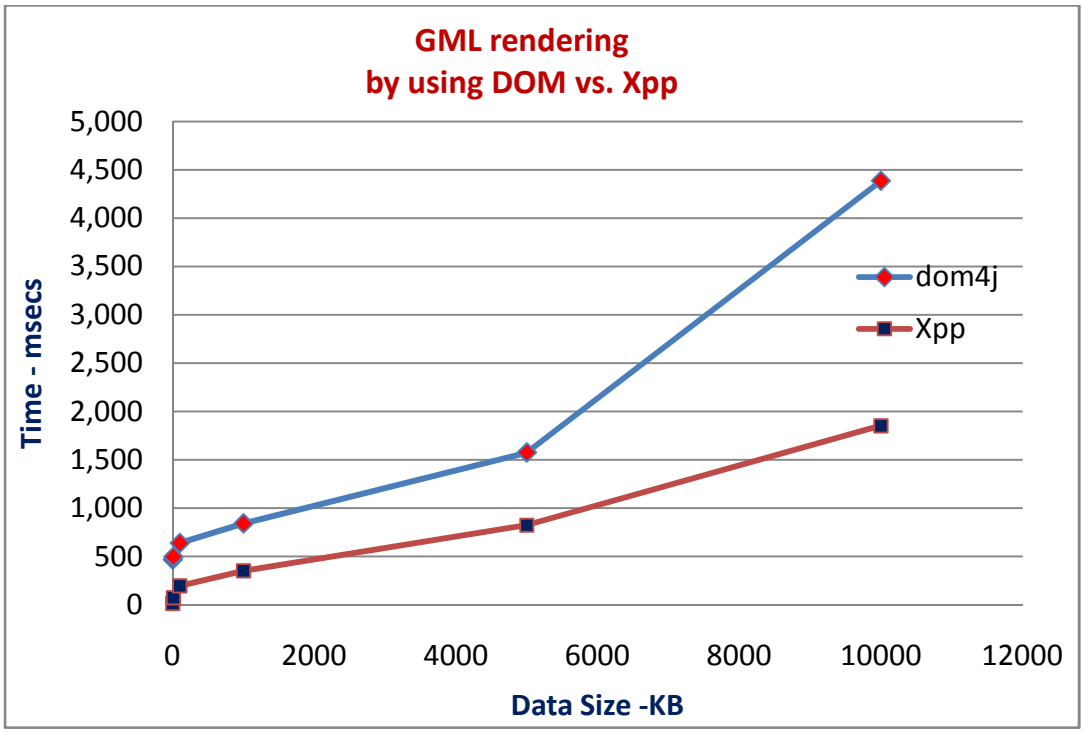

Figure 32: Performance comparison of two XML data processors, pull parsing and Document Object Model by using dom4j.

6.3.1.3. Analyzing the enhancement approaches in a complete System

This chapter presents overall performance gains obtained by applying data-oriented performance enhancement techniques mentioned in the previous chapters (Chapter 6.3.1.1 and 6.3.1.2s).

Test Setup: We run the test in Local Area Network (LAN). WMS, WFS, federator server and event-based interactive map tools are deployed in Indiana University Community Grids Labs. In local area network we have used so-called gridfarm machines from gf12 to 19.ucs.indiana.edu. These machines have 2 Quad-core Intel Xeon processors running at 2.33 GHz with 8 GB of memory and operating Red Hat Enterprise Linux ES release.

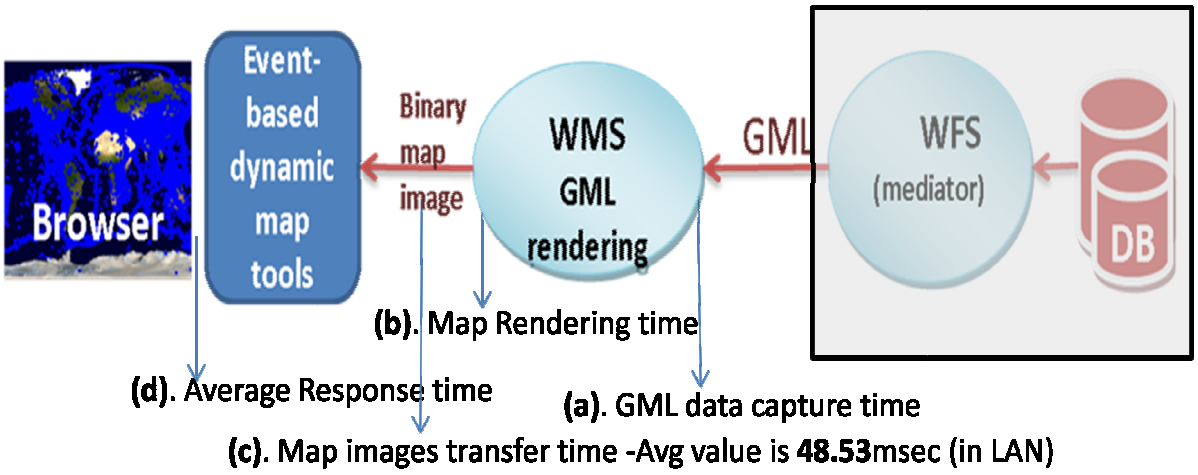

Figure 33: Geographic Information System components and data flow. d = a + b + c

Figure shows the proposed framework and data flow and detailed performance evaluation steps. These steps are summarized as

a. Data fetching from database as GML through WFS

b. Map-image creation – parsing/rendering of GML

c. Transfer of map images to event-based map tools.

d. Overall response time –map display at client's browser

The average response times shown in the figures include times for querying, transforming, rendering and displaying spatial data. The average response time is formulated as below:

$$d = a + b + c$$

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

The Measured response time can be further detailed as below (also see Figure 33 simultaneously):

- **[**$time_{(client\ makes\ request)}$**.]** Client makes requests through the interactive smart map tools.
- WMS parse and render requests and define set of actions required based on the requests and its capabilities file.
- WMS Creates map images (from the returned datasets) and returns them to the clients. This step is also detailed as below:
    - Defines the set of WFSs [WFS] and other WMSs [WMS] to communicate with to build the response in accordance with its capability file and client provided parameters.
    - Creates requests for WFSs and other WSMs
    - Invokes WFSs' *getFeature* Web Service interfaces for vector data encoded in Geographic Markup language (GML) [GML].
    - Streaming GML transfer through Naradabrokering messaging middleware from WFS to Federator/WMS

- Parsing and rendering returned GML data sets
- Aggregating and overlaying layers according to the request and capability file.
    - Such as Satellite map images from NASA OnEarth WMS
- Sending the map images to the WMS Client.

- **[*time*$_{(map\ is\ displayed)}$]** Client shows the returned maps on his browser

Table 6 and Figure 34 present performance values obtained by running the test setup given in Figure 33. Sample test setup uses earthquake seismic data records stored in databases and provided as GML by WFS. Earthquake GML data records are plotted over NASA satellite map layer received from NASA OnEarth WMS server. Since it is not important in the context, we have not shown NASA WMS in the above figure.

Table 6: The performance results in average timings.

	Average Timings				
Data KB	Data Fetching	Map Rendering	Total time Map Creation*	map images' transfer time	Response time for end-users**
10	797.85	927.35	1741.17	61.88	1808.13
100	1384.86	1168.29	2567.35	62.22	2635.46
500	3770.16	1153.96	4934.94	60.15	5001.29
1000	6794.94	1360.41	8155.35	68.38	8225.73
5000	31237.41	2116.12	33350.80	70.26	33419.31
10000	61777.20	2675.87	64441.96	62.15	64506.78

** Total time for map creation = WFS to WMS data capturing + Map Rendering.*
*** Response time for end-users = Total time for map creation at WMS + map images transfer time to end user*

Table 7: The standard deviation values for the average timings given in Table 6

	Standard Deviations				
Data KB	Data Fetching	Map Rendering	Total time Map Creation	map images' transfer time	Response time for end-users
10	48.39	123.29	132.36	26.33	140.32
100	73.86	383.61	384.61	21.90	313.48
500	80.81	230.33	234.03	20.74	238.94
1000	93.24	207.60	199.49	24.59	200.27
5000	211.45	346.06	432.43	22.19	394.48
10000	152.54	252.97	279.04	18.64	283.24

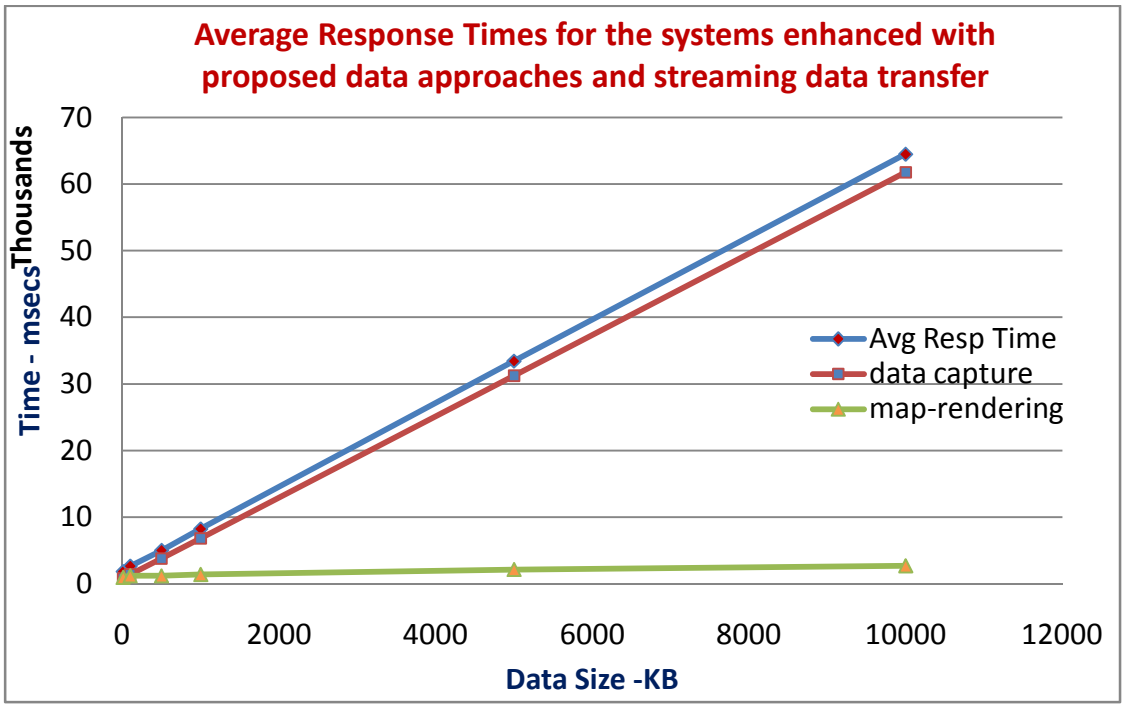

Figure 34: Average response, data capturing and map rendering timings for different data sizes.

Systems developed with Open Geographic Standards have high degree of interoperability and autonomy but poor performance results. As it is shown from the figure the system performance bottleneck is GML data fetching from distributed autonomous resources integrated to the system WFS-based mediation. It is mostly because of query and response conversions fulfilled in WFS to integrate the heterogeneous data sources in common data model GML.

This performance results teach us valuable lessons in terms of the capabilities and limits of the general distributed and interoperable GIS systems. From the figure we draw the following conclusions. First, for the small data payloads (less than 500KB) the response time is acceptable. However for larger data payloads the performance gets worse and the response time gets relatively longer. On the other hand, scientific applications require handling (transferring, parsing, rendering and displaying) large scale data.

From our experience we saw that most of the response time consists of on-demand GML data fetching. For example for 10MB of GML (in Figure 34), %96 of total response time is explained by data fetching. Therefore, even if we use the most efficient and fast parsing and rendering algorithms, it won't improve the performance very much as long as the data transfer time still stays that much high.

In order to improve performance results further and remove the performance bottleneck of remote data fetching through WFS-based mediation, we propose federator-oriented novel caching and parallel processing techniques applied together. Federator-oriented performance enhancing designs are developed over data transfer and rendering enhancements presented in Chapter 6.3. Figure 34 showing the overall performance

results will be used asbase-line and measurement to evaluate the success of the federator-oriented performance enhancement approaches presented in the following chapter.

6.4. Federator-oriented Performance Enhancement Approaches

OGC Open Standard's GIS services are inherently stateless and based on on-demand processing. They don't maintain state information between message calls. In order to adapt and use them in stateful fashion, we introduced a federator service over the OGC's WMS and WFS data services. Federator is actually a WMS developed in Open Standards and extended with federating and stateful service capabilities. It actually makes view-level federation of their data providing in service oriented framework. You can see Chapter 4 for more information.

The federator in the proposed federated GIS system inherently enables load balancing and parallel processing and this helps with enhancing the overall system performance. This chapter presents the techniques and system design to develop high performance federated GIS system through the federator.

We group the federator-oriented approaches into two. One is Pre-fetching which is an approach mediating distributed approach into central approach through federator. Federator behaves as proxy for other distributed data sources by collecting their data in ready to use common data model (GML) in periodic intervals, and successive queries are responded centrally from federator's disk space in which pre-fetched GML data sets are stored. Pre-fetching approach is presented in 6.4.1.

Second group of approach is pure distributed approach which we call it on-demand fetching and rendering, and presented in Chapter 6.4.3. The main idea is making stateless GIS services state-full by developing client-based caching and parallel data fetching. Since the data is kept only in its originating sources and not stored in intermediary places, architecture provides consistency and strong autonomy.

These techniques will be explained in the following sections with their performance evaluations and analysis.

6.4.1. Pre-Fetching (Central Approach over Distributed Sources)

Pre-fetching is briefly defined as getting the data before it is needed. Pre-fetching is used to overcome the performance bottleneck of the proposed federation system. As it is mentioned before (in Chapter 6.3.1.3) the bottleneck is transferring large sized data from source (database) to destination in common data model GML. On-demand access to originating databases through WFS is very costly. It is not only because of the requirement of moving the large data but also query and response conversions.

In order to solve that bottleneck problem, it periodically fetch/update whole data in databases into GML sets and store it locally in federator. Successive queries are served from the pre-fetched data in federator's local disk. To reduce the inconsistency problem, fetching module fetches and updates the data periodically. The fetching is done according to the proposed streaming data transfer technique explained in Chapter 6.3.1.1.

It indirectly enables getting rid of the query/response transformation overhead at WFS. As it is mentioned before, WFS serve heterogeneous data sources in GML format with standard interoperable Web Service interfaces.

6.4.1.1. Architecture

Architecture is consisted of two parts. First part is pre-fetching which is independent of run time application. It means on-demand data accesses from the client are not affected by pre-fetching. The second part is users/clients on-demand access which is served from pre-fetched data. Architecture is given in Figure 35.

As in the proposed data exchange framework defined in Section 6.3.1.1, the pre-fetching module make the requests with standard SOAP messages but for retrieving the results a NaradaBrokering subscriber class is used. Through the "*getFeature"* interface of WFS Web Services, pre-fetching module 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 [Vretanos]. 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. In order to do that, we define the "task" and "timer" (see Chapter 6.4.1.2). Task defines pre-fetching job, and timer defines the running periodicity of the task. Different data might have different periodicities set.

There will be two separate locations for the pre-fetched data. One is temporary into which pre-fetched data is stored. Another is stable which will be used for serving the clients' requests. Even if the system is busy with the pre-fetching job, it keeps itself up and running for the clients by using the stable storage. 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 is repeated at some time intervals pre-defined by periodicity parameter of Pre-fetching Module (PM).

Since the data fetching is done independent of real application time, it does not affect the real time application performance.

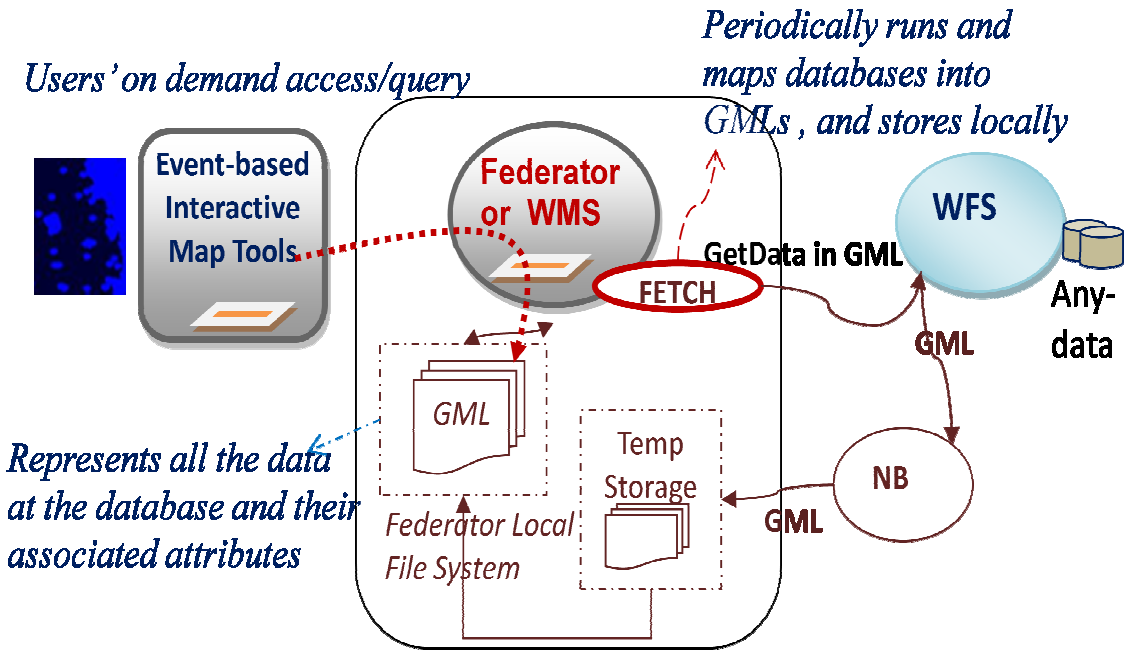

Figure 35: Pre-fetching architecture embedded to the federated GIS system

In order for the pre-fetching algorithm to work properly, pre-fetching module fetches the data as a whole; no constraint should be defined in the query. On the other hand, the requests from clients contain some query constraints. These queries and their constraints are handled at the A WMS side. Queries are processed by using parser techniques and XPATH queries over the pre-fetched data.

