Service Oriented Architecture for Geographic Information Systems Supporting Real Time Data Grids

DISSERTATION

Galip Aydin

Submitted to the faculty of the University Graduate School
in partial fulfillment of the requirements
for the degree
Doctor of Philosophy
in the Department of Computer Science
Indiana University
February 2007

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Doctoral Committee

Prof. Geoffrey C. Fox (Principal Advisor)

Prof. Dennis Gannon

Prof. David Leake

Prof. Beth Plale

January 15, 2007

Table Of Contents

List of Figures

Chapter 1

Introduction

Geography is the science of studying earth and its features and of the distribution of life on the earth, including human life and the effects of human activity [1]. Although geography is often associated merely with studying places and maps these are not the only areas its scope encompasses. Geography studies physical and human landscapes, the reasons for their spatial variation, why and how they change over time and the dynamics behind these changes. From creating simple maps to statistical analysis of population distribution to effects of pollution on rain forests geographic data and tools are being widely utilized by academia, industry and the governments for understanding almost every aspect of the modern life.

A Geographic Information System is a system for creating, storing, sharing, analyzing, manipulating and displaying spatial data and associated attributes. It can be used for creating or capturing geographic information from various sources in digital form or for viewing available geographically referenced data in human recognizable

formats. Perhaps the simplest example of Geographic Information Systems is widely available map viewers which process layers of geospatial data to create map images.

Geographic Information Systems are used in a wide variety of tasks such as urban planning, resource management, environmental impact assessment, emergency response planning in case of disasters, crisis management and rapid response etc. Although these seem to be relatively independent and different areas a common feature of almost all GIS use cases is the need for the system to relate information from different sources. For instance a GIS framework to help planning adequate response in case of a natural disaster such as a powerful earthquake would require latest information about the strength of the earthquake, detailed images of the affected areas with and inhabited places clearly marked, the names of these places, population density, usable roads and railroads, information about the energy and natural gas lines, hospitals, police headquarters, buildings that can be used for relocating affected people such as schools and public buildings etc. The list can go indefinitely but it gives an idea about unique characteristics of the GIS.

The 20th century saw the birth of the GI Systems, and their journey from centralized mainframe systems to desktop systems and finally to distributed systems [2]. Today a modern GIS requires distributed systems support at two levels; first for accessing various geospatial databases to execute spatial queries and second for utilizing remote geographic analysis, simulation or visualization tools to process spatial data.

In a relatively short period of time the Internet has dramatically changed how scientists, industry specialists and the public access, exchange and process information. As in most other cases the geospatial data access and dissemination methods also

significantly evolved. This helped academia, governments and businesses to have easy access to substantial amount of geospatial data. Today hundreds of spatially enabled web sites allow users to make spatial queries, create/view/manipulate interactive high-quality online maps or search and find national or global geographic data. The governments are working to establish national spatial data and services infrastructures to satisfy ever-increasing demand from public for more and higher quality geospatial data. For instance the Federal Geographic Data Committee under the National Geospatial Programs Office is established to promote the coordinated development, use, sharing, and dissemination of geospatial data on a national basis. It maintains the National Spatial Data Infrastructure (NSDI) Clearinghouse Network which is *"a community of distributed data providers who publish collections of metadata that describe their map and data resources within their areas of responsibility, documenting data quality, characteristics, and accessibility.*" [3]. Similar clearinghouses or data warehouses have been created by others as well [4].

Although the advances in internet and distributed computing provided easy access to distributed data products several issues still need to be resolved. These issues also constitute the motivations of our research and explained in the next section.

1.1 Motivation

1.1.1 Problems with the Traditional GIS Approaches

The desktop GIS applications conventionally used to access and analyze local data do not have the ability to interact with online data sources and with other spatial analysis applications. To be able to interact with online geospatial resources the traditional Desktop GIS programs are evolving into distributed applications, compatible with

various distributed systems architectures. As a result client-server based distributed GIS applications have been introduced to fill the gap, and the GIS companies and research groups have developed their own spatial databases along with various data access and manipulation tools.

However because of the proprietary design of these applications interoperability at the application level has always been a significant bottleneck. For instance in addition to the various types of GIS analysis applications there are at least three major types of GIS servers used by different Indiana State counties, and these servers are not compatible with each other [5]:

- ESRI [6] ArcIMS and ArcMap Server (Marion, Vanderburgh, Hancock, Kosciusko, Huntington and Tippecanoe counties)
- Autodesk MapGuide [7] (Hamilton, Hendricks, Monroe and Wayne counties)
- WTH Mapserver Web Mapping Application [8] (Fulton, Cass, Daviess and City of Huntingburg counties) based on several Open Source projects.

We also observe the same interoperability problem at the data level. This is perhaps due to the aggressive policies GIS companies have embarked in early years of the GIS development to discourage switching between different suites. There are numerous ways of describing geospatial data in various formats such as ESRI shape files [9], ASCII files, XML files, Geography Markup Language (GML) [10] files etc. Following table gives a comparison of some of the GIS software file formats:

Table 1-1 – Software File-Format Chart (Source: Geo Community, [11])

	tiff	.tfw header	.dxf	dem	.eoo	.shp	.jpg	igds/.dgn	.mif	.dlg	.sdts	dted	Tiger
AutoCad													
V.13	2	2	1	3	1	1	1	1	1				
V.14	1	2	1	3	1	1	1	1	1				
Cad Overlay GSX	1	1	1										

Caliper Corp.													
GIS Plus	1	2	1	1									
Maptitude	1	2	1	1	1	1			1	1			1
Trancad	1	2	1	1									
ENVI	1	1	3										
ERDAS	1	1	1	1									
ERMAPPER	1	1	1	1	1		1	1		1		1	
ESRI													
ARC/INFO	1	1	1	1	1	1	1	1	1	1	1		1
ArcView	1	1	1	2		1	1		1				2
Data Automation Kit				1	1	1			1				1
Geographix	1	1	1										
Genasys	1	1	1	1									
GRASS	1	1	1	1	1								
HASP	3	3	1	3									
IDRISI	1	1	1	3									
Intergraph													
IRAS C	1	3	3	3									
PowerRas	1	3	3	3									
Vista Map	1	3	3	3									
Landform Gold	1	1	3										
Microstation				3									
95	1	2	1										
Descartes	2	2	3	3									
MapInfo	1	3	1		1	1		1	1				
PCI Remote	1	1	1	1									
Terramodel	1	1	1	1									
TNT Mips	1	1	1	1									
TOPO+	1	1	3	3									
Vertical Mapper	2	3	2	2									
1= Full compatibility													
2= Compatible only with third party software													
3= Not compatible at all													

Therefore, the unique properties of the geospatial data such as different resolutions and scales of the same domain caused data providers to create different ways to describe the same geospatial entity which in turn resulted in numerous incompatible formats.

We identify several problems with traditional distributed GIS approach:

1 Problems with assembling data: Because of the distributed nature of geospatial data, users are required to utilize different tools to access data in various FTP or HTTP servers, relational or XML databases etc. In addition to the archived data

real-time data providers employ different communication and data transport protocols which further complicates the access.

2 Data format problems: Depending on the user's choice of software, applications that digest geospatial data require input in different formats. Users spend significant amount of time converting data from one format to other to make it available for their purpose.

3 Amount of resources for processing data: After the data is collected and converted into a usable format, enough hardware and software resources need to be allocated for analyzing the data. In most cases the amount of collected data reaches to an amount in the order of gigabytes or even terabytes, handling this data becomes a challenge for most users and organizations. Also, simulation and visualization software used in conjunction require high performance computing platforms which are unreachable for common users.

As a result, today, due to the distributed nature of the geospatial data and the variety of data and application standards the GIS community faces the following challenges:

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

Undoubtedly these issues are the focal point of numerous research and development efforts [12] [13]. Especially the problems related to data formats and standards are being addressed by a number of groups and organizations some of which also offer solutions to the application level interoperability issues [14-18]. We summarize these standards based efforts in Chapter 4.

However most of the distributed GIS services approaches are based on more traditional client-server models and lacks the potential of easily linking distributed computational components.

1.1.2 Sensors and Real-Time Data Access in GIS

Another very important and relatively less explored issue in geospatial world is the real-time data access and their integration with Geographic Information Systems. Thanks to the advancements in sensor technology a revolution is slowly taking shape in terms of data acquisition in a growing number of fields [19, 20]. Profound effects of sensors in GIS related sciences such as in environmental monitoring, earth observation, real-time pollution monitoring are becoming more and more visible [21-24].

GIS related use of sensors vary to a great extent; they can be used in monitoring the water level of the rivers, or the number of vehicles passing through bridges at certain times of the day, or recording humidity in the air etc [25] [26]. But what is common is that all of these measurements are used by some GIS framework for statistical or practical purposes. However since the traditional approach of the GIS frameworks is based on accessing and using geographic data from archives or spatial databases integrating these sensor measurements with the geo-processing tools is a problematic issue, especially in real-time [27].

There are some recent efforts to present unified interfaces to sensors and sensor measurements such as OGC Sensor Web Enablement [28]. We summarize these efforts in Chapter 6. However the GIS community today needs Service Oriented approaches for coupling real-time sensor measurements with the data analysis tools.

1.2 Research Issues

In this dissertation we investigate the issues pertaining to the traditional Geographic Information Systems approaches and propose solutions to these problems based on modern Service Oriented Grids approaches.

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

Distributed data access in GIS is traditionally regarded as dealing with distributed data archives, databases or files. However modern scientific applications especially real-time data processing tools require continuous data streaming. High-rate data streaming is also important for applications such as decision making tools that require fast data access.

We identify two major types of geographic data based on their sources: real-time measurements acquired from sensors and archival data stored in spatial databases. The connection between sensors and Geographic Information Systems is particularly strong

because the measurements are most likely to be used by these systems for analysis or statistical reasons.

This thesis is about developing a Web Services architecture that provides access to both types of the geographic data products, manage data sources, connect them to the geo-processing applications and allow users to access them in common formats. The thesis implementation encompasses development of GIS data services, high-performance streaming data services, integrating messaging system with these services, composition of GIS services in scientific workflows, real-time data filters and coupling scientific geophysical applications with real-time and archival data.

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

- Can we implement unified data-centric Grid architecture to provide common interfaces for accessing real-time and archival geospatial data sources?
- How can we incorporate widely accepted geospatial industry standards with Web Services?
- Are the performance of the Web Services acceptable for Geographic Information Systems and how can we make performance improvements?
- How can we build services for supporting scientific GIS applications that demand high-performance and high-rate data transfers?
- How can we build a Grid architecture to couple real-time sources with scientific applications that also provides high interactivity and performance?
- What is the way for managing real-time data filters using Web Services?

- Can we organize and manage real-time sensor data products using publish-subscribe systems? Are the mechanisms of topic based publish/subscribe systems appropriate?
- Is the performance of the Real-Time Data Grid acceptable for uninterrupted, continuous operation?
- Will the Real-Time Data Grid implementation scale for large number of data providers such as sensors and clients?

1.3 Organization of the Dissertation

This thesis is organized as follows. The first chapter consists of an overview of the Geographic Information Systems, a summary of the outstanding issues that relate to the research outlined in this thesis and the research questions. Chapter 2 contains short reviews of some of the related projects and motivating geophysical applications. Our system is an example of the Grids of Grids paradigm [29] which consists of two major architectural components; the first is a Data Grid for archival geographic data, and the second is the Real-Time Data Grid for sensors. In Chapter 3 we give an overview of the overall architecture and explain the major components of the system.

Chapter 4 and Chapter 5 present the High Performance GIS Data Grid architecture for archival geospatial data, and a detailed performance study. In chapter 4 we explain the design principles of the Data Grid architecture for GIS and give implementation details. In this chapter we present our approach for creating GIS Web Services and methods for improving the performance of these services. We introduce streaming GIS Web Services for high performance and high rate data transfer. In Chapter 5 we present a detailed

performance study of the GIS Data Grid and the streaming services introduced in Chapter 4.

Chapters 6 and 7 introduce the Real-Time GIS Data Grid and its performance studies. In Chapter 6 we present a novel Grid architecture which consists of filter Web Services, Grid Messaging Substrate and Information Services. This architecture is designed to provide continuous streaming access to sensor messages. We also present metadata descriptions for real-time filters and filter chains. This chapter presents several use cases of the Real-Time Data Grid architecture with real-time GPS streams. Chapter 7 presents a detailed performance study of the Real-Time Data Grid architecture. We outline several test cases and give performance results. The tests help us determine the limits of the system in terms of the maximum number of data producers and clients that can be supported.

In Chapter 8 we give answers to the research questions identified in Chapter 1, outline future research directions and conclude the dissertation.

Chapter 2

Related Work and Motivating Use Cases

While writing about Geographic Information systems one must acknowledge the tremendous amount of work done for more than half a century. Since the first true operational GIS framework “Canadian Geographic Information System” (CGIS) was put in service in 1964, there have been many successful systems developed and used. Later the first mainframe GIS examples are replaced by the more modular Desktop based systems working on UNIX workstations and Personal Computers (PC). With the development of sophisticated networking methodologies, access to distributed geographic data and geo-processing applications become much easier. Today widely known online mapping applications such as Google Maps, Microsoft Visual Earth, and Yahoo Maps provide GIS services to ordinary Internet users.

This research is mainly constructed around the Geophysical Grid notion and it aims to provide tools for coupling scientific geophysical applications such as RDAHMM,

Pattern Informatics etc with real-time and archived geographic data. The research is divided into two major parts; the first part is about issues related to creating a Service Oriented Architecture for geographic data, proposed improvements and benchmarks, and the second part is defining principles and developing a prototype implementation for supporting sensors and real-time data in Grid environments. This logical separation of geo-referenced data helps us clearly define the boundaries of the research and the requirements for the two distinct Grids we build.

In this chapter we summarize several well-known projects in the community, which are also closely related to our work. However it should be noted that it is not possible to mention all related projects here because of the sheer number of work done or currently being researched. Also it should be noted that the term GIS relates to many different scientific fields, but we are only interested in Information Technology aspects of it. Additionally we summarize some of the scientific applications which motivated this research.

2.1 Related Projects

2.1.1 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 [30].

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 [31].

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 [32]. 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 [32].

LEAD provides a web-portal as the entry point for students, users or advanced researchers to the meteorological data, services, models, and workflow, analysis and visualization tools related to the project. Users can interactively explore the weather as it evolves, create custom scenarios or acquire and process their own data [33].

An important issue researched by LEAD scientists is adaptive workflow, termed as Workflow Orchestration for On-Demand, Real-Time, Dynamically-adaptive Systems

(WOORDS). This approach allows use of tools such as analysis or visualization applications, models or data repositories as dynamically adaptive, on-demand Grid enabled systems instead of static data consumers with fixed configurations. The dynamic nature of the system allows a workflow configuration to be changed in response to weather, respond to user decisions, initiate other processes and interact with remote observing technologies to optimize data collection for current problem [34]. The workflow tools provided by LEAD automates many of the otherwise time consuming and complicated tasks by linking data management, assimilation, forecasting, and verification applications into a single experiment [LEAD Portal].

2.1.2 OPeNDAP

Open-source Project for a Network Data Access Protocol (OPeNDAP) is a framework that aims to simplify all aspects of scientific networking. OPeNDAP or formerly known as DODS (Distributed Oceanographic Data System) allows access to scientific data over the internet from applications that were not specifically designed for that purpose. There are also some applications designed to communicate with OPeNDAP servers. To access data OPeNDAP provides a URL, however to retrieve the data using the provided URL users need to know the type of the data and how to request it. By default OPeNDAP data is stored and transmitted in binary format. To provide some information about the data OPeNDAP provides *Dataset Descriptor Structure* (DDS). DDS is OPeNDAP's version of metadata in a C-like syntax. Users can access the DDS of a particular data set by appending .dds to the URL [35]

Distributed Oceanographic Data System or DODS was originally initiated in early 1990's as a system that would facilitate scientific data exchange between researches, archives, industry specialists etc [36]. Two fundamental design criteria was 1) servers must be easy to install. And 2) the system must be compatible with the existing software. Built to satisfy these criteria today OPeNDAP is one of the major tools used especially by ocean scientists to share data across globe. The key futures of OPeNDAP is that it allows access to data from a wide variety of programs including existing applications, and it provides network versions of Application Program Interface (API) libraries for most commonly used data formats such as NetCDF, HDF, JGOFS and several others. Thus it allows users to continue to use their applications with OPeNDAP support.

The OPeNDAP architecture uses the World Wide Web model, or the client/server model, where the browsers submit requests to the servers and the servers respond with the data that make up the web pages. In addition to the requesting data from the servers OPeNDAP allows clients to browse data, request data to be translated and delivered in some particular format, request specific part of the data etc. Also OPeNDAP allows researchers to convert their data analysis programs such as Matlab, Ferret, IDL into specialized web browsers.

How an OPeNDAP server and client communicate with each other is defined by OPeNDAP protocol which consists of four components: 1- A data model used to transport data from one source to another, 2- A Data Descriptor Structure (DDS) which describes the data structures to be filled by the accompanying data, 3- A procedure for

retrieving data and the DDS from the remote source, 4- An API to implement this protocol [36].

The advantages of using OPeNDAP to share data over the web is explained as follows [37]:

1. OPeNDAP server and client cooperate to deliver the data in the particular format in which the analysis application expects, so the user need not learn about various archival formats.
2. OPeNDAP allows users to sample the datasets in the formats supported by his/her analysis package, thus unnecessary data exchange over the Internet is prevented.
3. Most of the search and sampling is performed on the server machines which reduce the Internet traffic and decrease the load on the local machine.

The interaction between an OPeNDAP client and a server can be summarized in four steps: User sends a request to the OPeNDAP server via URLS, the URL is passed to the HTTP server via an OPeNDAP client, the data are returned via HTTP and the OPeNDAP client reformats the received data for user's analysis package. There are three basic data object types provided by OPeNDAP:

1. Data Descriptor Structure (DDS), which describes the structure of the data set and provides syntactic metadata,
2. Data Attribute Structure (DAS) semantic metadata which gives the attributes values of the fields described in the DDS,

3. The data (DODS data), the actual data in a binary structure.

Major advantages of OPeNDAP over conventional file transfer protocols such as FTP are its ability to sample data, or request only subsets of data, and the ability to aggregate data from several remote resources in one transfer operation.

2.1.3 ROADNet

The Real-time Observatories, Applications, and Data management Network (ROADNet) (http://roadnet.ucsd.edu), is a large scale project involving several different types of scientific research areas and communities. ROADNet research focuses on resolving challenges related to building wireless sensor networks for various types of observations and the information management system which will deliver this sensor observation in real-time to the users.

The goal of creating sensor networks to measure various entities is to enhance our capacity to monitor and respond to changes in our environment. ROADNet collects seismic, oceanographic, hydrological, ecological and physical data and streams to a variety of end users in real-time [38].

ROADNet utilizes Object Ring Buffers (ORB) to capture real-time data from sensor networks [39]. ORBs capture data from the sensors and store them for immediate use as well as archive them for further analysis. ORB allows users to access data through a well-defined API and provides metadata about the raw sensor observations.

To enhance the capabilities of the ORB for real-time data exchange and dynamic reconfiguration ROADNet team has developed Virtual Object Ring Buffers or Virtual ORBs (VORB) [40]. VORB provides transparency and independence from the physical

ORB implementation which can be useful in federated sensor networks to act as data exchange points or in developing test beds for virtual sensor networks [39].

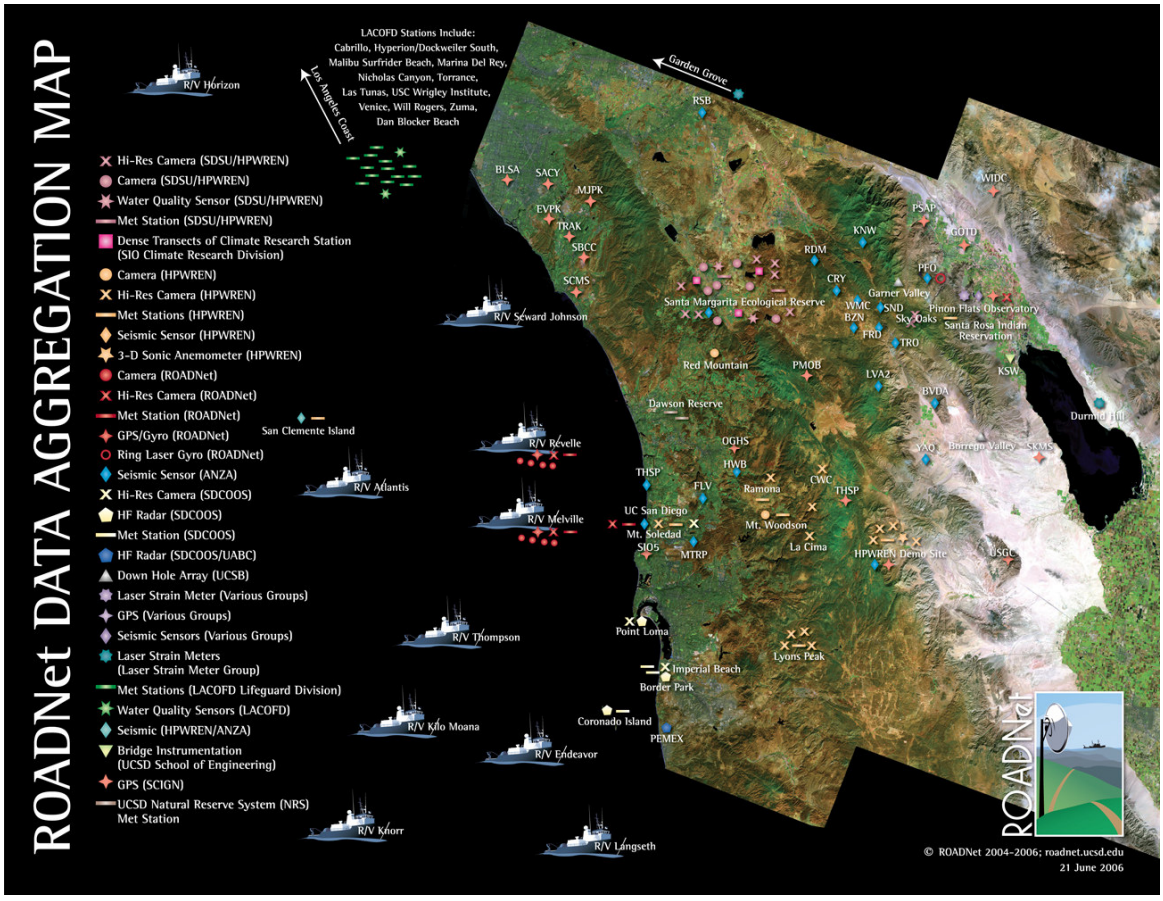


Figure 2-1 Current ROADNet Sensor map (image taken from http://roadnet.ucsd.edu/index.html) shows several different types of sensors being utilized.

ROADNet research spans following scientific areas (source: http://roadnet.ucsd.edu/field_research.html):

Ecology: By employing a multiplicity of sensors to monitor environmental conditions such as physical, chemical and biological variables.

Geodesy: GPS stations are used for high-resolution, high-rate, precise position measurements. The GPS streams are made available through internet which can be used by surveyors to obtain real-time three-dimensional position fixes with high-level vertical and horizontal precision.

Hydrology: Sensors are used to monitor mountainous watersheds, precipitation, runoff, and weather and water quality. ROADNet is proposing to build a remote sensor network to continuously monitor hydrological and meteorological conditions of the water resources.

Oceanography: Implementing continuous real-time data delivery helps observational oceanography at several levels, such as understanding and modeling the ocean and underlying crust and mantle using real-time and past data; taking necessary steps in any emergency situations, or in educational, recreational or business purposes.

Seismology: ROADNet provides Virtual Seismic Network (VSN) for integrating real-time data from multiple disparate seismic networks. Currently 550 globally distributed stations are accessible through internet.

2.2 Data Stream Processing and Management

One of the major parts of our research is about processing real-time sensor streams on-the fly. To achieve this goal we have developed real-time distributed data processing filters extended from a generic filter class and deployed them around publish-subscribe system. In 6.2 we describe our approach to create filter chains for more complex processing and real-time data analysis. We use distributed filter services and filter chains to process the data streams on-the-fly. In this section we summarize some of the related work about issues pertaining to stream processing and management.

The Active Streams is a middleware approach and its associated framework for building distributed applications and services [41]. In the Active Streams context the distributed systems have three components: applications, services and data streams. The

data streams are sequences of self-describing data units flowing between components and services. The data streams are made active by attaching functional units called *streamlets*. Streamlets are self-contained units that operate on incoming streams and generate records placed onto outgoing streams. Streamlets are created using E-code which is a subset of general procedural language, and they can be obtained from streamlet repositories. A coarse form of dynamic adaptation is obtained by attachment/detachment of streamlets that work on data streams. Finer grain adaptation involves fine-tuning a streamlet's behavior via parameters, and by re-deploying streamlets to best leverage dynamically-changing available resources over the data path [42]. Active Streams framework relies on Echo [43] a high-performance event-delivery middleware designed to scale to the data rates typically found in grid environments.

Echo which is developed at Georgia Tech aims to address requirements of high-performance event-based communication in distributed systems. Echo provides maximum bandwidth to the applications by allowing receivers to customize delivery through derived event channels, which are mechanism that can operate at network transmission speeds [43]. Related to our research is Echo's filtering and transformation ability. To support specific customizations on the event streams Echo provides derived event channels which bears some similarities to content-based and pattern-based filtering. Additionally Echo supports dynamic code generation (DGC) which allows general computations on the streams. By using a derivation function applications can create a new channel which is derived from an existing channel. This allows custom filter execution on the stream and eliminates unwanted event traffic [43].

Adapting Computational Data Streams (ACDS) system which is built on top of Echo is a system for implementing adaptive computational data streams [44]. ACDS aims to address the high-performance requirements for creation and management of the large-scale data streams put forward by the distributed scientific applications. The streams are sequence of data events, generated either in response to requests from consumers or by the producers. ACDS supports migration, specialization, splitting and merging of these stream computations. Some of the capabilities provided by ACDS are stream parallelization, runtime filtering adaptation, and runtime migration of the stream components. ACDS provides capabilities for parallelization of the stream computations since in some cases the stream computations themselves are computationally intensive. This motivates 'split' and 'merge' adaptations in ACDS. Runtime adaptation of the parameters for single or sets of stream components allow data filtering at runtime. Runtime migration of the stream components allows ACDS to deal with dynamic variations in the node and network loads [44].

Managing and processing data streams in distributed systems are central in several research efforts. One such example is dQUOB: dynamic Query Object system [45, 46]. dQUOB provides mechanisms to reduce end-to-end latency of the scientific data by forwarding only the useful data. This requires ability to continuously run queries on the data stream to strip the unnecessary parts for a specific purpose. dQUOB enables users to create SQL-like queries and attach them into runtime components called *quoblets*. These components may then be dynamically embedded into the streams at various points which also provide distributed filtering capabilities [45]. By adopting dQUOB scientists may potentially eliminate large amount of data processing and transfers. In [47] authors

explain that although the queries over data streams is a useful abstraction, they had discovered that the transformation power of the rules can be enhanced by coupling a query and a complex, user-defined function which is triggered when the query is evaluated to true. This approach can be thought of a custom filter, based on a rule which is created by the user.

dQUOB uses dQUOBEC, a publish-subscribe system implemented as event channels [46]. dQUOBEC is a lightweight and efficient system for transferring binary data as streams. It is a subject based publish-subscribe system and follows a push-based streaming model, and also supports peer-to-peer architecture. dQUOBEC uses Portable Binary Input Output (PBIO) [48] for binary data transfer. dQUOBEC is also used by Calder, a stream processing engine, which aims to provide timely access to data streams [49]. Architecture and experimental evaluation of Calder is discussed in [49, 50].

Data streams such as those obtained from the sensors and their management are discussed in various publications. Golab and Ozsu in [51] discuss the issues in data stream management and challenges related to executing queries on streaming data. Telegraphcq [52, 53], Eddies [54] and Calder [50] are a few examples of Stream Processing Engines (SPE's) designed to process data flows. In [55] Babcock et al explain the differences between the traditional Database Management Systems (DBMS) and the continuous data streams. They also give a list of the data stream management projects and propose STREAM (STanford stREam datA Manager) a data stream management system (DSMS). In [56] and [57] Plale explains how distributed global snapshots can be used to access streams and proposes a framework based on this idea to bring the data streams to the Grid.