6.4.1.2. Fetching Module

The pre-fetching module is composed of two components. One is “timer” defining the periodicity that PF will be running, and other is “task” defining what to do. The periodicity should not be less than the time to transfer one set of critical data. Assigning a periodicity at PM is the most critical task. This is defined under the considerations of data characteristics and developer’s experience on the domain specific application.

Since the system is developed in JAVA, we use Timer and TaskTimer JAVA class libraries to implement the routinely running pre-fetching module.

Here is the “task” defined in a pseudo code:

```
public void pseudo_TASK() {
    Vector CDMdataList = new Vector();
    CDMdataList = getListPerformanceCritical_GMLDataNames();
    String tempDatastore = applpath + "/prefetchedData";
    String stableDatastore = applpath + "/prefetchedDataUsed";
    //Fetching all the data in CDM format (GML) - with NB
    fd.FetchDataWithStreaming( NBip,NBport,NBtopic,
                        wfs_address,tempDatastore,CDMdataList );
    //After pre-fetching is done move the data to stable storage
    fd.moveData(tempDatastore, stableDatastore);
}
```

We also define timer determining the periodicity of task to run. The below sample code sets the periodicity of “task” defined above to 3 days. It means PF will be running once every three days.

```
Timer timer = new Timer();
timer.schedule(task, 0, 40000);
```

Timer class schedules the specified task for repeated fixed-delay execution, beginning after the specified delay. Subsequent executions take place at approximately regular intervals separated by the specified period.

There are two concerns in developing an efficient pre-fetching architecture. First one is limited storage capacity for a node. The size of the pre-fetched data is constrained by local node's storage capacity. Second one is regarding the pre-fetched data characteristics. Some archived data is updated so often that they look like real-time data. In that case, pre-fetching becomes unfeasible and cannot be benefited. For this type of data (archived but updated frequently), we propose a novel parallel processing approach applied together with the caching (see Chapter 6.4.3.2).

6.4.2. Evaluation of Pre-fetching

We test the pre-fetching technique over the proposed federated GIS system by using real-world Pattern Informatics (PI) earthquake geo-science application (see Figure 35). PI is an earthquake forecasting application and uses archived earthquake seismic records stored at WFS as feature collections encoded in GML. Whole earthquake records are 127MB of GML data.

The performance results for the data fetching are given earlier in Figure 34, here we give performance results for the response times when requests are served from pre-fetched data (Figure 37 and Figure 38). We also compare response times over pre-fetched data with response times on-demand fetching (Figure 39).

As test setup for the performance measurements, in case of on-demand fetching approach, one end is database and other end is user (see Figure 35).

In case of pre-fetching evaluations, one end is federator and other end is user (see Figure 33 and Figure 36).

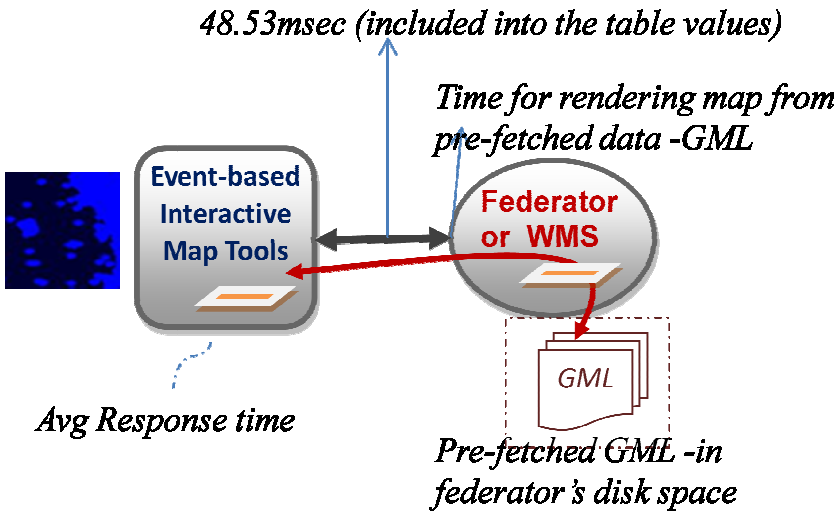

Figure 36: Test setup for testing performance results of on-demand map rendering from pre-fetched data.

Table 8: Performance results for the response times when the pre-fetched data is used in its original form GML.

GML Data Size MB	Average Rendering	StdDev	Average Transfer	StdDev	Average Response	StdDev
0.01	19,215.60	477.71	46.30	15.39	19,261.90	481.57
0.1	19,040.74	670.65	71.57	29.74	19,112.30	673.69
0.5	19,191.24	630.50	31.24	8.30	19,222.48	631.35
1	19,387.64	307.45	39.84	10.01	19,427.48	305.94
5	20,107.54	514.46	38.46	10.66	20,146.00	516.50

10	20,113.19	548.52	52.71	27.13	20,165.90	546.53
50	22,830.33	505.86	52.19	15.88	22,882.52	509.98
100	22,934.52	598.25	55.90	12.66	23,990.43	603.59

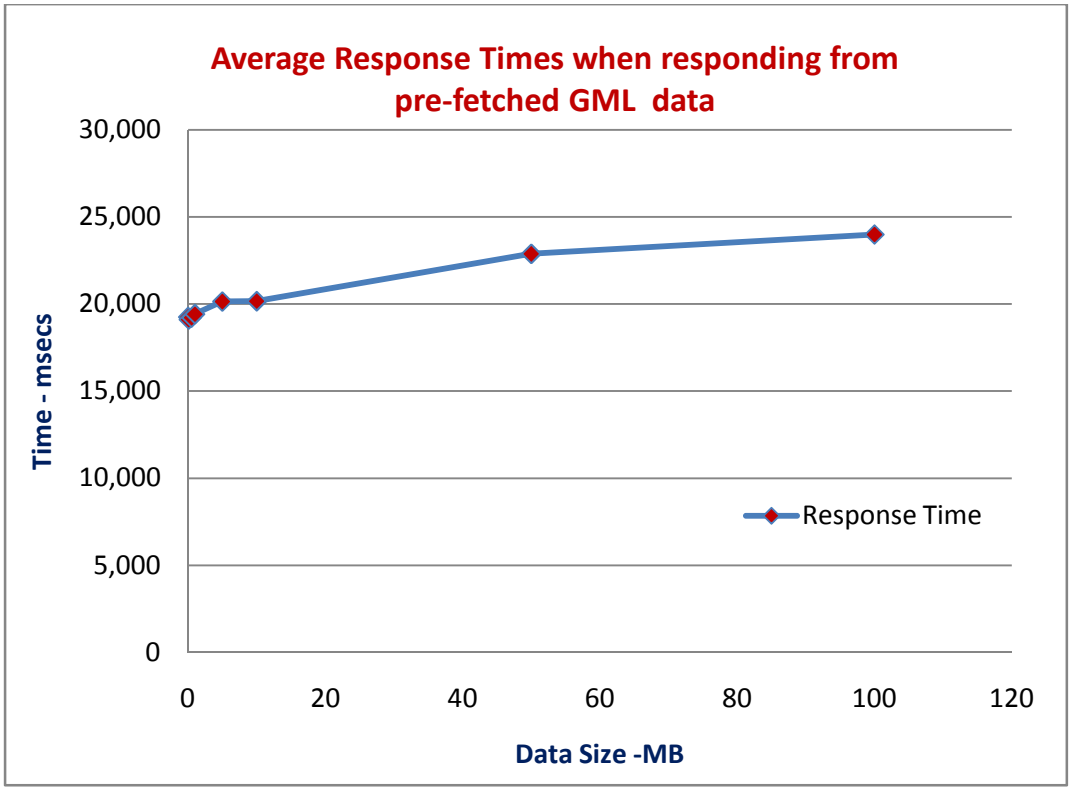

Figure 37: Performance of the pre-fetching technique. Pre-fetched data is kept as GML

As you see in Figure 37, the response times for using pre-fetched data seem very close. This is because of the time spent on parsing and extracting geometry elements from GML data which is dominating time in overall response times. Since the pre-fetched data size does not change (as long as the data in corresponding database does not change), successive on-demand queries for maps are always responded from the same size of GML. Since we use earthquake data records in our sample test case, every time

request comes (even for 1KB data) map is rendered from 127MB of GML data. In order to prevent these repeated times we went one step further and created a data structure from GML that keeps all kinds of geometry elements ready to use for plotting and that removes dominating parsing time. For the 127MB of GML data static dominating parsing and data extraction time is average 11.69sec.

When we get rid of that part by using appropriate structure providing plotting values we get better results as shown in Table 9 and Figure 38.

Table 9: Comparison of the pre-fetching and on-demand response times when rendering is done over processed GML.

Data Size MB	Average Response of Pre-fetching	StdDev	Average Response On-demand	StdDev
0.01	7,410.46	227.71	1,808.13	140.32
0.1	7,426.86	420.65	2,635.46	313.48
0.5	7,537.04	380.50	5,001.29	238.94
1	7,742.04	257.45	8,225.73	200.27
5	8,360.56	264.46	33,419.31	394.48
10	8,480.46	298.52	64,506.78	283.24
50	11,197.08	255.86	316,906.00	623.08
100	12,304.99	348.25	643,344.00	548.65

As shown in the table threshold value is 500KB of data. For large size of data pre-fetching works much better. For example, for 100MB of data, the pre-fetching is about 50 times faster than on-demand fetching. The larger the data size the higher the performance gains.

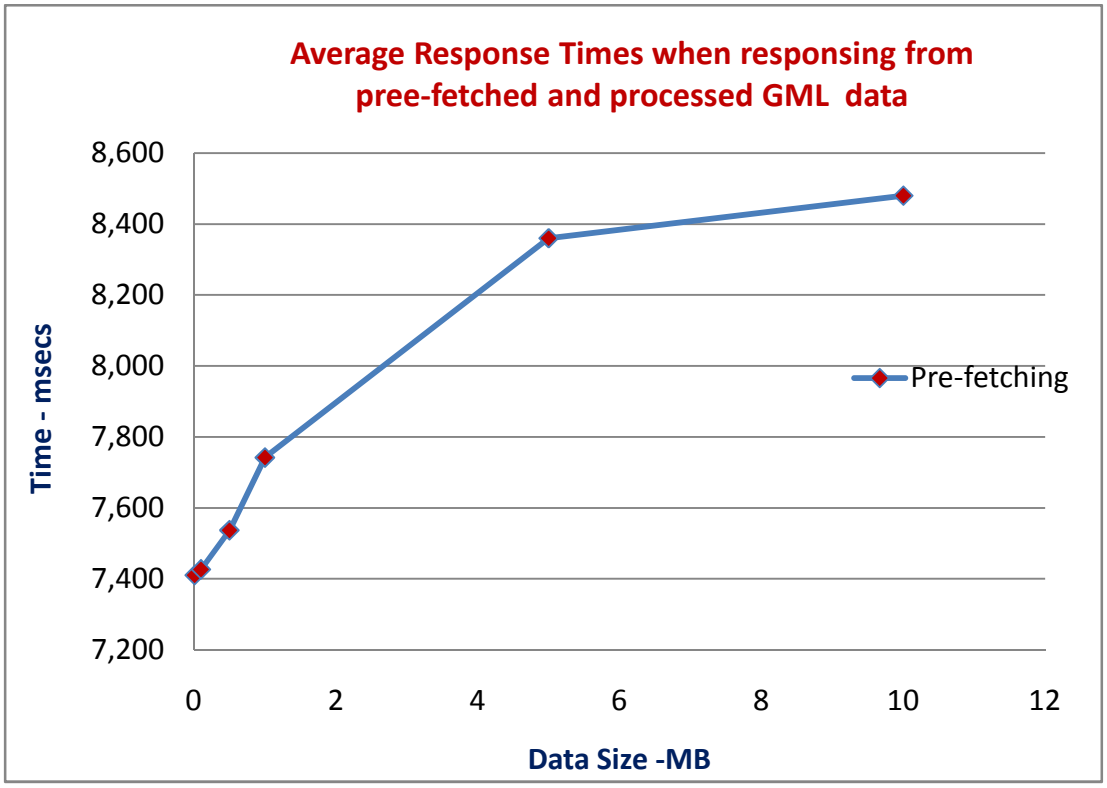

Figure 38: Figure 39: Performance of the pre-fetching technique. Pre-fetched data is kept as geo-elements (processed GML)

For 100MB of data request, pre-fetching is about 50 times faster. The larger the data size the higher the performance gains. Dominating performance bottleneck is removed by pre-fetching the data. Successive on-demand queries are responded from federator's pre-fetched GML data, no need to go through the WFS to get the data from database.

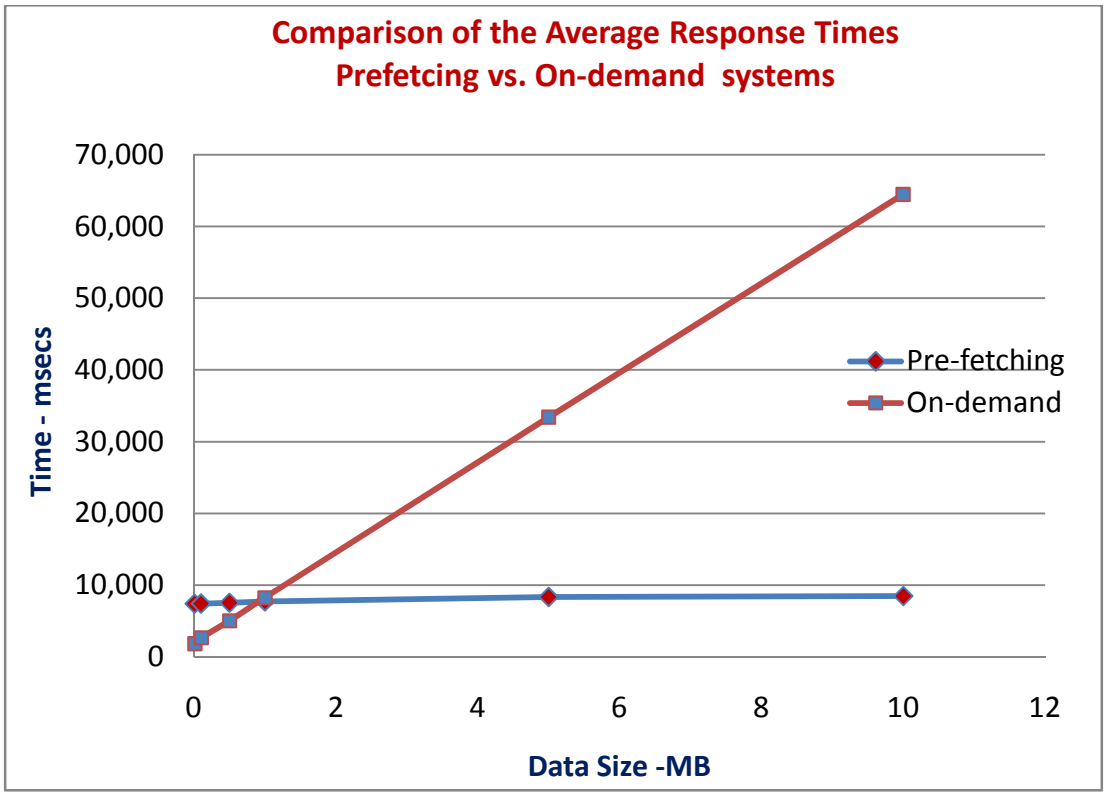

Figure 39: Performance comparison of the map rendering in the proposed GIS system with pre-fetching and ordinary ways.

As it is expected the pre-fetching increased the performance and responsiveness of the system for accessing, querying and rendering archived data. Compared to on-demand fetching (ordinary), pre-fetching removes the times spent on query/response conversion at WFS (getFeature request to SQL and relational tables to GML) and transferring GML.

Our criterion for selecting the technique to apply depends on two measurements. One is the minimum time required to fetch a whole critical data from the source (defined in pre-fetching runner's timer) and another is the time periodicity in which data is updated in its storage. If the data changes less than a time periods in which whole critical data is fetched, then the data is called frequently changing.

The pre-fetching technique gives the best performance outcomes for archived infrequently changing data, but might cause inconsistency depending on the fetching periodicity and data update periodicity in the originating source. In order to remove inconsistency risk totally we propose using data from its originating resource and making just-in-time fetching/rendering presented in the following chapter.

6.4.3. On-Demand Fetching/Rendering (Distributed)

On-demand (just in time) fetching is decentralized approach enabling autonomy and scalability. Autonomy in this context, means keeping the data in their originating sources all the time. Originating sources are autonomous and control their own data definitions. Autonomy indirectly results in scalability and enables easy data maintenance. Besides having many advantages of being decentralized, on the other hand, it has also performance drawbacks coming from accessing/querying the heterogeneous data sources through WFS-based mediation.

In the following chapters we present enhancement techniques to reduce the negative effects of time consuming query/data conversions and data transfer latencies. We focus on the issues at the upper level of data handling. We are not proposing enhancement over query and/or response conversions at the autonomous resources integrated through mediators (WFS). Our enhancement approaches are at the federator and view-level.

Enhancement approaches are summarized in two closely related groups. One is adaptive client-based caching (Chapter 6.4.3.1) and other is load balancing through query

decomposition/partitioning and parallel processing (Chapter 6.4.3.2). The last chapter is about the analysis and evaluation of the proposed techniques.

6.4.3.1. Adaptive Client-based Caching

OGC Open Standard's GIS services are inherently stateless and based on on-demand processing. They don't keep information regarding the previous requests/responses, and that causes poor performance results in response times because of redoing the time and resource consuming repeated jobs. In order to make stateless services interaction state-full, we proposed an adaptive client-based caching.