2.3 Motivating Use Cases

2.3.1 RDAHMM

Regularized Deterministic Annealing Hidden Markov Model (RDAHMM) is an implementation of regularized deterministic annealing expectation-maximization algorithm (RDAEM) [58] for fitting hidden Markov Models (HMMs) [59] to time-series data. Fitting an HMM to a time-series allows us to describe the statistics of the data in a simple way that ascribes discrete modes of behavior to the system.

What is different in RDAEM than standard HMM time series fitting methods such as those used in speech analysis, is that it does not require a priori knowledge about the data. This property of RDAEM allows its use in data collected from poorly understood systems and quick adoption to new problem domains.

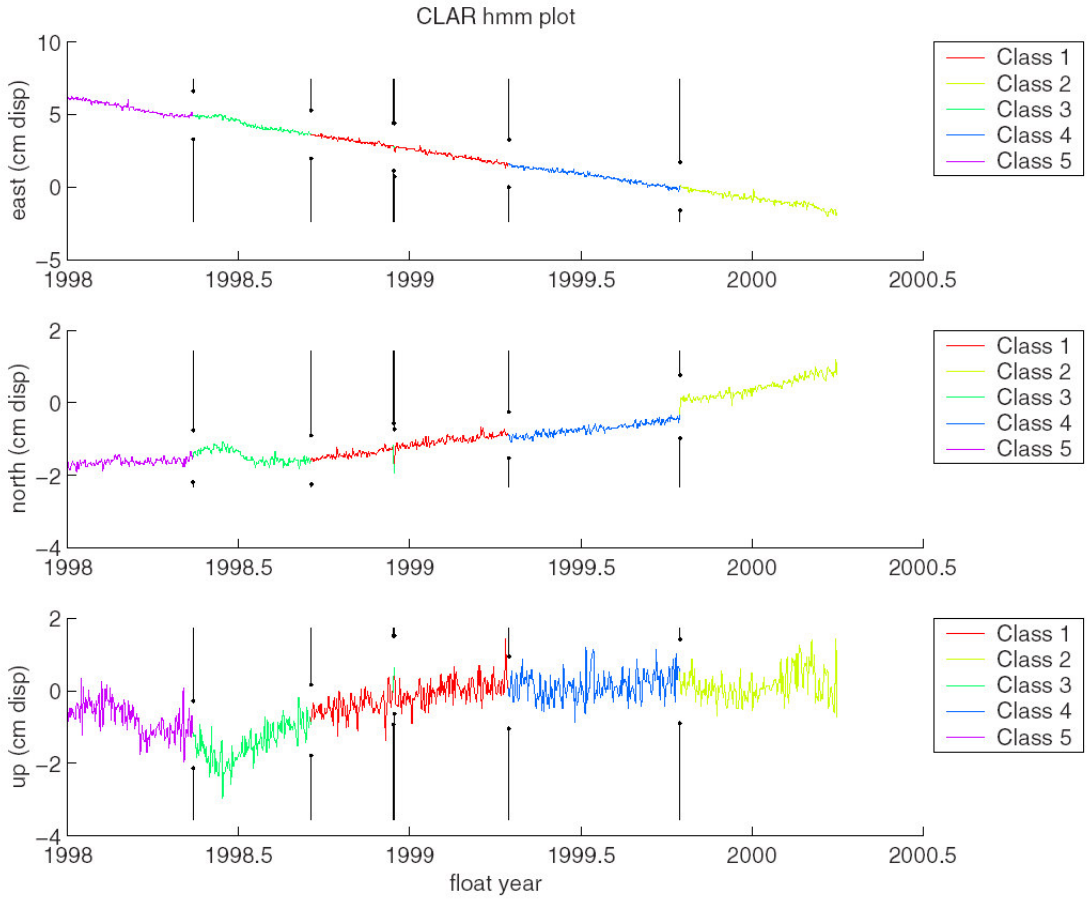


Figure 2-2 – RDAHMM output plots for time series data collected for two years between 1998 and 2000 from GPS station CLAR which is located in the city of Claremont, California. The graphics display the displacement time series segmentation for north, east and up coordinates. This figure is taken from [60].

RDAHMM has successfully been applied to geodetic time series in Southern California such as daily displacement time series collected by the Southern California Integrated Geodetic Network (SCIGN). To analyze the GPS time series data the HMMs are first trained using a data set deterministic of the actual time series. The algorithms segment the series based on statistical changes as identified by the trained HMMs. The identified segments correspond to state changes or different behavioral modes in the series. The correlations in the state changes across multiple stations at a given point of time indicate region-wide activity. The application of this algorithm has shown that it can

detect seismic events across a region as well as signals associated with aseismic events or long-range interactions between smaller events [61].

Figure 2-2 is an example plot of RDAHMM output given in [60]. The figure shows that the states before and after the Hector Mine quake of October 1999 are clearly separated, and distinct in turn from a period in 1998 in which well ground water drainage caused displacement in the vertical direction is also identified.

Another HMM plot example is given in Figure 2-3 for the same time period. Results show that the algorithm identified the dip due to the aquifer drainage between days 120-250 and the Hector Mine earthquake at day 626.

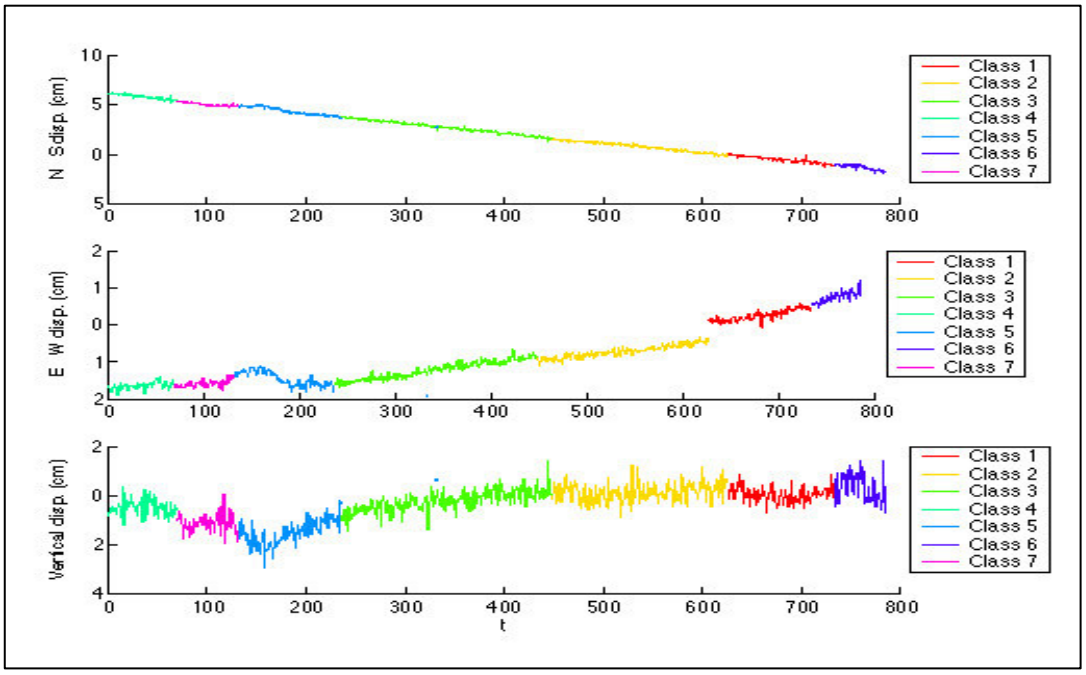


Figure 2-3 – RDAHMM output plots for 800-day long GPS time series data. Figure courtesy of Robert Granat, JPL

From the point of view of our research RDAHMM is a geophysical application which requires geographic data as input to produce a set of output. The input data is usually archived GPS observations made available through files on FTP or HTTP servers.

We aim to provide a fully automatic integration of RDAHMM with the data sources with ability to select single or multiple station analysis.

Another interesting research challenge we address in this research is that although RDAHMM has traditionally been used to analyze archival GPS observations we can also use it to analyze near-real time observations. Our architecture provides easy access to real-time GPS observations of Southern California Integrated GPS Network (SCIGN) and we have integrated RDAHMM with our data services to analyze the real-time data. Details of this integration are explained in Chapter 6.

2.3.2 Pattern Informatics

As reported in [62] 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. 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) [63-66] 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 [62].

The PI 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 [62].

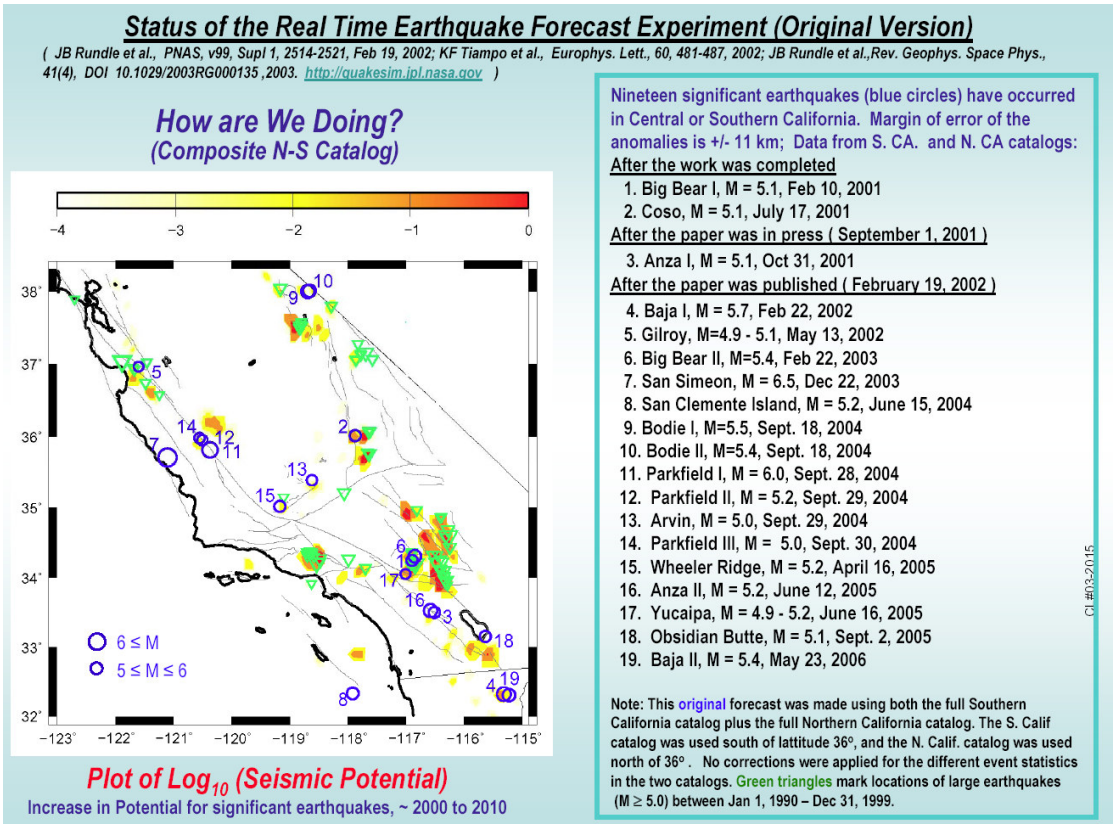


Figure 2-4 – PI forecast map or hotspot scorecard shows the results of a forecast experiment for California. The time period of this experiment is January 1 2000 – December 31 2009. The green triangles represent the earthquakes occurred between 1990 and 1999, while the blue circles are the large seismic events occurred after 2000. The scorecard indicates that 16 of these 19 significant earthquakes occurred after the work was first published in February 19, 2002. The figure is taken from QuakeSim Website (http://quakesim.jpl.nasa.gov/scorecard.html)

The PI technique quantifies the temporal variations in seismicity patterns to identify geographic regions with strongly correlated seismic activities. These regions are shown to be the locations for subsequent large earthquakes. The result is a map which shows the fluctuations in seismic activity which are found to be related to the preparation steps for large earthquakes. In other words the PI map shows regions with hotspots where earthquakes are likely to occur during a specified period in the future [62]. An example PI map is shown in Figure 2-4.

The PI method has been applied to existing seismic observations to forecast future seismic hotspots in various regions such as Southern California, Turkey and Japan [67]. The fact that the PI uses publicly available seismic records to forecast future earthquakes makes it an ideal candidate for our research because our system provides access to both real-time and non-real time geophysical records. We have harvested global and Southern California specific seismic records off the internet and created services to access these data. By using these services we were able to integrate PI with our architecture. This integration is reported in Chapter 4. Sayar et al [68] reports the results of integration of PI, our GIS data services and GIS visualization Web Services [69].

2.3.3 Interdependent Energy Infrastructure Simulation system (IEISS)

The National Infrastructure Simulation and Analysis Center (NISAC) at Los Alamos National Laboratory (LANL) develops 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. [70].
One such suite of analysis software is the Interdependent Energy Infrastructure Simulation System (IEISS) developed at LANL with the collaboration of Argonne National Laboratory (ANL). The goal of IEISS is to provide a comprehensive simulation of national energy infrastructures and intra- and inter-infrastructure dependencies [70].

During our research we have worked with NISAC to develop a Service Oriented Architecture for IEISS suite. Traditionally IEISS is run as a desktop application with

input data supplied as XML files collected from various sources, and the result is locally generated. We have used our Web Services to provide the required input data from our geospatial databases. Additionally our map interfaces allowed users to select geographical regions on the maps where the simulation is executed. This integration is detailed in Chapter 4.

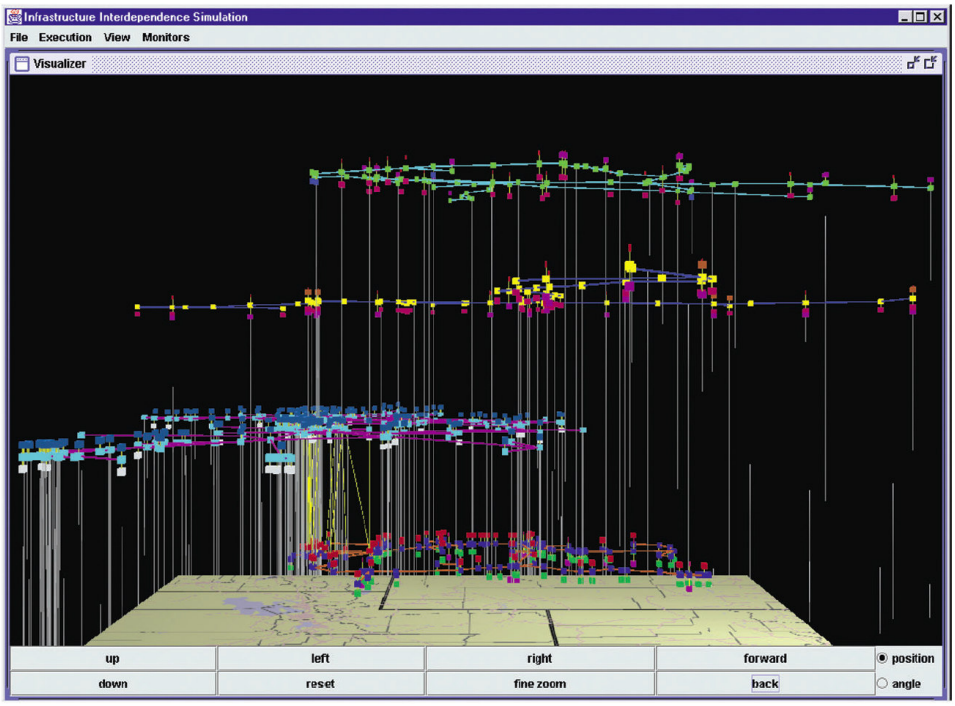


Figure 2-5 – IEISS screen capture shows various energy infrastructures and interdependencies between them in three-dimensional visualization. The infrastructures starting from the top layer are network of crude oil pipelines, petroleum product pipelines, electric power transmission lines and natural gas pipelines. The figure is taken from [71].

Chapter 3

GIS Data Grid Architecture

3.1 Overview of the System

The Grid architecture we have developed for Geographic Information Systems is a high performance, Service Oriented Architecture [72] to support coupling archived and real-time geospatial data with scientific applications such as simulation, visualization or data mining software.

GIS applications that require access to and processing of very large data sets are increasing in number with the evolution of computing resources, network bandwidth, and storage capabilities etc. At the same time some of the applications are being designed to consume real-time data to provide near-real time analysis results; such applications are gaining ground in systems like Crisis Management or Early Warning Systems because they allow authorities to take action on time. Earth observation and earthquake data assimilation tools are good examples of this group since they use data from Seismic or

GPS sensors which continuously collect data. However most of these tools currently consume data from repositories and either they do not have access to real-time data or they do not have the capability to analyze data on the fly.

We utilize GIS standards and Web Services methodologies to couple data assimilation tools with real-time and archived geospatial data. The system uses publish/subscribe [73] based messaging substrate to provide high performance data transfer between data sources and the client applications. Standard GIS interfaces and encodings like Geography Markup Language (GML) [10], GML-Observations and Measurements (OM) [74] and Sensor Markup Language (SensorML) [75] allow data products to be available to the larger GIS community. The architecture supports seamless access to both archival and real-time geospatial data through standard Web Services interfaces. Although the issues related to online and offline geographic data differ, our architecture provides a common platform for suppliers to make their data sets available without much effort and easy to use tools for users to access this data.

Geographical data can be classified in two major categories according to their sources: Online or real-time measurements collected from sensors and offline or archived records.

Archival geospatial data has been in the core of almost every GI System since the first example. Long history of mapping, topography and related scientific activities has created huge spatial repositories and with the development of computing resources numerous instances of software were produced to consume and analyze these data. Because of the need to create universal data and service standards for the GIS community, recent years have brought about very intensive research in this direction.

This research has yield to very successful outcomes and the standards produced are being adopted in every part of the world. In addition to the standards development the internet revolution helped geospatial data to be used and integrated into very diverse web based developments. Every day we see new examples of web sites offering access to some form of geographic data. Online mapping tools, driving direction tools, store locators are just a few examples. Web GIS, Internet GIS or Distributed GIS are some of the terms used to describe these online GIS related activities [2].

Although there is enormous demand to utilize the spatial data online much of the research in this area has been about developing data format and service standards. While the GISystems are migrating from the traditional stand alone desktop applications or LAN based desktop GIS to distributed systems [2] there is an obvious lack of thrust towards service based approaches. The distributed GI services are mostly developed as traditional, well known client/server architectures such as Java Servlets. In this chapter we describe our approach which not only adopts the latest industry standards but also conforms to Service Oriented Architecture principles.

3.2 Major Components of the Architecture

Taking into account the two types of the geospatial data, our architecture consists of two major parts:

1 - GIS Data Grid for providing unified access to archived, offline geographic data stored in various distributed databases,

2 – Real-Time Data Services to provide access to online, real-time sensor measurements, or streaming observations collected from various sources.

Each of these parts consists of several Web Services and separately they provide access to different types of data hence they can be thought of independent Grid architectures. Therefore the complete SensorGrid architecture is an example of the Grid of Grids [29] paradigm.

We discuss the details of these parts in the consecutive chapters; here we give a short overview of the architecture and the service components.

The most important component of the Archival Data Grid is the Web Feature Service [15]. Web Feature Service (WFS) is an Open Geospatial Consortium [76] service for sharing vector geographic data on the web. Details of this service are discussed in Chapter 4, here we only discuss how it is used in the overall architecture. The WFS accesses various geospatial databases to retrieve and present the data to the users in a standard format. However because the specification describes a HTTP GET/POST based service we have extended it by implementing a Web Service version which allowed us to integrate several installations of this service and other Web Services to create Grids for particular purposes. For instance a Web Map Service [14] can be used to provide an interface to the WFS and allow users to interact with it via online maps. Reference [68] demonstrates one such example.

Although the first Web Service version of the WFS was successfully used in several GIS Grid projects, for several reasons explained in Chapter 4 and 5 we have created a streaming version, which uses a publish/subscribe system to stream data to the clients. This method allows the WFS to serve arbitrarily large amount of geographic data in high rates [77].

Therefore our archival GIS Grid architecture has two major types of data services, streaming and non-streaming WFSs. Users may choose to use any one of these services depending on the capabilities implemented on the client side. The non-streaming WFS is a traditional Web Service which does not require any additional capabilities on the client side, however to use the streaming version the clients need to implement streaming publish/subscribe API. Furthermore our services have additional capabilities for performance improvements such as Binary XML framework integration for shrinking the query results in XML. To use these capabilities the clients need to have appropriate API implementations. However existence of these capabilities does not prevent Web Service clients from using the basic WFSs since they can serve the results in simple XML Schema types.

The second part of the overall architecture is the Real-Time Data Grid which consists of Real-Time Filter services and publish/subscribe messaging system. We explain the details of this architecture in Chapter 6. The Real-Time Filter Services are data processing or analysis applications exposed as Web Services and connected with each other via publish/subscribe messaging substrate. Real-time messages collected from sensors are processed using these services. Filters are usually connected as chains to realize complex tasks. The fact that they use a topic based publish/subscribe system provides us an important ability to access original and processed data products via different topics. This method allows creation of many different types of chains for various tasks. The system provides continuous access to sensor streams for large number of clients. In fact because we can create publish/subscribe networks there is no limit on

the number of sensors or clients the system can support. Chapter 7 gives detailed performance evaluation of the Real-Time Data Grid architecture.

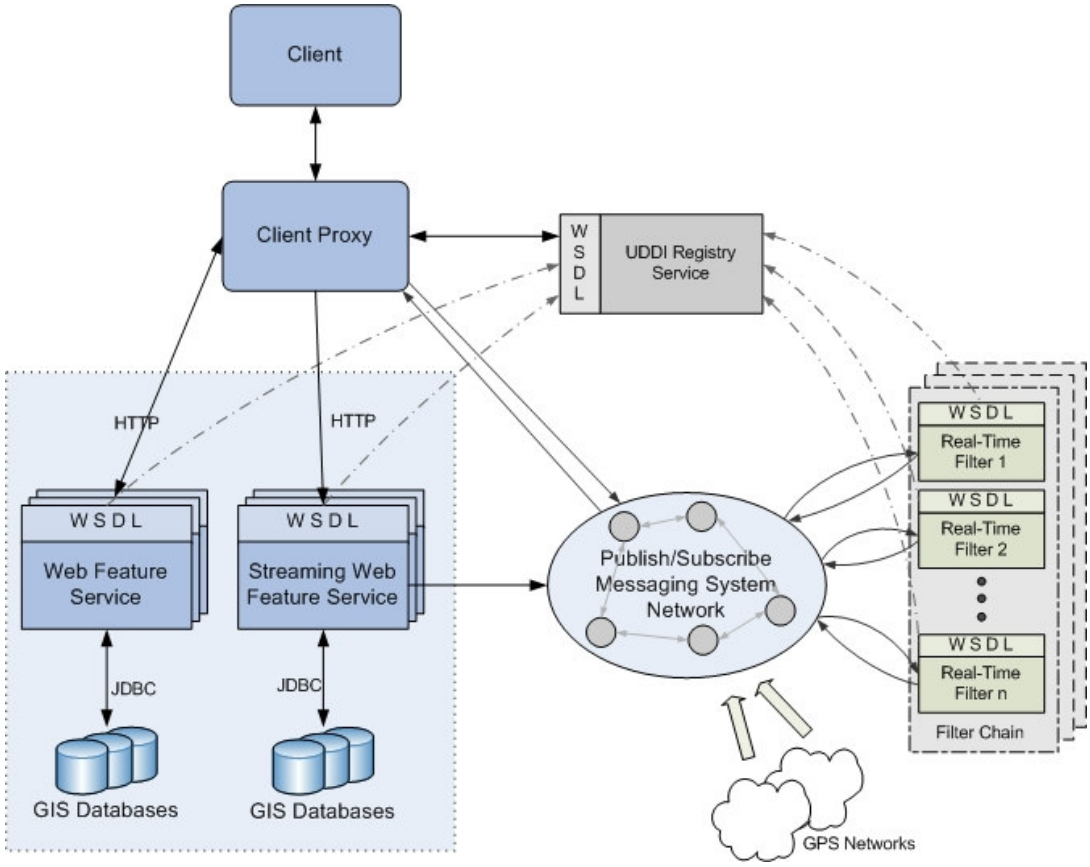


Figure 3-1 - SensorGrid Architecture consists of archival and real-time services. A publish/subscribe messaging system is used to stream large archival data and real-time sensor messages to the clients.

Figure 3-1 illustrates the overall SensorGrid architecture with both archival and real-time data services. The major components in the system are Web Feature Service, Streaming-Web Feature Service, Real-Time Filter Services, Publish/Subscribe Messaging System and Registry Service.

All services in the system have traditional Web Service endpoints or WSDL [78] URLs. To provide easy access and search capabilities for the active services in the system we use a UDDI service as services registry. The UDDI implementation we use in this

architecture is a specialized implementation of the UDDI specification [79] which has GIS specific extensions. This UDDI registry service is part of a larger Information Services project developed in Community Grids Lab [80], more information can be found in [81-86].

Each service in the system publish its WSDL URL to the UDDI [79] registry at the time of initialization. The registry service URL is supplied to the services before initialization. The registry service also provides search capabilities, which is useful for discovering particular GIS capabilities by the users. For instance a user may want to see WFS instances which have access to data for a particular geographic region, or sensors physically located in a particular region. We also have a JSP interface (Client Proxy) that displays the available services and sensors in the registry and makes use of UDDI service's search capabilities for the user. Using this interface the clients can view the capabilities of each service, available geographic features, and real-time sensors or filter services. The Client Proxy also provides the required information to the client for receiving streaming real-time messages.

To summarize the architecture we go back to Figure 3-1:

- The dotted lines represent one time access to the UDDI registry by other services to register their WSDL URLs at the time of initialization. The streaming and non-streaming WFSs access to GIS databases using JDBC connections. We use several MySQL [87] databases for this purpose.
- The WFS communicates with the Client as a traditional Web Service, request and response SOAP messages are transported over HTTP. The streaming-WFS is accessed via HTTP, and the request messages are submitted in the conventional

Web Service way, however it does not return the results over HTTP rather utilizes the publish/subscribe system to stream the results.

- On the real-time data grid side we have several filter services which communicate through the publish/subscribe system. Usually these filters are connected as chains as depicted in the figure. The standards WSDL interfaces provide capabilities such as starting, stopping or resuming the filter operation and providing metadata about filter.
- The real-time data sources are integrated into the architecture through the pub/sub system. Typically sensor messages are collected through a proxy server and then disseminated for use, in such cases we use a filter to connect to the proxy servers and receive messages to publish to a topic on the messaging substrate. We then deploy subsequent filters around the messaging substrate to process these raw sensor messages. In the pictures the sensors are represented as clouds; as an example we use GPS networks for representing the sensor networks.

3.3 Summary

In this chapter we have given an overview of our Data Grid Architecture for Geographic Information Systems. The architecture consists of two major parts corresponding to archival and real-time geospatial data. We adopt open geographic standards for data and service interfaces and Web Service standards for implementing the data services which allows us to create and manage scientific workflows for complex data analysis cases. We also summarize a novel approach for processing the real-time sensor messages. This

approach is based on using filters as Web Services and creating chains of filters for more complex analysis cases.

Chapter 4

Grid Architecture for Archival GIS Data

4.1 Introduction

In this chapter we discuss our approach to build a Service Oriented Architecture for archival geographic data and give detailed descriptions of the services developed as part of this architecture.

To build a GIS Data Grid we adopt the most common industry standards for geospatial data descriptions which allow our data products to be available to the larger GIS community, and our services to be compatible with others. The Grid architecture we built composes of several Web Services for managing, accessing and providing geospatial data. The data service components can access and query distributed geospatial databases, and make the data available to the users in the commonly used formats. This approach ensures interoperability on service type, and the data format levels. Web Services approach allows our GIS data services to be used in conjunction with other services using workflow management tools. These additional services may include

mapping, or visualization services to illustrate the data in graphical formats. In this chapter we discuss some examples of this approach.

4.2 Data Grids for Geographic Information Systems

GIS applications developed by various vendors and academic institutions have become more complex as they are required to process larger data sets, utilize more computing power and in some cases need to collect data from distributed sources. Traditionally GIS applications are data centric: they deal with archived data. However, with sensor-based applications gaining momentum the need of integrating real-time data sources such as sensors, radars, or satellites with high end computing platforms such as simulation, visualization or data mining applications introduces several important distributed computing challenges to GIS community.