The idea is based on allocating separate chunk of caching area to each active client and serving each client from its own allocated area. Client's cache is kept up-to-date as in working window concept in operating systems. Server differentiates the clients based on their IDs defined in the request.

In this context, we use client and session interchangeably. One client might have more than one session by assigning different IDs to his messages to the server. For example, when event-based interactive mapping tools are used, those IDs are assigned automatically whenever user opens a new browser.

We introduced this novel idea for performance reasons. It removes the repeated jobs and helps efficient load balancing over the un-predicted workload by utilizing the locality [Denning] and nearest neighborhood [Belur] principles. It also helps us finding out the best efficient number of partitions for parallel processing and reducing the overhead timings for handling unnecessary number of partitions. Locality principle in this context is explained as following. If a region has a high volume of data, then the regions in close neighborhood also expected to have high volume of data. The simplest example

to give is the distribution of human population data across the earth. The urban areas have higher human population than the rural areas, and oceans (2/3 of the world) have no human populations etc.

For large scale applications it might be impossible to cache whole data at intermediary servers to lower the response times. Furthermore, keeping data at different places force application developers to be more careful to keep the data consistent. It also brings maintenance and handling costs to the service administrators. Instead of doing this, we propose a selective client-based dynamic caching. In the following chapter we explain the architectural details about how to develop such a framework.

6.4.3.1.1. Architectural Details

Architecture is based on recently used data sets and clients requesting them. The research issues this chapter deals with are summarized as (1) how server differentiate the clients and (2) what to cache and how to cache.

What to cache: Maps are composed of multiple layers and each layer is created from different data set such as satellite map layer, state boundaries layer and earthquake-seismic layer. The proposed caching is applied to the selected layers. These layers are defined as critical in server's properties file.

How to cache: each critical data is cached in the common data model (GML format) instead of ready to use image tiles. The reason behind this is that the proposed GIS framework allows attribute-based querying/display and data-mining. It is not just for displaying based on location attribute. In order to accomplish this, the data/layer needs to be cached with its geometry and non-geometry elements together with the core data. By

doing this, even if client changes its queries in terms of attributes system utilizes the cached data as long as queries and cached data bboxes are intersects.

For each separate session (differentiated by their IDs defined in the request message), there will be separate set of cached data. Cached data is upgraded at every request from the same session.

Since the proposed federated GIS is interacted through browser-based interactive decision making tools over the integrated data views, the remaining of the chapter first gives the details about how to set browser-based session ID to the SOAP message and forward it to the server, and then how to keep track of separate clients' session information at the server.

The proposed interactive event-based client tools are developed in Apache Tomcat [apache] Servlet container and pages are developed with Servlet and Java Server Pages (JSP). JSP defines session ID whenever a user opens a page to interact with the federated GIS system. A session is normally stored in a cookie which is available to all windows in the browser. The system access this ID by *session.getId()*. This returns a string unique user ID (uuid) which can be used for application specific purposes.

Whenever federated GIS client interacts with the system through federator, it sets its browser's session ID to the header of its SOAP messages sent to the Web Service. All the requests coming from the same browser has same session ID. Session IDs are created when the browser is opened and kept same until it is closed. Each browser has a separate and unique session ID. By setting this session ID to the header of SOAP messages federator can distinguish what client (browser) makes the requests and check its cached data and session information stored before.

Here is the pseudo code briefly explaining the steps:

```
WMSServicesSoapBindingStub binding;
binding = (WMSServicesSoapBindingStub)
    new WMServiceLocator().getWMSServices(newURL( service_address));
String sessionID = session.getid();
String channel_name = "WMS_getMap_Request";
//Add SessionID to the SOAP message's header
binding.setHeader(service_address, channel_name, sessionID);
//See Appendix A-2 for the sample GetMap request
Object value = binding.getComprehensibleData(getMap_request);
```

Whenever a user access the system through the same browser its session number will be the same and federator keeps its local data and actions in the system differentiated based on its unique session ID.

In order to implement dynamic client-based caching we keep static table keeping updated session information about each active client. This table is called *MapTable* and each entry represents a client. Each entry keeps unique user identification number (uuid) and its dynamic session information. Dynamic session information for each client is kept as an instance of a class called *FormerRequest*. It has four attributes as listed below.

MapTable: Client-id session tracking Obj

uuid-1	FormerRequestObj1
uuid-2	FormerRequestObj2
…..	……

FormerRequest Class attributes

String uuid; *//unique-user-id*

String bbox; *//bounding box of the last request*

Double density; *//data size falling into per unit square*.

Vector [] feature_data; *//geometry elements of the last request used to plot map*

Density is used to find out allowable largest bbox area to be assigned to a thread for parallel processing. Details about the load balancing and parallel processing are given in Chapter 6.4.3.2.2.

6.4.3.1.2. Comparing with Static/central approach: Google Map Server

Static/central approach is fast because in that case the server doesn't depend on data and processing services from geographcically distributed heterogeneous resources. They can replace computation with storage. They don't manage the dynamically changing data provided by autonomous resources in various formats in integrated format such as multi-layered map layers whose layers are created from distributed autonomous resource.

In this chapter, we take Google Map Servcer's tiling approach as a sample of static/central approach and compare it with our adaptive client-based caching approach to show the necessities in developing such a caching system.

The fundamental concept behind the caching is removing the resource consuming repeated jobs and serving the client from the ready to use data sets kept in local storages. In case of map rendering process, ready to use data sets are map images. Google Map Servers [Googlemap] are the best examples for caching map images to provide high

performance map services. They keep the data as ready to use map images chunked in tiles. Each tile is defined by its x,y coordinates and a corresponding zoom-level (18 different zoom levels). They formalize the accepted requests (in terms of parameters), and responses in terms of the tile compositions. Their major concern is developing high performance map services. In order to do that, they introduced AJAX (Asynchronous JavaScript and XML) [AjaxSerrano] for client/server communications and used locally stored static map images.

However, Google Map's static caching approach (tiling) would not work in case of considering (1) data's dynamic and distributed characteristics, and their various heterogeneous formats; and (2) seamless addition of new data sources rendered as layers and overlaid with other layers in various combinations and orders.

Google Map Servers provide two unique layers, satellite and Google map, and one hybrid layer as overlay of those two. Maps are served from three groups of tiles corresponding to these layer sets. In order to highlight the limitations of their algorithms, let's assume they provide three unique layers instead of two. Let's say layer names are 'a, b and c. Then server would need to have 7 different tile groups as named a, b, c, ab, ac, bc and abc.

In summary, for the N number of unique layers, the required number of tile groups is calculated as below. It is sum of all k-subset combinations in which k gets the values from 1 to N.

$$\sum_{k=0}^{N} \mathrm{C}\binom{N}{k}, \qquad \mathrm{C}\binom{N}{k} = \frac{N!}{(N-k)!\,k!}$$

In case of 10 unique layers (N=10), the number of tile groups would be 1023. Moreover, in each tile group there are thousands of tiles, and each tile in the group has different copies for 18 different zoom levels. As the layer number increases, the number of required tile groups increases dramatically and at some point it becomes impossible to store that much tiles in a single storage with current possible technologies.

Client-based dynamic caching approach: We allow all the data to be kept at their original resources and integrated to the system through standard service API, communication messages and in expected common data formats. This enables extensibility and interoperability, easy data handling/maintenance, and workload and data sharing. We do on-demand data fetching and rendering. Instead of caching whole combinations of data sets we fetch and cache the data based on client's actions (locality and nearest neighborhood principles). Clients are given the flexibility to compose their own maps based on their applications' requirements. The framework also enables attribute based querying of the data integrated to the system through the common data model carrying both content and presentation features of the data.

With the client-based caching, besides removing the repeated processing jobs, we utilize the locality principles and develop efficient load balancing algorithm for sharing unpredicted workload among the worker nodes. We will show how to use this approach for load balancing in the following section.

6.4.3.2. Load-balancing through Query Decomposition and Parallel Processing

The parallel processing is implemented based on the main query partitioning. Each partition is assigned to separate thread of work. The number of partitions and their

sizes are defined by using locality principles. Locality information is obtained from the cached data kept for the same session and user. See the Chapter 6.4.3.1.

Federator apply the parallel processing for creating multi-layered map images corresponding to un-cached queried region. Since all the data in the system is geo-referenced and queried in ranges defined by bounding boxes (defining coordinates of rectangles in the form of (minx, miny, maxx, maxy)), we do range query partitioning to implement parallel processing.

Parallel processing algorithm has three parts in order and closely related. These are listed below.

1. *Cached-data extraction and Rectangulation (Chapter* 6.4.3.2.1*)*
2. *Query decomposition over un-cached data regions in rectangle regions created at step1. (Chapter 6.4.3.2.2)*
   - *If there is no cache utilized decomposition will be applied to main query*
3. *Parallel-processing for sub-queries created at step2.(Chapter 6.4.3.2.3)*

In order to make these concepts more clear I give the illustration of these steps and their relations in Figure 40. Figure shows a map image composed of two layers. One is NASA satellite base map layer, and other is a layer showing earthquake seismic records (in blue dots). (a) shows partially overlapping of cached data and the main request bboxes. (b) Shows cached data extraction and rectangulation for the remaining part in the main query. (c) Shows partitioning of the rectangles from (b) based on the locality information obtained (explained in Chapter 6.4.3.2.2) from the cached data. All

the rectangulated regions from (c) will be assigned to a thread to created map images as final responses.

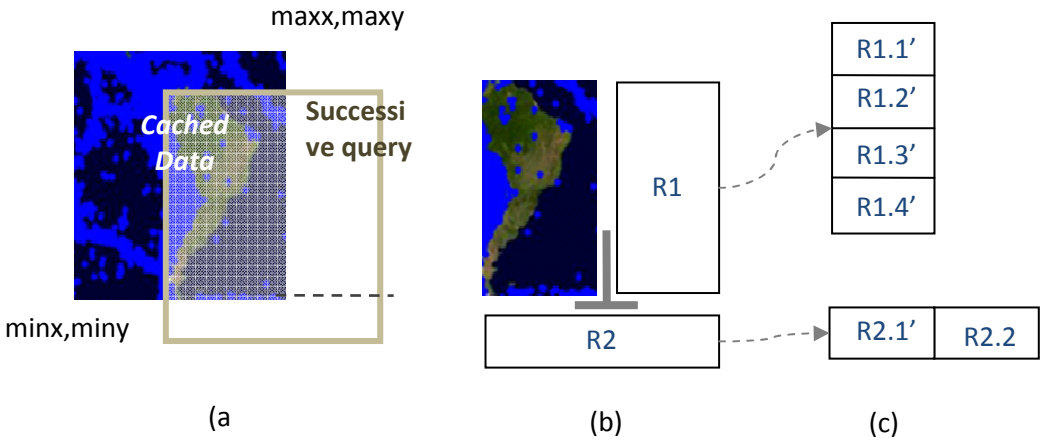

Figure 40: (a) Cached data extraction, (b) rectangulation, and (c) query decomposition/partitioning for parallel processing.

6.4.3.2.1. Cached-data Extraction and Rectangulation

According to OGC standards in GIS domain, queries are created with location parameter and location is defined in bounding box (bbox) formats. bbox is a formula defining the region as a rectangle through coordinates of bottom left corner and top right corner. Example: Q(minx, miny, maxx, maxy).

After extraction of cached data falling in the main query range, the remaining of the main query needs to be converted to the rectangular shapes in order to create valid sub-queries in the ranges defined by the bboxes (see *Figure 40* -b). This is why we make rectangulation after cached data extraction from queried-region.

The cached data extraction and rectangulation algorithm changes depending on the positions of bboxes of the main query and cached data region against to each-other.

The main query and cached data bboxes can be positioned to each other in four possible ways (see Figure 41).

Notation to be used for representing bboxes: Main query bbox is described as (minx, miny, maxx, maxy) and Cached data bbox is described as ($minx^c$, $miny^c$, $maxx^c$, $maxy^c$)

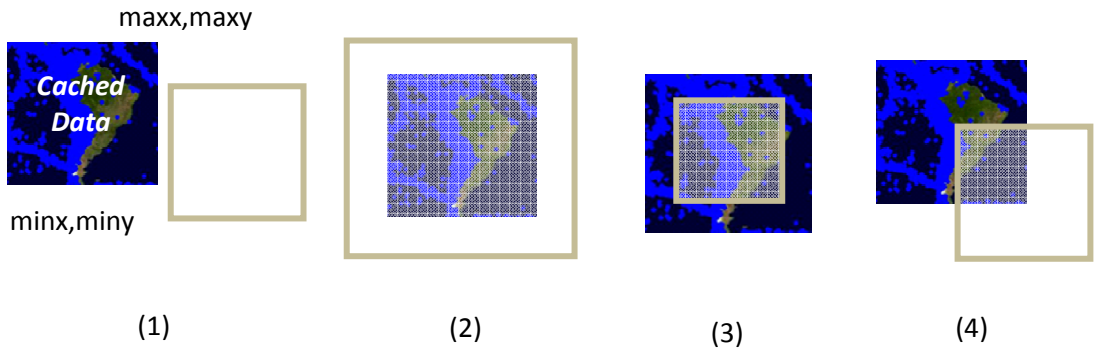

Figure 41: Positioning of the successive main query and stored client-based cached data

- Positioning-1: (No rectangulation). The main query and cached data do not overlap in anyway. In this case "cache data extraction and rectangulation" process is going to give only one rectangle which is the main query bbox.
- *Positioning-2*: The main query covers cached data (zoom-out action):

  Rectangles: R1: minx, miny, $minx^c$, maxy R3: $minx^c$, $maxy^c$, $maxx^c$, maxy

  R2: max^c, miny, maxx, maxy R4: $minx^c$, miny, $maxx^c$, $miny^c$

- *Positioning-3*: (No rectangulation). The main query falls in cached data (zoom-in action)

Rectangles: This case enables the fastest response. There is no need for query partitioning and data transfer from WFSs. It just uses cached GML to create map image based on the bboxes values of main query. A lot of performance gains.

- Positioning-4: The main query partially overlaps with cached data (move action).

  This case is also explained in *Figure 40*.

  Here is the formula of the rectangles for a specific case of partial overlapping of cached data bbox and main query bbox (Figure 41-4):

  Rectangles: R1: minx, miny, maxx, $miny^c$ and R2: $maxx^c$, $miny^c$, maxx, maxy

  In this case, there are four different sub-cases depending on the movement directions. These are (1) down-right, (2) down-left, (3) up-right, and (4) up-left. The *Figure 40* illustrates the down-right case, and the rectangles above belong to his case. The rectangles for the other cases are also created similarly.

The rectangles obtained in this section go through the decomposition process explained in the following chapter.

6.4.3.2.2. Query Decomposition

This chapter explains how to determine the number of partitions, and how to partition the rectangles to assign to the separate threads to create map images in parallel processing.

There two ways we propose. One is naïve approach, just partition into equal sizes (Chapter 6.4.3.2.2.1). The other is smart approach partition the queries according to the previous query's bbox values and utilizing the locality principles. But because of the

overhead timings and costs we need to define the best partition number to decompose the main query. In order to do that we propose smart query decomposition using client-based caching algorithm defined in Chapter 6.4.3.2.2.2.

6.4.3.2.2.1. Blind Query Decomposition

If there is no cached data available for the client, in other word rectangles are coming from positioning-1 explained in the previous chapter, then we use blind partitioning. In all the other cases we use smart decomposition technique explained in the following chapter.

Blind query decomposition is a static approach, it just chunks the query area (represented in bbox) into equal sized regions in terms of bbox values without identifying identity of client. Partition number is pre-defined and does not change at run-time.

6.4.3.2.2.2. Smart Query Decomposition Using Client-based Caching

Instead of decomposing the main query into predefined static number of sub-regions, we utilize the neighborhood and locality principles through client-based caching and figure out the most efficient number of partitions changing based on the data returned and cached at last time.

Here we explain how to define the number of partitions (i) and how to decompose the query (ii).

i. Determining the partition number:

In order to define the partition number we use locality principles. Locality principle in this context is explained as following. If a region has a high volume of data, then the regions in close neighborhood also expected to have high volume of data.

Example is the human population data. The urban areas have higher human population than the rural areas. The oceans (2/3 of the world) have no populations etc.

We partition the rectangles into equal regions in the form of bboxes, because we don't know the size of the data falling in that region before getting it. In order to define the size (in bbox) we use cached data sizes expressed in bbox and KB. We assume (by using locality) cached data density is similar to the main request density, and by using the threshold value and un-cached main request part we calculate the partition number (P_n) as below:

Cached data bbox area = $(maxx^c - minx^c)*(maxy^c - miny^c)$

Density of cached-data: $dcd = \frac{Cached-data\ size\ in\ KB}{Cached\ data\ bbox\ area}$

Cached-data size in KB and bbox values are obtained from the client-based caching.

Allowable largest area to assign: $lat = \frac{threshold\ data\ size}{density\ of\ cached\ data} = \frac{thr}{dcd}$

Threshold data size is a static value pre-defined in server's properties file for the corresponding critical data.

P_n*: the number of partition* $= \frac{rectangule\ query's\ bbox\ area}{lat}$

If P_n is less than 1 then, don't make partition. In contrast, if it is bigger than 1, then partition into P_n regions. The following section explains how to partition a rectangle into P_n number of regions.

ii. Query decomposition of the rectangulated regions with P_n:

After getting P_n value in previous step, we cut the region into P_n number of sub-regions in the form of bbox values.

Here, we explain how to partition a given rectangle into P_n number of bboxes. There are two alternative techniques here, one is partitioning the rectangle vertically and the other is partitioning it horizontally.

In case of horizontal partitioning the step value is calculated as below, and partitioning is done along the Y-coordinate. (See Figure 42)

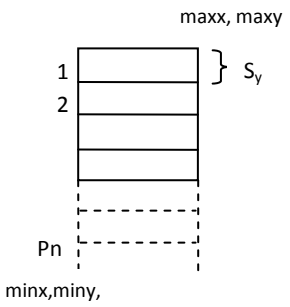

Figure 42: Partitioning a rectangle along the coordinate-y

Calculating the bboxes of the partitioned regions:

*for (i=0; i<P_n*s_x; i=i+s_x;)*

print (minx-i, miny, maxx-(i+s_y), maxy) ;

$$S_y = \frac{(maxy - miny)}{Pn}$$

6.4.3.2.3. Parallel-Processing

The proposed parallel processing is based on range-query (defined in bbox) decomposition we call it partitioning. The main query ranges are partitioned into smaller sub-regions and defined in new smaller bboxes and assigned to separate threads to be created in parallel.

This section explains how to create sub-queries corresponding to the partitions, and how to assign the sub-queries to threads and assemble the results.

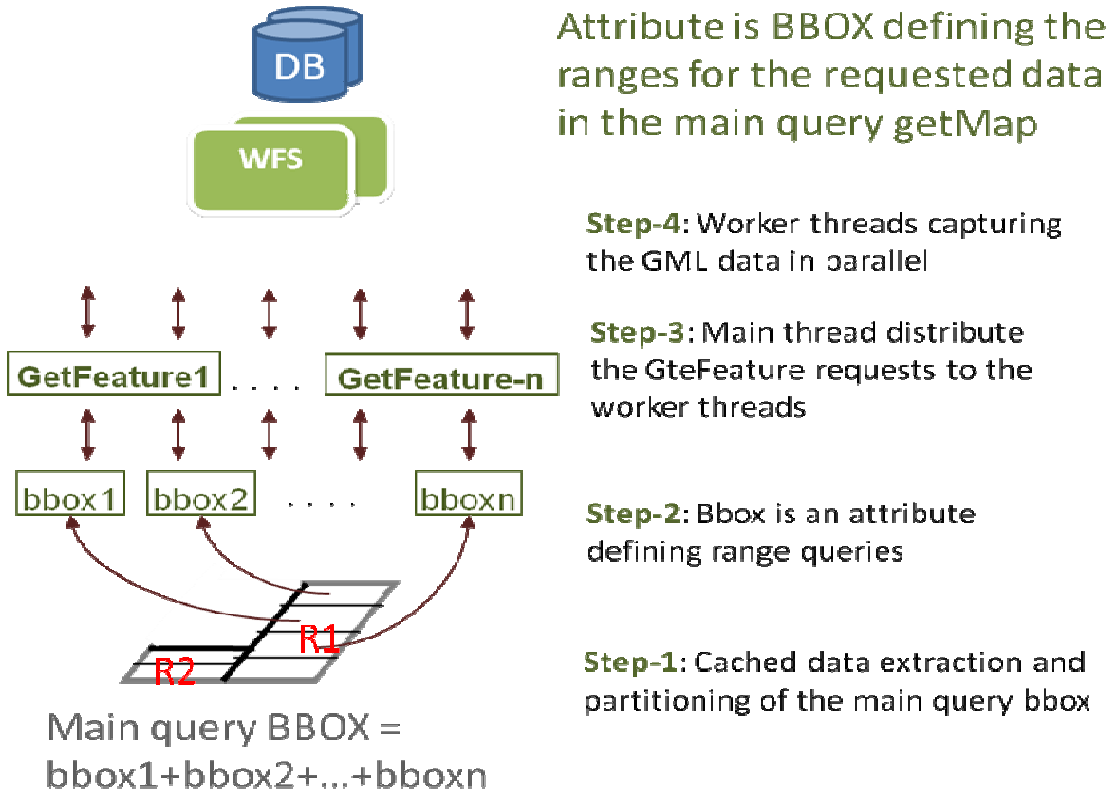

Figure 43: Parallel processing illustration in brief. See also *Figure 40*.

Figure 43 illustrates the parallel processing in brief. The rectangulated regions R1 and R2 in the main query are determined after the extraction of the main query parts overlapping with the cached data. The rectangulation processes explained in Chapter 6.4.3.2. There is no need for data transfer for the regions overlapping with the cached data bbox. This is obtained from the cache. For the other parts (region 1 and 2), the system first makes partitioning and then fetches the data in parallel. el processing for data access, query and plotting after creating partitions.

i. Creating the queries for the partitions.

Throughout the rectangulation and partitioning, the only changing attribute of the main query is the bbox coordinate value. These are calculated in the previous chapter.

Based on the set of bbox values obtained at the end of partitioning process (ii) we need to create sub queries. Each partition is differentiated by only their bbox value, and they go through the query creation process. The federator creates *getFeature* requests corresponding to these rectangles based on their bounding boxes. Other parameters and attributes required for creating *getFeature* request are inherited from the main query (*getMap*). All the parameters, attributes and their values (except for bbox values) will be the same for all the getFeature requests created for the partitions.

An example case of decomposing a rectangle obtained by rectangulation process and creating parallel queries is illustrated at Figure 44. In this example, rectangle is partitioned into 5 regions vertically.

$$P_n = 5 \quad \text{and} \quad s_y = \frac{(\text{maxy} - \text{miny})}{Pn} = \frac{(45-40)}{5} = 1$$

You can see a sample getFeature created for bbox value "-110, 35, -100, 40" request at Figure 45.

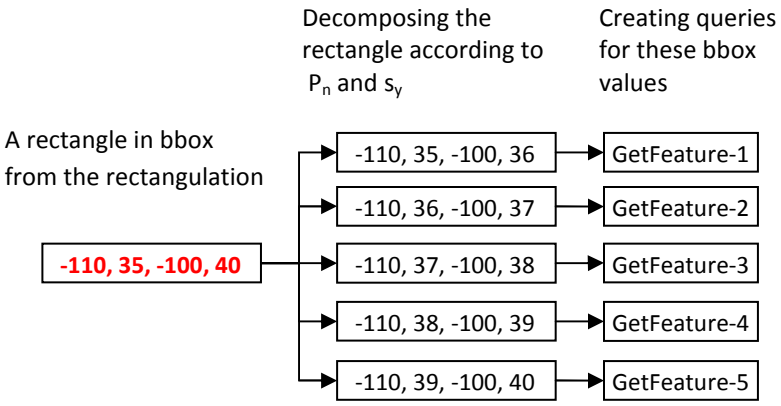

Figure 44: Example scenario of the partitioning a region into 5 sub-regions through the bbox value of a rectangle.

```
<?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="global_hotspots">
       <wfs:PropertyName>YEAR</wfs:PropertyName>
       <wfs:PropertyName>MONTH</wfs:PropertyName>
       <wfs:PropertyName>DAY</wfs:PropertyName>
       <wfs:PropertyName>LATITUDE</wfs:PropertyName>
       <wfs:PropertyName>LONGITUDE</wfs:PropertyName>
          <wfs:PropertyName>MAGNITUDE</wfs:PropertyName>
       <ogc:Filter>
        <ogc:BBOX>
         <ogc:PropertyName>coordinates</ogc:PropertyName>
         <gml:Box>
          <gml:coordinates>-110,35 -100,40</gml:coordinates>
         </gml:Box>
        </ogc:BBOX>
       </ogc:Filter>
      </wfs:Query>
      <wfs:Query typeName="global_hotspots">
       <ogc:Filter>
        <ogc:PropertyIsBetween>
         <ogc:Literal>MAGNITUDE</ogc:Literal>
         <ogc:LowerBoundary>
          <ogc:Literal>7</ogc:Literal>
         </ogc:LowerBoundary>
         <ogc:UpperBoundary>
          <ogc:Literal>10</ogc:Literal>
         </ogc:UpperBoundary>
        </ogc:PropertyIsBetween>
       </ogc:Filter>
      </wfs:Query>
</wfs:GetFeature>
```

Figure 45: Sample GetFeature request for the partitioned region of bbox (-110, 35 -100, 40). Request is done for global hotspot (earthquake seismic data)

ii. How to assign the sub-queries to threads and assemble the results.

Sub-queries created at previous step are assigned to separate threads to capture the GML data from WFS and process the corresponding map pieces. Partitions are assigned to worker nodes through separate thread of works in round-robin fashion [tanenbaum].

Let's say PN is the partition number and WN is the number of WFS worker nodes.

$$share = base\,(\frac{PN}{WN})$$

Share is the number of partitions each worker node is supposed to get.

$$rmg = WN - \ base\,(\frac{PN}{WN})$$

rmg is the remaining of the PN/WN division. If there is no remaining every worker node is assigned share number of partitions. Rmg is dofferent from 0 then partitions are assigned to worker nodes as below:

The first rmg #of WN are assigned share+1 number of partitions, and remaining WN are assigned share number of partitions.

Figure 46 illustrates the algorithm over a case of seven partitions and three WFS worker nodes (called WFS-1, WFS-2 and WFS-3). So, the algorithm's parameters would be

share = base (7/3) = 2 and *rmg* = 3 – 2 = 1;

So WFS-1 is assigned 3 (*share+1*) partitions through thread-1, 4 and 7,

WFS-2 is assigned 2(*share*) partitions through thread-2 and 5

And finally WFS-3 is assigned 2 (*share*) partitions through thread-3 and 6.

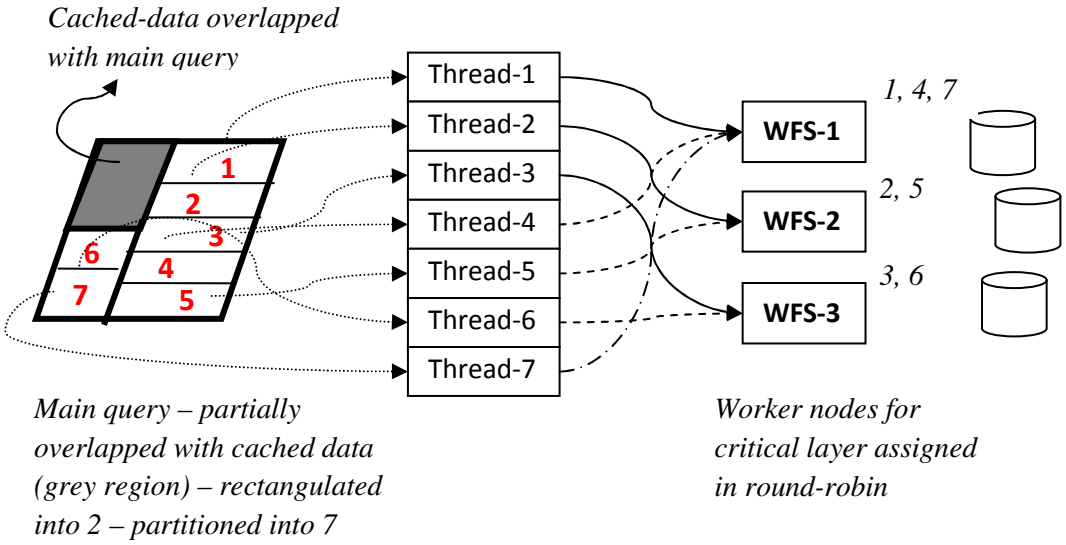

Figure 46: Assigning partitions to threads and capturing/processing in parallel

Each query corresponding to the partitions are assigned to the threads. Threads are responsible for interacting with the WFS and getting the requested data to create map images for the partition. After getting the data, federator starts rendering and plotting the critical data over the other layers by parsing and extracting the geometry elements from returned GML.

6.4.4. Evaluation of On-demand fetching/rendering Enhancements

Analysis and evaluations will be done on two-layered map images for simplicity. The bottom layer is from NASA satellite map images provided by OnEarth project's WMS, and the top layer is earthquake seismic data provided by WFS in GML.

We run the test in Local Area Network (LAN). WFS servers, federator server and event-based interactive map tools are deployed in Indiana University Community Grids Labs. In local area network we have used so-called gridfarm machines from gf12 to 19.ucs.indiana.edu. These machines have 2 Quad-core Intel Xeon processors running at 2.33 GHz with 8 GB of memory and operating Red Hat Enterprise Linux ES release.

Test Setup:

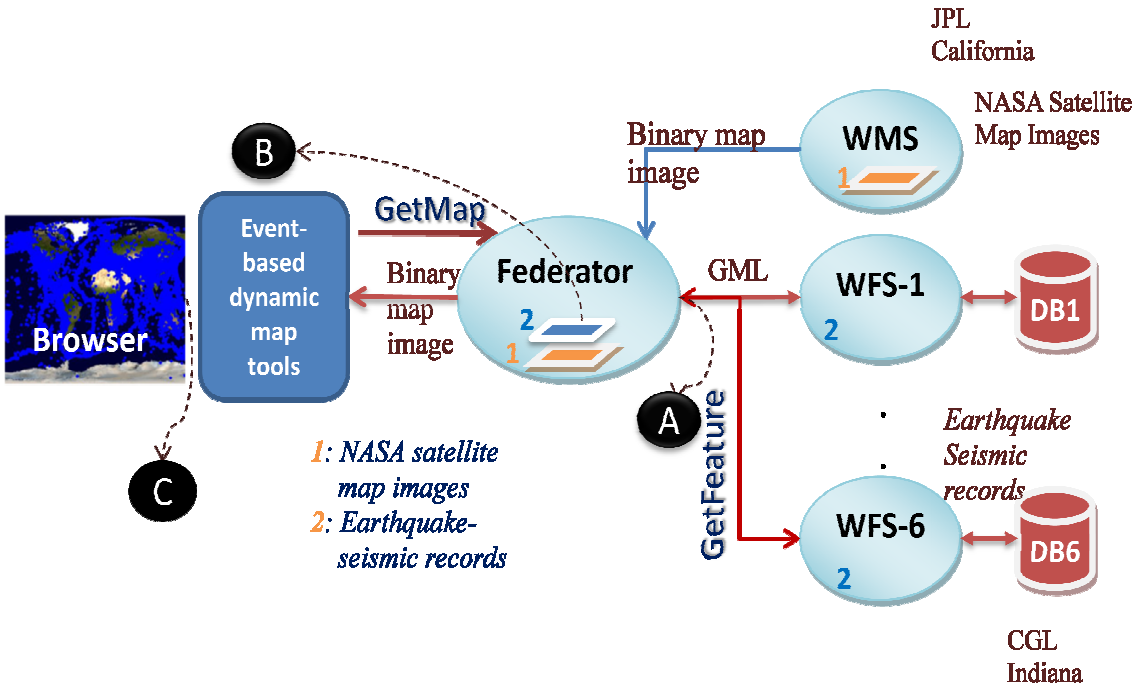

Figure 47: Test setup for Federator-oriented enhancement analysis and evaluations - on-demand fetching/rendering of earthquake GML data over NASA satellites image

Figure 47 is test setup showing GML data fetching (A), rendering and overlaying on NASA satellite map image obtained from another standard WMS and creating two-later multi-layered map images (B) as an outcome shown on the user's browser (C).

There are 3-phases from bottom to top – based-on data flow:

A. Fetching data as GML (Chapter 6.4.4.1)

B. Creation of a layer from GML data (Chapter 6.4.4.2)

C. Displaying the requested data at user-end (Overall response time) (Chapter 6.4.4.3)

Overall response times: C = A + B + Image transfer from federator to user

In accordance with the 3-phase performance test setup (A, B and C in Figure 47), we analyze and evaluate the proposed on-demand fetching and rendering techniques presented earlier in three groups. Each group has different concern and has different effects on overall end-to-end response time. We will be analyzing them separately in the following chapters.

Performance enhancing design techniques focus on XML-encoded common data model (GML) handling which is plotted as a top layer in the two-layered map image. Binary map images are fast enough to access and display. Because there is no performance burdening query and data conversion as it is done for GML.

From our earlier base-line performance (Figure 34) tests we know that dominating performance is fetching the data from databases through WFSs. We mostly focus on that phase and evaluate the techniques (Chapter 6.4.4.1) by comparing the base-line tests given in Figure 34. Measurement of success is evaluated relatively.

6.4.4.1. Parallel data fetching (A)

Architectural details for data fetching through publish/subscribe based messaging middleware and parallel processing through query decomposition are given earlier.

The **motivation** behind the parallel data fetching is the dominating performance degrading data fetching from autonomous heterogeneous data sources through WFS-based mediators.

The **strategy** is load balancing and parallel processing through range query decomposition. Each partition will be obtained by using Streaming data transfer through Naradabrokering (NB) publish/subscribe based messaging middleware. As it is mentioned earlier there are two approaches to partition the range queries there are blind/static partitioning and smart/dynamic partitioning whose performance evaluations are given in the following chapters.

Test setup:

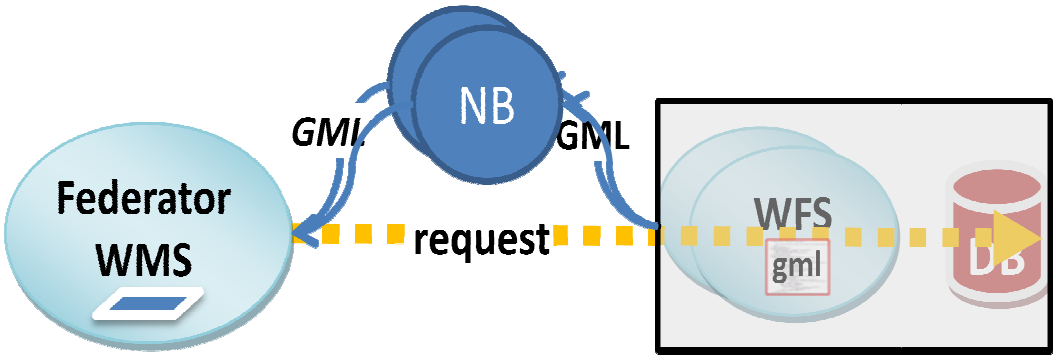

Figure 48: Streaming Data fetching through publish/subscribe based messaging middleware

There is a limited number of WFS's, and possibly large number of partitions. The partitions are assigned in round-robin fashion see Chapter 6.4.3.2.3. There will be

actually 6 number of WFS workers deployed in LAN running in parallel to server parallel queries from the federator deployed again in the same LAN.

In the following chapter we analyze the overhead times resulting from partitioning and parallel processing, and the following two chapters analyze the data fetching times by using blind and smart partitioning respectively.