Although commercial GIS applications provide various solutions to these problems, most of the solutions are based on more traditional distributed computing paradigms such as static server-client approaches. Traditional point to point communication approaches tend to result in more centralized, tightly coupled and synchronous applications which results in harder management practices for large scale systems. Modern large scale systems on the other hand require more flexible asynchronous communication models to cope with the high number of participants and transfer of larger data sets between them.

As in other distributed computing domains the trend in distributed GIS is moving towards component based applications [2]. This is due to the fact that the previously used distributed GIS technologies such as CORBA/IIOP and COM+/ActiveX type frameworks were not able to address the major interoperability issues. Although these are very

successful frameworks, their use are constrained with proprietary client applications and specific types of middle tier servers. For instance a typical distributed GIS architecture involves three tiers [2]; a Client Tier which contains a Java Applet, a Middle Tier which contains a CORBA/Application server and a Server Tier which contains a GIS Server or a Database. In this type of architecture the client has to be aware of the CORBA programming techniques to communicate with the middle tier application server.

A complete GIS architecture contains three major types of service components: presentation, logic and data. For instance, consider an online mapping application; here the web server is responsible for the presentation by displaying the map images. The underlying logic engine which creates the map image can be either on the client side (thick client), or on another server. The engine usually communicates with a geospatial database, which contains the map data. As we can see, even the simplest type GIS application has several distributed components and interoperability between these components must be realized. When we think about the bigger picture where hundreds, even thousands of data repositories, data analysis and visualization applications are available, we realize the need for GIS standards to make interoperability possible.

4.2.1 Web Services

However it is also obvious that the diversity of the GIS applications and data sources is a great challenge. That is where a new breed of distributed systems approach may help: the Web Services. “A Web Service is an interface that describes a collection of operations that are network accessible through standardized XML messaging.” [88]. In practice the interface, operations and the XML messaging are standardized. The important thing about the service interface is that it hides the implementation logic from

the users, which allows the service to be used on different platforms than which it was implemented. Also any application capable of communicating through the standard XML messaging protocol and regardless of with which programming language it was developed in can use the service through the standard interface. These properties allow Web Services based frameworks to be 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.

One significant aspect of the Web Services is that they allow program-to-program communications. With the help of several Web Services specifications a complete cycle of describing, publishing, and finding services can be made possible. As new specifications are being developed and the industry matures the system integration that includes these steps will eventually happen dynamically in runtime.

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) [89] or Remote Method Invocation (RMI) [90] or Internet Inter-Orb Protocol (IIOP) [91].

Obviously the capabilities offered by the Web Services can be of great benefit to the geo-science community as well. Because the possibility of accessing various types of geospatial data sources and applications using standard service interfaces may help solving the interoperability issues the GIS community has long suffered. But what kinds of standards are really required for service components and databases to be interoperable?

According to Kirtland [91] the Web Service specifications and technologies address following requirements for service-based technologies:

- A standard way to represent data
- A common, extensible, message format
- A common, extensible, service description language
- A way to discover services located on a particular Web site
- A way to discover service providers

Currently there are several universally used standards to address these requirements: XML is the common choice for representing the data while Simple Object Access Protocol (SOAP) [92] is universally being used for information exchange. SOAP provides rules for describing how to use XML to represent data as well as conventions for representing remote procedure calls (RPCs) and bindings to the HTTP protocol. Web Service Definition Language (WSDL) [78] is used to describe what type of message a Web Service accepts and generates. Available protocols such as Web Services Dynamic Discovery can be used to locate services. Universal Description, Discovery, and Integration (UDDI) specification [79] can be used by the service providers to advertise the existence of their services.

4.2.2 Open Geographic Standards

From the GIS perspective the problems being addressed by the Web Services are also being discussed by the geo-scientists. In recent years several organizations have started developing standards to address interoperability issues on data and application

levels. The standard bodies aim 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 270 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 [15] and the Web Map Service (WMS) [14]. Geography Markup Language (GML) [10] is widely accepted as the universal encoding for geo-referenced data. The OGC is also defining the SensorML [75] family of specifications for describing properties of sensors and sensor constellations and sensor observations. On the other hand ISO Standards proposes a standard framework for the description and management of geographic information and geographic information services. 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 [2]. The scope ISO/TC 211 is described as following on the working group's web page [93]:

Scope: Standardization in the field of digital geographic information.

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.

The work shall link to appropriate standards for information technology and data where possible, and provide a framework for the development of sector-specific applications using geographic data.

In short the OGC is interested in developing both abstract definitions of OpenGIS frameworks and technical implementation details of data models and to a lesser extent services and the ISO/TC 211 focuses on high-level definition of geospatial standards from an institutional perspective [2].

Both of these major geospatial standard bodies have been formed in 1994 and until 1997 they have worked independently and produced several often overlapping standards. But after 1997 because of the strong demand from the industry they have been working closely to align their work to produce compatible standards.

4.2.3 Web Services for GIS

Today major GIS software companies such as ESRI, ERDAS, AutoDesk and INTERGRAPH are member of the OGC and participating in the interoperability programs, thus helping shape the next generation geospatial data and service standards. Furthermore we are seeing an increasing number of governmental and municipal

contributions at a global level based on OGC standards. There is also increasing interest in the academic community towards OGC standards and specifications [12]. This is also an indicator of the consensus between the software vendors, governments and academia for an interoperable GIS infrastructure.

Considering the strong background from the industry and backing of scientists, experts and several research institutions we expect to see wider deployment and acceptance of OGC specifications, both at national and global level. For these reasons we have used OGC service and data specifications to build a GIS Data Grid.

The OGC specifications can be studied in two groups: data and service specifications. The geospatial data issue is a multi dimensional and complex problem. There are several types of geospatial data: satellite imagery, aerial images, coverages, maps, vector data, sensor measurements, raster data etc. The GI Services are also diverse applications ranging from the ones making the data available to the end user to others doing more complex coordinate transformations and computations. In the next section we discuss the OGC approach to the common data format and services problem.

4.2.4 Common Data Format

The OGC has produced many specifications for web based GIS applications such as Web Feature Service (WFS) [15] and the Web Map Service (WMS) [14]. The data model developed by OGC is the Geography Markup Language (GML) [10] and it is currently widely accepted as the universal encoding for geo-referenced data.

The first step for building GI Services is to decide appropriate encodings for describing the data. The importance of the data format lies in the fact that it becomes the basic building block of the system which in turn determines the level of interoperability.

Use of a universal standard like XML greatly increases the number of users from different backgrounds and platforms who can easily incorporate our data products into their systems. Furthermore, services and applications are built to parse, understand and use this format to support various operations on data. So in a sense the type and variety of the tools being used in the development and data assimilation processes depend on the format initially agreed.

For these reasons we use Geography Markup Language (GML), a commonly accepted XML based encoding for geospatial data, as our data format in our GI Services. One important fact about GML is that, although it offers particular complex types for various geospatial phenomena, users can employ a variety of XML Schema development techniques to describe their data using GML types. This provides a certain degree of flexibility both in the development process and in the resulting data products. For instance, depending on the capability of the environment schema developers may exclusively use certain XML Schema types and choose not to incorporate more obscure ones because of incompatibility issues. As a result a particular geospatial phenomenon can be described by different valid GML schemas.

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.

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 [94], XML Data Binding Frameworks, XSLT, XPath, XQuery etc. a GML dataset becomes easier to process in heterogeneous environments.

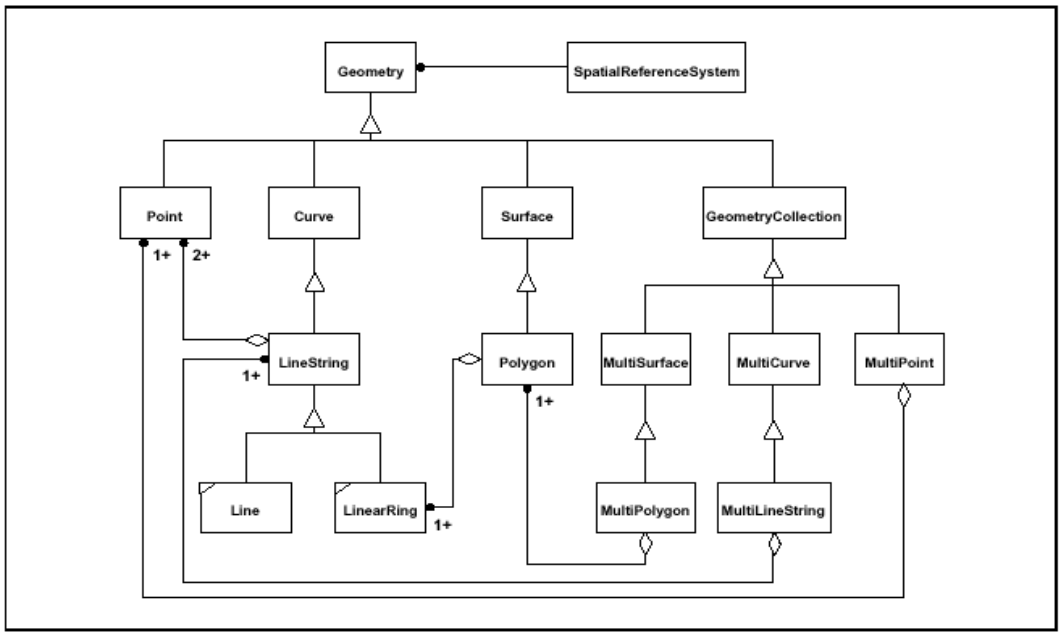


Figure 4-1 - OGC Geometry Model is based on three major geometry constructs, point, curve and surface. The other necessary geometry constructs are created using these main types.

Basically GML is an abstract model for geographic data which can be used to encode:

- Features: abstract representations of map entities.
- Geometry: encode abstractly how to represent a feature pictorially.
- Coordinate reference systems
- Topology
- Time, units of measure

- Observations and Measurements data collected from Sensors.

By incorporating GML in our systems as de facto 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. Several GML schemas we have developed are given in the Appendix A. (See also http://www.crisisgrid.org/html/servo.html for more GML schemas for GPS and Seismic data)
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.

4.2.5 Data Binding

Establishing XML or some flavor of it as the default message/data format for the global system requires consideration of a Data Binding Framework (DBF) for generating, parsing, marshalling and un-marshalling XML messages. Marshalling and un-marshalling operations convert between XML-encoded formats and (in our case Java) binding classes that can be used to simplify data manipulation.

Being able to generate XML instances and parsing them in a tolerable amount of time is one of the criteria while choosing such a framework, because message processing time would affect overall system performance as well as the performance of the individual XML processing component.

Another criterion to consider is the ability of the binding framework to successfully generate valid instances according to the Schema definitions. This is a major problem for DBFs since not all of the XML Schema types can be directly mapped to Object Oriented Programming constructs. Some of the XML Schema types (such as Substitution Groups which are heavily used in GML Schemas) do not correspond to types in Object Oriented world and this causes difficulties while processing the XML documents. Various Data Binding Frameworks offer different solutions, some of which are more elaborate than the other and depending of the nature of the data a suitable framework must be chosen.

4.2.6 Web Feature Service

Web Feature Service is one of the major OGC service standards 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 geodata from different sources by invoking several standard operations [15].

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, of course, also implies that features can be defined with no geometric properties at all.

As a minimal requirement a basic WFS should be able to provide requested geographical information as GML Feature Collections. However more advanced versions support “create, update, delete and lock operations” as well.

The operations that must be supported by basic WFS are defined as follows [15]:

- **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.

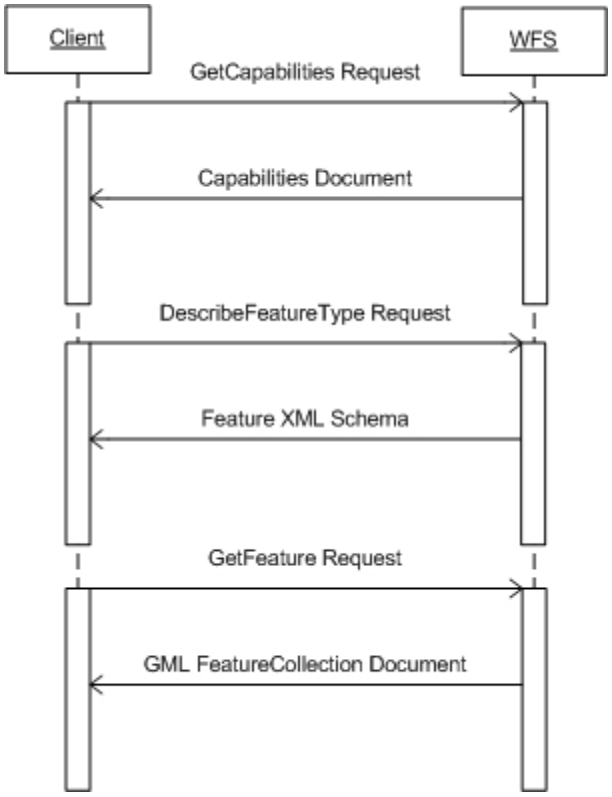


Figure 4-2 – WFS Interaction Steps: Client's interaction with WFS usually starts with a discovery step which involves retrieving the capabilities document. After this the client may request details about a certain feature by issuing a DescribeFeatureType request. However the most common queries used are GetFeature requests to retrieve particular features.

Following is a typical scenario which describes the use of the above operations and the interaction between a client and a Web Feature Service, Figure 4-2Figure 4-2 – WFS Interaction Steps: Client's interaction with WFS usually starts with a discovery step which involves retrieving the capabilities document. After this the client may request details about a certain feature by issuing a DescribeFeatureType request. However the most common queries used are GetFeature requests to retrieve particular features. displays these steps:

1. **GetCapabilities**: 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.
2. **DescribeFeatureType**: 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. The WFS returns an XML Schema that describes the requested feature as the response.
3. **GetFeature**: 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 allows 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 4-3 displays a sample case where the WFS server is accessed by different types of clients and has access to various types of spatial databases. Clients interact with WFS by submitting database queries encoded in OGC Filter Encoding Implementation [95] and in compliance with the Common Query Language [96]. The query results are returned as GML FeatureCollection documents.

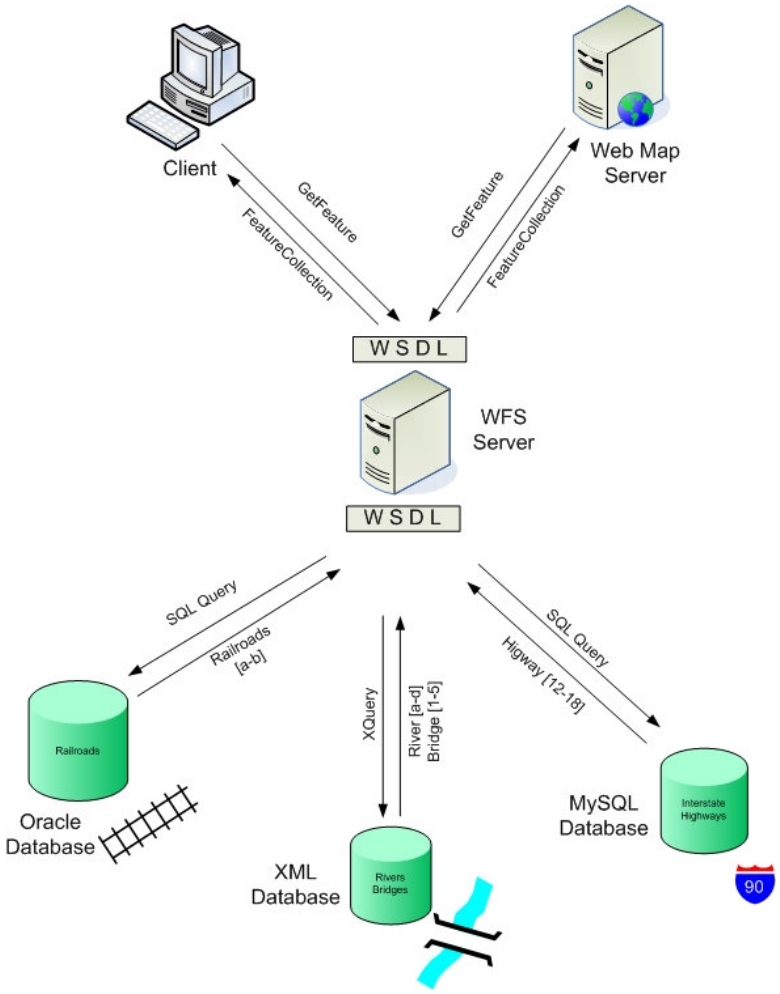


Figure 4-3 – WFS may interact with multiple databases and various types of clients. In this figure the WFS server ha access to three different types of databases which hold various types of data. The clients interact with the WFS via standard WSDL interfaces.

OGC Web Feature Service implementation specification [15] 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. OGC WFS implementation specification [15] states that:

At present, the only distributed computing platform (DCP) explicitly supported by OGC Web Services is the World Wide Web itself, or more specifically, Internet hosts implementing the Hypertext Transfer Protocol (HTTP). HTTP supports two request methods: GET and POST. One or both of these methods may be defined for a particular web feature service and offered by a service instance.

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 WFS implementation on Web Services principals.

Furthermore complex scientific applications require access to various data sources and run several services consecutively or at the same time. This is not in the scope of HTTP but can be supported using rapidly developing workflow technologies for Web and Grid Services. For these reasons we have based our Web Feature Service implementation

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.

4.2.7 Web Service Implementation of Web Feature Service

We have initially implemented Web Service version of a basic WFS which supports the three mandatory operations through a WSDL interface: GetCapabilities, DescribeFeatureType and GetFeature.

Following picture depicts the components of the WSDL document for this implementation:

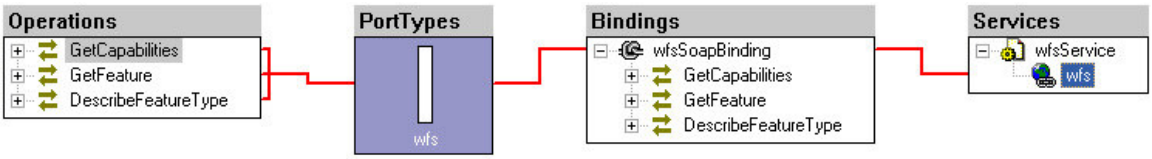


Figure 4-4 –WSDL Components of our WFS implementation. We expose three basic capabilities required by the WFS specification as the Web Service operations.

Each supported operation takes an XML document as argument and returns another XML document as response. While implementing these operations in a Web Service context we have to choose appropriate types. Since the requests and responses are well-defined XML documents one possibility is to create object representations of these in our favorite programming language, i.e. we can create a Java Object for each GetFeature request document and the returning GML document can be another Java object. So the communication between WFS and the client is based on exchanging Java objects. However this approach severely undermines the interoperability with clients who might use other programming languages such as C++ or Python to communicate with our service.

As a simpler solution we have used strings as argument and return types in these operations. This allows clients who use other programming languages to create client

stubs to our WFS-WSDL to simply send and receive XML documents without any conversions. However this method also has its shortcomings which are described in the WFS Performance section.

We chose MySQL as our data store to use with our WFS implementation. We have collected several types of geographic data from various online sources and inserted these to our database. Some of the data types are:

- QuakeTables Fault Database [97], SERVO [13, 98] fault repository for California.
  Compatible with GeoFEST, Disloc, and VirtualCalifornia [99]
- GPS Data sources and formats (RDAHMM [58] and others).
  JPL time series [100]
  SOPAC time series [101]
  USGS time series [102]
- Seismic Event Data (RDAHMM and others)
  Southern California Seismic Network (SCSN) format seismic records [103]
  Southern California Earthquake Data Center (SCEDC) format records [104]
  Dinger-Shearer format seismic records [105]
  Haukkson format seismic records [106]

Also to support producing meaningful maps by Web Map Service [69] we have U.S. and World map data including borders, county boundaries, cities etc. Since we heavily work on seismic and GPS data for California we have several additional features like fault lines, rivers, lakes for this state.

4.3 Web Feature Service Architecture

4.3.1 Creating a Geospatial Database

The geospatial data may be obtained from various types of sources, such as online repositories, coordinate, raster or vector data files, online sensors, satellites etc. To facilitate data collection from various sources we wrote several tools such as HTTP, FTP Clients which download files from online servers. However some of geospatial data such as seismic records or GPS time series are dynamic, and continuously updated. To keep our geospatial database up to date, we also wrote services that can be set up to execute periodic downloads and database insertions. Once the data are available locally they must be inserted into the database for future queries.

Today a large volume of geographic data is available online and WFS can be thought of as a unified solution to serve this data in a common format. We have introduced a simple approach to create relational database tables for various types of geospatial data which are used by WFS for creating GML FeatureCollection objects. This approach allowed us to support various types of geographic queries generated by the clients. Mostly these are SELECT queries asking for a set of one or more feature types but we also support intersections junctions, overlays etc.

Consider the following segment from SCEDC seismic catalog for year 2004[104] [107] :

Table 4-1 – Sample Data from SCEDC Seismic Catalog

#YYY/MM/DD	HH:mm:SS.ss	ET	MAG	M	LAT	LON
2004/01/01	00:28:59.26	le	1.52	l	34.163	-116.424

2004/01/01	01:31:28.13	le	1.60	h	34.384	-116.922
2004/01/01	01:58:38.83	le	2.03	l	32.232	-115.726

DEPTH	Q	EVID	NPH	NGRM
13.1	A	14018180	29	407
1.0	A	14018196	30	554
7.0	C	14018200	15	205

We can easily deduce that the latitude and longitude values are the only geographic information available for this particular data type. To store the data we create a database table as following:

Table 4-2 – MySQL Database Table structure for SCEDC Seismic Records

```
mysql> describe scedc;
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| YEAR           | year(4)      | NO   |     |         |       |
| MONTH          | tinyint(2)   | NO   |     |         |       |
| DAY            | tinyint(2)   | NO   |     |         |       |
| DATE           | double       | NO   |     |         |       |
| HOUR           | tinyint(2)   | NO   |     |         |       |
| MINUTE         | tinyint(2)   | NO   |     |         |       |
| SECOND         | decimal(9,2) | NO   |     |         |       |
| ET             | char(2)      | NO   |     |         |       |
| MAGNITUDE      | decimal(9,2) | NO   |     |         |       |
| MAGNITUDE_TYPE | char(1)      | NO   |     |         |       |
| LATITUDE       | decimal(9,3) | NO   |     |         |       |
| LONGITUDE      | decimal(9,3) | NO   |     |         |       |
| DEPTH          | decimal(9,1) | NO   |     |         |       |
| QUALITY        | char(1)      | NO   |     |         |       |
| EVID           | int(11)      | NO   |     |         |       |
| NPH            | tinyint(3)   | NO   |     |         |       |
| NGRM           | tinyint(4)   | NO   |     |         |       |
+----------------+--------------+------+-----+---------+-------+
                  17 rows in set (0.01 sec)
```

For this feature type the queries are quite simple because each line in the original ASCII file actually corresponds to a geographic point and the metadata associated with that point and more complex queries such as intersections or junctions are not necessary.

However for more complex features that contain several points, lines or polygons we need to find minimum and maximum values of the location elements to support queries.

Another example feature type is California Fault lines. Each fault consists of a number of segments. We can think of each segment as a direct line between two points. For instance following data describes several segments of the San Andreas Fault:

Table 4-3 – Sample geospatial data, California fault lines

Fault Name	Segment Number	Coordinates
San Andreas	62	-115.8,33.42 -115.73,33.37
San Andreas	61	-115.93,33.52 -115.86,33.47
San Andreas	60	-116.6,34.01 -116.52,33.98

Following database table is created to hold the fault segments data:

Table 4-4 - MySQL Database Table structure for California fault lines

```
mysql> describe ca_faults;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| Name        | varchar(100) | YES  |     |         |       |
| Segment     | varchar(100) | YES  |     |         |       |
| Author      | varchar(100) | YES  |     |         |       |
| coordinates | varchar(100) | YES  |     |         |       |
| LatStart    | double       | YES  |     | 0       |       |
| LatEnd      | double       | YES  |     | 0       |       |
| LonStart    | double       | YES  |     | 0       |       |
| LonEnd      | double       | YES  |     | 0       |       |
+-------------+--------------+------+-----+---------+-------+
8 rows in set (0.00 sec)
```

Since the geospatial entry for his data type is “line type” we need to define two points for each segment, a starting point and an end point. Once we have the geographic data in the database the WFS can connect and execute queries. However we still have to find a way which allows easy addition of new feature types to the database. The

challenge here is that geographic data is extremely diverse; it may be as simple as a point and as complex as topographic data or complex features with 3 dimensions. Our database and WFS integration scheme should be flexible enough to allow easy addition of these various kinds of features to the database and consecutively fast generation of the corresponding GML documents. We have developed a simple and easy to adopt system to solve this problem. Our approach requires creation of a set of files as described below:

4.3.2 Adding New Features

For each new feature type to be added we first create a definitive xml Schema. This schema inherits necessary GML schemas and describes both geographic information and non-geographic metadata about the feature.

The second step is to create a sample xml instance from the xml Schema with all the possible element and attributes present. This is the xml skeleton of the feature type or simply an 'empty' feature-xml document and should not have any actual element or attribute values.

The third file to be created is a mapping file that associates the sample xml instance with the relational database table. This xml file contains several *MapElement* elements each of which has two attributes: *XSDNodeXPath* and *DBColumnName*. The first attribute contains the XPath [108] path to a particular element in the xml instance while the second one contains the relational database column name for that particular element. For instance the mapping entry for the Magnitude column from the SCEDC [107] seismic catalog example is:

```
<MapElement No="6" XSDNodeXPath="//Magnitude" DBColumnName="MAGNITUDE"/>
```

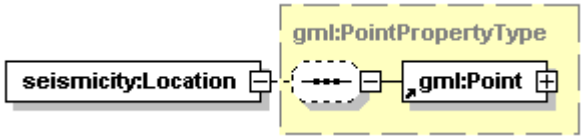
Here the XPath value to this element in the xml-instance is //Magnitude and the actual magnitude values are stored in the MAGNITUDE column.

The last file is the configuration file for this feature type which includes the database connection information, physical paths of the aforementioned files, names of the columns that contain maximum and minimum values for the geographic data and metadata.

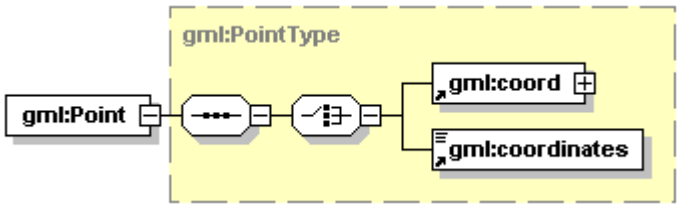
To further explain our approach we give the required files created for SCEDC seismic catalog. Following picture depicts a graphical representation of the XML schema describing seismic events.

Note from the above schema that the geographic information entry is described using a complex type "`gml:PointPropertyType`" inherited from the GML2 schemas:

```
<element name="Location" type="gml:PointPropertyType"/>
```



The gml:Point type has two choices, gml:coord and gml:coordinates.



gml:coord type is extended from gml:CoordType which contains X, Y and Z values for three dimensional coordinate systems.

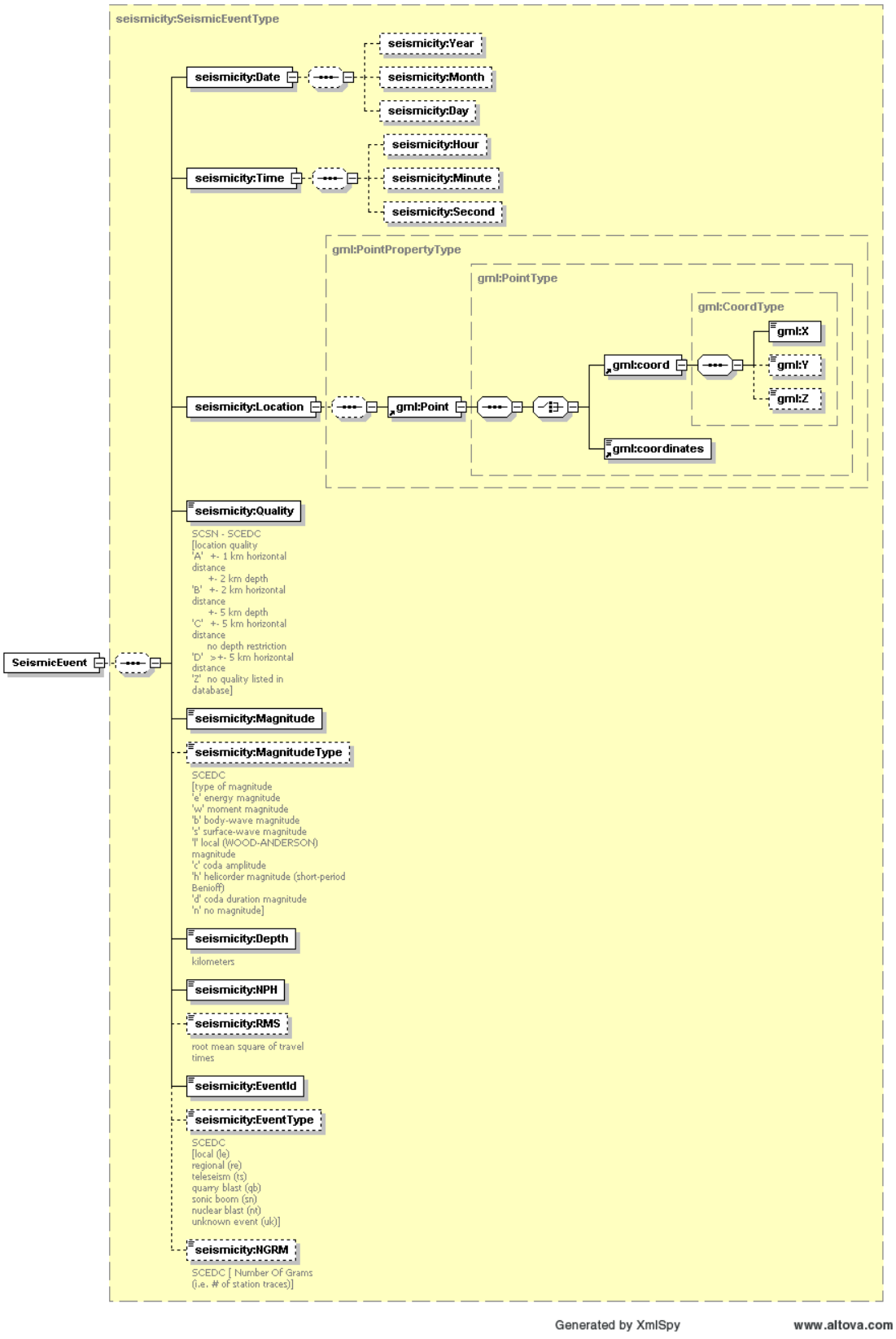
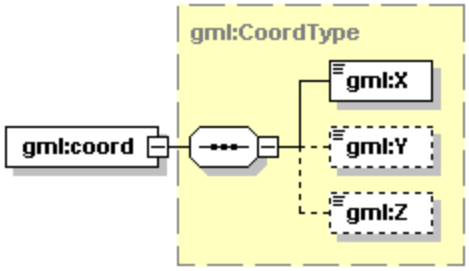


Figure 4-5 - XML Schema for SCEDC and SCSN Seismic Catalogs



For instance we can encode GPS measurements with this complex type since these measurements contain Latitude, Longitude and Height values.

The geographic element for the fault schema would be

```
<xs:element name="SegmentCoordinates" ref="gml:lineStringProperty"/>
<gml:lineStringProperty>
    <gml:LineString srsName="EPSG:4230">
       <gml:coordinates>-82.7335,27.8846,1.0      -82.7586,28.1352,1.0      -
82.6368,28.4571,1.0      -82.7335,27.8846,1.0      -82.7218,28.1763,1.0      -
82.5235,28.6658,1.0      -82.2489,27.2001,1.0      -81.5399,28.6771,1.0      -
81.1583,28.4414,1.0</gml:coordinates>
  </gml:LineString>
</gml:lineStringProperty>
```

since the fault segments are actually lines and contain multiple coordinates. For instance:

The second file is an xml instance generated from the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<SeismicEvent
       xmlns:gml=http://www.opengis.net/gml
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://sensorgrid.org/seismicity.xsd">
<Date>
<Year/>
<Month/>
<Day/>
</Date>
<Time>
<Hour/>
<Minute/>
<Second/>
</Time>
<Location>
<gml:Point srsName="EPSG:6356">
<gml:coord>
<gml:X/>
<gml:Y/>
</gml:coord>
</gml:Point>
</Location>
<EventType/>
<Magnitude/>
<MagnitudeType/>
<Depth/>
<Quality/>
<NPH/>
<NGRM/>
<EventId/>
</SeismicEvent>
```

Here the element that holds geographic information which is inherited from the GML schema is

```
<Location>
       <gml:Point srsName="EPSG:6356">
              <gml:coord>
                     <gml:X/>
                     <gml:Y/>
              </gml:coord>
       </gml:Point>
</Location>
```

Note from the xml instance that none of the elements contain actual values except an attribute which is the same value for all the features.

The third file created for this feature type is the following mapping type:

```
      <?xml version="1.0" encoding="UTF-8"?>
      <MapElements xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <MapElement No="0"
              XSDNodeXPath="//Date/Year"
              DBColumnName="YEAR"/>
        <MapElement No="1"
              XSDNodeXPath="//Date/Month"
              DBColumnName="MONTH"/>
        <MapElement No="2"
              XSDNodeXPath="//Date/Day"
              DBColumnName="DAY"/>
        <MapElement No="3"
              XSDNodeXPath="//Time/Hour"
              DBColumnName="HOUR"/>
        <MapElement No="4"
              XSDNodeXPath="//Time/Minute"
              DBColumnName="MINUTE"/>
        <MapElement No="5"
              XSDNodeXPath="//Time/Second"
              DBColumnName="SECOND"/>
        <MapElement No="5"
              XSDNodeXPath="//EventType"
              DBColumnName="ET"/>
        <MapElement No="6"
              XSDNodeXPath="//Magnitude"
              DBColumnName="MAGNITUDE"/>
```

```
        <MapElement No="7"
                    XSDNodeXPath="//MagnitudeType"
                DBColumnName="MAGNITUDE_TYPE"/>
        <MapElement No="8"
                    XSDNodeXPath="//Location/gml:Point/gml:coord/gml:X"
                DBColumnName="LATITUDE"/>
        <MapElement No="9"
                    XSDNodeXPath="//Location/gml:Point/gml:coord/gml:Y"
                DBColumnName="LONGITUDE"/>
        <MapElement No="10"
             XSDNodeXPath="//Depth"
             DBColumnName="DEPTH"/>
        <MapElement No="11"
             XSDNodeXPath="//Quality"
             DBColumnName="QUALITY"/>
        <MapElement No="12"
             XSDNodeXPath="//EventId"
             DBColumnName="EVID"/>
        <MapElement No="13"
             XSDNodeXPath="//NPH"
             DBColumnName="NPH"/>
        <MapElement No="14"
             XSDNodeXPath="//NGRM"
             DBColumnName="NGRM"/>
      </MapElements>
```

This file is used after the WFS retrieves the queried features from the database. WFS then uses this mapping file to populate the xml instance with the results.

The last file is the following configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<feature>
 <db>
   <type>mySQL</type>
   <serveraddress>gf8.ucs.indiana.edu</serveraddress>
   <dbname>cce</dbname>
   <tablename>scedc</tablename>
   <driver>com.mysql.jdbc.Driver</driver>
   <username>uname</username>
   <password>passwd</password>
 </db>
      <xml_instance>
     <localaddress>/home/galip/wfs/seismic_instance.xml</localaddress>
 </xml_instance>
 <map_file>
   <localaddress>/home/galip/wfs/scedc_mapping.xml</localaddress>
 </map_file>
 <xmlschema>
   <localaddress>/home/galip/wfs/seismicity.xsd</localaddress>
 </xmlschema>
 <maxmin_column_names>
   <minx>LONGITUDE</minx>
   <miny>LATITUDE</miny>
   <maxx>LONGITUDE</maxx>
   <maxy>LATITUDE</maxy>
 </maxmin_column_names>
 <Metadata>
   <Name>scedc</Name>
   <Title>California Earthquake Data in SCEDC Format</Title>
   <Abstract> </Abstract>
   <Keywords>Seismic,WFS</Keywords>
   <SRS>EPSG:6356</SRS>
   <Operations>
     <Operation type="Query"/>
   </Operations>
   <MetadataURL>http://www.crisisgrid.org</MetadataURL>
 </Metadata>
</feature>
```

WFS uses this file to locate and query the database that contains this particular feature type. After the query results are returned it uses the file locations provided in this file to generate the GML feature collection.

4.3.3 Web Feature Service Operation Steps

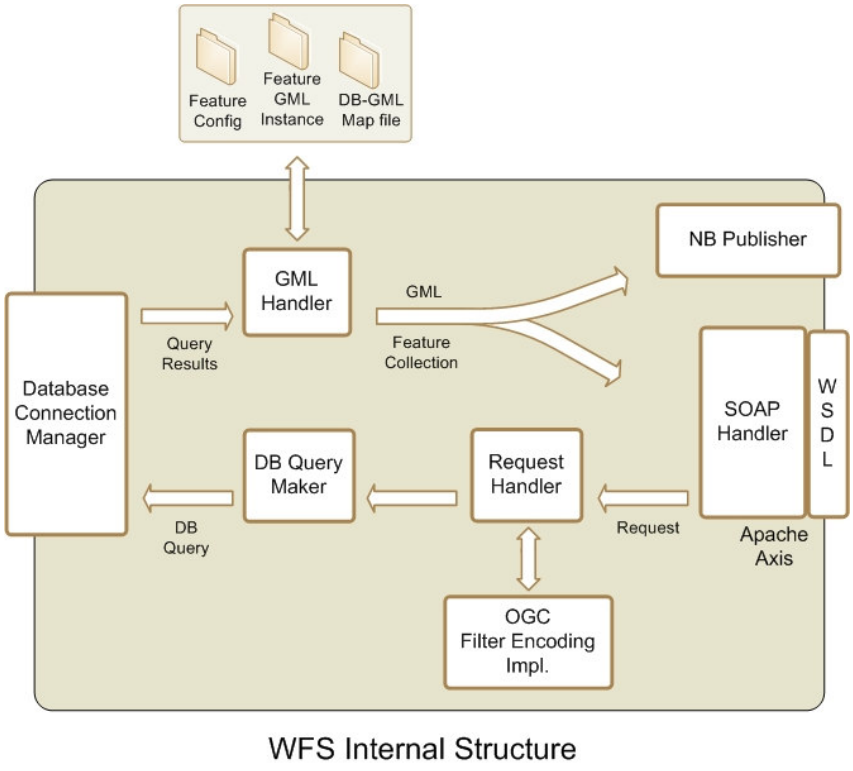


Figure 4-6 - Architectural diagram of the WFS implementation

Figure 4-6 - Architectural diagram of the WFS implementation shows the internal structure of the WFS implementation. A typical WFS request-response cycle starts with the client request. The Client communicates with the service via the WSDL interface. A request may include several types of queries such as SELECT, UPDATE, DELETE etc. Since we have implemented the basic WFS currently we only support SELECT queries. After the request is received the WFS extracts the SQL query from the request using the

OGC Filter Encoding implementation [95] classes. Then the query is executed and the results are received. At this point the WFS uses the configuration files explained above to create the GML FeatureCollection object. Depending of the type of the WFS (Streaming or Non-Streaming) the results are returned to the user via appropriate channel. The streaming WFS is explained in the next section.

Here we explain the request-response cycle with example documents. A sample query encoded according to OGC Filter Encoding Implementation [109] is as follows:

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

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

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

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

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

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

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

```
<gml:lineStringProperty>
   <gml:LineString srsName="null">
     <gml:coordinates>-123.05,39.57 -122.98,39.49</gml:coordinates>
   </gml:LineString>
 </gml:lineStringProperty>
</fault>
</gml:featureMember>
<gml:featureMember>
<fault>
 <name>Bartlett Springs</name>
 <segment>1.0</segment>
 <author>Rundle J. B.</author>
 <gml:lineStringProperty>
   <gml:LineString srsName="null">
     <gml:coordinates>-122.98,39.49 -122.91,39.41</gml:coordinates>
   </gml:LineString>
 </gml:lineStringProperty>
</fault>
</gml:featureMember>
<gml:featureMember>
<fault>
 <name>Bartlett Springs</name>
 <segment>2.0</segment>
 <author>Rundle J. B.</author>
 <gml:lineStringProperty>
   <gml:LineString srsName="null">
     <gml:coordinates>-122.91,39.41 -122.84,39.33</gml:coordinates>
   </gml:LineString>
 </gml:lineStringProperty>
</fault>
</gml:featureMember> …
```

4.3.4 Web Feature Service Capabilities

One of the most important properties of WFS is its ability to provide metadata about the supported features and capabilities. We have developed a method for WFS to dynamically generate the capabilities document on the fly instead of providing a static document. This method allows WFS to ignore feature types stored in an inaccessible

database. To do this the WFS first collects the database information for all feature-types and tests if it can open a connection. Then it uses the individual configuration files for each supported feature type to generate the capabilities document. A Sample capabilities document can be seen in the Appendix.

The capabilities document first describes the access methods to this WFS. Since the specification is based on HTTP Get and POST methods it has Get and Post *OnlineResource* elements. We provide the WSDL endpoint address instead. After the access methods the supported feature types are listed. These feature types may be located on various distributed databases which are opaque to the client. The last section contains the OGC Filter Encoding capabilities supported by this particular WFS installation.

4.3.5 Performance Issues

We have tested our initial Web Service implementation of WFS in several scenarios such as producing fault maps of Southern California, displaying seismic history of particular regions on the map etc. A very interesting application domain was integrating our GIS services with Pattern Informatics [65] code to forecast future seismic activities in a selected geographic region. This test case is explained in the Use Cases section of this chapter. Our experience with these tests showed us several important lessons:

- If the size of the GML documents provided by WFS do not exceed several Megabytes the response time is acceptable. However for larger data sets the response time is relatively long; and in some cases the Web Service throws time-out exceptions. This is caused by our choice of Web Service container, Apache Axis 1.2.

• Web Service require us to create the whole response string on the client side and transport to the client at once. However maximum size of the XML string depends on the system configuration and cannot be very large because the string is created in the memory.

4.4 Streaming Web Feature Service

The usefulness of Web Services is constrained by several factors. They can be used in several cases such as

- The volume of data transferred between the server and the client is not high. Actual amount of data can be transferred depends on a number of factors like the protocol being used to communicate or maximum allowed size by HTTP;
- Time is not a determining factor. Despite the obvious advantages, current HTTP-based implementations do not provide desirable results for systems that require fast response and high performance. This is simply due to the delays caused by data transfer over network, network constraints, and HTTP request-response overhead.

The original WFS specification is based on HTTP Get/Post methods, but this type of service has several limitations such as the amount of the data that can be transported, the rate of the data transportation, and the difficulty of orchestrating multiple services for more complex tasks. Web Services help us overcome some of these problems by providing standard interfaces to the tools or applications we develop. Our experience shows that although by using Web Services we can easily integrate several GI Services and other services to solve complex tasks, providing high-rate transportation capabilities

for large amounts of data remains a problem because the pure Web Services implementations rely on SOAP messages exchanged over HTTP. This conclusion has led us to an investigation of topic-based publish-subscribe messaging systems for exchanging SOAP messages and data payload between Web Services. We have used a publish/subscribe messaging system 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. In the next section we discuss NaradaBrokering, our choice for publish/subscribe messaging system support.

4.4.1 NaradaBrokering

Community Grids Lab has been developing NaradaBrokering [110]; a distributed messaging infrastructure which goes beyond the remote procedure call methodology pure Web Services approach is based on. It provides two related capabilities. First, it provides a message oriented middleware (MoM) which facilitates communications between entities (which includes clients, resources, services and proxies) through the exchange of messages. Second, it provides a notification framework by efficiently routing messages from the originators to only the registered consumers of the message in question.

NaradaBrokering facilitates the idea of loosely coupled systems by supporting asynchronous communication and it can be used to support different interactions by encapsulating them in specialized messages called events. Events can encapsulate information pertaining to transactions, data interchange, method invocations, system conditions and finally the search, discovery and subsequent sharing of resources.[NaradaBrokering]

Some of the important features of NaradaBrokering can be summarized as follows [110]:

- Ensures reliable delivery of events in the case of broker or client failures and prolonged entity disconnects.
- Provides compressing and decompressing services to deal with events with large payloads. Additionally there is also a fragmentation service which fragments large file-based payloads into smaller ones. A coalescing service then merges these fragments into the large file at the receiver side.
- Provides support for multiple transport protocols such as TCP (blocking and non-blocking), UDP, SSL, HTTP, RTP, HHMS (optimized for PDA and cell-phone access) and GridFTP with protocol chosen independently at each link
- Implements high-performance protocols (message transit time of 1 to 2 ms per hop)
- Order-preserving optimized message delivery
- Quality of Service (QoS) and security profiles for sent and received messages
- Interface with reliable storage for persistent events, reliable delivery via WS-Reliable Messaging.
- Discovery Service to find nearest brokers /resources

Additionally, NaradaBrokering allows all services to be linked by managed reliable streams. These capabilities allow fault tolerance and asynchronous messaging with publish-subscribe semantics [22]. A recent addition to the NaradaBrokering features is a sophisticated management environment that controls and monitors all streams in a Grid

[23] and extends fault tolerance across streams, services and message brokers. The latter allows one to control the flow of data into filters so that there is no overflow. NaradaBrokering supports the subscription of redundant services to aid in fault tolerance. The Web Service messages flowing in NaradaBrokering can be archived at any link. This provides for dynamic caching to support system performance and is also used in message throttling. NaradaBrokering has been successfully used for audio-video conferencing and other collaborative tools in the commercial Anabas product [25] and the open source GlobalMMCS project [111][13, 23, 26].

NaradaBrokering has been used extensively in several projects that require real-time streaming data access. Because of its relevancy to our topic and as an example how it has been used to provide reliable streaming support we summarize GlobalMMCS here;

4.4.2 GlobalMMCS: Using NaradaBrokering to Manage Audio/Video Streams

We think that the nature of sensor data is somewhat similar to that of audio/video and a Service Oriented Architecture which employs NaradaBrokering should exhibit high performance for sensor filter services.

Global Multimedia Collaboration System [111] is designed to provide scalable videoconferencing services to a diverse set of users. The system uses NaradaBrokering as the media distribution medium. Topics provided by NaradaBrokering serve as the messaging channels among participants in a session to exchange data [112].

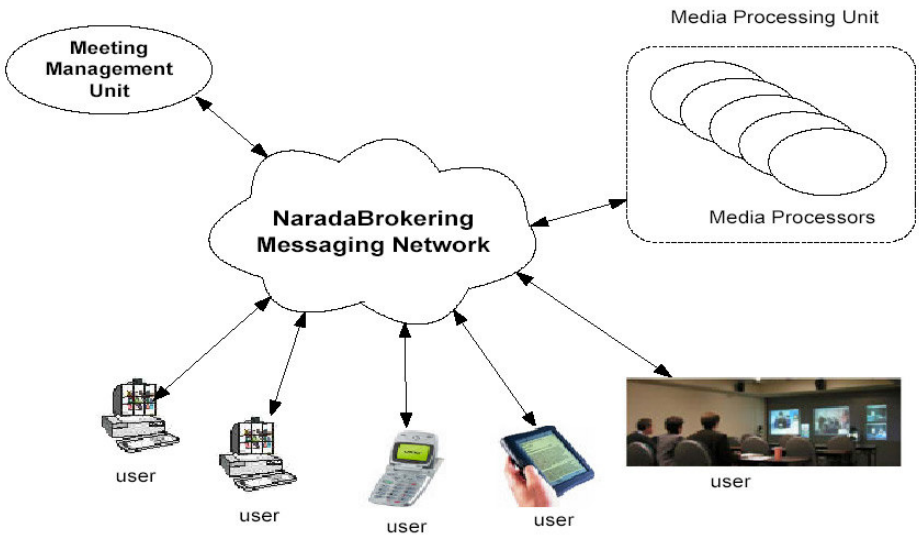


Figure 4-7 - Main Components of GlobalMMCS architecture [113]

NaradaBrokering has proved to be very efficient in delivering audio and video streams to a group of participants in a meeting.

The topic based publish-subscribe system works as follows: The source user or the data provider publishes copy of a stream to a topic and the broker network delivers this stream to all the subscriber of this topic by duplicating whenever necessary. To save network bandwidth NaradaBrokering avoids sending multiple copies of the same stream on the same link. Additionally it calculates the near-optimal routes from sources to destinations by organizing the brokers in hierarchical cluster architecture.

This architecture provides a scalable and flexible framework to distribute media processing units. Additional futures of NaradaBrokering allow the system to be dynamic; the capacity of the system can be increased by easily adding new computing resources and new processing services can be integrated to support ever changing needs of end users. Performance of the GlobalMMCS system is investigated extensively and the results show that the system exhibits high performance for audio/video meetings [112].

4.4.3 Comparison of Streaming and Non-Streaming Web Feature Services

The Streaming-WFS uses standard SOAP messages for receiving queries from the clients; however, the query results are published (streamed) to a publish/subscribe messaging substrate topic as they become available. Since we use MySQL database for keeping geographic features, we employed MySQL streaming result set capability by streaming the results row by row. This allowed us to receive individual results and publish them to the messaging substrate instead of waiting for whole result set to be returned. The performance results (see Chapter 5 for detailed performance results) show that (especially for smaller data sets) streaming removes a lot of overhead introduced by object initializations.

Table 6 gives a comparison of the streaming and non-streaming versions of our WFS implementations. The data requested is the Southern California seismic records for the eventful year of 1992, initially obtained from Southern California Earthquake data center [SCEDC] [107] and converted into GML for our Web Feature Service. The first column is the minimum magnitude of the earthquake, the second column shows the data size of the query result. Timings for Streaming WFS contains two columns; the first column shows the time it takes to generate and stream out GML feature collection, the second column shows the total response time. The fourth column shows the total response time for non-streaming WFS. The difference between streaming and non-streaming WFS versions is that streaming version does not accumulate the query results and stream as soon as they become available. The timings are in milliseconds and include object initializations, query processing, database query and transport times.

To measure the performance of two WFS versions we made several tests using seismic catalog for year 1992 from Southern California Earthquake Data Center (SCEDC). Tests were performed for the following lower bounds with seismic event magnitudes: M = 5.0, 4.5, 4.0, 3.5, and 3.0. These correspond to increasing data file size, as shown in Table 4-5. We measure WFS performance by timing the steps needed to extract seismic records with specific latitude/longitude bounding boxes, time periods, and lower bounds for the earthquake threshold magnitudes. These extracted records are returned as GML responses. This test is representative of other applications which need to extract records from remote databases using the WFS. In these tests data from 1/1/1992 to 12/31/1992 were requested and latitude/longitude bounding box (-117.0, 32.0)-(-114.0-37.0) was used.

Table 4-5 - Performance Comparison of Streaming and Non-Streaming WFS Versions

Event Magnitude Lower Bound	Number of Seismic Events	Data Size (KB)	Streaming WFS		Response Time Non-Streaming WFS
			Time for streaming the result	Total Response Time	
3	1790	880	2414.2	4570	5662.6
3.5	587	287	826.7	3405	4414
4	209	106	320.1	2945	4098.7
4.5	67	36	100	2661	3917.1
5	19	11	31.3	2425	3912.5

We can deduce from the table that for 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.

Other improvements were also made in Streaming-WFS to make it a high performance service. These improvements are discussed in Chapter 5.

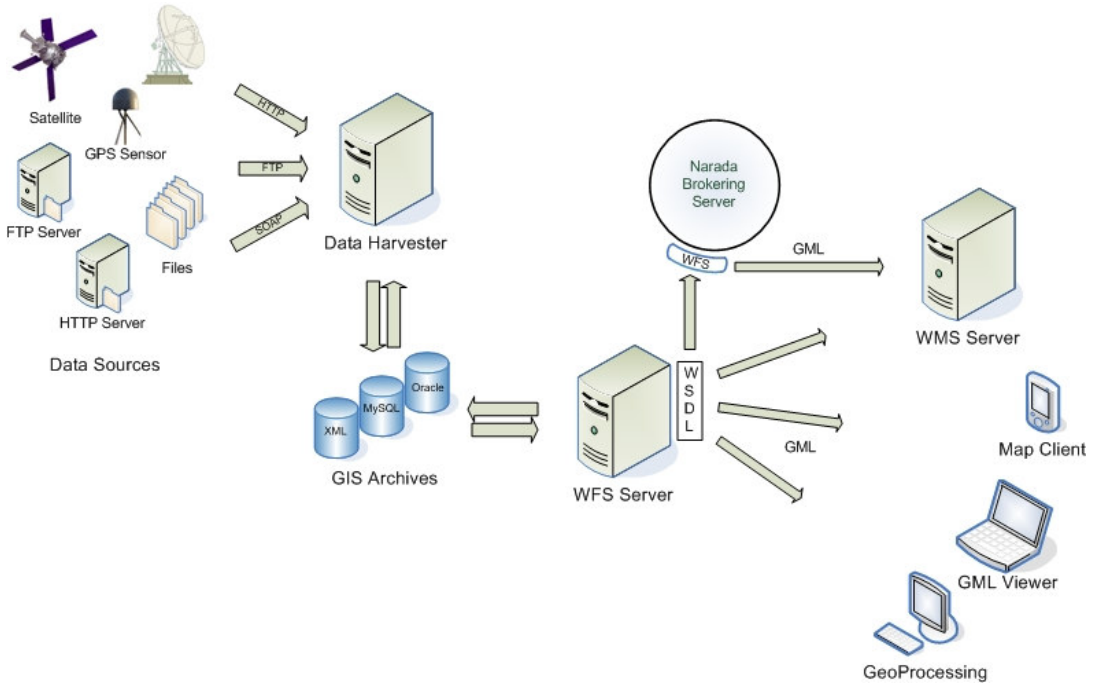


Figure 4-8 - WFS in a Grid environment. The WFS serves feature data collected from various sources such as online file servers, local files or sensor observation archives. The collected data are inserted into our GIS archives and served by the WFS in GML format.

Figure 4-8 shows the integration of the WFS with data sources and clients. The WFS is able to return the results as SOAP messages or stream over the NaradaBrokering publish/subscribe messaging substrate. This figure also summarizes our efforts to create a GIS Data Grid. The first steps for this work required harvesting data from various publicly accessible archives such as Southern California Earthquake Data Center (SCEDC) seismic records, ANSS seismic archives, NASA-JPL GPS time series and SOPAC GPS time series archives. For these purposes we have created a Data Harvester module inside the WFS which has HTTP and FTP client implementations. We have also

implemented a programmable harvester tools which connects to online servers to retrieve regularly updated seismic records.

Once the data files are retrieved from the online archives the WFS uses the methods explained in Chapter 4.3.2 to insert these data into our spatial database. After this point the data are served to the clients as GML documents through WFS interfaces.

4.5 Geophysical Data Grid Examples

Our WFS implementations have been used in several GIS projects as data provider services, here we discuss two examples.

4.5.1 Los Alamos National Laboratory, NISAC SOA Architecture

We have applied our GIS Grids ideas to create a Service Oriented Architecture for Los Alamos National Laboratory, National Infrastructure Simulation and Analysis Center [71]. We have integrated several Web Services including the Streaming-Web Feature Service with IEISS (Interdependent Energy Infrastructure Simulation System) [114]. IEISS is a suite of analysis software used to understand the normal operations of the infrastructures and various implications of the interdependencies between the components[71]. 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 the usual operation IEISS and similar software are being used as local simulation applications. The data are either being kept in databases such as ESRI spatial database, or in proprietary XML files. The person who runs the application collect 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. So 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. Figure 8 shows the components and data flow in this architecture.

The sample electric and natural gas infrastructure components are provided to us as XML files. We have inserted the components into a MySQL database which allowed us to query for specific components in particular geographic regions. Figure 9 visualizes the components. Figure 10 shows the overlays of these components on a satellite picture provided by the NASA OnEarth WMS Server [115]. We used an open source GML viewer GAIA [116] to create these figures.