6.4.4.1.1. Analysis of Partitioning Overheads

The **motivation** for doing the tests presented here is to see if the overhead times stemming from partitioning and parallel processing is in considerable amount.

There are three steps extra compared to straightforward single process approach. These are (1) Partitioning, (2) Sub-query creation and (3) Merging the results

Partitioning: Defining the partition number and cutting the main query range into that number of pieces in the form of bounding box (bbox) values (range query attribute).

Sub-query creation: Create corresponding XML-based query (getFeature - APPENDIX G) for each partition to fetch the remote GML data from WFS.

Merging: Aggregating the results to sub-queries and creating one complete map images as an answer to the main query

We illustrate the partitioning and parallel processing as shown in Figure 49 (b). Range queries are defined in bounding box attribute (we call it BB in the figure). Sample main query range (BBtotal) is partitioned into 5 smaller sub-regions. These are illustrated as bb1, bb2, bb3, bb4 and bb5 in the below figure. getFeature queries to fetch the data are represented as letter Q.

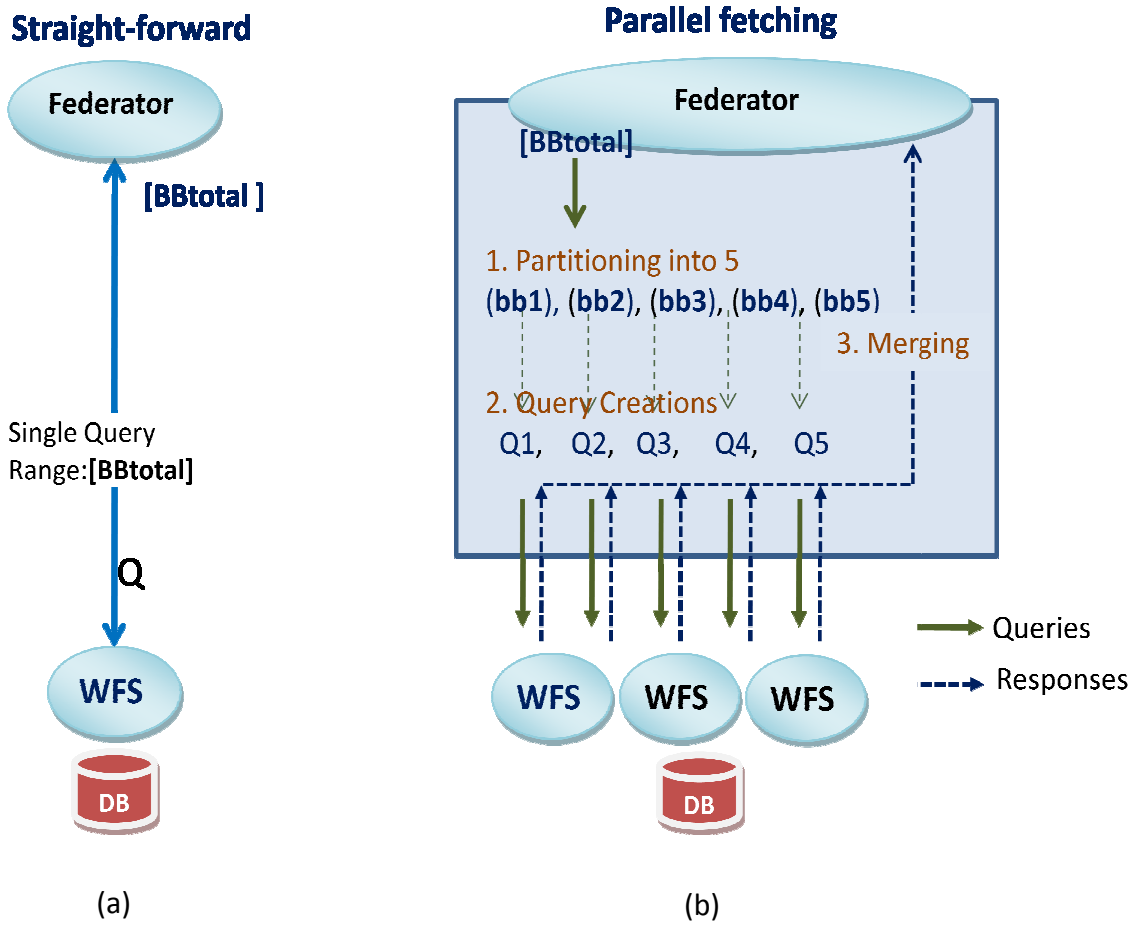

Figure 49: Architectural comparisons of parallel fetching with straightforward single thread fetching. In the case of partition number=5

Our **strategy** to evaluate the overhead times: In order to get the timings for the load balancing and parallel processing, we took a sample region of maps and cut it into various levels of partition numbers. In the figure we started with 5 and increased 5 at each step till 30 to see the overhead times' patterns over the partition numbers.

Figure 48 is the test setup.

Table 10: Avg and standard deviation values for overhead timings from partitioning with changing partition numbers

Partition Number	Partitioning		Sub-Query Crt		Merging partitions	
	Avg	StDev	Avg	StDev	Avg	StDev
5	51.28	14.74	161.67	25.32	27.00	12.88
10	58.65	15.16	421.55	63.98	44.26	23.44
15	60.15	19.74	720.35	102.87	64.90	23.77
20	68.75	21.75	1,058.84	199.49	118.90	25.53
25	69.05	15.98	1,366.10	198.37	131.88	30.59
30	85.42	30.04	1,837.16	343.26	170.00	30.56

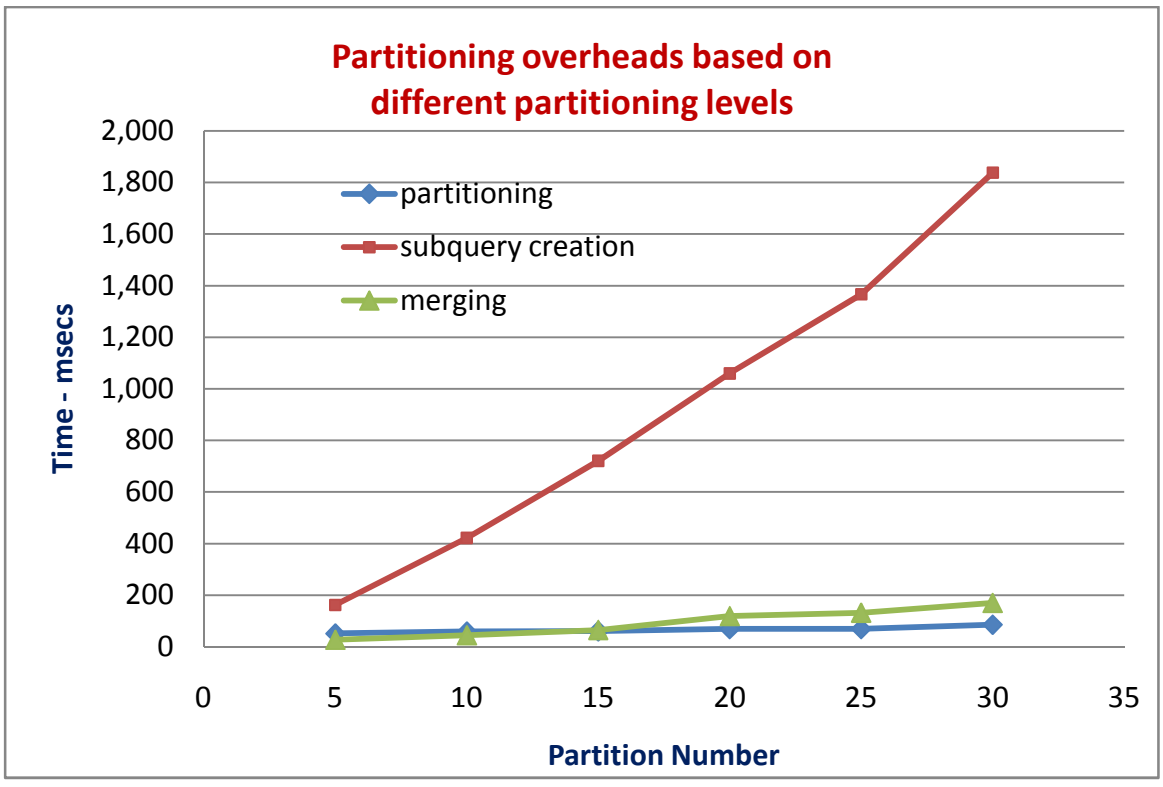

Figure 50: Overhead timings from Partitioning

Evaluation:

- Graph shows the pattern of changes in overhead timings according to the changing partition numbers, and their relative weights in total overhead.

- The only overhead items increasing with the increase in partition number is sub-query creation. It is related to XML-based query creation for partitioned sub-queries. These sub-queries are copies of main query with only one difference. That difference is the bounding box values defining the ranges of the queries. For example from the figure main query bounding box is defined as BBtotal, and sub-query bounding boxes are defined as bb_i. The other two overhead items do not have considerable effects on total response times.

- Because of the overhead times, if we do unnecessary number of partitioning then there is not going to be a performance gain for less than a threshold-data size but we see from the figure that it is less than some small amount that does not affect the overall performance considerably.

6.4.4.1.2. Analysis of Data Fetching with Blind Partitioning

The **Motivation** is that single process flow for map rendering from distributed data resources is not adequate for responsiveness. This is because of the decentralized nature of the framework, and heterogeneous and autonomous resources.

Our **Strategy** to get better performance results is using Load-balancing and parallel processing technique by partitioning range query into equal sized (in terms of bounding box ranges) regions. Each sub-query is a copy of the main query with only range value difference. For more information see Chapter 6.4.3.2.

By the blind we mean a straightforward approach. The queried range is decomposed into smaller equal number of regions according to the pre-defined static partition number. It might be changing from data to data.

Here we give times for data fetching for selected static partition numbers, 2, 10 and 20 as presented in Table 11. After plotting the performance values in Figure 51, we evaluated the results as below.

Table 11: Parallel data fetching average response times according to different data size and partition numbers

Data Size MB	Data Fetching Average Timing			
	Single-thread	2-thread	10-thread	20-thread
0.01	797.85	769.94	1,385.50	2,423.30
0.1	1,384.86	1,160.95	1,712.27	2,483.36
0.5	3,770.16	2,664.47	2,488.47	2,628.10
1	6,794.94	5,749.79	3,440.89	3,820.36
5	31,237.41	20,350.38	15,036.95	14,390.50
10	61,777.20	45,072.75	22,060.27	20,517.26
50	308,671.63	221,321.80	152,592.80	96,753.20

Table 12: Standard deviation values for parallel fetching response times according to various partitioning numbers.

Data Size MB	Data Fetching Standard Deviation		
	2-thread	10-thread	20-thread
0.01	92.52	91.09	281.40

0.1	99.03	89.56	177.48
0.5	94.35	97.66	117.43
1	101.61	90.05	108.09
5	99.43	190.37	265.17
10	131.44	973.42	582.05
50	312.20	1,852.54	1,154.53

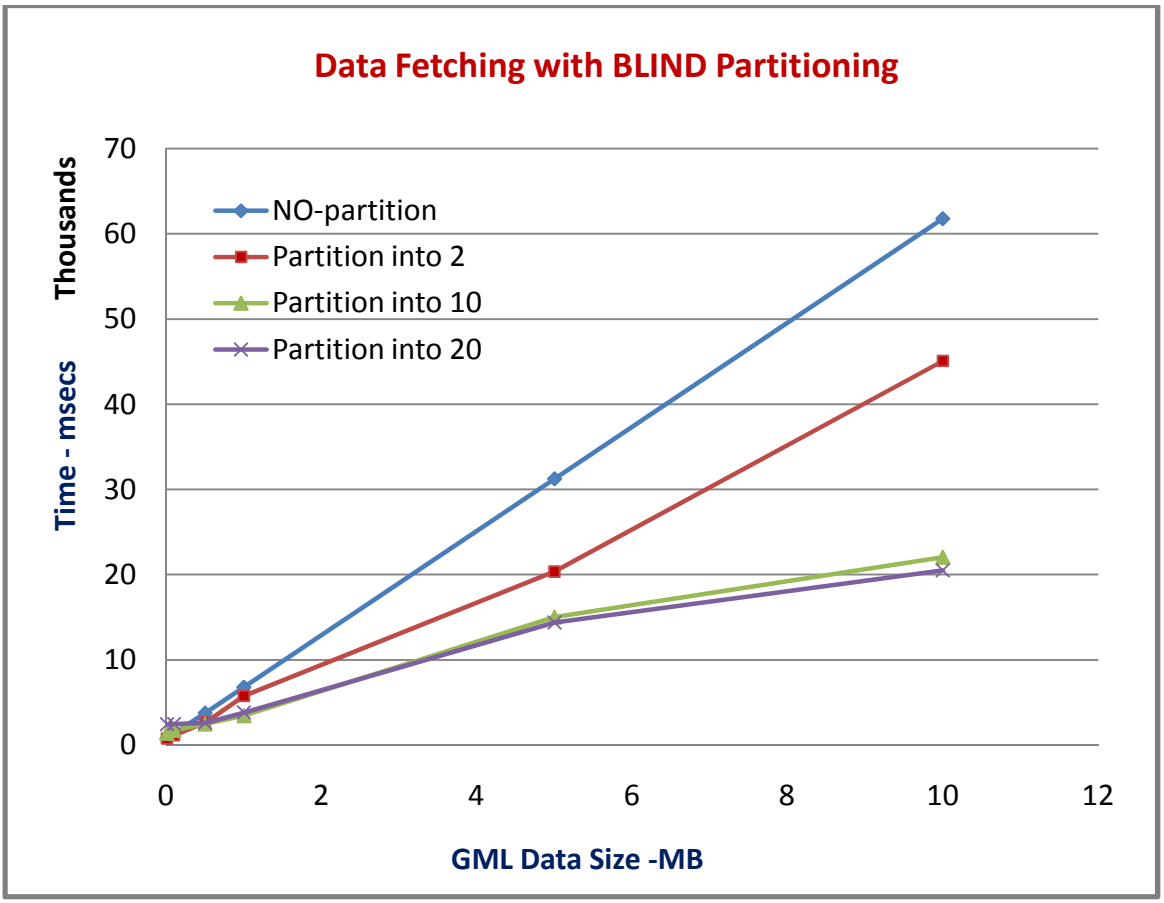

Figure 51: The performance results of data fetching with parallel partitioning through blind partitioning

Evaluations:

- Load balancing with blind partitioning always gives the better performance results than the straightforward approach does.

- For small data sets it might give less efficient result but still gives better compared to base-line tests. The larger the data the higher the performance gain.

- There is no performance gain for less than a threshold-data size handled. In Table 11 first 2 lines show that partitioning into 10 and 20 do not give better results than single process flow's performance result. This is mostly correct for small sized data fetching, but even in this case, it does not affect the overall response considerably.

- As the density or data size falling in a specific region increase (higher possibility of even workload sharing), the gain from the partition increases.

- As the partition number increases, after a saturation point the performance gain rate decreases. In the figure 10 and 20 partitions are a good example for this.

-The performance does not increase in the same ratio at which the thread number increases. This because of the unevenly non-homogeneous distribution of data sets in the ranges. If the data was 100% homogeneous in the queried range then, the partitioning would share the workload evenly and the most efficient performance results would be obtained. Another reason might be the overhead times coming from partitioning and parallel processing.

- It does not guarantee even-workload sharing. Actually elbows in the lines shows the difference of degree of equal partitioning for different size of data. Since we selected different ranges and each range has different homogeneous data distribution it might

cause elbows in the graph. For example, if the elbow is high that means homogeneity in distribution is bad compared to previous point, or vice versa.

- In case of single process flow there is ignorable elbows. This is because of that streaming data transfer takes linear time according to increased data size. However, we used parallel threads to fetch the data. Each thread transfer data in linear time in itself but some threads might finish earlier than the others and some might take so long. That happens because of that assigned partitioned regions have varying size of data even if they have same size of bbox. That is why figure have some elbows compared to the single partition (the top line).

- The performance results in Figure 51 are obtained with 6 WFS worker nodes. When we increase it, results would be little better for 20 thread case but the others would be almost same. As the partition number increases, results would get the saturation point at some point depending on the number of WFS. Increasing the WFS number would postpone that saturation time to higher partition numbers.

6.4.4.1.3. Analysis of Data Fetching with Smart Partitioning

Our **Motivation** in developing a smart partitioning technique is that the blind-partitioning does not guarantee just enough workload sharing. Due to the geo-data's tough characteristic which is explained as unevenly distribution of data based on location attribute, blind partitioning into static number of partition might not help in some cases.

Strategy: We take the data distribution into account and create an algorithm differentiating data-dense/sparse regions and act adaptively. It is based-on forecasting the most efficient partition number by using priori knowledge obtained through proposed client-based caching

- Makes use of locality information
- Differentiates data-dense regions

Estimates the most efficient partition number whenever user sends a new query and reduces the proportions of overhead times in total time.

Here we give times for data fetching for selected static partition numbers, 2, 10 and 20, and illustrate the effect of using smart partitioning technique in Table 13. After plotting the performance values in Figure 52, we evaluated the results as following.

Table 13: Average performance timings for parallel data fetching according to different data size and partition numbers. Comparison with smart partitioning.

Data	Data Fetching Average Timing					
MB	Single-thread	2-thread	10-thread	20-thread	Smart Partition	
0.01	797.85	769.94	1,385.50	2,423.30	769.94	2-thrd
0.1	1,384.86	1,160.95	1,712.27	2,483.36	1,160.95	2-thrd
0.5	3,770.16	2,664.47	2,488.47	2,628.10	2,488.47	10-thrd
1	6,794.94	5,749.79	3,440.89	3,820.36	3,440.89	10-thrd
5	31,237.41	20,350.38	15,036.95	14,390.50	14,390.50	20-thrd
10	61,777.20	45,072.75	22,060.27	20,517.26	20,517.26	20-thrd
50	308,671.63	221,321.80	152,592.80	96,753.20	96,753.20	20-thrd

The performance results for smart partitioning are not from real time application. It is a result of comparison and studying of table values for partitioning 2, 10 and 20. This only gives an idea about how well smart partitioning works. Since we don't know the data size and density values corresponding to a specific bounding box earlier, we can't

present the test result in a well understood manner. This is the only way to present the smart partitioning performance outcome and comparing with sample static partitioning values for selected bounding boxes.

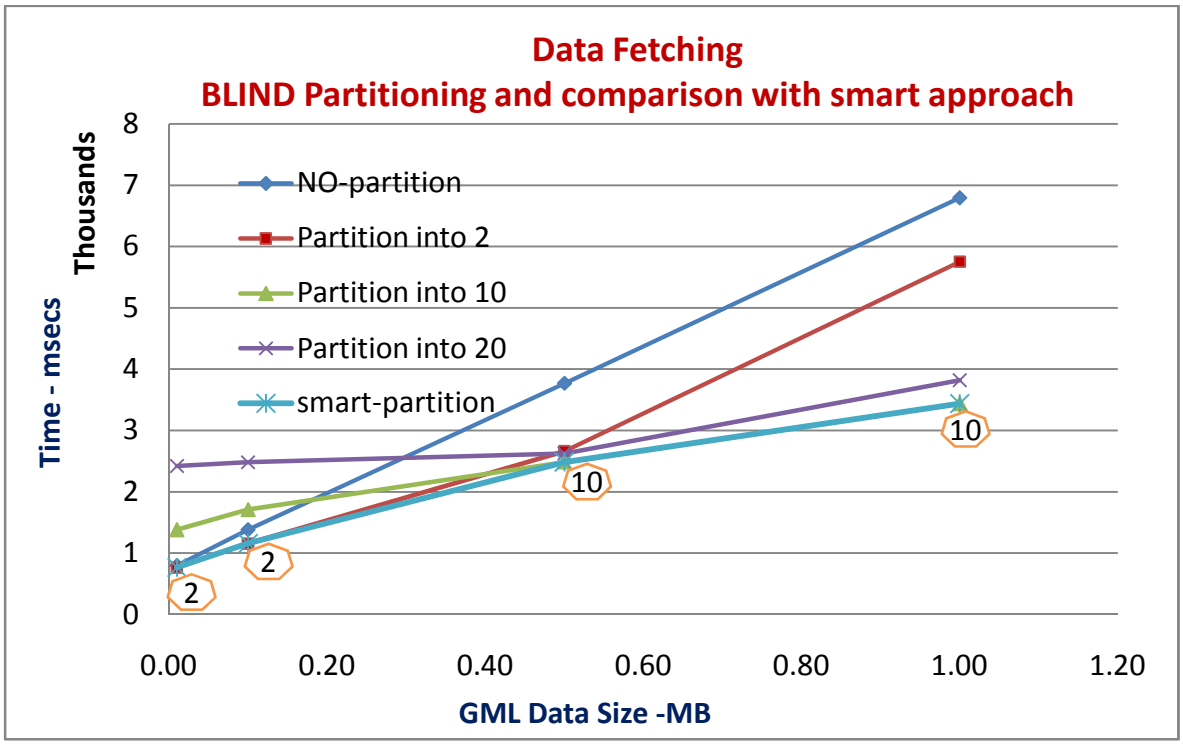

Figure 52: Comparison of data fetching performance results – blind vs. smart partitioning

Figure values are plotted from Table 13 for relatively small data sizes (less than 1MB of GML data sizes).

Evaluation:

- As plotted in the figure, using blind partitioning for small size of data because of the partitioning and parallel processing overhead performance times might change unexpectedly such as 20-partition gives worse result than 2 for less than 500KB data etc. overhead timing analysis are given in Chapter 6.4.4.1.1 and Figure 50.

- Standard User/client map tools enable system invoke by event-based interactions. These are zooming-in and out, moving and panning. All these actions cause request for map images in near neighborhood of the previous map images. Since the near neighborhood has similar characteristics in data density and sizes falling in bounding boxes, the priory-knowledge from the client-based caching helps with smart partitioning. The important thing is defining an efficient threshold value for the corresponding data according to algorithm.

- Table illustrates the advantages of using smart partitioning by comparing partition values 2, 10 and 20. Until 100KB of data partitioning into 2 or single-process are better. From 100KB to 5MB partitioning into 10 gives the better results, and from 5MB to 50MB 20-partition or most probably higher partitions would give better results.

- Smart partitioning always helps and gives better results than the blind partitioning.

- Blind partitioning might give very bad results if you set it high or low depending on the available worker nodes. For example let's assume developer sets blind/static partition number for earthquake data as 30, in that case for small data sets such as less than 1MB there would be unnecessary partitioning overheads and overall response performance would be poor.

- If it is inevitable to use blind partitioning, then setting the partitioning number high always gives better performance results than the setting low as long as you have enough worker nodes enabling high degree of parallelization. That is because of that the overhead times are not so high compared to data fetching/rendering latencies. The

number can be defined according to available WFS worker nodes. For example for 6 WFS, blind partitioning 20 would be good.

6.4.4.2. Just-in-time Map Rendering from GML (B)

This chapter analysis on-demand rendering of GML data illustrated as **phase-2** (**B**) in Figure 47. Our **motivation** is seeing how much effect data rendering has over the overall response times. See how much ratio is spent on data extraction and how much on rendering, and how these values change depending on the various data sizes.

The performance tests here are not related to parallel processing. We assume there is only one process flow and one GML is processed. The processing steps to create a map image layer from a GML are shown in Figure 53.

For more detail about the map rendering from GML see Chapter 3.3.2 (Web Map Service). This chapter is also closely related to Chapter 6.3.1.2 explaining data parsing and extraction from GML data.

Strategies are using pull parsing for parsing and extracting geometry elements from GML, and Java Graphics2D and AWT libraries to plot the geometry elements as a standalone layer (or over the other layers such as NASA satellite map images or Google maps). The GML to be rendered for the proof of concept is earthquake seismic data. Multi-layered map images and layer overlaying issues are not taken into considerations.

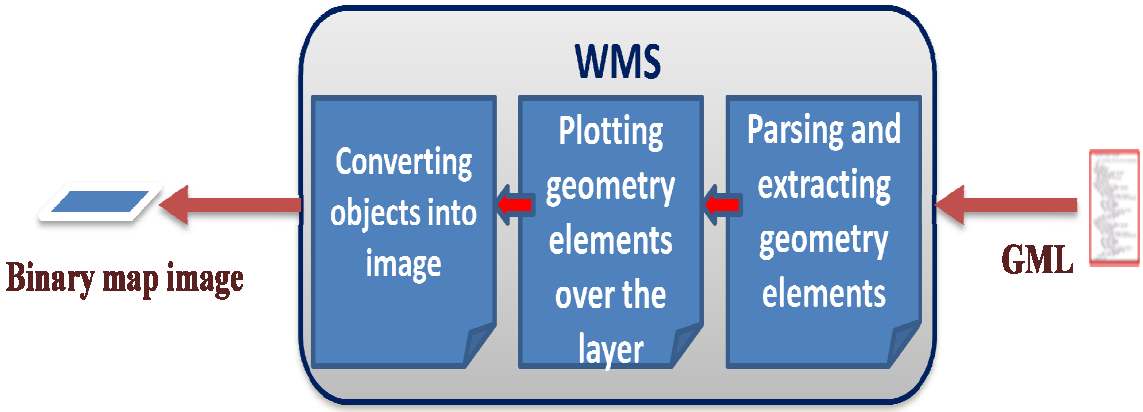

Figure 53: Map rendering process steps

Table 14 shows performance values for the map rendering steps illustrated in Figure 53.

Table 14: Average timing values for map image processing steps

Data (KB)	Data extraction	Data plotting	JAVA Image to JPEG	Layer Creation
1	15.59	0.00	25.43	41.02
10	72.81	3.00	25.43	101.24
100	183.06	15.33	25.43	223.82
1,000	270.47	83.11	25.43	379.01
5,000	671.74	153.67	25.43	850.84
10,000	1,025.67	828.50	25.43	1,879.60
100,000	7,059.72	3,738.25	25.43	10,823.40

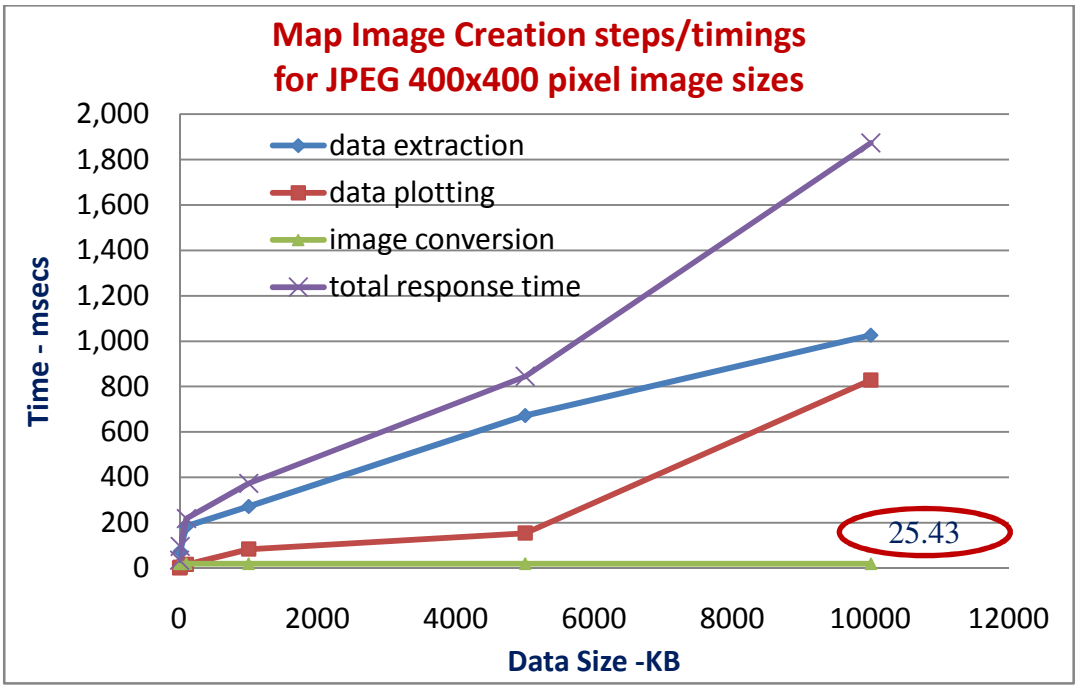

Figure 54: Average timings for map-image creation steps

Evaluation:

- Data plotting increase-rate seems higher than the data extraction's increasing rate. But even for 100MB of data it does not seem high compared to data fetching timings given earlier. Moreover, since we apply partitioning, processes will never see 100MB GML data as shown in Table 14.

- Image conversion time shown in Figure 54 does not change with the GML data size. For 400x400 PIXEL jpeg map image creation its value is steady-state and 25.43 msecs.

- Image object to JPEG conversion time changes with the requested map's sizes. The map size is a request parameter defined by the user. In order to see the affects of map sizes in overall map rendering performance, see Table 15 and Figure 55. They analyze

conversion times in case of converting to mime/JPEG for different map sizes in pixel values.

- We see that map rendering gives good enough performance results compared to system bottleneck (remote data fetching) by using proposed techniques and strategies.

Table 15: Average timings and standard deviation values of object to image/JPEG conversion

Resolution	Average (msec)	StdDev
200x200	19.24	8.53
400x400	25.43	9.29
600x600	46.38	10.42
800x800	71.58	16.70
1000x1000	131.67	17.24

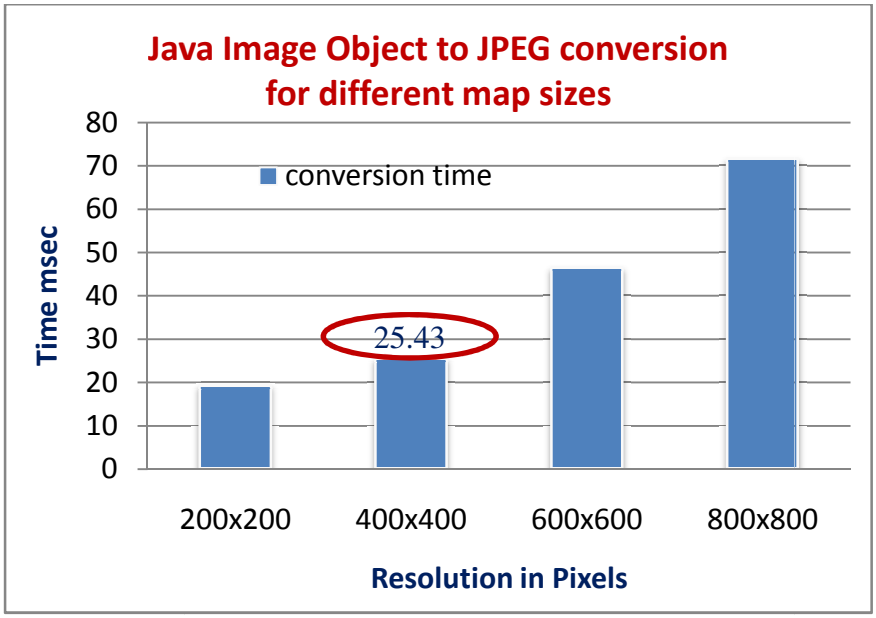

Figure 55: Image conversion timings based-on pixel resolution values

6.4.4.3. Overall Response for Map display (C)

Motivation: overall performance results changes considerably depending on the cached data available and queried region, and their positioning to each other. Here we analyze those affects on overall performance results. In Figure 56, "browser" shows the previous request's answer to user (cached data in federator) and boxes 1, 2 and 3 show the user's successive query region.

We also need to show how all the techniques are applied together and what performance results we get depending on these positioning.

Here we analyze end-to-end response times based on all possible cases of query and cached data ranges positioning (see Figure 56). Queries can be positioned in three possible different groups of ways. These are:

1. Main query range falls in outside of cache data ranges [worst case]
2. Main query range falls in cached data ranges [best case]
3. And partially overlapping [in between]

Overall performance changes depending on cached data and main query positioning. These possible groups of positioning are displayed in the below figure.

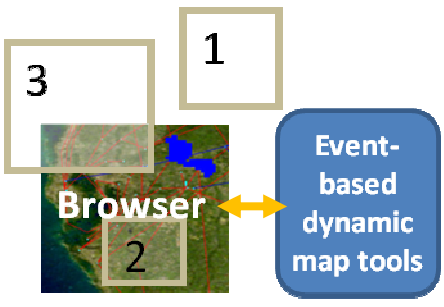

Figure 56: End-to-end query/data flow from user's point of view.

First group of requests are mostly first time queries. Federator does not have priori state information about them. In such cases, federator fetches whole data falling in the requested ranges from remote databases through WFSs.

Second group of requests are mostly coming from users zooming-in or panning (cutting a rectangular region over the display map) actions over the event-based interactive map tools. In such cases, the query falls in the cached data ranges and it gives the best performance result. In that case federator does not need to make successive queries to fetch the required data from the remote distributed databases through WFS-based mediators. Cache meets the whole requested ranges. Cached data is GML data kept in federator from the previous request. It is used to created the map image shown as "browser" in Figure 56.

The third group of requests is mostly coming from moving (drag and drop) and zooming-out actions over the even-based interactive map tools displaying the previous map images. In such cases, cache cannot meet the whole requested range. This is called partially overlapping with cache.

Strategy: We tested overall system performance by taking all cases and grouping them together and analyzing the results. All the performance enhancement techniques are analyzed together.

We had given the performance analysis for base-line single process flow earlier in Figure 34. We also had analyzed worst case scenario (box-1 in above figure) in Figure 51 and Figure 52. Here, we first give detailed analysis for the best case scenario (box-2

shown in Figure 56) in which queried regions are responded fully from cache data in federator, and then present the overall case-base comparative analysis in Figure 58.

Responding only from the cache (best case)

In this case there is no need for rectangulation, main query decomposition and threaded parallel processing. This case happens when the user query a smaller region falling in the previous map he got on his browser. It is mostly caused by zooming-in action or drawing a rectangular region in the browser over the displayed map. In those cases, the cached data is enough for responding to the main request, and no other cascaded requests are needed. The federator renders the map just by using the cached data. The only task needed is the cached data extraction and overlaying (plotting) over the other requested layers.

This case's performance results are similar to the pre-fetching, and even give better results than pre-fetching. In pre-fetching every time request comes, federator creates the response from processing (extracting geo-element and rendering) whole GML. But in this case, only small amount of cached GML is used. Smaller amount of GML means better performance in extracting geometry elements and in map rendering.

Table 16: Overall response time performances in case of processing only from cached GML

GML MB	Processing	StdDev	Transfer	StdDev	Response	StdDev
0.01	1,011.67	233.28	39.13	9.21	1,040.33	233.24
0.1	1,110.00	233.83	38.44	9.57	1,148.44	233.11
0.5	1,334.34	430.87	35.56	10.04	1,369.90	443.66

1	1,655.56	421.22	31.89	8.22	1,687.44	421.92
10	2,754.58	281.07	30.79	7.64	2,785.37	282.39
40	7,002.61	219.75	31.22	11.90	7,039.11	220.47
80	12,900.94	361.10	26.50	14.15	12,927.44	380.99
100	16,019.50	373.06	31.30	12.54	16,050.80	373.72

In table, *processing* means rendering map from the cached GML. It includes parsing and extracting geometry elements from GML, plotting the geometry elements over the map layer and finally converting JAVA image objects into JPEG. The "processing" steps in the table are given in detail in Table 14.