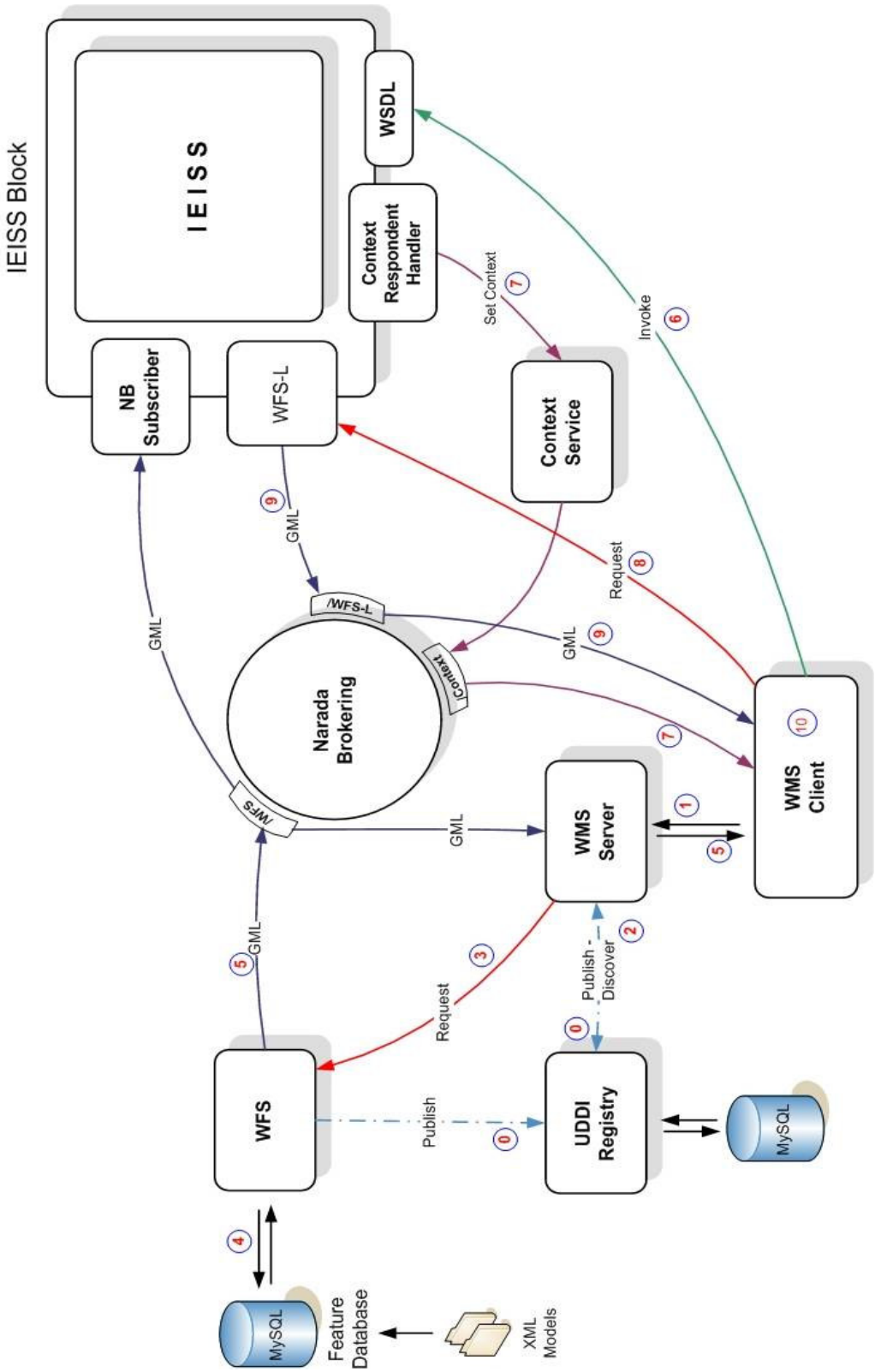


Figure 4-9 – NISAC SOA Demonstration Architectural Diagram and Data Flow

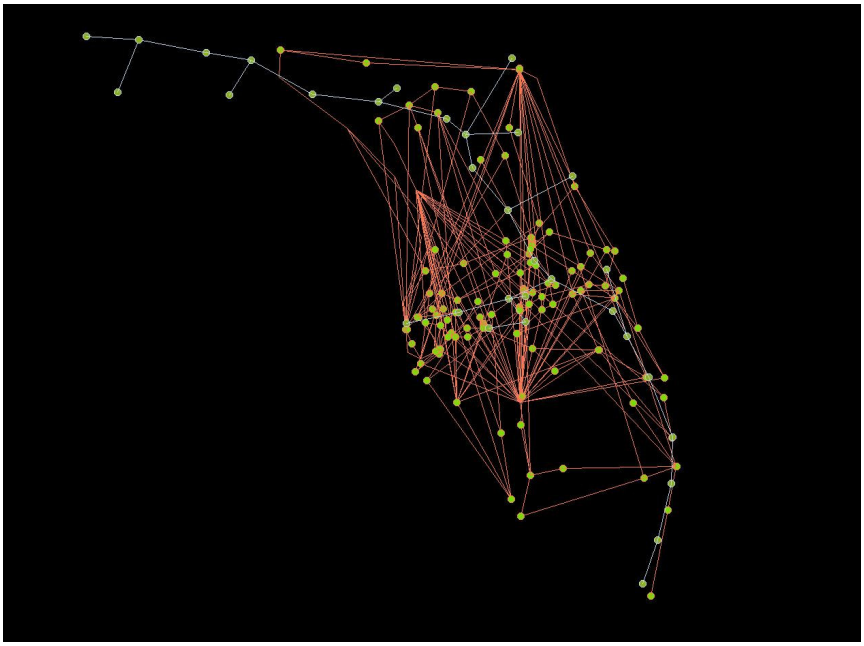


Figure 4-10– Original IEISS Data for the Florida State Electric Power and Natural Gas Components

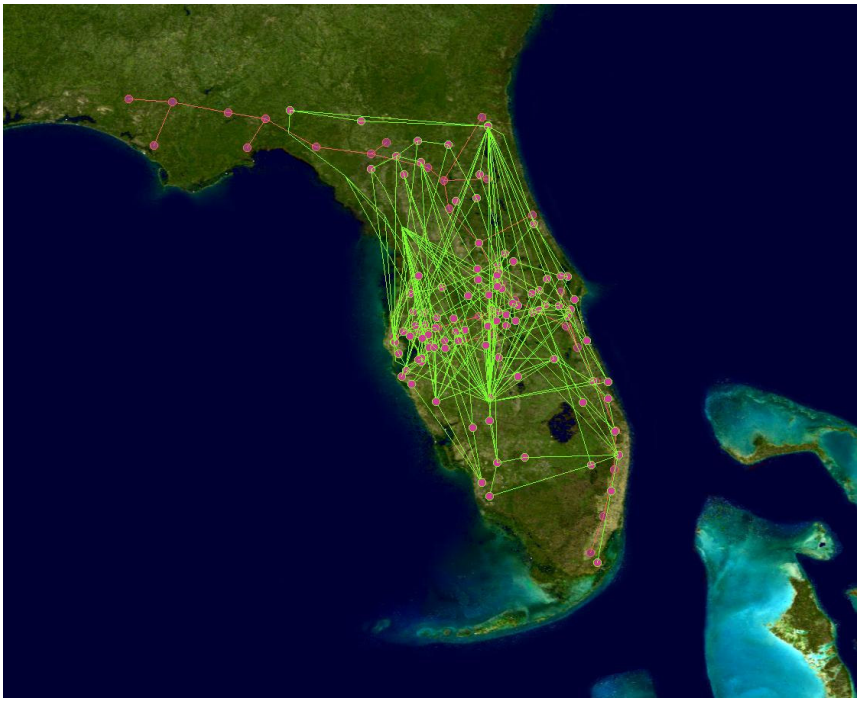


Figure 4-11 – Sample Florida State Electric Power and Natural Gas Components as overlays on a Satellite Picture provided by NASA OnEarth WMS Server. Electric power components are connected with green, natural gas components are connected with red lines.

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.

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 a 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 geo-spatial and visualization services. It extends existing Universal Description, Discovery and Integration (UDDI) [79] Information Model to provide GIS domain specific Information Services.

Web Map Client: This is a thin client to the Web Map Server. It provides a user interface that displays the map overlays and allows client interaction with the maps.

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: 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 (http://deegree.sourceforge.net/) to convert these shape files to GML, which are sent to WMS Client by the lightweight WFS.

Data Flow in this architecture is explained here (Figure 8):

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.
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.

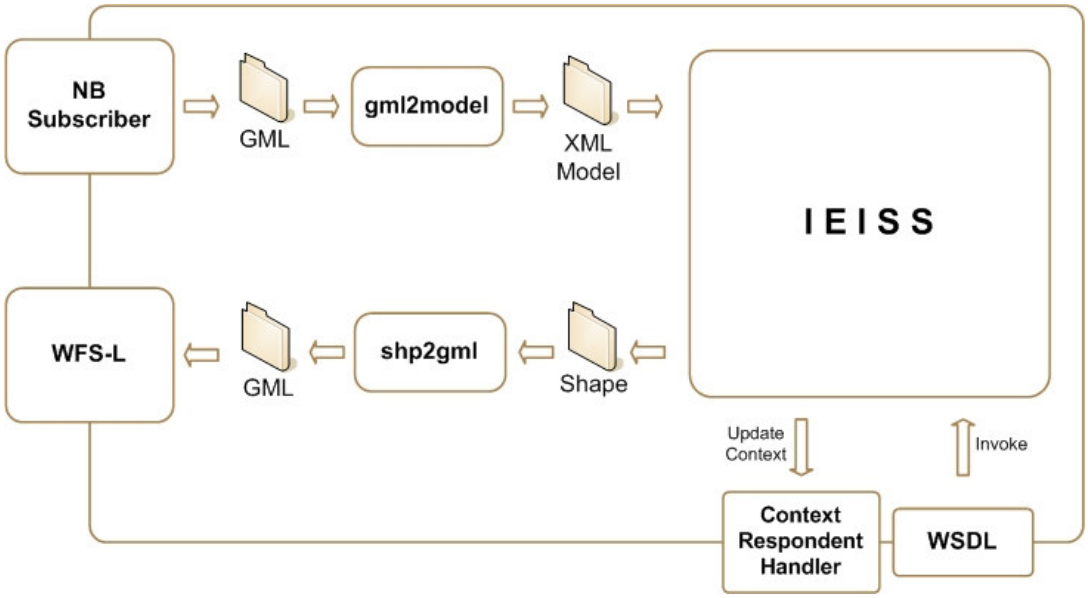


Figure 4-12 – Data flow in the IEISS Block

Figure 4-12 shows the data flow in the IEISS block. To use the IEISS application as a Web Service we have created a small WSDL wrapper, which is responsible for receiving the SOAP messages and invoking IEISS code with the received parameters. First, the NB Subscriber receives the GML model which corresponds to the user's selection on the mapping client. Because the IEISS simulation requires a particular XML format the gml2model tool converts the input to this model format. And as soon as the WSDL engine receives the invocation message the IEISS Simulation runs on the data already received and outputs the result as a shape file which is a specific type of vector data format. Depending on the input parameters provided the application calculates the affected geographic area and outputs it in the shape file.

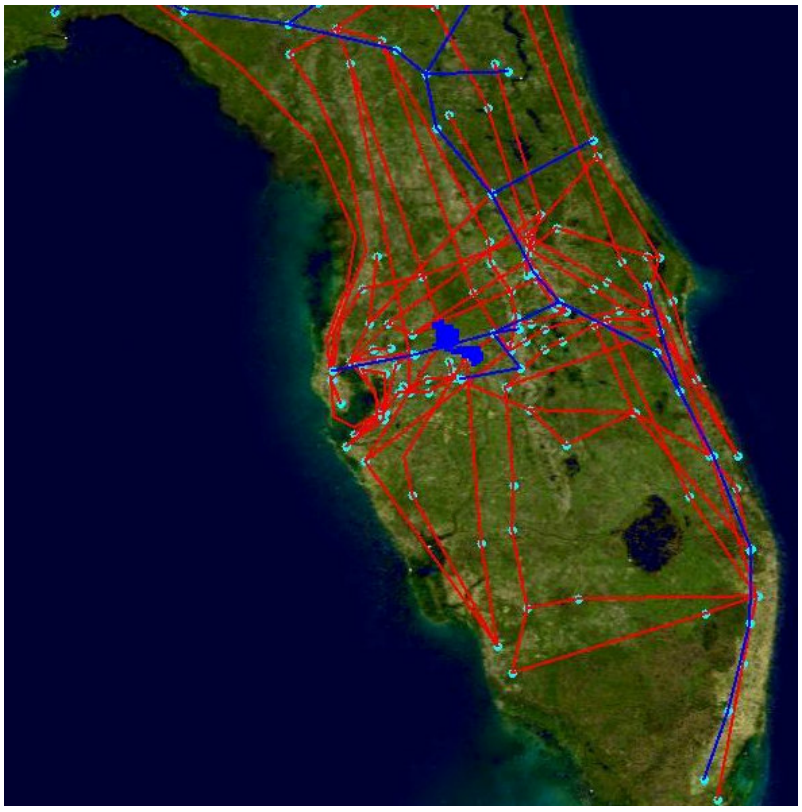


Figure 4-13 – Sample IEISS output generated by the WMS; The blue region is the affected area calculated by IEISS because of a possible problem with the energy infrastructure.

Figure 4-13 shows a sample IEISS output; here the blue region depicts the affected outage area. This image is generated by the Web Map Service developed in Community Grids Lab, see [17, 68, 69] for more information.

4.5.2 Pattern Informatics Integration

Pattern Informatics [65, 66] 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 [69] is responsible for collecting parameters for invoking the PI code. These parameters are then sent to an HPSearch [117-120] engine which invokes the various services to start the flow. The process is diagrammatically illustrated in Figure 4-14. The Code Runner Service is a sample wrapper service that invokes the Pattern Informatics application.

As shown in the figure, the Web Map Service submits a flow for execution by invoking the HPSearch Web Service.

Figure 4-14's steps are summarized below. This is the basic scenario that we use for integrating Pattern Informatics, RDAHMM, and other applications.

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.
2. WMS Server dynamically discovers available WFSs that provide requested features through UDDI Registry and obtains their physical locations (WSDL address).

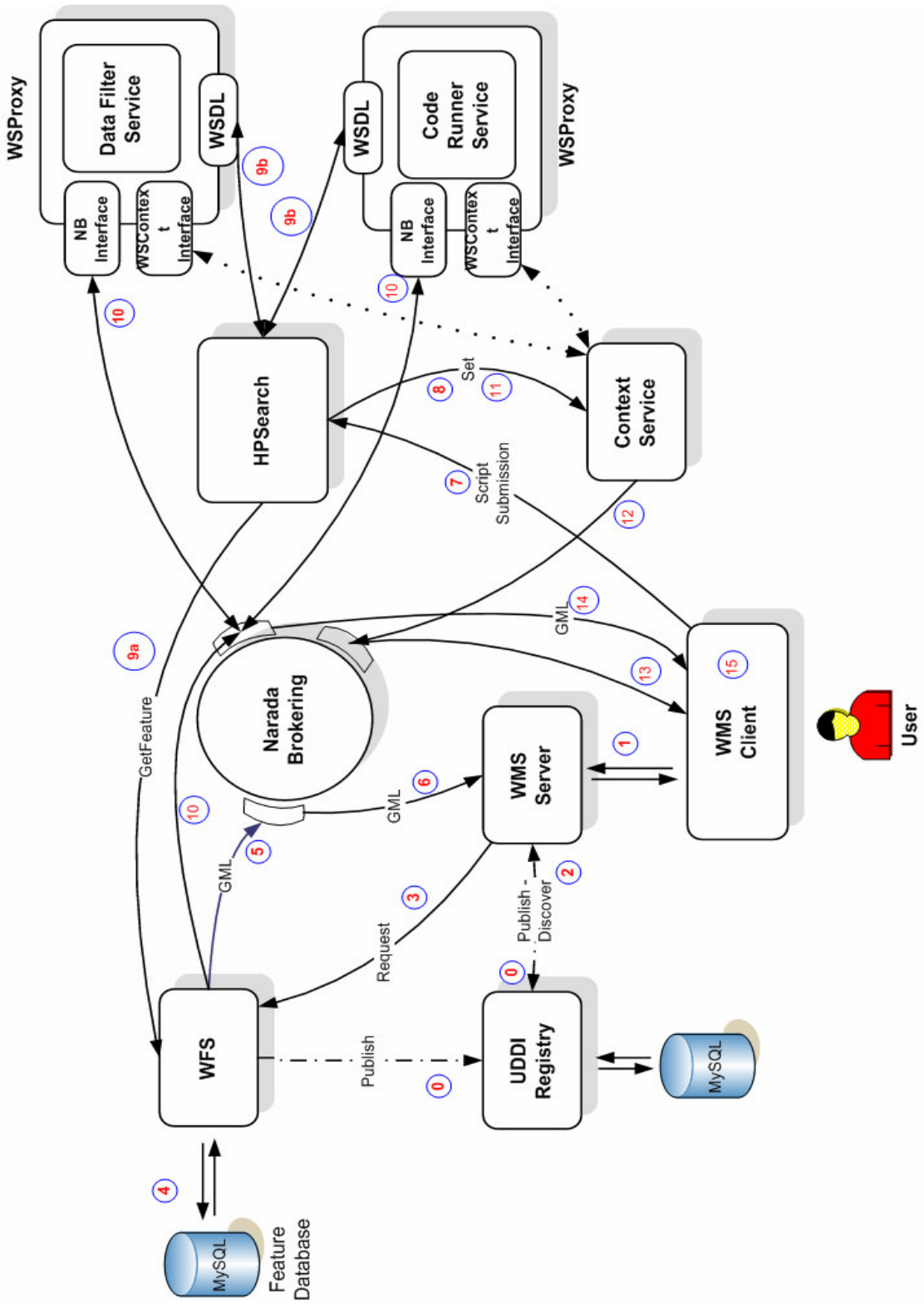


Figure 4-14 - A general GIS Grid orchestration scenario involves the coordination of GIS services, data filters, and code execution services. These are coordinated by HPSearch.

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 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. The WMS submits a flow for execution by invoking the HPSearch Web Service. 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 the WMS.
8. Initially, the context corresponding to the script execution is marked as "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 (Geographical Markup Language) 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.

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.

4.6 Summary

In this chapter we have first discussed the data and application level interoperability problems in GIS and presented a Service Oriented Architecture to answer these problems. Although the open GIS standards gain ground and being accepted as the common formats the lack of a strong application level interoperability framework is a serious issue for distributed GIS frameworks. Therefore we have discussed that this problem could be answered by developing Web Services based GIS Grids. We have implemented a fundamental OGC specification the Web Feature Service as both streaming and non-streaming Web Services and using this service created a GIS Grid. We have integrated several scientific GIS applications with our GIS Grid Services and proved that this approach can be used with real-world analysis and simulation applications including the cases demanding high-performance and high rate data input.

Figure 4-15 shows the interaction of services explained in this chapter. Two major data services we have developed for archival geographic data are streaming and non-streaming Web Feature Services.

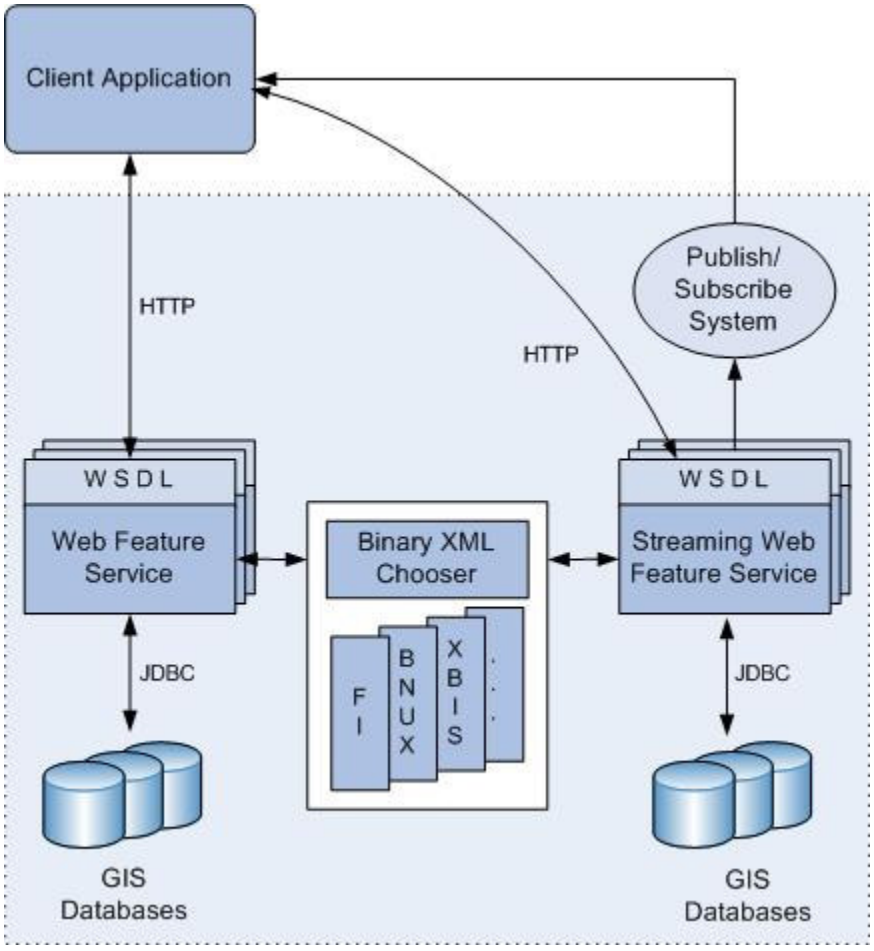


Figure 4-15 - High Performance Data Grid for Geographic Information Systems

The Client communicates with the services through traditional Web Service interfaces. The non-streaming Web Feature Service responds to client requests via SOAP messages transferred over HTTP, while the Streaming Web Feature Service returns the results over a publish/subscribe messaging system. Both services provides Binary XML support for encoding the returning XML documents as binary documents for performance and bandwidth consumption improvements. In the next chapter we explain the streaming services and Binary XML integration.

Chapter 5

Streaming Web Feature Service and Performance of the GIS Data Grid

5.1 Introduction

Recent research discussed that SOAP is not an efficient solution for high-end data transport [121-124] and there are several ways to improve the Web Services performance [125-134]. One way of improving the performance of a traditional HTTP based Web Service is to incorporate a better transport protocol. Although HTTP provides a universally agreed upon communication platform for Web Services, it has serious limitations in terms of providing support for larger payload transfers and high rate data exchange. Later becomes especially apparent in the real-time or near-real time systems with the use of SOAP envelope which increases the message size. For instance [135] shows that the SOAP protocol increases the message sizes by a factor of four to ten as

compared to the binary representation. Therefore a second area where the performance increase can be gained is to decrease the size of the SOAP message payload.

Our Web Service based implementation of the OGC Web Feature Service Specification gave us a chance to make extensive tests to investigate the limitations of Web Services in both of the areas mentioned above. Essentially a Web Features Service provides a unified front for accessing different data stores to retrieve geo-spatial feature data as Geography Markup Language (GML) formatted documents. We have initially implemented the Web Features Service as a traditional Web Service which returned the requested GML responses as strings embedded in the SOAP envelope. However our tests have shown that memory related issues limited the amount of information we could contain as in-memory strings. On the other hand we also knew that some scientific applications require fast access to large amounts of geographic data. These conclusions led us to the development of a streaming version of the Web Feature Service.

Our Streaming Web Feature Service has a WSDL end-point which provides the same operations with the non-streaming version, however additionally it can employ a topic based publish-subscribe system to stream out the GML results instead of sending them as SOAP messages over HTTP.

We chose NaradaBrokering to provide streaming capabilities. NaradaBrokering is a topic based publish/subscribe messaging system which provides several important features appropriate for our use cases. Some of these properties are explained in [136] and we summarize them here:

- The communication with NaradaBrokering is asynchronous; this is an important feature where the time interval between request and response is long and this property allows non-blocking interaction between clients and the WFS.
- NaradaBrokering supports large client configurations publishing messages at a very high rate and there are no restrictions placed by the broker on the number, rate or the size of the messages issued by the clients.
- Entities can specify Quality-of-Service (QoS) restraints on the message delivery such as reliable delivery of the messages, exactly-once delivery etc. The broker allows entities to retrieve event issued during an entity's absence.
- NaradaBrokering also provides fragmentation/coalescing service for large data transfers. This service breaks the large files into manageable fragments, publishes the individual chunks. On the receiver side, these chunks are written in a temporary storage area, and once it is determined that all the chunks are retrieved all fragments are coalesced into a single file.
- Additionally NaradaBrokering provides capabilities for communicating through a wide variety of firewalls and authenticating proxies.

In short using NaradaBrokering in our system gives us enormous flexibility in terms of supporting arbitrarily large message sizes and high transfer rates. Our previous tests show that by streaming the GML documents over publish/subscribe based messaging broker we make significant performance gains.

In this chapter we report the results of our research efforts to further improve the performance of the Web Features Service by incorporating Binary XML frameworks. It should be noted that these improvements can be applied to other Web Services as well.

Related research shows that SOAP message transfer over HTTP has inherent problems. One of these problems is the SOAP header that needs to be transferred along with every message embedded in the SOAP body. Considering the fact that most of the time Web Service clients make more than one requests from the server it becomes obvious that the same header will be redundantly exchanged with every message between the server and the client. One way to overcome this redundancy is to save the header only once in a third party online repository accessible to both the Web Services server and the client [130]. The header may be put to this repository by the server and client may request it at the beginning of the transaction to process the incoming messages.

Another possible performance improvement can be made by reducing the size of the data payload or the SOAP body section. XML is the universal format for the message exchange in Web Services. However one significant drawback of the XML encoding is that it increases the size of the raw data.

In recent years several binary XML frameworks have been developed to help reduce XML document sizes [137]. In September 2003 The W3C ran a workshop to study methods to compress XML documents, comparing Infoset-level representations with other methods. The goal was to determine whether a W3C Working Group might be chartered to produce an interoperable specification for such a transmission format [138]. The W3C has formed The XML Binary Characterization Working Group as a result of this workshop [139]. Although the workshop concluded that there should be further wok in this area to decide if W3C should attempt to define formats or methods for non-textual interchange of XML many independent groups or individuals developed several binary XML formats such as Fast Infoset [134], XBS [140] and BNUX [141]. In 2005 W3C has

released a Working Group Note which includes an analysis of which properties a binary XML framework must possess and recommends that W3C produce a binary XML recommendation [126].

Depending on the content of the XML document these binary frameworks usually achieve significant compression rates which can be very useful in accomplishing very good transfer times. However the encoding and decoding of the XML may introduce significant overheads as well. Therefore the advantages and disadvantages of each binary XML framework must be studied for each type of the data.

We have made several tests to investigate possible performance improvements to our Streaming-Web Feature Service by using two major binary XML frameworks, namely Fast Infoset and BNUX. Additional tests will be made with the non-streaming Web Feature Service to see if any improvements can be made using binary XML encodings in a traditional Web Service usage..

5.2 Streaming Web Feature Service

The Web Feature Service provides access to geographic information stored in various distributed databases. The data is encoded as GML documents, which is a popular XML dialect for describing geospatial entities. The Streaming- Web Features Service has a WSDL document to describe the operations it supports, which is used by the client to send feature requests. Essentially the client requests are made as traditional Web Service calls, but the responses are transmitted over NaradaBrokering topics.

Along with the feature request the Client provides the Broker address and topic information to the server. The server retrieves the desired features from the database,

converts these to GML and publishes these features to the given topic. Figure 1 depicts this process.