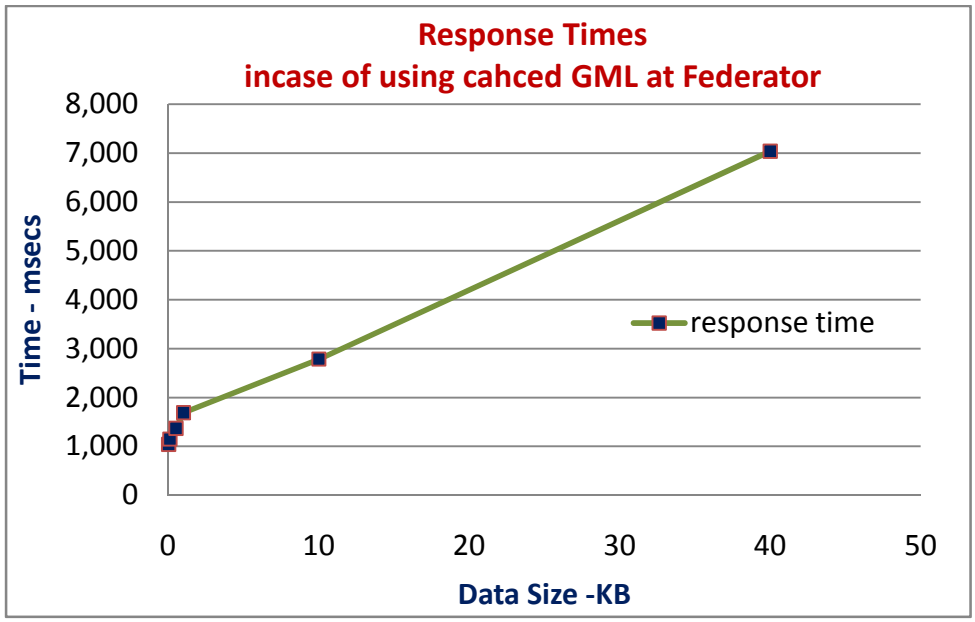

Figure 57: Overall response time from user-end point incase of that all the requests responsed from federator's cached GML

Overall case-base performance analysis

Blue line in Figure 58 is given earlier in Figure 34. It gives the response times with single process flow. We take it as a base-line performance result and use for evaluating the federator-oriented approaches' success.

Red-line presents the response times when using proposed load balancing and parallel processing techniques with smart partitioning. But there is no available cached data to utilize, all the data fetched from the remote databases through WFS mediation. It is taken from Figure 52.

Green-line presents the response times when rendering all the data from federator's cached data, all map data is served from the cache. There is no need to go to the databases through WFS. That line is already analyzed in Figure 57.

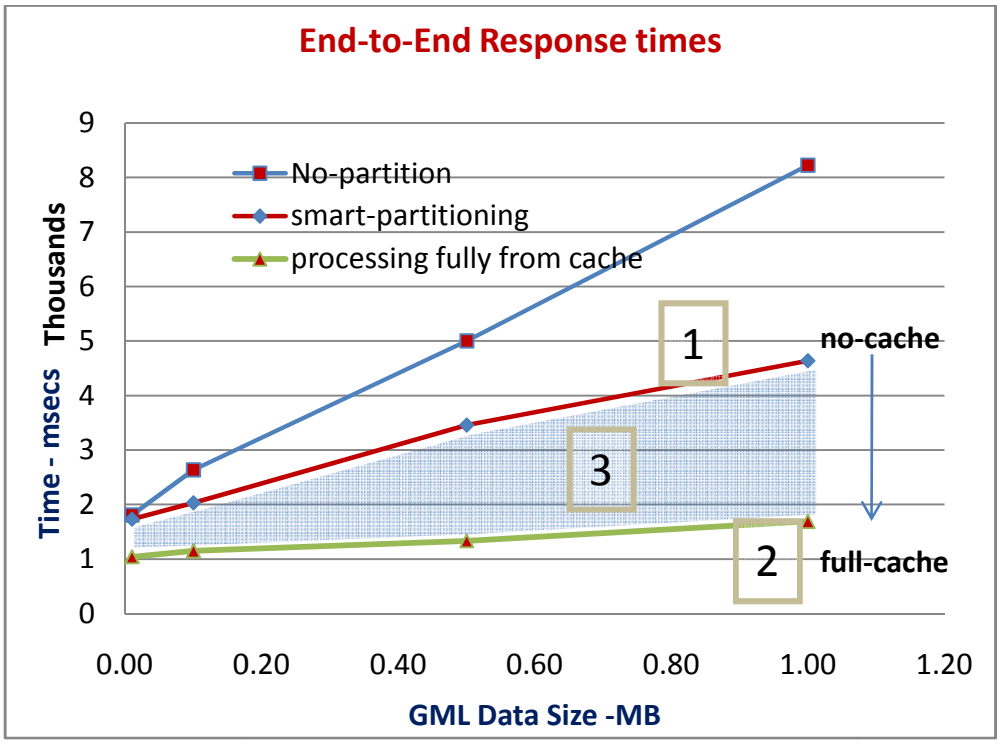

Figure 58: End-to-end response times according to possible query cases.

End-to-end response times fall in one of three general groups as shown in Figure 58.

1. Query is responded from only remote data sources.
    - There is no useful cached data. Cached data range and main query range are distinct. The requested map corresponding to the queried region is created from the data that needs to be fetched from the remote database through WFS mediation.
    - Federator applies load balancing and parallel processing through attribute based query decomposition. That is, partitioning main query bounding box into smaller regions and assign them separate threads.
2. Query is responded only from cache
    - This is the best case scenario. In this case map is created only from the cached data. There is no need to go to the originating data sources and databases through WFS mediation. The cache data is enough to respond whole map.
3. Query is partially responded from each, cache and remote resources
    - Performance results are obtained in between line 1 and 2. As the overlapped region gets larger, results gets close the green line. As the overlapped gets smaller, performance result gets close to red line.

Evaluation:

- The performance results here show round-trip times or in other words request-response times from the user-end point.

- Gain from making Open Standard's stateless GIS services state-full through novel client-based caching enabled us removing the repeated time consuming jobs and making the workload sharing and parallel processing. Overall these enhance the system responsiveness to a greater extend.

- Depending on the cache and main query overlapping size, response times changes between red-line (1) and green-line (2) in Figure 58

- The performance gain increases as the overlapped cached region increase. The green line shows that the entire queried region is responded from federator's cache for the corresponding bbox and data size. As the overlapping size increases, overall response time gets close to green line.

- In case of total overlapping, it looks like pre-fetching case. It will even be faster than pre-fetching case because federator does not need to render as large GML data as in the pre-fetching case. In pre-fetching case as mentioned, federator keeps whole GML representation of the data kept in the corresponding database.

- In case of the need to go to database and fetch the data system uses parallel processing with the partitioning obtained by smart-partitioning techniques.

- For the partitioning process we use 6 WFS workers to which partitions are assigned in round-robin fashion. In case of increasing the number of WFS, performance will be increasing.

-Lines 1 and 2 are not straight, there are some elbows. This is because of that we used separate regions having different characteristics of data in terms of distribution heterogeneity of the data in the selected region (bbox). It affects the efficiency of the

partitioning and performance results as well. In other words, if the data is distributed more homogeneous than the partitioning will make more just load balancing and the overall response times will be better.

- For 1MB of GML data, proposed federator-oriented performance enhancing techniques enhanced the system responsiveness 3 fold compared to single-flow data fetch shown as blue-line in the figure (also see Figure 34).

- As the data size falling in the selected bbox and/or homogeneity of data in terms of distribution increases, performance gain from load balancing through partitioning and parallel processing increases.

Chapter 7

Conclusion and Future Work

7.2. Summary and Conclusions

We proposed a Service-oriented architecture for understanding and managing the production of knowledge from the distributed observation, simulation and analysis through integrated data-views in the form of multi-layered map images. Infrastructure is based on common data model, standard GIS Web-Service components and a federator. Federator federates GIS services and enables unified data access/query and display/analysis over integrated data-views through event-based interactive display tools. Integrated data-views are defined in the federator's capability metadata as composition of layers provided by standard GIS Web-Services.

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 and inter-service communications through capability exchanges are the other issues serving optimization and architecture enhancement purposes. These are presented in Chapter 6.4 as federator-oriented high performance support techniques.

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 plottings) and integrate multiple information sources into interactive user interfaces such as digital photography, demographic information, and information from simulations.

Thesis is organized in three parts. The first part presents Web Service based reusable data Grid components for data access and map rendering in interoperable and extensible service oriented Geographic Information Systems. We present a component-based Service oriented architecture whose components are OGC compatible GIS Web Services. We merge two important software worlds: GIS and Web Service standards.

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.

Adopting Open Standards for GIS to Web Service standards make applications span cross-language, platform and operating systems. It also enables application

developers to integrate the third party geospatial functionality and data into their custom applications easily.

The second part presents the fine-grained dynamic information presentation through a federation framework enabling heterogeneous/autonomous data sources to be queried as a single resource over integrated data-view. Integrated data-views are multi-layered map images whose layers are created from the data provided by distributed standard GIS Web Service components presented in the first part of the thesis.

The federation is built over the proposed standard GIS data grid components based on common data model and their capability metadata definitions through the WMS-extended federator. The framework applies just-in-time (late-binding) federation in which the data is kept in its originating sources all the time. This enables autonomy and easy data maintenance.

The **last part** defines possible bottlenecks and corresponding optimization and enhancement techniques for the distributed heterogeneous information management systems in which the interoperability is achieved through XML encoded common data models and Service-oriented architecture. We analyzed/investigated the ways to turn the compliance requirements into competitiveness in general Geographic Information Systems domain.

We showed that publish-subscribe based messaging middleware can be used successively in Service-oriented GIS data Grid for high performance support in the production of multi-layered map images. The streaming technique also allows map-data rendering even on partially fetched GML data.

In such a framework built over common data model and standard service interfaces according to standard specifications, the repeated data validations are not crucial. In such cases, using pull parsing approach for handling XML-encoded data models give the best performance results in data rendering compared to other XML data handling approaches such as Document Object Model (DOM) and push approach (ex. Simple API for XML -SAX) .

The federator's inherent characteristics allowed us developing novel caching and parallel processing approach giving satisfactory performance results. Client-based caching enabled state-full access over stateless GIS services. Parallel processing technique's success depends on how well we share the workload to worker nodes. This is the challenge because of the un-evenly distributed and variable sized geo-data characteristics according to the location attribute. The proposed smart partitioning built over client-based caching gives the satisfactory performance results for the responsive GIS systems.

The best performance outcomes are achieved by applying pre-fetching technique. It is a kind of central approach to collect XML-encoded GML data provided by distributed standard GIS Web Services before run-time. Upon on-demand user queries, the federator responds from its local storage without going to remote databases through WFS mediation. Although it gives much better performance result, it might cause data inconsistency due to storing the data in intermediary storage at federator.

7.3. Summary of Answers to Research Questions

1. ***How to develop federated GIS data Grid architecture enabling fine-grained dynamic information presentation?***

By the federation, we mean providing one global view over several data sources and let them processed as one source. There are three 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); and the third is the common presentation model of data sources, i.e. mapping of common data model to a display model enabling integration/overlaying with other data sets (integrated data-view). The first two groups of research issues are related to lower level (database and files) data format/query/access heterogeneities summarized as semantic heterogeneity. In the proposed framework we take them for granted by applying Open Geographic Standards specifications for data models (GML) and online services (WMS and WFS). In order to adopt open standards with our aims and for the sake of responsiveness and quality of services we extended standard service components as Web Services with streaming data transfer and handling capabilities.

The proposed extended standard GIS Web Service components are integrated to the system through a federator which is actually a WMS extended with capability aggregating and state-full service capabilities to enable high performance support for responsive GIS applications. Actual federation is done through capability federation of standard GIS components at federator to create a global view over several data sources and let them processed as one source.

2. ***How can we incorporate widely accepted Open Geographic Standards (OGC) with Web Services standards?***

One of the major problems that have been acknowledged by almost all parties in the GIS world is interoperability. Hence any work in this area must consider the work that has been done previously and build upon. We have conducted extensive field work and background research to understand the open standards developed recently. We saw that the OGC based data and service standards are widely being used and adopted. Therefore we adopted the OGC standards for defining our GIS data products, and implemented our GIS Web Services to conform to these standards.

We demonstrated that the common industry standards can be incorporated with WSDL-SOAP based Web Services. We used these services in several Geo-science applications proved the usability of these services in real scientific world. This is described in detail in Chapter 5.

At the implementation level we have applied the incorporation technique to Web Map Services and Web Feature Services. We have created XML-based standard query schemas from the standard HTTP/GET-POST based query definitions which are actually attribute-value pairs. We have also defined standard services as Web Service in Web Service Description Language (WSDL) based on the services/functions provided.

3. ***How to make streaming data transfer between Open Standards GIS data services through messaging middleware?***

Although we gain a lot of advantages by extending Open Geographic Standards online data services as 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.

WFS's standard *getFeature* Web Service interface is used for the negotiation. The streaming data transfer is done between publisher in WFS and subscriber in WMS over the broker whose port, id and topic are agreed on.

Streaming data transfer capability of the services give better performance results besides enabling rendering of map images with partially returned data, no need to wait for whole data to be returned.

4. Can we build GIS like Federated Service-oriented Information System for any other science domains? What are the requirements, constraints and limitations?

Our experiences with GIS have shown that federated, service-oriented, GIS-style information model can be generalized to many application areas such Chemistry and Astronomy. We call this generalized framework Application Specific Information System (ASIS) and give blueprint architecture in terms of principles and requirements. This is presented in Chapter 4.4.

5. ***How to merge Asynchronous Java Script and XML (AJAX) with Web Services client stubs for event and browser-based interactive map client tools.***

AJAX and Web Services are XML based structures and this property allows 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 also reduces the workload of the network traffic.

This integration framework also enabled us to integrate the Google Maps with scientific geo-data layers from OGC compatible WMS and WFS services. The case scenario is given in Chapter 3.3.3.1.2.

6. ***Can we build <u>responsive</u> Service-oriented Geographic Information System requiring data access and rendering in which data is provided by autonomous resources in XML-encoded common data model (GML)?***

We have investigated some high-performance support technique to create responsive GIS systems as summarized below in the questions/answers. These are investigated in three main research questions listed as below.

- ***How to access stateless Open Standard's GIS data services in a state-full manner?***

Open Standard's GIS services are inherently stateless and based on on-demand processing. They don't maintain state information between message calls, and that causes poor performance results in response times because of redoing the time and

resource consuming repeated jobs. In order to make stateless services interaction state-full, we recommend using standard GIS services through the proposed federator.

Federator differentiates clients/sessions based on their IDs defined in their standard requests. If it is the first time call from a new session, federator creates a new session object and stores it to its session table. If it is a call from a client/session already in the table, federator only updates the object with the new information. Session objects are JAVA class objects maintaining necessary information for serving the clients' successive queries in much shorter time by using advanced client-based caching, load balancing and parallel processing over un-predictable workload in distributed data access and map rendering.

- ***How to make load balancing in federated GIS system?***

Federator inherently makes workload sharing by fetching the different data layers from separate resources to create multi-layered map image. This is a natural load balancing and parallel processing result from the architectural features.

A layer (in the multi-layered map image) itself can be split into smaller bounding box tiles and each tile can be farmed out to slave WFS/WMS. Layer-based partitioning is based on attribute-based query decomposition in which the attribute is the bounding box defining the requested data's range in a rectangular shape. This is presented in Chapter 6.4.3.2.

That approach gave us much better performance results compared to straightforward single process flow.

- ***How do you apply locality principles on un-predictable workload sharing?***

Geo-data is called un-predictable in terms of range queries the location attribute of the data. In this case decomposing the queried range into equal sized regions might not help. We developed a smart algorithm for that and applied to geo-data but can be applied to any other domain having that type of characteristics.

Locality means that close neighborhood in terms of bbox ranges have similar characteristic. So federator make an estimation over the unpredicted data falling in the queried region and make the best efficient partitioning on the main query to create response in a shortest time. We called this smart partitioning.

The smart partitioning is based on locality information and locality information is obtained from client-based adaptive caching. The client-based caching object keeps previous request's bbox value (describes queried region), corresponding data and data size falling into per unit square (density) in that region. By using these values, algorithm forecasts the workload of the next request and tries to partition main query as just as possible for parallel processing.

We saw that smart partitioning gave better performance results than the blind/static partitioning.

7.4. Future Research Directions

In this thesis we have outlined our initial research and implementations to build geophysical data Grid architecture enabling fine-grained information/knowledge presentations in multi-layered map images through novel federator architecture based on

common data model, standard GIS Web-Service components and a federator. We addressed several issues related to archival data access and processing from a single access point, and investigated high-performance design techniques to support responsive Geographic Information Systems.

Our work presented in this thesis was an example especially aimed towards Geo-science, and we believe it can be adopted for other domains. However the effects of domain specific requirements are not well understood. 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 domains. Our initial discussion for that is given in Chapter 4.4.

In the proposed federated GIS system, we use a static approach to create application specific hierarchical data layers in federator's aggregated capability metadata. Federated capability defining the data and corresponding data sources are not allowed to be changed or updated after application run. It would be useful to have a way for the system to automatically create/deploy and/or fix the required layers and add the services providing those layers and update the capability of the federator with those changes on-the-fly.