For the performance tests we use Southern California Earthquake Data Center's (SCEDC) seismic data records from 1932 to 2005. In GML terms each of these records are considered a geographic feature and represented in the GML document as an individual<gml:featureMember> element with attributes and sub-elements as following.

```
<gml:featureMember
xmlns:gml="http://www.opengis.net/gml">
<SeismicEvent>
 <Date>
   <Year>1992</Year>
   <Month>4</Month>
   <Day>15</Day>
 </Date>
 <Time>
   <Hour>6</Hour>
   <Minute>51</Minute>
   <Second>17</Second>
 </Time>
 <Location>
   <gml:Point srsName="SRS">
     <gml:coord>
       <gml:X>34.291</gml:X>
       <gml:Y>-117.564</gml:Y>
     </gml:coord>
   </gml:Point>
 </Location>
 <EventType>et</EventType>
 <Magnitude>3.47</Magnitude>
 <MagnitudeType>n</ MagnitudeType>
 <Depth>7.2</Depth>
 <Quality>B</Quality>
 <NPH>16</NPH>
 <NGRM>26<NGRM>
 <EventId>2038429</EventId>
</SeismicEvent>
</gml:featureMember>
```

A GML document usually consists of several such feature members inside a </gml:FeatureCollection> tag.

The Web Features Service retrieves the features from the database in a streaming fashion, i.e. we use the MySQL streaming query statement property to receive individual query results immediately as they become available, instead of waiting for the whole

result set to be created by the database driver. This approach improves the performance by allowing us to create gml:featureMember elements for each individual query result immediately upon receiving.

Another point where performance gain can be obtained is to find an optimal number of features to accumulate inside one message before publishing to the broker topic. Usually the clients request multiple features in one query. The WFS may choose to create a broker event for each gml:featureMember element and publish these to the broker topic, or may choose to publish multiple gml:featureMember elements as one broker event.

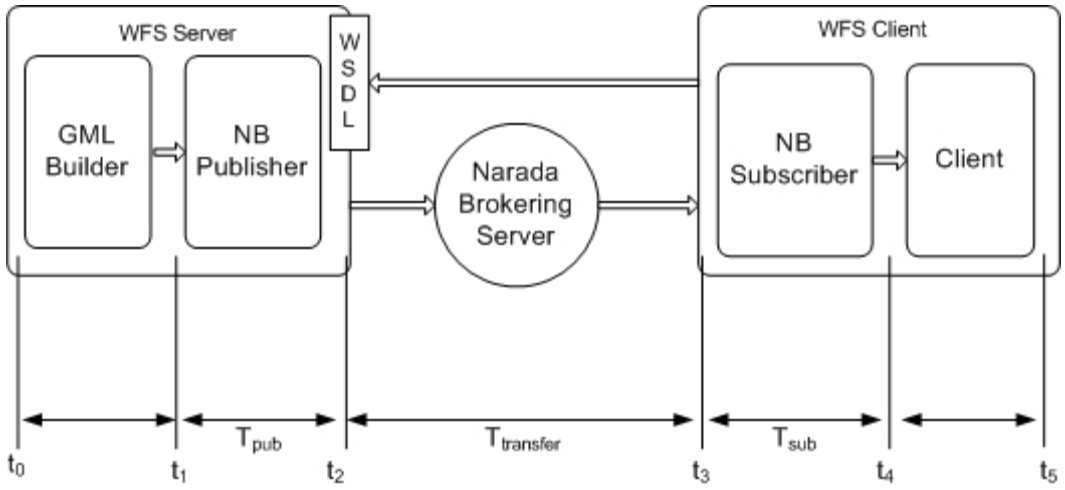


Figure 5-1 - Streaming Web Feature Service Performance Test Setup

Figure 5-1 shows the data flow in the performance test setup. In this scenario we measure total response time, publish, subscribe and transfer times.

$$T_{total} = T_{pub} + T_{transfer} + T_{sub}$$

The timings taken in this scenario are intended to find out the contributions of publish, subscribe and transfer times to the overall performance of the system, because these are the parts of the system which will be affected with the addition of a binary XML framework.

The request-response cycle starts with the client making a GetFeature request through WFS-Server's WSDL interface. The server decodes this message, creates a MySQL query and queries the database. Every individual query result returned is converted to GML and published to the broker topic. However the total number of features in the response affects the overall performance since the transfer time will be changed according to the size of the message. We test the system for various message sizes, by increasing the number of features included in each message to be published.

The individual messages published by the WFS-Server are parts of a GML Feature Collection document, hence essentially are XML fragments. And because we stream these fragments through a broker topic without embedding in a SOAP envelope we can easily test the effects of using a binary XML framework by directly converting these XML documents into a binary format.

A self-contained binary XML framework has two major parts, an encoder which using a particular algorithm converts the text based XML document into a binary document, and a decoder which reads the binary document and reformats it to the original XML input. The most obvious reason for using such a framework is size compression. Depending on the type of the data the compression rate is usually very high (See table below). This in turn allows us to use less network bandwidth and better transfer times which means higher performance.

However encoding an XML document into binary and decoding it back to XML can be CPU intensive and time expensive operations. We must carefully evaluate each framework to see if we lose more time with encoding and decoding than what we gain from the transfer by decreasing the size of the message.

Figure 5-5-2 shows how we can integrate a binary XML framework with our streaming Web Feature Service. In addition to the Streaming Web Feature Service scenario discussed above we now have a Binary Encoder unit on the Server side which converts the GML fragments into binary documents and a Binary Decoder on the Client side which receives these binary documents, converts back to XML and gives to the client process.

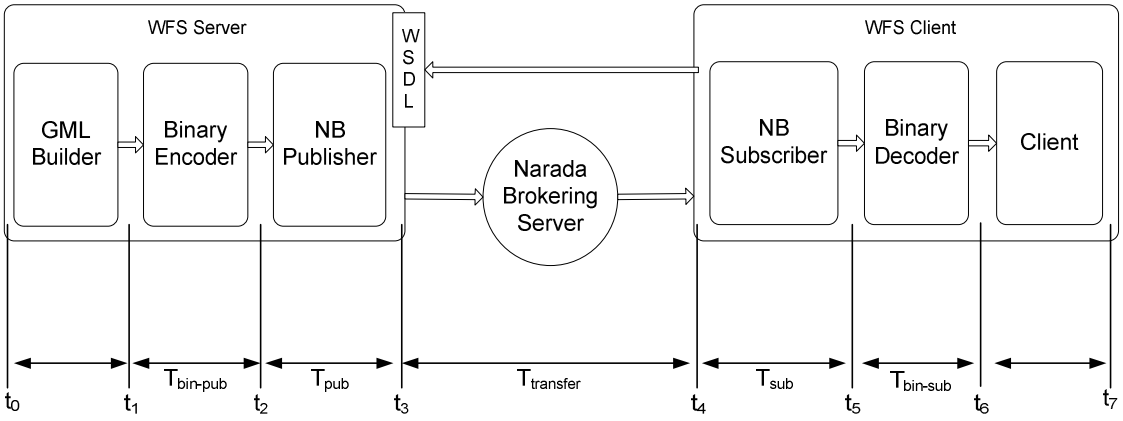


Figure 5-5-2 - Streaming Web Feature Service integrated with a Binary XML framework.

In this case we take additional timings to measure the time spent during encoding and decoding steps. In this scenario the total time is the sum of all steps described in the first step plus the binary encoding and decoding times:

$$T_{total} = T_{bin-pub} + T_{pub} + T_{transfer} + T_{sub} + T_{bin-sub}$$

5.3 Performance Tests

As depicted in Figure 5-1, the test scenario starts with the building of the GML Feature Collection object using the results obtained from the database. Figure 5-3 shows the message creation time for various GML document sizes. Web Service call times are

not included in the following figures since these are almost constant values for all queries. The database query time is also included in this figure.

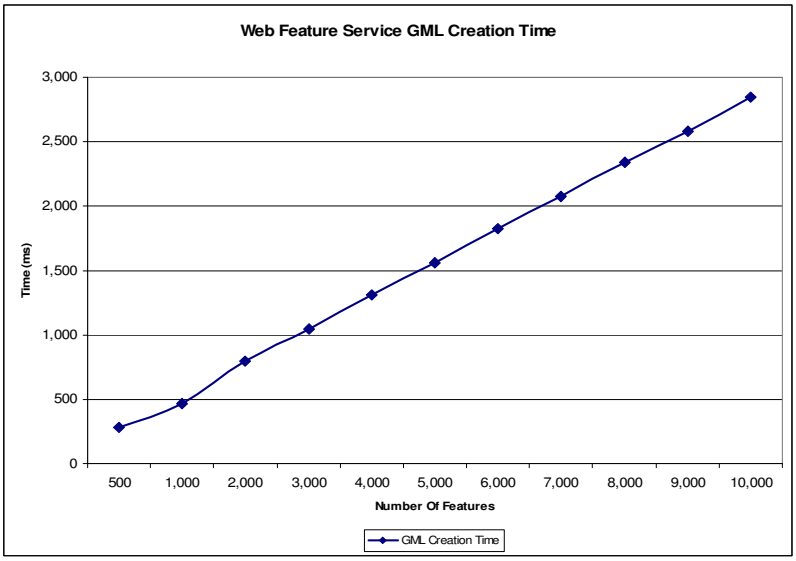


Figure 5-3 – Web Feature Service GML Message (GML Feature Collection) creation times as a function of number of the features in the message

We test the performance of the system by publishing various GML documents from the WFS Server and measuring the total time it takes to transfer the whole file to the client. To understand the behavior of the system for smaller and larger data transfers we use two different groups of files. The first group contains 10 GML documents with sizes between 10KB to 100KB. The second group contains 11 files with sizes between 500KB to 6MB, with the number of features contained in these files increasing from 500 to 10000. The WFS Server publishes the GML documents in increasing size order.

Table 5-1 Size comparison of the documents used in the tests as textual XML files and as compressed binary formats

Number Of Features	Document Size (KB)		
	XML	BNUX	Fast Infoset
100	61	24	24
500	300	75	70
1,000	600	148	139

2,000	1203	302	281
3,000	1805	482	422
4,000	2406	606	563
5,000	3036	758	704
6,000	3609	901	836
7,000	4210	1056	986
8,000	4817	1257	1129
9,000	5417	1483	1274
10,000	6018	1623	1402

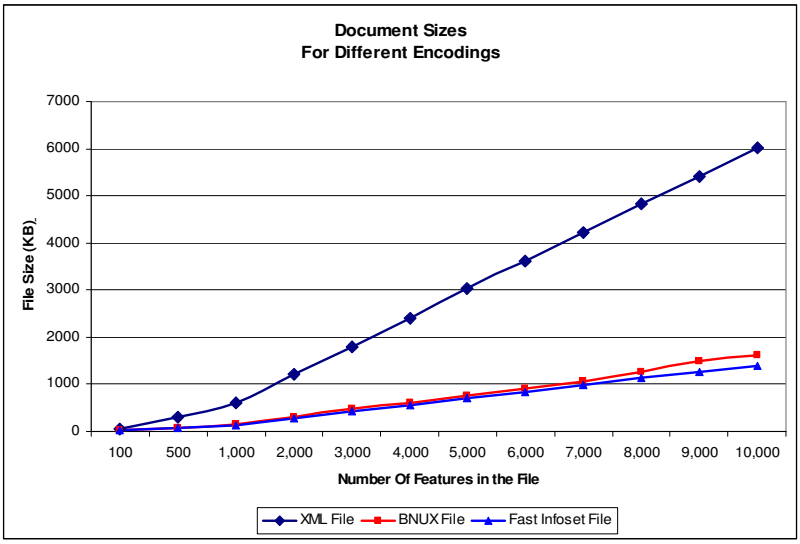


Figure 5-4 - Document Sizes for different encodings.

Table 5-1 shows the size of the GML files and binary documents used in the tests. Document sizes are reported as kilobytes. Figure 5-4 depicts the values reported in Table 5-1.

To measure the effects of using binary XML frameworks we test the system by publishing these GML documents in textual, Fast Infoset and BNUX formats. The files are published in two groups with the first 10 files representing smaller payloads up to 100KB, while the second group of 11 files representing larger payloads. In the tests we publish each file for 50 times and calculate the average and standard deviations which are shown in the figures below.

To understand the effect of the network distance between the broker, the publishers and subscribers we use three different NaradaBrokering servers running at different cities. We run the WFS-Server and the Client at our local GridFarm servers in Bloomington, Indiana. This ensures the similar performance in all three test cases for binary XML processing steps, encoding and decoding the documents. A typical GridFarm server specs are given in Table 5-2.

Table 5-2 – GridFarm008 Machine configuration summary

Processor	Intel® Xeon™ CPU (2.40GHz)
RAM	2GB Total
Bandwidth	100Mbps
Operating System	GNU/Linux (kernel release 2.4.22)
Java Version	Java 2 platform, Standard Edition (1.5.0-06)

Table 5-3 - Test Cases

Test Case 1 - NB Server in Bloomington, IN

1.a Textual XML Transfer Performance

1.a.1 Small Files

1.a.2 Large Files

1.b BNUX Integration Performance

1.b.1 Small Files

1.b.2 Large Files

1.c Fast Infoset Integration Performance

1.c.1 Small Files

1.c.2 Large Files

Test Case 2 - NB Server in Indianapolis, IN

2.a Textual XML Transfer Performance

2.a.1 Small Files

2.a.2 Large Files

2.b BNUX Integration Performance

2.b.1 Small Files

2.b.2 Large Files

2.c Fast Infoset Integration Performance

2.c.1 Small Files

2.c.2 Large Files

Test Case 3 - NB Server in La Jolla, CA

3.a Textual XML Transfer Performance

3.a.1 Small Files

3.a.2 Large Files

3.b BNUX Integration Performance

3.b.1 Small Files

3.b.2 Large Files

3.c Fast Infoset Integration Performance

3.c.1 Small Files

3.c.2 Large Files

5.4 Performance Test Results

5.4.1 LAN Tests

In this configuration the WFS and NaradaBrokering servers and the WFS Client run on GridFarm servers, shown in Figure 5-5. Since the test components run on the same servers and share the same file system, this is the ideal situation in terms of data transfer time.

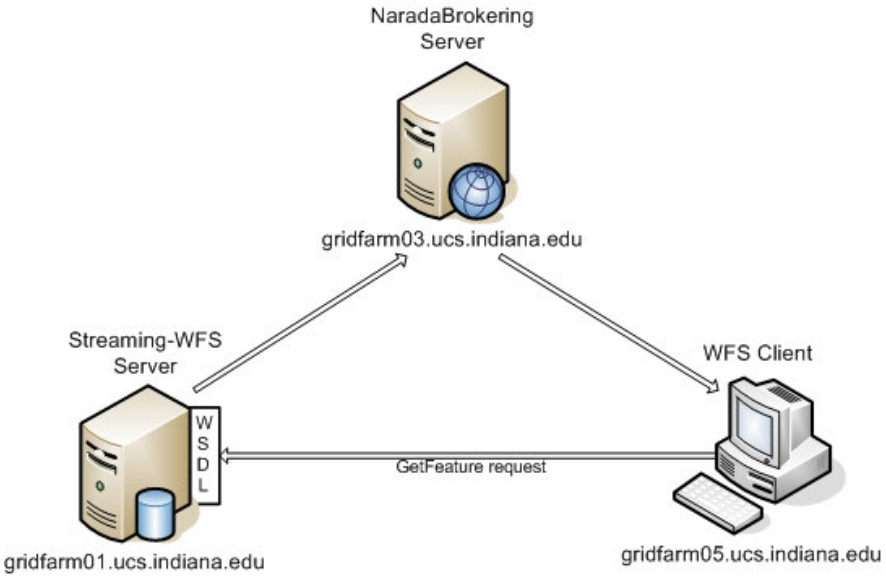


Figure 5-5 Test Configuration for the first case

5.4.1.1 Streaming WFS Performance with Textual XML Transfer

The first test measures the data transfer time on the network for exchanging textual GML documents which includes publish time on the server side; receive time on the client side and actual wire transfer time. Figure 5-6 and Figure 5-7 shows the test results for smaller and larger data sizes.

Table 5-4 - Average XML transfer times and standard deviations

Number of Features	Average Transfer Time (ms)	Standard Deviation
10	3.00	1.23
20	3.86	1.45
30	3.68	1.25
40	4.22	1.15
50	4.86	1.01
60	5.18	0.83
70	5.38	1.03
80	5.60	0.99
90	6.12	0.94
100	7.34	1.21

Number of Features	Average Transfer Time (ms)	Standard Deviation
500	19.14	5.03
1,000	35.26	3.23
2,000	71.90	9.65
3,000	108.66	10.00
4,000	146.50	13.53
5,000	198.24	24.88
6,000	245.08	26.14
7,000	284.24	24.07
8,000	326.10	35.42
9,000	346.74	36.03
10,000	414.34	63.00

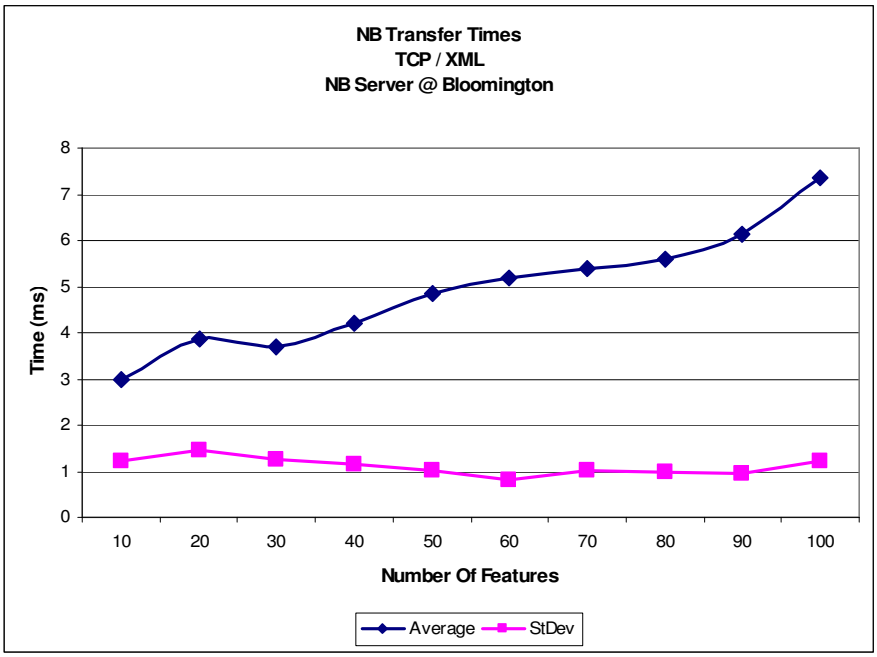


Figure 5-6 Average transfer times and standard deviations for small payloads

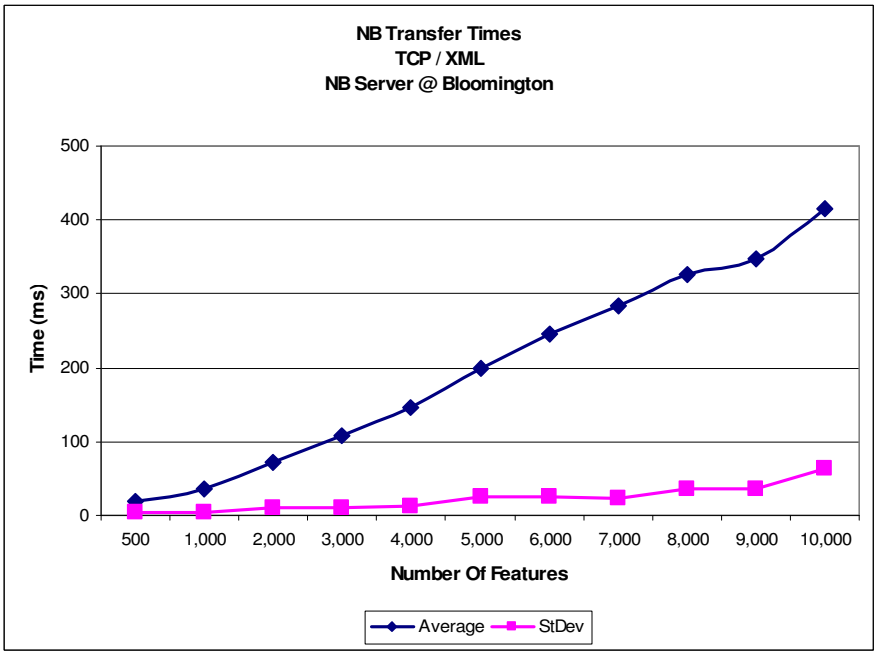


Figure 5-7 Average transfer times and standard deviations for larger payloads

The total time shown in the graphs is the accumulation of three components: publish, subscribe and transfer times. The WFS-Server reads the file into memory and gives to the publisher class; this operation is very fast and negligible in comparison with total time showed in the graph. The publish time is the amount of time it takes to publish the in-memory file bytes. Second component is the time spent on the client side to receive the published bytes and create a complete in-memory representation of the XML object. Third component is the time spent to transfer the file bytes from server to the client.

As can be seen from the graphs the timings show an expected near-linear increase, since the actual size of the published data increases linearly.

5.4.1.2 Streaming WFS Performance with Fast Infoset Integration

One of the major binary XML frameworks under development today is SUN's ASN.1 based Fast Infoset project [134]. We used Fast Infoset to test the scenario described in Figure 5-6. In addition to the network timing described above we measure encoding and decoding costs required to convert XML documents to Fast Infoset documents and vice versa.

Table 5-5, Table 5-5 and Figure 5-6 and Figure 5-6show the average timings and standard deviations for Fast Infoset encoding, decoding and network transfer times.

Table 5-5 - Average timings for Fast Infoset processing and network transfer times, small files

Number Of Features	Average Timings				Standard Deviation			
	Encoding	Network	Decoding	Total	Encode	Network	Decoding	Total
10	3.00	**6.00**	3.00	**11.00**	2.65	**1.17**	1.64	**4.12**
20	2.00	**7.00**	3.15	**12.15**	2.36	**1.65**	1.79	**4.32**
30	2.60	**6.60**	3.70	**12.90**	0.60	**0.99**	0.98	**1.37**

40	2.90	**7.10**	4.25	**14.25**	0.85	**1.48**	0.91	**1.86**
50	4.30	**7.35**	5.00	**16.65**	1.08	**1.90**	0.79	**1.63**
60	5.05	**7.65**	5.10	**17.80**	1.57	**1.18**	0.31	**1.96**
70	5.05	**7.75**	5.70	**18.50**	1.57	**0.79**	1.13	**2.09**
80	6.00	**8.65**	5.45	**20.10**	0.65	**1.50**	0.51	**1.83**
90	6.85	**8.40**	6.10	**21.35**	0.67	**0.60**	0.55	**0.99**
100	7.20	**9.20**	6.35	**22.75**	1.06	**1.85**	0.67	**1.92**

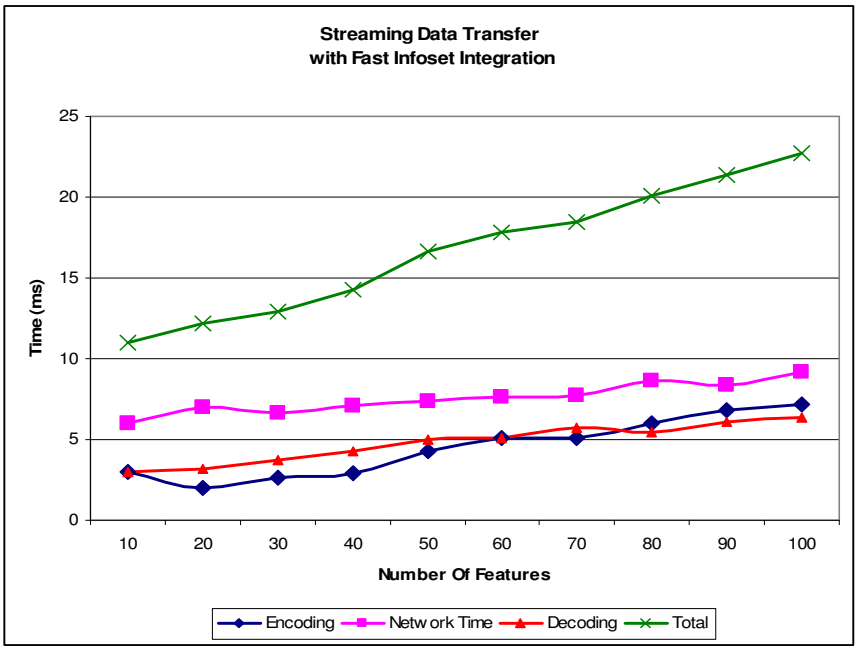


Figure 5-8 - Streaming WFS performance with Fast Infoset integration, small files

For small data sizes the network time is the dominant factor for the total time, while the encoding and the decoding processes take similar times and smaller than the transfer time.

Table 5-6 - Average timings for Fast Infoset processing and network transfer times, large files

Number Of Features	Average Timings				Standard Deviation			
	Encoding	Network	Decoding	Total	Encode	Network	Decoding	Total
500	23.68	**9.98**	20.58	**54.24**	3.98	**1.62**	2.93	**6.16**
1,000	43.56	**13.48**	38.48	**95.52**	1.51	**1.87**	2.87	**3.50**
2,000	89.26	**18.96**	75.40	**183.62**	2.59	**2.20**	1.47	**4.10**
3,000	135.94	**24.54**	108.76	**269.24**	2.85	**7.45**	2.38	**7.77**
4,000	182.34	**33.18**	149.02	**364.54**	6.26	**10.07**	10.68	**15.03**
5,000	229.24	**36.32**	178.80	**444.36**	5.67	**5.68**	8.04	**10.32**

6,000	318.52	**47.32**	217.00	**582.84**	9.13	**9.09**	11.20	**14.79**
7,000	348.54	**50.36**	272.70	**671.60**	9.13	**9.09**	11.20	**14.79**
8,000	394.34	**54.32**	308.14	**756.80**	14.08	**6.74**	8.08	**15.68**
9,000	445.76	**60.32**	345.40	**851.48**	16.00	**12.75**	6.07	**20.15**
10,000	491.34	**66.50**	378.16	**936.00**	13.46	**12.24**	2.59	**17.55**

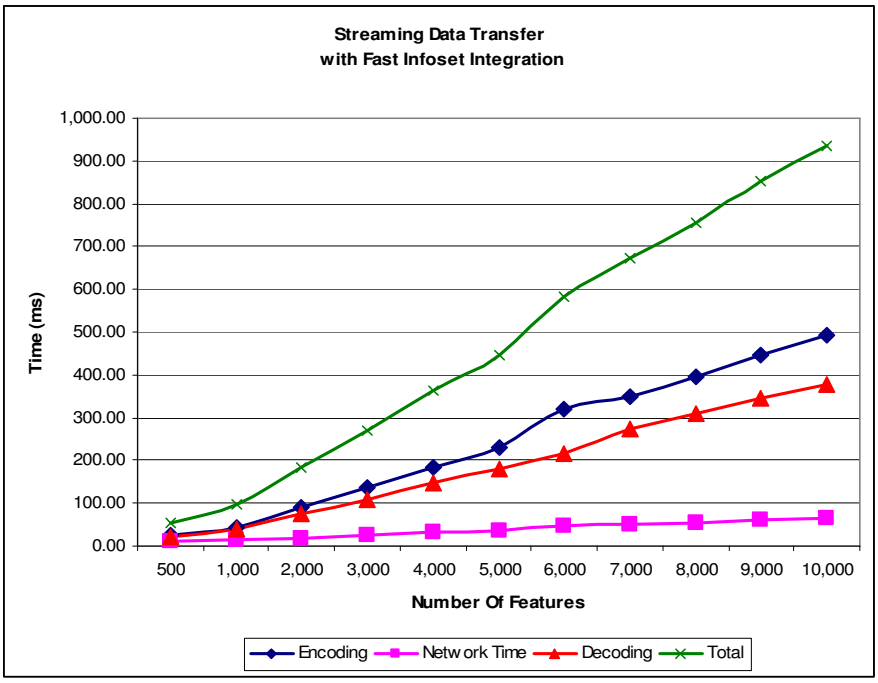


Figure 5-9 - Streaming WFS performance with Fast Infoset integration, large files

Figure 5-6 shows that as the data size grow the Fast Infoset encoding and decoding times also grow significantly larger than the transfer time which displays a small increase. The total time shows near-linear characteristic because of the linear payload size increase. The performance in this case is dominated for all data sizes by encoding the Fast Infoset documents on the server side. The network time does not have a major contribution to the total time since the actual network path is almost zero.

5.4.1.3 Streaming WFS Performance with BNUX Integration

The second binary XML framework we tested is BNUX [141], an extension of the NUX – An XML processing framework designed especially to be used in high-

throughput XML messaging middleware such as large-scale Peer-to-Peer infrastructures, message queues, publish-subscribe and matchmaking systems etc. [142].