APPENDICES

APPENDIX A: Sample Request Instances to standard WMS Service Interfaces

i. GetCapability Request Instance

```
<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>
```

ii. GetMap Request Instance

```
<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>
```

iii. GetFeatureInfo Request Instance

```
<?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>
```

APPENDIX B: A Template Capabilities.xml file for WMS.

```
<?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>
```

APPENDIX C: A sample WMS Capabilities.xml Instance

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

        <!-- WORLD SEISMIC -->
        <Layer queryable="0" cascaded="1" noSubsets="0">
                <Title>World_Seismic</Title>
                <Abstract>Seismic data for the world</Abstract>
                <CRS>EPSG:4326</CRS>
                <Layer queryable="1" cascaded="1" noSubsets="0" fixedWidth="0"
                                                                fixedHeight="0">
                        <Name>Nasa:Satellite</Name>
                        <Title>Nasa:Satellite</Title>
                        <EX_GeographicBoundingBox>
                                <westBoundLongitude>-150</westBoundLongitude>
                                <eastBoundLongitude>-100</eastBoundLongitude>
```

```
                                        <southBoundLatitude>30</southBoundLatitude>
                                        <northBoundLatitude>50</northBoundLatitude>
                                    </EX_GeographicBoundingBox>
                                    <BoundingBox CRS="EPSG:26986" minx="189000"
                                miny="834000" maxx="285000" maxy="962000" resx="1" resy="1" />
                                    <MinScaleDenominator>0</MinScaleDenominator>
                                   <MaxScaleDenominator>100000000</MaxScaleDenominator>
                                </Layer>
                                <Layer queryable="1" cascaded="1" noSubsets="0" fixedWidth="0"

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

APPENDIX D: A sample Instance of 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>
```

APPENDIX E: A Simplified 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>
```

APPENDIX F: A Simplified WFS Web Services Service Definition file (WSDL)

```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="http://wfs.cgl"
        xmlns:apachesoap="http://xml.apache.org/xml-soap"
        xmlns:impl="http://wfs.cgl" xmlns:intf="http://wfs.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: 1.2beta3
      Built on Aug 01, 2004 (05:59:22 PDT)
    -->
  - <wsdl:message name="GetCapabilitiesRequest">
      <wsdl:part name="request" type="soapenc:string" />
    </wsdl:message>
  - <wsdl:message name="GetFeatureOnStreamResponse">
      <wsdl:part name="GetFeatureOnStreamReturn" type="soapenc:string" />
    </wsdl:message>
  - <wsdl:message name="DescribeFeatureTypeResponse">
      <wsdl:part name="DescribeFeatureTypeReturn" type="soapenc:string" />
    </wsdl:message>
  - <wsdl:message name="DescribeFeatureTypeRequest">
      <wsdl:part name="request" type="soapenc:string" />
    </wsdl:message>
  - <wsdl:message name="GetCapabilitiesResponse">
      <wsdl:part name="GetCapabilitiesReturn" type="soapenc:string" />
    </wsdl:message>
  - <wsdl:message name="GetFeatureRequest">
      <wsdl:part name="request" type="soapenc:string" />
      <wsdl:part name="nbHost_" type="soapenc:string" />
      <wsdl:part name="nbPort_" type="soapenc:string" />
      <wsdl:part name="pubTopic_" type="soapenc:string" />
    </wsdl:message>
  - <wsdl:message name="GetFeatureOnStreamRequest">
      <wsdl:part name="request" type="soapenc:string" />
      <wsdl:part name="nbHost_" type="soapenc:string" />
      <wsdl:part name="nbPort_" type="soapenc:string" />
      <wsdl:part name="pubTopic_" type="soapenc:string" />
    </wsdl:message>
  - <wsdl:message name="GetFeatureResponse">
      <wsdl:part name="GetFeatureReturn" type="soapenc:string" />
    </wsdl:message>
  - <wsdl:portType name="wfs">
    - <wsdl:operation name="GetCapabilities" parameterOrder="request">
        <wsdl:input message="impl:GetCapabilitiesRequest" name="GetCapabilitiesRequest" />
        <wsdl:output message="impl:GetCapabilitiesResponse" name="GetCapabilitiesResponse" />
      </wsdl:operation>
    - <wsdl:operation name="DescribeFeatureType" parameterOrder="request">
```

```
    <wsdl:input message="impl:DescribeFeatureTypeRequest" name="DescribeFeatureTypeRequest" />
    <wsdl:output message="impl:DescribeFeatureTypeResponse" name="DescribeFeatureTypeResponse" />
  </wsdl:operation>
- <wsdl:operation name="GetFeature" parameterOrder="request nbHost_ nbPort_ pubTopic_">
    <wsdl:input message="impl:GetFeatureRequest" name="GetFeatureRequest" />
    <wsdl:output message="impl:GetFeatureResponse" name="GetFeatureResponse" />
  </wsdl:operation>
- <wsdl:operation name="GetFeatureOnStream" parameterOrder="request nbHost_ nbPort_ pubTopic_">
    <wsdl:input message="impl:GetFeatureOnStreamRequest" name="GetFeatureOnStreamRequest" />
    <wsdl:output message="impl:GetFeatureOnStreamResponse" name="GetFeatureOnStreamResponse" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="wfsSoapBinding" type="impl:wfs">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
- <wsdl:operation name="GetCapabilities">
    <wsdlsoap:operation soapAction="" />
  - <wsdl:input name="GetCapabilitiesRequest">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://wfs.cgl" use="encoded" />
    </wsdl:input>
  - <wsdl:output name="GetCapabilitiesResponse">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://wfs.cgl" use="encoded" />
    </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="DescribeFeatureType">
    <wsdlsoap:operation soapAction="" />
  - <wsdl:input name="DescribeFeatureTypeRequest">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://wfs.cgl" use="encoded" />
    </wsdl:input>
  - <wsdl:output name="DescribeFeatureTypeResponse">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://wfs.cgl" use="encoded" />
    </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="GetFeature">
    <wsdlsoap:operation soapAction="" />
  - <wsdl:input name="GetFeatureRequest">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://wfs.cgl" use="encoded" />
    </wsdl:input>
  - <wsdl:output name="GetFeatureResponse">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://wfs.cgl" use="encoded" />
    </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="GetFeatureOnStream">
```

```
      <wsdlsoap:operation soapAction="" />
    - <wsdl:input name="GetFeatureOnStreamRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="http://wfs.cgl" use="encoded" />
      </wsdl:input>
    - <wsdl:output name="GetFeatureOnStreamResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="http://wfs.cgl" use="encoded" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
- <wsdl:service name="wfsService">
  - <wsdl:port binding="impl:wfsSoapBinding" name="wfs">
      <wsdlsoap:address location="http://gf12.ucs.indiana.edu:7312/wfs-streaming-service/services/wfs" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

APPENDIX G: 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>
```

APPENDIX H: Sample GML document for earthquake fault data. This is simplified document to give an idea about the common data model.

```
<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>
```

APPENDIX I: Sample GetFeature Response from

```
<?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>
```

REFERENCES

[Lu 2006] Wei Lu, Kenneth Chiu, and Yinfei Pan, "A Parallel Approach to XML Parsing". In *the 7th* IEEE/ACM International Conference on Grid Computing, *2006.*

[Pallickara2003] 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.

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

[Sosnoski] 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

[Alexander] Aleksander Slominski. XML Pull Parser, visited 04-15-02. http://www.extreme.indiana.edu/xgws.

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

[GML] 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.

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

[Kris03] Kris Kolodziej, OGC OpenGIS consortium, OpenGIS Web Map Server Cookbook 1.0.1, OGC Document #03-050r1, August 2003

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

[Booth]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.

[Tran] Tran, P., Greenfield, P., and Gorton, I., *Behavior and Performance of Message-Oriented Middleware Systems.* . Proceedings of the 22nd international Conference on Distributed Computing Systems, ICDCSW. 2002.

[ogc] The Open Geospatial Consortium, Inc. web site: http://www.opengeospatial.org

[deegree] deegree project web site available at http://deegree.sourceforge.net/

[minmapserv]University of Minnesota Map Server, available at http://mapserver.gis.umn.edu/

[Patterninfo] Tiampo, K.F., Rundle, J. B., McGinnis, S. A., & Klein, W., , *Pattern dynamics and forecast methods in seismically active regions.* Pure and Applied Geophysics (PAGEOPH), 2002(159): p. 2429-2467).

[Tiampo] Tiampo, K.F., Rundle, J. B., McGinnis, S. A., Gross, S. J. & Klein, W., *Eigenpatterns in Southern California seismicity.* . J. Geophys. Res. , 2002. 107(B12): p. 2354.

[Nanzo06] K. Z. Nanjo, J.R.H., C. C. Chen, J. B. Rundle, and D. L. Turcotte. , *Application of a modified Pattern Informatics method to forecasting the locations of future large earthquakes in the central Japan,* . Tectonophysics, 2006. 424: p. 351-366

[Denning] P.J. Denning and S. C. Schwartz, Properties of the working-set model. Communications of the ACM, 15(3), March 1972.

[Belur] Belur V. Dasarathy, editor (1991) *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, ISBN 0-8186-8930-7.

[AjaxSerrano] Nicolas Serrano, Juan Pablo Aroztegi, Ajax Frameworks for Interactive web apps, IEEE Software Magazine V24n5 (Sep/Oct 2007) pp12-14.

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

[Googlemap] Project web site is available at http://code.google.com/apis/maps/index.html

[tanenbaum] Andrew S. Tanenbaum, Modern Operating Systems, 2nd ed., Upper Saddle River, NJ: Prentice Hall, 2001.

[apache] Apache Tomcat, http://tomcat.apache.org/.

[Foster04] Foster, I. and Kesselman, C., (eds.) The Grid 2: Blueprint for a new Computing Infrastructure, Morgan Kaufmann (2004).

[Berman03] Fran Berman, Geoffrey C, Fox, Anthony J. G. Hey., Grid Computing: Making the Global Infrastructure a Reality. John Wiley, 2003.

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

[Booth04] 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.

[Aydin] 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

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

[Belwood] 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.

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

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

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

[Rmi] 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.

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

[Peng03] Peng, Z.R. and M. Tsou, Internet GIS: Distributed Geographic Information Services for the Internet and Wireless Networks. 2003: Wiley

[Lansing02] Jeff Lansing., OWS1 Coverage Portrayal Service (CPS) Specifications 1.0.0, Document #02-019r1 February 2002.

[Sayar06] 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.

[Vretanos01] Vretanos, P.A., *Filter Encoding Implementation Specification. OGC 02-059. Ver 1.0. 0*. 2001. p. 02-059.

[Rao00] Rao, A.P., et al., *Overview of the OGC catalog interface specification*. 2000.

[Galipthesis] Galip Aydin, “Service Oriented Architecture for Geographic Information Systems Supporting Real Time Data Grid” Indian University thesis, February 2007.

[Votable] Williams, F. Ochsenbein, C. Davenhall, D. Durand, P. Fernique, D. Giaretta, R. Hanisch, T. McGlynn, A. Szalay and A. Wicenec, “VOTable: A Proposed XML Format for Astronomical Tables”, version 1.0, R., 15 April 2002.

[Cml] G. L. Holliday, P. Murray-Rust, H. S. Rzepa, Chemical Markup, XML and the Worldwide Web. Part 6. CMLReact; An XML Vocabulary for Chemical Reactions, J. Chem. Inf. Mod., 2006, 46, 145-157.

[Mathml] Buswell, Steven; Devitt, Stan; Diaz, Angel; et al (7 July 1999). Mathematical Markup Language (MathML), 1.01 Specification (Abstract).

[Mitchell] 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

[Ecma1] EcmaScript project web site is available at http://www.ecmainternational.org/

[Ecma2] ECMAScript Language Specification. ECMA International, third edition, 1999.

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

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

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

[Meyer03] Thomas W. Meyer, et al., *The Los Alamos Center for Homeland Security.* LOS ALAMOS SCIENCE, 2003. 28

[Bush04] Bush, B.W., *NISAC Interdependent Energy Infrastructure Simulation System, Report LA-UR-04-7700,*. 2004, Los Alamos National Laboratory.

[OnEarth] OnEarth, *NASA OnEarth Web Map Service for global satellite images, available at http://onearth.jpl.nasa.gov/.*

[Holliday05] Holliday, J.R., et al., *A RELM earthquake forecast based on pattern informatics*, in *AGU Fall Meeting;*. 2005: San Francisco, California,.

[Rundle03] Rundle, J.B., D.L. Turcotte, and R. SHCHERBAKOV, KLEIN, W., AND SAMMIS, C. , *Statistical physics approach to understanding the multiscale dynamics of earthquake fault systems.* . Rev. Geophys. , 2003. 41(4).

[Hpsearch] 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.

[Gadgil05] 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.

[Hpsearch1] Gadgil, H., et al., *HPSearch: Service Management & Administration Tool*, in Abstract for VLAB Meeting Minnesota July 21-23 2005. 2005

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

[Rundle2002] Rundle, JB, PB Rundle, W Klein, J Martins, KF Tiampo, A Donnellan and LH Kellogg, GEM plate boundary simulations for the Plate Boundary Observatory: Understanding the physics of earthquakes on complex fault systems, Pure and Appl. Geophys., 159, 2357-2381 2002.

[Donnellan] Donnellan, A., et al., Numerical simulations for active tectonic processes: increasing interoperability and performance.

[Servo] en, 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.

[Cce] 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.

[Fox2005] 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.

[Bush03] 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

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

[Rabiner]Rabiner, L.R., A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. Proceedings of the IEEE, 1989. **77**(2): p. 257–286.

[Granat] Granat, R.A., A method of hidden Markov model optimization for use with geophysical data sets. Comp. Sci., 2003(2659): p. 892–901

[Quaketables] Andrea Donnellan, et al, QuakeTables Fault Database for Southern California. Approved project documentation available from http://quakesim.jpl.nasa.gov/QuakeTables_Doc.pdf

[Papakonstantinou95] 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

[Li1998] Chen Li , Ramana Yerneni , Vasilis Vassalos , Hector Garcia-Molina , Yannis Papakonstantinou , Jeffrey Ullman , Murty Valiveti, Capability based mediation in TSIMMIS, ACM SIGMOD Record, v.27 n.2, p.564-566, June 1998

[Papakonstantinou96] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object Fusion in Mediator Systems. In International Conference. on Very Large Data Bases (VLDB), 1996.

[Tomasic98]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.

[Levy96]A. Levy, A. Rajaraman and J. Ordille: Querying Heterogeneous Information Sources Using Sources Descriptions. Proceedings of VLDB, 1996: 251-262.

[Wiederhold92] Wiederhold G (1992), Mediators in the Architecture of Future Information Systems. IEEE Computer, 25, 3, 38-49.

[Alameh03]Alameh N., Chaining geographic information web services, IEEE Internet Computing, Sept-Oct 2003, 22- 29.

[Jerome05] S. Jerome, Web Map Context Service (WMC), OGC project document reference number OGC 05-005. Version 1.1.0. 2005.

[Sayartech] Ahmet Sayar, Marlon Pierce, Geoffrey Fox OGC Compatible Geographical Information Services Technical Report (Mar 2005), Indiana Computer Science Report TR610.

[Bunting03] 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.

[Liping03] 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.

[Boyd04] 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, Proceedings. Springer-Verlag, 2004.

[Friedman99] Mark Friedman, L. Y. Alon and M. D. Todd. Navigational Plans For Data Integration. In AAAA/IAAI, pages 67-73, 1999.

[Lenzerini02] Maurizio Lenzerini, Data Integration: A Theoretical Perspective. In PODS Proceedings, PAGES 233-246, 2002. Invited Tutorial.

[Busse99] 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.

[Bigagli06] 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.

[Fox04] Fox, G., *Grids of Grids of Simple Services.* Computing in Science and Engg., 2004. 6(4): p. 84-87.

[Kelvin04] Kelvin K. Droegemeier, et al. *Linked environments for atmospheric discovery (LEAD): A cyberinfrastructure for mesoscale meteorology research and education*. in *20th Conf. on Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology,* . 2004. Seattle, WA.

[Plale06] Beth Plale, et al., CASA and LEAD: Adaptive Cyberinfrastructure for Real-Time Multiscale Weather Forecasting IEEE Computer, 2006. 39(11): p. 56-64.

[Beth06] Beth Plale, Rahul Ramachandran, and S. Tanner, Data Management Support for Adaptive Analysis and Prediction of the Atmosphere in LEAD, in 22nd Conference on Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology (IIPS),. 2006.

[Zaslavsky04] Zaslavsky, Ilya, Ashraf Memon, "GEON: Assembling Maps on Demand from Heterogeneous Grid Sources", Twenty-fourth Annual ESRI International User Conference, August 9-13, 2004, San Diego, CA.

[Bhatia00] Bhatia, Karan, Ashraf Memon, Ilya Zaslavsky, Dogan Seber, Chaitan Baru, "Creating Grid Services to Enable Data Interaoperability: Example from GEON Project", GSA Annual Meeting in Seattle, WA November 2-5, 2000.

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

[Laits] LAITS project web side is available at http://grid.laits.gmu.edu

[Chu06] Kai-Dee Chu, Liping Di, Peter Thornton: Introduction of Grid Computing Application Projects at the NASA Earth Science Technology Office. 289-298, 2006.

[Allock03] W. Allcock et al. GridFTP: Protocol extensions to FTP for the Grid. GGF Document Series GFD.20, Apr. 2003. Also available as http://www.ggf.org/documents/GFD.20.pdf

[Foster97]I. Foster and C. Kesselman. Globus: A meta-computing infrastructure toolkit. International Journal of Supercomputer Applications, 11:115–128, 1997

[Wsrf] Web Services Resource Framework, www.globus.org/wsrf/

[Novotny04] Jason Novotny , Michael Russell , Oliver Wehrens, GridSphere: An Advanced Portal Framework, Proceedings of the 30th EUROMICRO Conference (EUROMICRO'04), p.412-419, August 31-September 03, 2004

[Russell04] M. Russell, G. Allen, G. Daues, I. Foster, E. Seidel, J. Novotny, J. Shalf, G. Laszewski The Astrophysics Simulation Collaboratory: A Science Portal Enabling Community Software Development. Cluster Computing 5(3): 297-304 (2002).

[Doyle] Doyle, A. and Reed, C.: Introduction to OGC web services: OGC interoperability program white paper. http://ip.opengis.org/ows/

[Rew1990] R. Rew and G. Davis, "The Unidata netCDF: Software for Scientific Data Access," Sixth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography and Hydrology, Anaheim, CA, February 1990.

[Williams02] Williams, R., et al. 2002, VOTable: A Proposed XML Format for Astronomical Tables, version 1.0, available at http://cdsweb.u-strasbg.fr/doc/VOTable/

[Yasuda04] Yasuda, N. et al. "Astronomical Data Query Language: Simple Query Protocol for the Virtual Observatory", in ASP Conf. Ser., Vol. 314, ADASS XII, ed. F. Ochsen-bein, M. Allen, & D. Egret (San Francisco: ASP), p.293, 2004.