Table 5-5 and Figure 5-6 show the test results for small and large data payloads.

Table 5-7 - Average timings for BNUX processing and network transfer times, small files

Number Of Features	Average Timings				Standard Deviation			
	Encoding	Network	Decoding	Total	Encode	Network	Decoding	Total
10	1.35	3.00	1.75	6.00	1.23	0.50	0.64	1.40
20	1.00	4.55	1.50	6.10	1.28	0.50	0.99	1.76
30	1.40	4.80	2.00	8.20	1.05	0.52	1.30	1.70
40	2.05	5.35	2.25	9.65	1.32	0.59	0.79	1.69
50	3.05	5.70	2.60	11.35	1.36	0.57	0.82	1.31
60	3.55	5.80	2.95	12.30	0.60	0.62	0.69	1.08
70	4.20	6.15	3.35	13.70	0.52	0.37	0.81	0.86
80	4.75	6.50	3.65	14.90	0.79	0.95	1.09	1.52
90	5.75	6.85	3.70	16.30	0.55	0.67	1.13	1.08
100	6.75	7.35	3.40	17.50	1.12	0.59	1.85	2.50

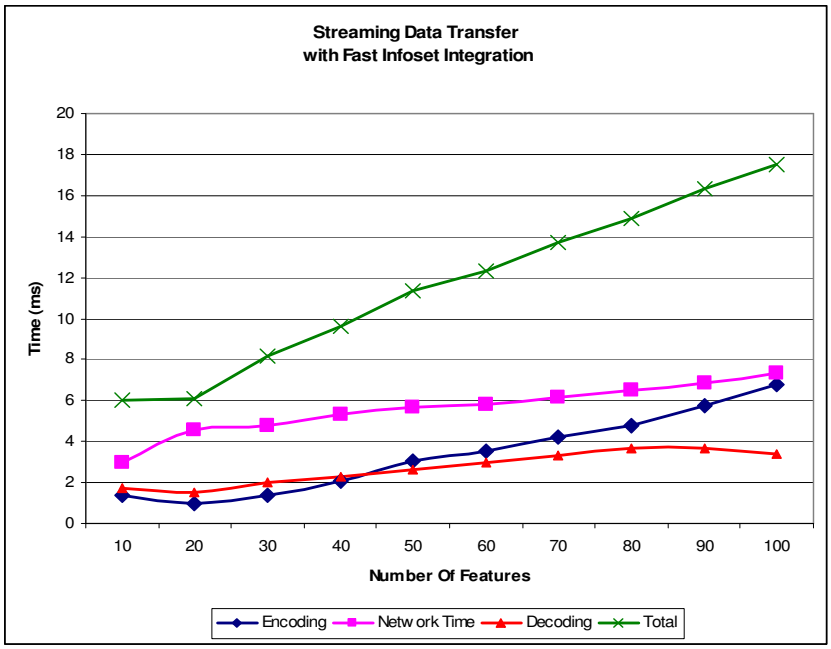


Figure 5-10 - Streaming WFS performance with BNUX integration for small payloads

As in the Fast Infoset case the dominant contributor to the total time for small data sizes is the network transfer time, while encoding and decoding steps take less time.

Table 5-8 - Average timings for BNUX processing and network transfer times, large files

Number Of Features	Average Timings				Standard Deviation			
	Encoding	Network	Decoding	Total	Encode	Network	Decoding	Total
500	27.14	9.98	11.06	48.18	6.60	2.09	7.88	14.80
1,000	49.20	13.48	16.36	79.04	2.35	1.40	6.51	6.69
2,000	100.74	18.96	34.30	154.00	7.66	2.36	15.13	18.84
3,000	152.40	24.54	55.50	232.44	3.99	8.70	26.88	29.27
4,000	199.22	33.18	70.52	302.92	8.77	17.68	24.71	29.21
5,000	250.26	36.32	122.64	409.22	6.18	10.87	37.96	47.21
6,000	296.46	47.32	184.78	528.56	8.80	26.46	66.31	68.02
7,000	347.84	50.36	156.74	554.94	20.70	9.04	60.74	68.23
8,000	395.62	54.32	308.06	758.00	9.53	58.39	86.41	75.19
9,000	446.12	60.32	381.70	888.14	12.03	95.56	92.85	134.24
10,000	501.74	66.50	370.82	939.06	18.08	47.78	84.70	86.64

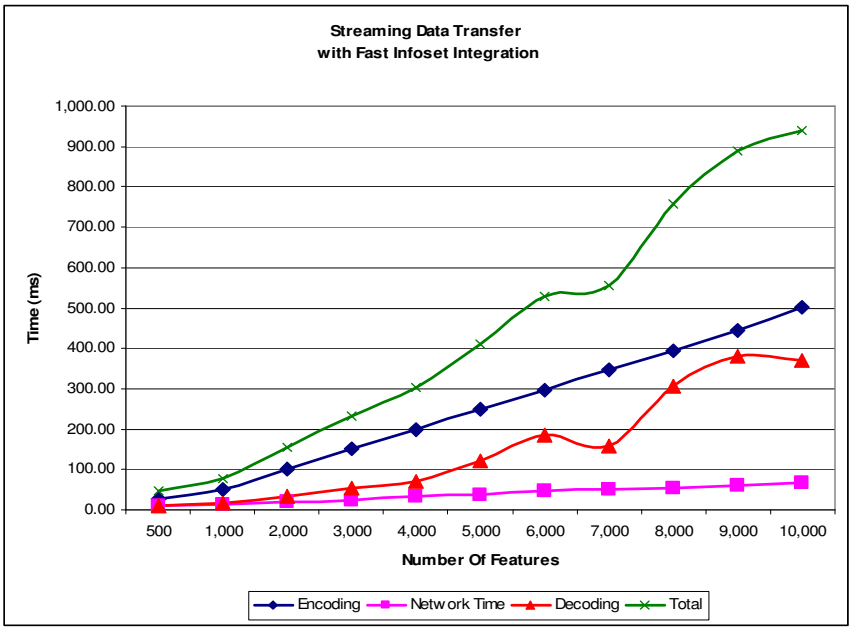


Figure 5-11 - Streaming WFS performance with BNUX integration for larger payloads

Figure 5-6 shows that the behavior of BNUX framework is similar to Fast Infoset for larger data sizes too. Here the encoding step is longer than encoding and the network transfer time. The total time is also very similar to Fast Infoset case.

5.4.1.4 Performance Comparison of Three Encodings

Here we compare the three test cases outlined above. The first case is the pure XML transport the second case showed the results with Fast Infoset integration, and the third case is the BNUX integration case.

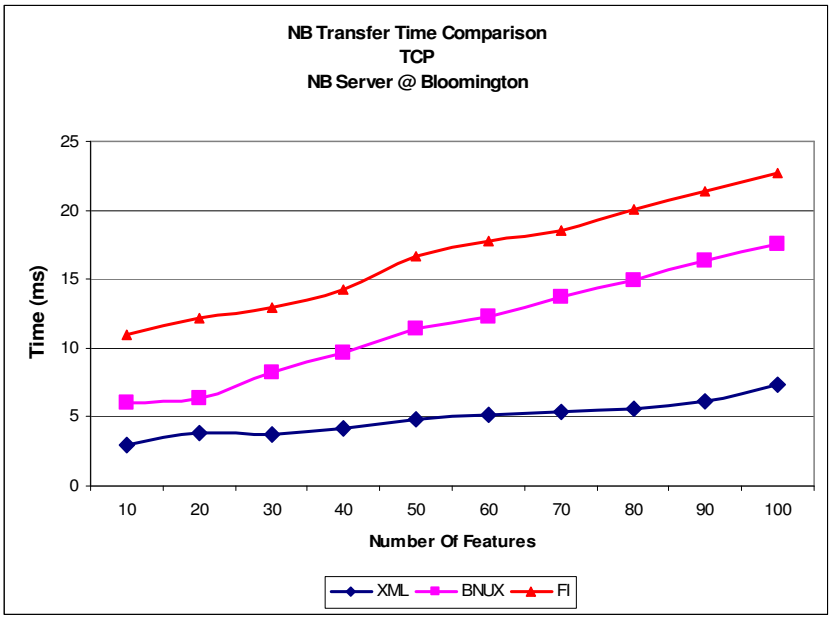


Figure 5-12 - Total processing times for different XML encodings, small files

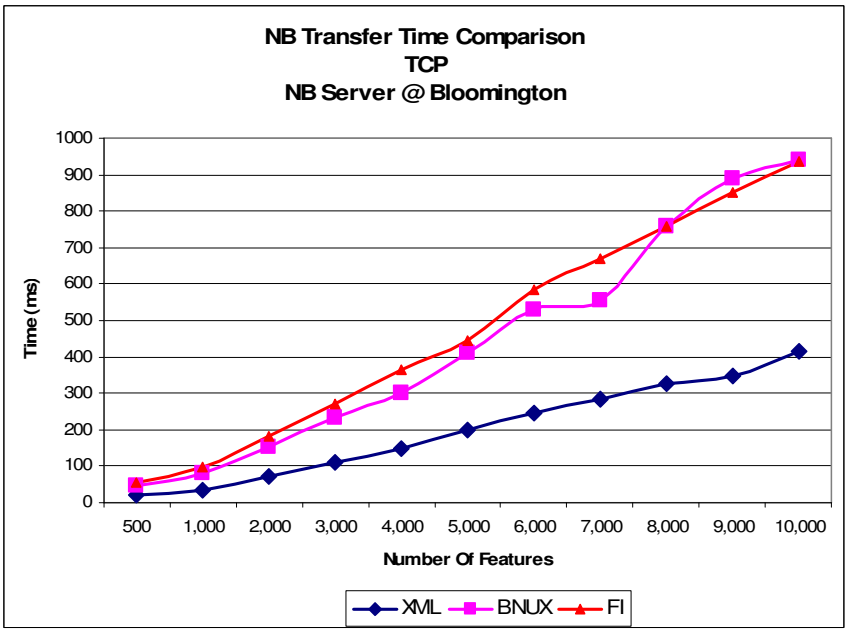


Figure 5-13 - Total processing times for different XML encodings, large files

The graphs show that because of the expensive binary encoding and decoding processes the two binary XML frameworks add significant overhead to the system, hence the textual XML transfer performs much better. This is also due to the fact that the tests are executed in the local network which causes the transfer times to be minimal.

5.4.1.5 Fast Infoset, BNUX comparison

Figure 5-6 and Figure 5-6 show the performance comparison of two binary XML frameworks in first test case for different data sizes. The graphs show that BNUX have demonstrated better performance for the smaller files while Fast Infoset has superior performance for the larger data sizes. The difference becomes more obvious after the number of features in the file are larger than 1000.

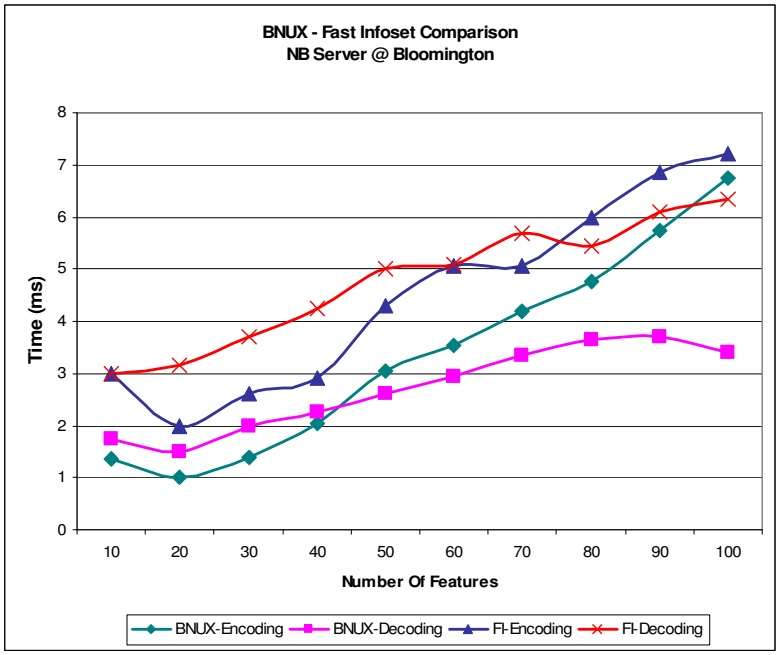


Figure 5-14 - Performance comparison of Fast Infoset and BNUX frameworks, small files

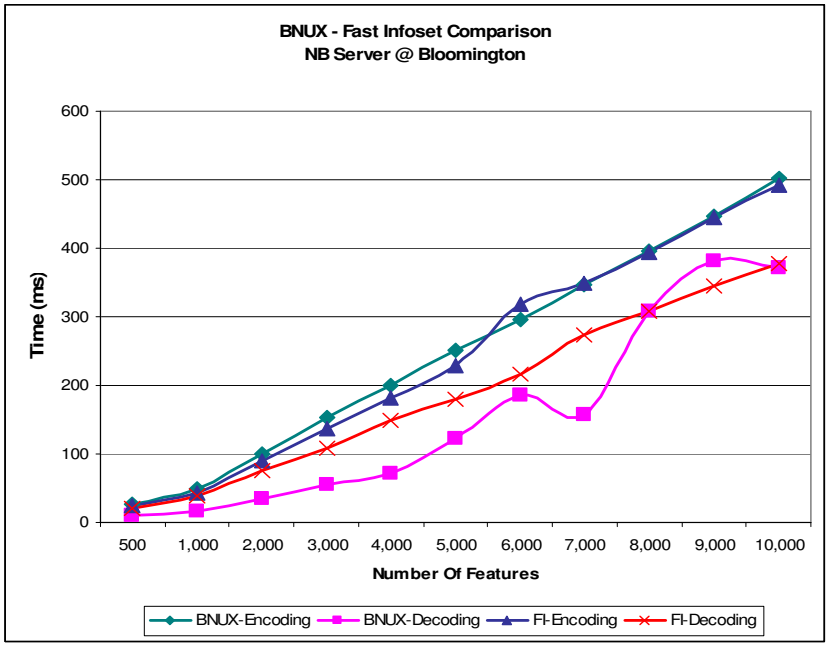


Figure 5-15 - Performance comparison of Fast Infoset and BNUX frameworks, large files

5.4.2 WAN Testing I

In this test configuration the WFS server and the WFS Client were run on gridfarm servers, while the NaradaBrokering Server run on complexity.ucs.indiana.edu at Indianapolis, Indiana.

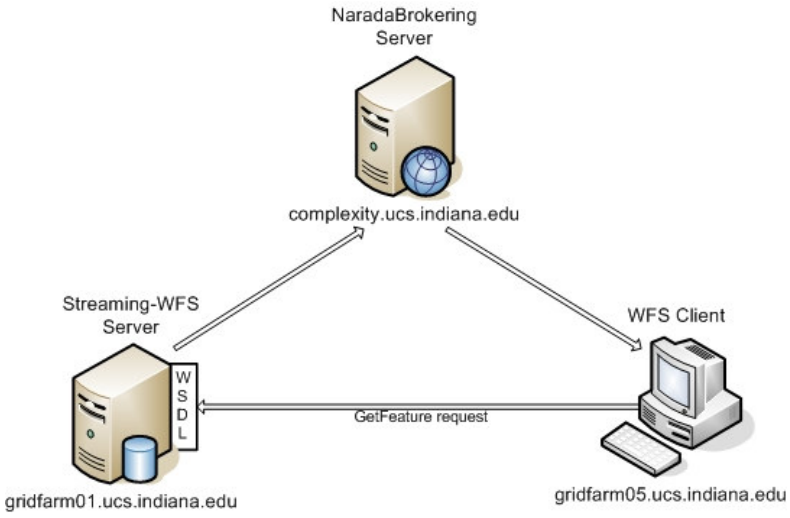


Figure 5-16 - Test Configuration for the second case; NaradaBrokering Server is in Indianapolis.

5.4.2.1 Streaming WFS Performance with Textual XML Transfer

Table 5-5, Figure 5-6 and Figure 5-6 show the average network time spent for transferring textual XML documents between WFS-Server, NaradaBrokering Server and the Client, and the standard deviations.

Table 5-9 – Average XML transfer times

Number of Features	Average Transfer Time (ms)	Standard Deviation
10	8.65	2.06
20	12.65	1.84
30	12.85	1.69
40	13.05	1.79
50	15.00	1.00
60	15.25	0.91
70	17.10	1.41
80	18.85	1.95
90	19.05	1.32
100	20.00	1.00

Number of Features	Average Transfer Time (ms)	Standard Deviation
500	47.20	13.89
1,000	94.40	25.81
2,000	197.65	31.58
3,000	300.35	25.46
4,000	381.55	28.62
5,000	473.10	26.83
6,000	546.35	22.94
7,000	671.70	41.14
8,000	747.60	39.44
9,000	793.15	44.10
10,000	857.85	46.32

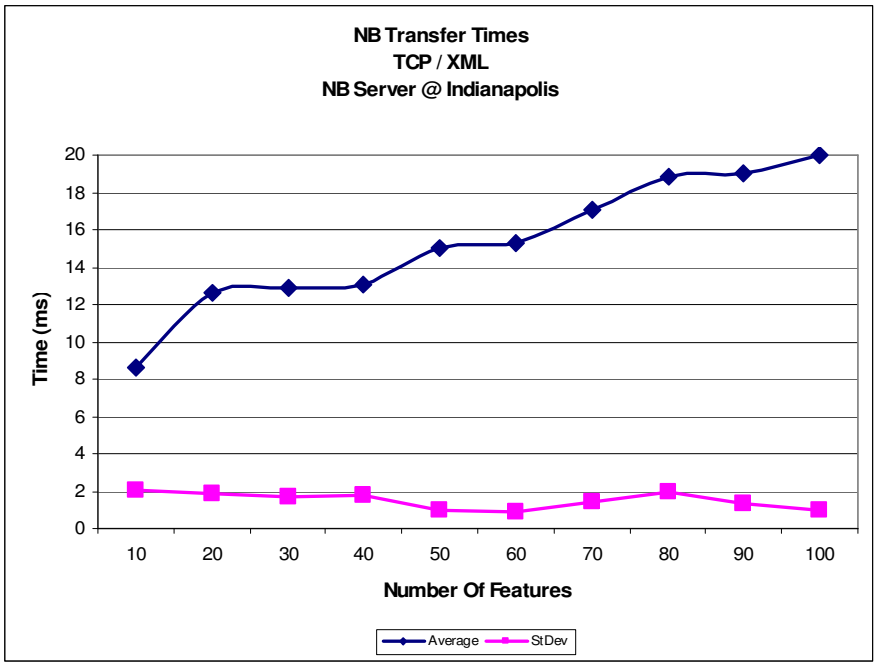


Figure 5-17 - Streaming WFS timings for XML data exchange, small files

Because the distance between publisher, subscriber and the broker server increased from the first test case where all of the components were run in the local network, the average transfer time increases to between 8ms and 20 ms whereas in the previous test case it was between 3ms and 7.5ms.

We see the network distance effect more clearly in Figure 5-6; here the transfer times of the larger data files doubles in comparison to the local network test case.

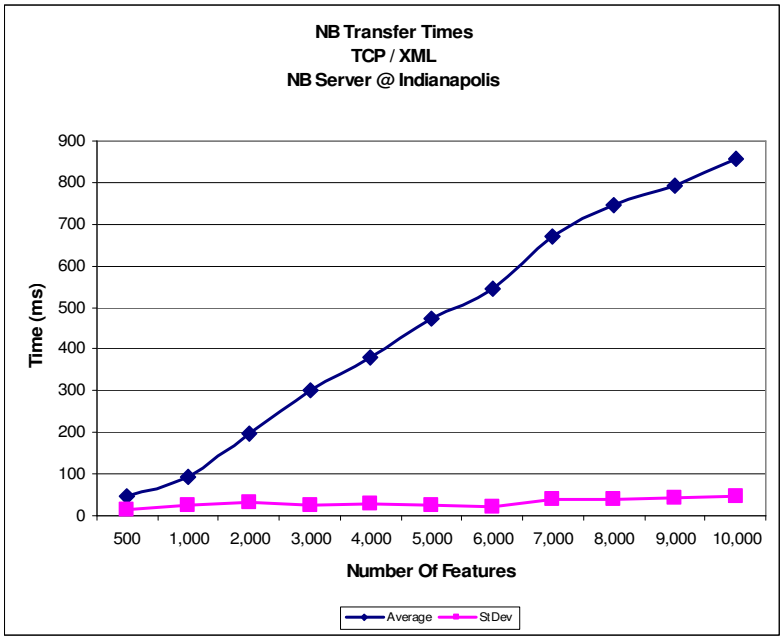


Figure 5-18 - Streaming WFS timings for XML data exchange, large files

5.4.2.2 Streaming WFS Performance with Fast Infoset Integration

Table 5-5 and Figure 5-6 show the system performance with Fast Infoset encoded document transfer.

Table 5-10 - Average timings for Fast Infoset processing and network transfer times, small files

Number Of Features	Average Timings				Standard Deviation			
	Encoding	Network	Decoding	Total	Encode	Network	Decoding	Total
10	4.85	9.75	3.95	18.55	2.28	1.02	1.15	2.21
20	1.95	12.21	3.45	17.58	2.68	2.51	1.73	4.96
30	2.50	12.05	3.40	17.95	0.51	0.89	0.50	1.05
40	2.60	13.40	4.75	20.75	0.50	1.64	1.41	1.80
50	4.45	14.00	5.20	23.65	0.60	1.72	0.83	1.53
60	6.20	14.10	5.50	25.80	3.02	0.79	0.76	2.91
70	4.70	17.25	5.85	27.80	0.66	2.24	1.18	3.00
80	5.70	16.35	5.75	27.80	0.80	1.23	1.02	1.40
90	6.25	18.15	6.05	30.45	0.72	2.32	0.51	2.06
100	7.25	17.50	6.75	31.50	0.85	1.24	0.85	1.70

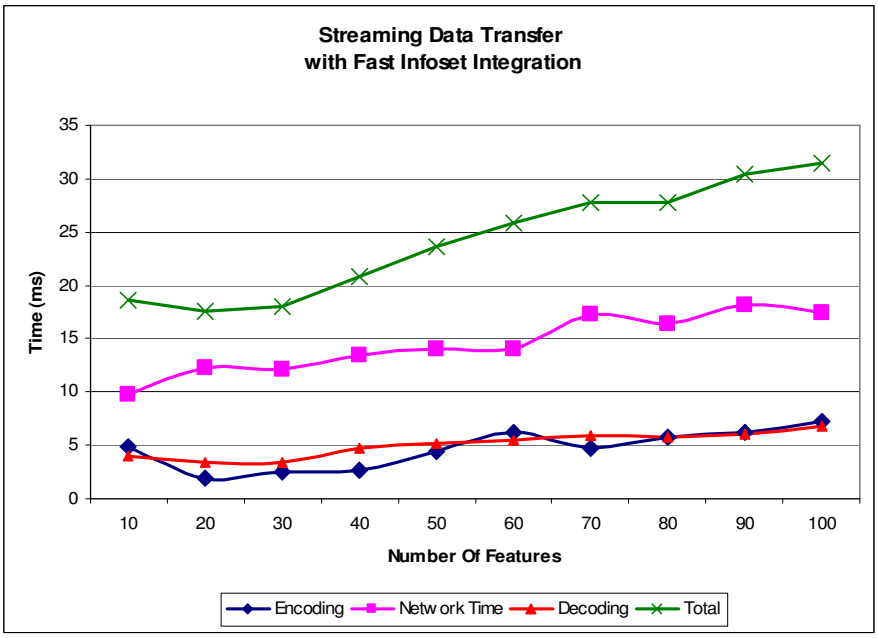


Figure 5-19 - Streaming WFS performance with Fast Infoset integration, small files

As the Figure 5-6 shows the use of Fast Infoset encoding introduces a 10ms to 15ms overhead and since the data transfer time is relatively small this causes the total transfer time to be larger than the textual XML transfer. However for the small files the encoding and decoding times are always smaller than the network time.

Table 5-11 - Average timings for Fast Infoset processing and network transfer times, large files

Number Of Features	Average Timings				Standard Deviation			
	Encoding	Network	Decoding	Total	Encode	Network	Decoding	Total
500	25.35	29.50	21.35	76.20	2.70	1.91	2.98	4.26
1,000	50.45	41.00	40.55	132.00	2.09	3.60	1.15	3.61
2,000	102.90	83.50	77.35	263.75	4.41	21.89	1.69	20.24
3,000	214.90	114.70	116.95	446.55	54.70	26.30	11.17	61.90
4,000	215.30	136.80	160.30	512.40	34.02	24.97	6.64	50.23
5,000	264.75	176.90	192.55	634.20	11.68	35.29	4.77	37.64
6,000	314.10	209.75	227.15	751.00	12.91	32.22	6.05	31.19
7,000	399.45	238.70	288.60	926.75	91.21	33.07	17.63	104.96
8,000	476.90	297.20	349.65	1,123.75	133.04	39.53	25.64	121.70
9,000	554.40	373.85	371.10	1,299.35	116.18	35.94	18.53	123.78
10,000	548.20	384.50	401.75	1,334.45	39.07	32.05	19.55	47.18

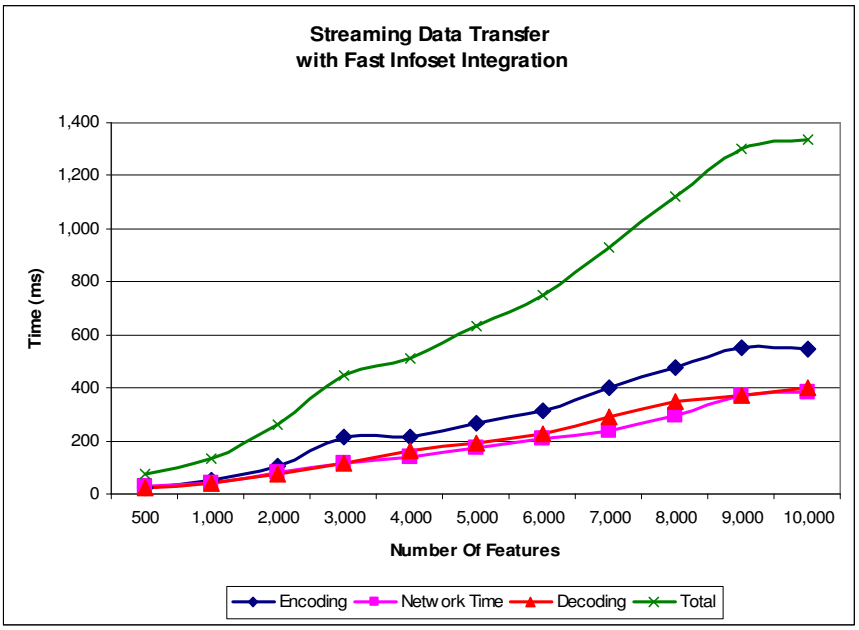


Figure 5-20 - Streaming WFS performance with Fast Infoset integration, larger files

Figure 5-6 shows an interesting behavior of Fast Infoset framework, where for large files the encoding takes more time than decoding. Overall the transfer time is almost always smaller than the total encoding and decoding overhead and it is significantly smaller than the textual XML transfer time. However because of the encoding and decoding overhead contributions the total performance is worse than the textual XML transport.

5.4.2.3 Streaming WFS Performance with BNUX Integration

Table 5-5, Table 5-5, Figure 5-6 and Figure 5-6 show the system performance with BNUX encoded document transfer.

Table 5-12 - Average timings for BNUX processing and network transfer times, small files

Number Of Features	Average Timings				Standard Deviation			
	Encoding	Network	Decoding	Total	Encode	Network	Decoding	Total
10	2.70	5.30	1.65	9.65	1.49	0.47	1.04	2.62

20	1.40	8.20	1.15	10.75	2.06	0.89	0.67	2.45
30	1.65	9.45	1.55	12.65	1.35	0.83	0.76	1.73
40	2.35	10.35	2.15	14.85	0.49	1.53	0.93	1.95
50	3.70	10.70	2.80	17.20	1.13	0.80	0.89	1.15
60	4.40	11.45	2.75	18.60	1.39	0.51	1.12	1.90
70	4.60	13.60	3.45	21.65	0.94	0.75	0.94	1.35
80	6.15	13.50	4.05	23.70	0.75	1.00	1.61	2.15
90	6.85	14.40	3.55	24.80	0.59	0.82	1.05	1.51
100	7.45	14.55	3.60	25.60	0.60	1.36	1.43	1.64

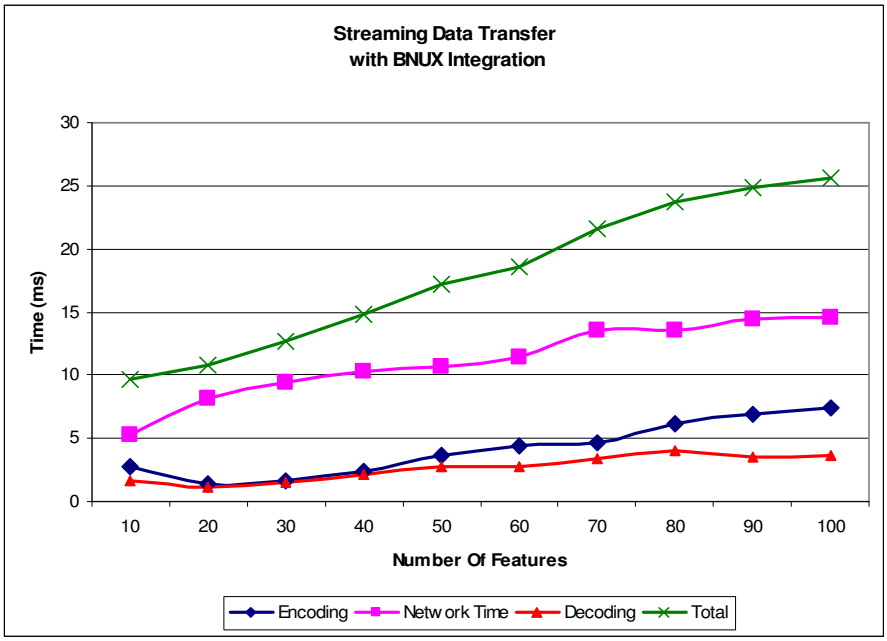


Figure 5-21 - Streaming WFS performance with BNUX integration, small files

In this test the performance of the BNUX integration is similar to the previous test. Encoding and decoding overheads add to the total time which causes the total time to be longer than the textual XML transfer.

Table 5-13 - Average timings for BNUX processing and network transfer times, large files

Number Of Features	Average Timings				Standard Deviation			
	Encoding	Network	Decoding	Total	Encode	Network	Decoding	Total
500	29.05	20.20	12.35	61.60	5.02	1.24	3.17	6.11
1,000	52.50	30.85	21.75	105.10	1.47	1.23	8.53	9.10

2,000	107.05	59.55	47.40	214.00	5.06	18.67	17.46	24.49
3,000	162.85	85.80	77.75	326.40	4.53	3.90	49.36	49.98
4,000	216.25	107.85	89.10	413.20	12.34	13.10	27.12	29.09
5,000	265.00	126.70	127.75	519.45	4.14	5.28	24.35	28.54
6,000	321.30	159.90	191.90	673.10	14.43	24.58	72.29	78.77
7,000	365.55	185.30	204.30	755.15	21.22	29.40	57.54	72.60
8,000	392.00	293.25	337.90	1,023.15	13.43	67.53	114.85	99.41
9,000	436.65	383.35	415.95	1,235.95	19.01	92.64	144.59	196.75
10,000	488.70	463.15	385.00	1,336.85	9.09	59.41	100.98	106.97

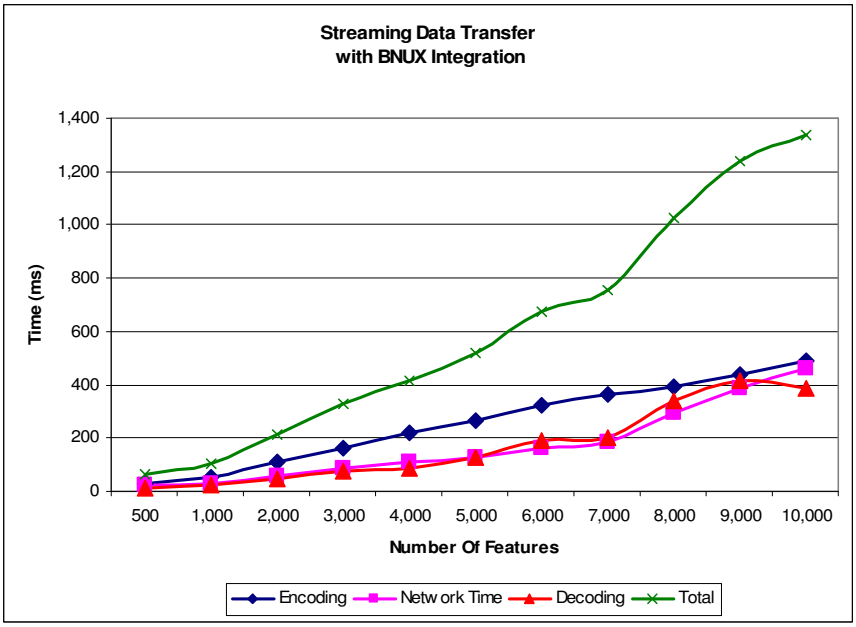


Figure 5-22 - Streaming WFS performance with BNUX integration, large files

Figure 5-6 shows that the two binary XML frameworks demonstrate similar performances under same conditions. The BNUX document transfer times do not show major increase between Bloomington and Indianapolis servers and the encoding time dominates the total time.

5.4.2.4 Performance Comparison of Three Encodings

Figure 5-6 and Figure 5-6 compare the system performance for three document encodings in this test case, where we run the NaradaBrokering server in Indianapolis.

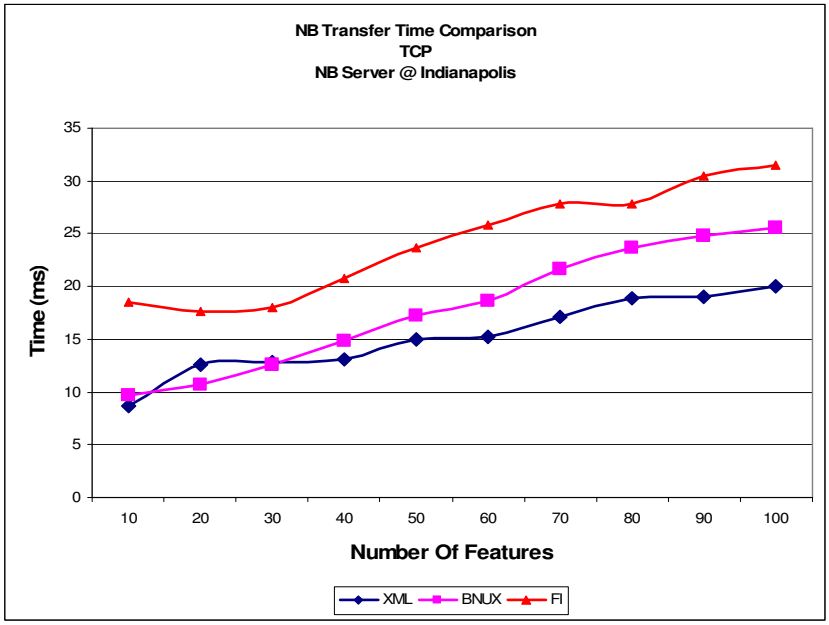


Figure 5-23 - Total processing times for different XML encodings, small files

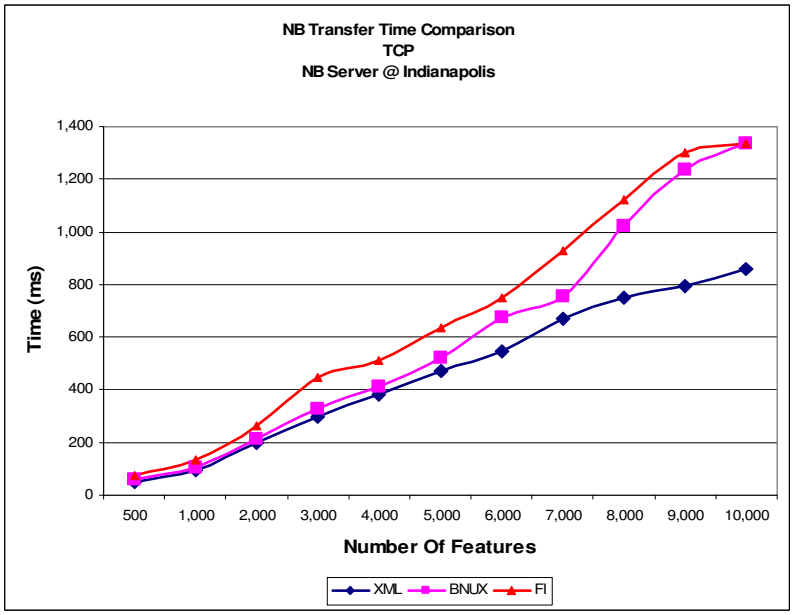


Figure 5-24 - Total processing times for different XML encodings, large files

For the smaller data sizes we see that although the XML and BNUX transfer times are closer in the first steps the difference grows for the rest of the files and overall the textual XML transfer performance is better.

For the second group of the test files the BNUX and XML file transfer times are very similar up to 4000 features, however again in this case the binary XML conversion overheads cause the integration of these frameworks to increase the total time.

5.4.3 WAN Testing II

In this test configuration the WFS server and the WFS Client run on gridfarm servers, while the NaradaBrokering Server run on one of the servers provided SIO (Scripps Institution of Oceanography) at La Jolla, California.

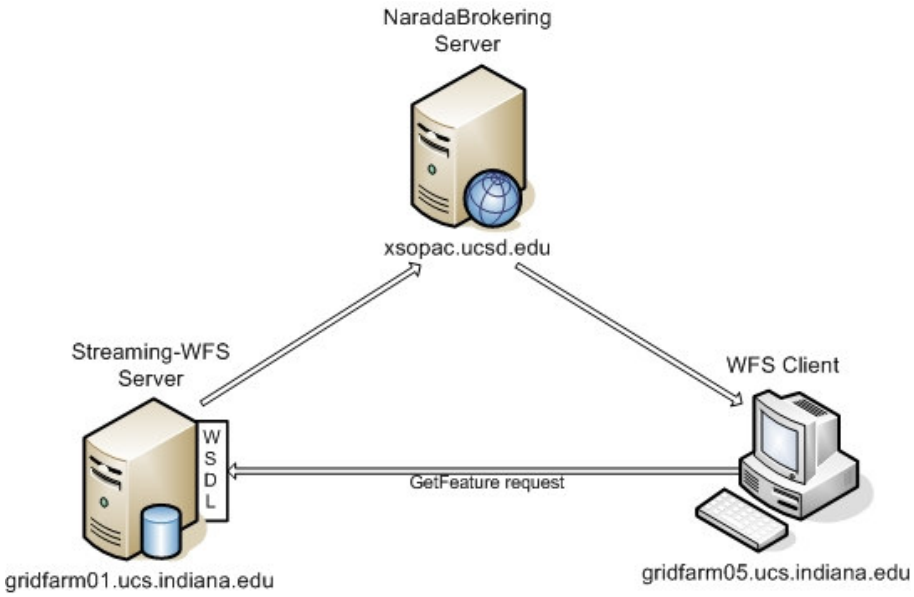


Figure 5-25 - Test Configuration for the third case; NaradaBrokering Server is run in La Jolla, California.

5.4.3.1 Streaming WFS Performance with Textual XML Transfer

Figure 5-6 and Figure 5-6 shows the total network time spent for transferring textual XML documents between WFS-Server, NaradaBrokering Server and the Client.

As the following table and the graphs show that the effect of the distance between the NaradaBrokering server and the clients is significant. The textual XML transfer times increase by tenfold for both the small and large file transfers.

Table 5-14 - Average XML transfer times, small files

Number of Features	Average Transfer Time (ms)	Standard Deviation
10	183.30	0.66
20	186.30	2.13
30	245.85	1.69
40	246.65	1.23
50	252.55	12.70
60	257.70	18.30
70	308.25	1.29
80	368.25	2.49
90	369.45	0.94
100	374.95	12.82

Number of Features	Average Transfer Time (ms)	Standard Deviation
500	957.60	167.01
1,000	2,204.70	103.23
2,000	3,787.85	47.07
3,000	5,554.10	40.53
4,000	7,291.05	43.72
5,000	9,066.45	39.19
6,000	10,678.10	58.11
7,000	12,487.70	153.52
8,000	14,317.35	167.96
9,000	15,940.15	238.70
10,000	17,572.35	181.22

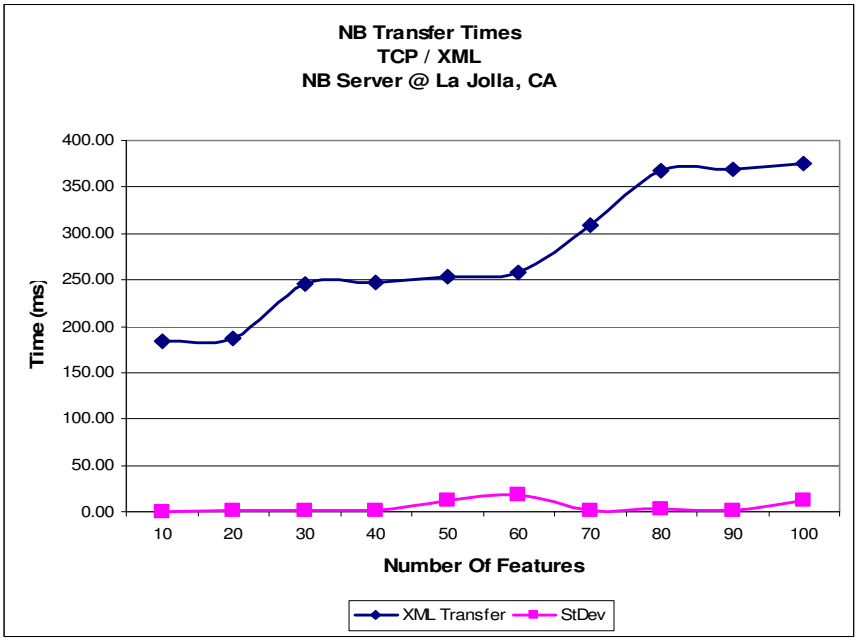


Figure 5-26 - Streaming WFS timings for XML data exchange, small files

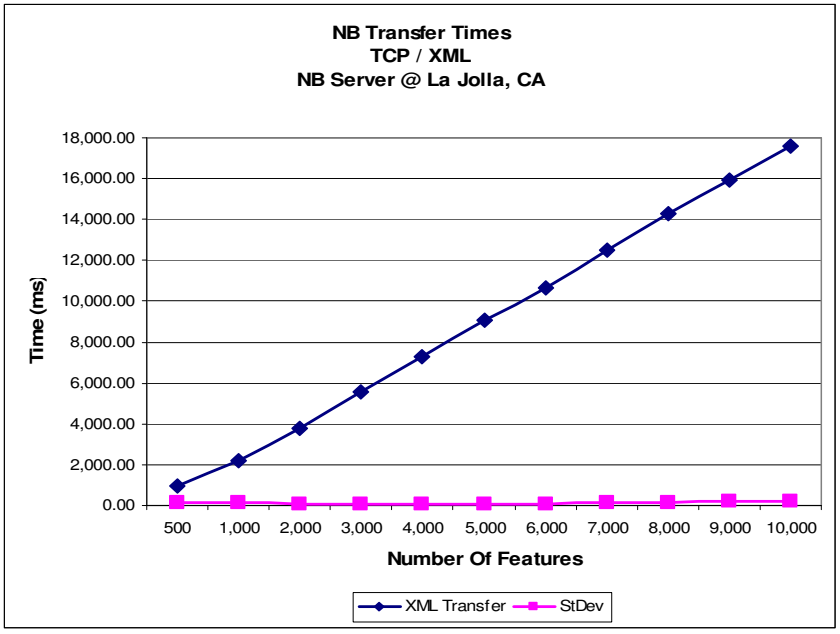


Figure 5-27 - Streaming WFS timings for XML data exchange, large files

5.4.3.2 Streaming WFS Performance with Fast Infoset Integration

As the Table 5-5, Table 5-5 and Figure 5-6 and Figure 5-6 show that the Fast Infoset integration significantly reduces the total transfer time.

Table 5-15 - Average timings for Fast Infoset processing and network transfer times, small files

Number Of Features	Average Timings				Standard Deviation			
	Encoding	Network	Decoding	Total	Encode	Network	Decoding	Total
10	8	128	5	141	7	3	2	10
20	3	187	3	193	2	1	1	3
30	3	188	3	194	0	2	0	2
40	3	188	5	195	1	1	1	2
50	4	244	5	253	1	13	1	13
60	5	251	5	261	1	13	1	13
70	5	251	5	262	1	13	0	13
80	6	248	6	260	1	0	1	1
90	7	248	6	261	1	1	0	1
100	23	221	21	266	15	19	15	48

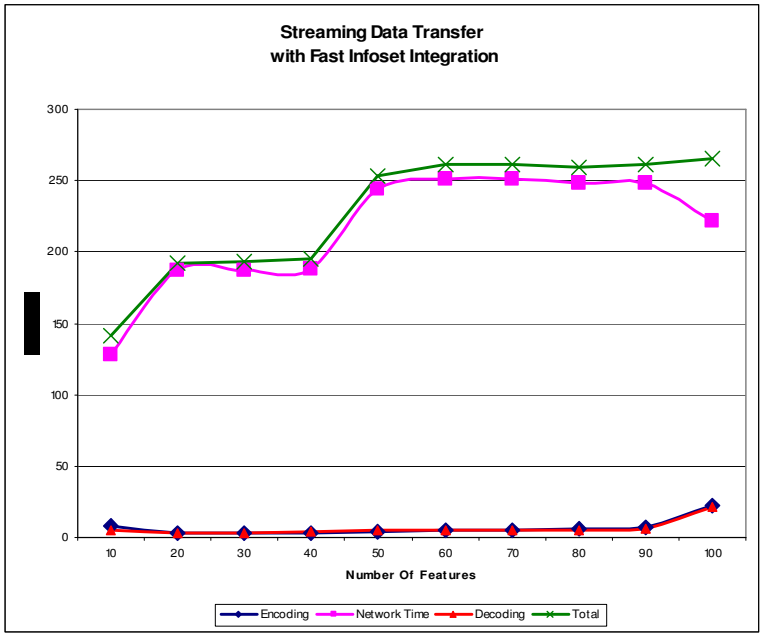


Figure 5-28 - Streaming WFS performance with Fast Infoset integration, small files

The total network time for small files shows that we gain significantly by using Fast Infoset encoding. This is due to the fact that the binary encoding and decoding overheads are ignorable when compared to the actual data transfer times. And since the size of the data is reduced because of the binary conversion the total transfer time for small files changes between 141-266ms whereas it was between 183-374ms in textual XML transfer. The average gain introduced by Fast Infoset integration is about 50ms.

Table 5-16 - Average timings for Fast Infoset processing and network transfer times, large files

Number Of Features	Average Timings				Standard Deviation			
	Encoding	Network	Decoding	Total	Encode	Network	Decoding	Total
500	35	317	31	383	7	2	10	17
1,000	52	477	40	568	5	68	1	70
2,000	103	955	78	1,136	8	54	2	56
3,000	152	1,390	112	1,653	5	66	4	65
4,000	209	1,832	150	2,191	8	68	3	72
5,000	261	2,380	190	2,831	10	171	7	176
6,000	311	2,698	224	3,234	24	68	17	74
7,000	370	3,346	268	3,984	21	52	8	53
8,000	419	3,794	322	4,535	16	39	21	46

9,000	484	4,206	363	5,053	31	21	20	38
10,000	536	4,573	392	5,501	28	47	21	45

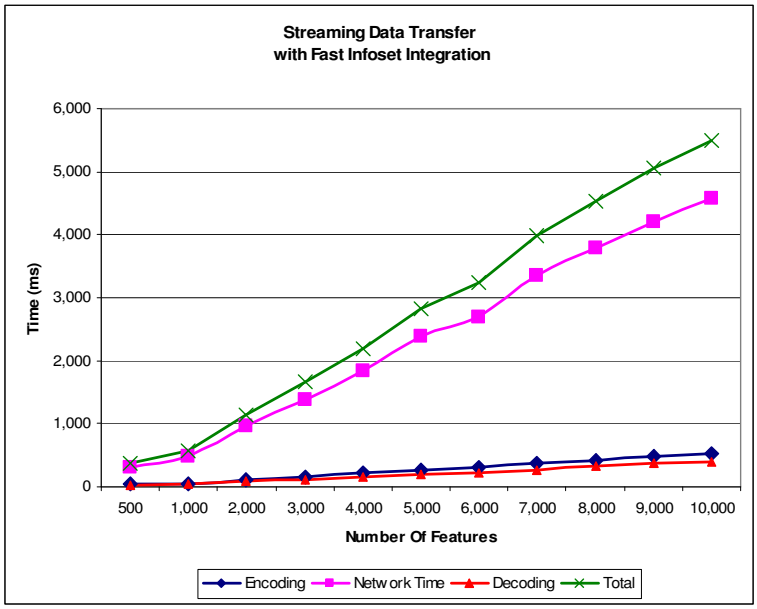


Figure 5-29 - Streaming WFS performance with Fast Infoset integration, large files

Figure 5-6shows that the average data transfer time for large files is much smaller than the textual XML transfer times. The encoding and decoding costs stay at a low level compared to the network time especially for the large data sizes. The average network time changes between 383ms-5501ms, whereas it was between 957ms and 17000ms in textual XML case. The average time gain for the whole test is around 6600ms.

5.4.3.3 Streaming WFS Performance with BNUX Integration

Following tables and the figures show that the results of BNUX binary encoding integration are very similar to the Fast Infoset case discussed above. Overall the reduction obtained by the binary conversion of the textual XML results in significant transfer time gains.

Table 5-17 - Average timings for BNUX processing and network transfer times, small files

Number Of Features	Average Timings				Standard Deviation			
	Encoding	Network	Decoding	Total	Encode	Network	Decoding	Total
10	1.90	129.00	2.65	133.55	7.73	14.44	1.69	20.57
20	1.00	185.85	1.35	186.85	1.39	1.35	1.14	1.81
30	0.70	189.40	1.45	191.55	1.26	13.17	0.51	12.97
40	1.65	195.80	2.45	199.90	1.31	28.79	1.67	30.85
50	2.45	246.60	2.50	251.55	1.15	0.68	0.61	0.94
60	2.75	249.65	3.20	255.60	0.64	13.04	1.06	13.09
70	3.60	261.85	3.25	268.70	1.10	32.71	1.16	32.43
80	4.25	254.05	3.25	261.55	0.72	18.47	0.91	18.33
90	5.10	260.35	3.60	269.05	0.91	24.45	0.88	24.24
100	23.00	251.65	9.75	284.40	30.89	43.44	11.45	77.63

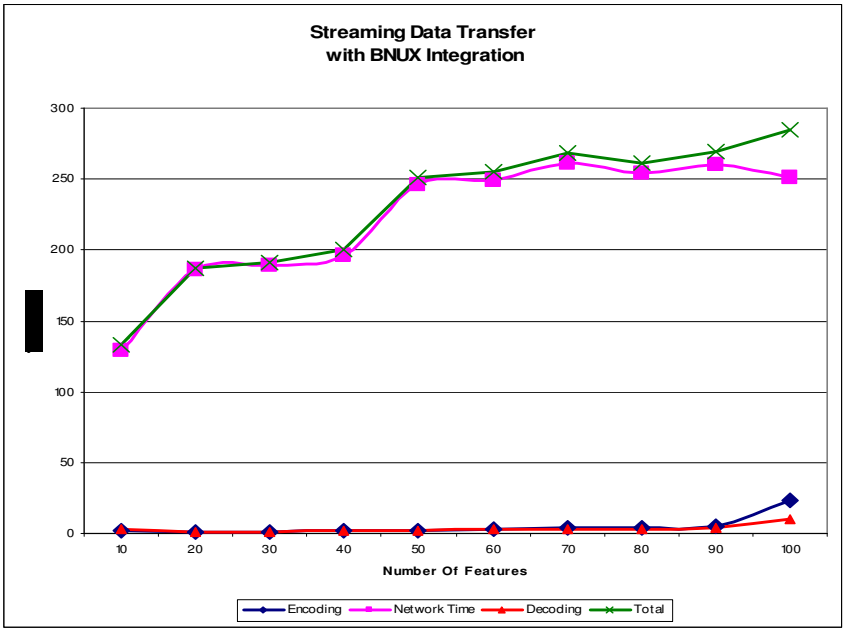


Figure 5-30 - Streaming WFS performance with BNUX integration, small files

This figure shows that the overhead caused by the encoding and decoding to and from the binary format are ignorable, the total time is almost always equal to the transfer time. So the use of BNUX framework helps reduce the data size and transfer times. The average gain for the whole test is 49ms.

Table 5-18 - Average timings for BNUX processing and network transfer times, small files

Number Of Features	Average Timings				Standard Deviation			
	Encoding	Network	Decoding	Total	Encode	Network	Decoding	Total
500	24.80	315.00	10.76	350.56	3.92	11.92	6.53	15.30
1,000	47.86	595.40	17.12	660.38	2.18	70.90	6.34	71.23
2,000	95.14	1,065.18	36.04	1,196.36	3.14	102.97	14.64	106.92
3,000	146.10	1,612.98	60.00	1,819.08	3.70	111.07	27.00	109.84
4,000	189.88	1,960.34	74.66	2,224.88	8.20	54.72	25.14	63.90
5,000	241.16	2,516.32	123.48	2,880.96	8.80	138.06	40.97	137.93
6,000	284.06	3,349.54	184.22	3,817.82	6.07	142.25	73.76	150.70
7,000	328.05	3,509.85	168.65	4,006.55	7.47	27.91	106.49	112.37
8,000	381.30	4,137.25	330.45	4,849.00	18.45	94.50	110.76	117.93
9,000	418.80	4,792.15	402.40	5,613.35	7.80	89.99	124.46	158.84
10,000	480.95	5,183.70	532.85	6,197.50	26.75	63.82	161.36	203.32

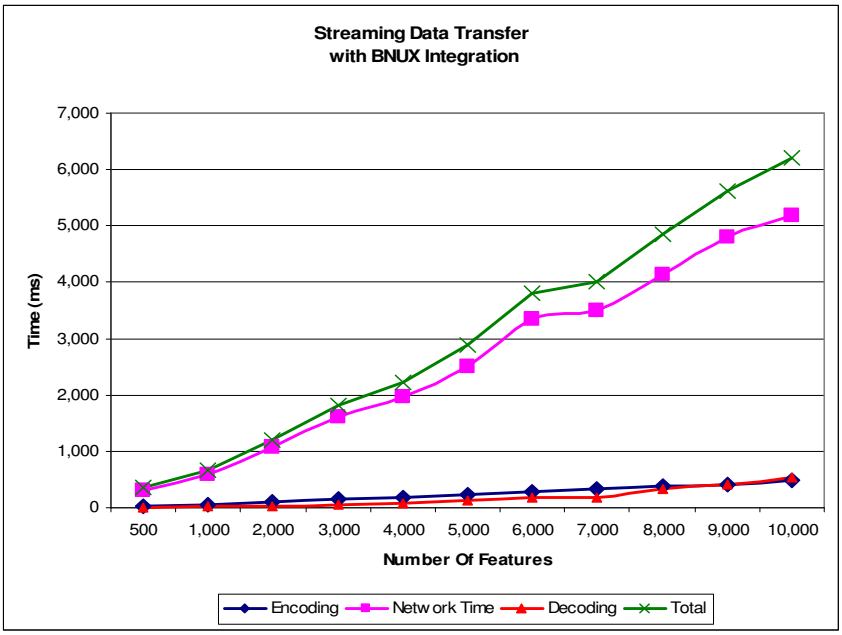


Figure 5-31 - Streaming WFS performance with BNUX integration, large files

The test results for larger data file transfer with BNUX encoding depicted in figure 31 shows the overall system performance gain in comparison to the textual XML transfer. Since the encoding and decoding overheads are ignorable, the performance is determined by the network time. The average performance improvement from the textual XML transfer is 6879ms.

5.4.3.4 Performance Comparison of Three Encodings

Now we compare the results of three test cases discussed above. As it can be seen from Figure 5-6, the XML transfer takes more time for most of the files, and almost equal time for 3 files. The binary encodings help reduce the data transfer time at a lower level.

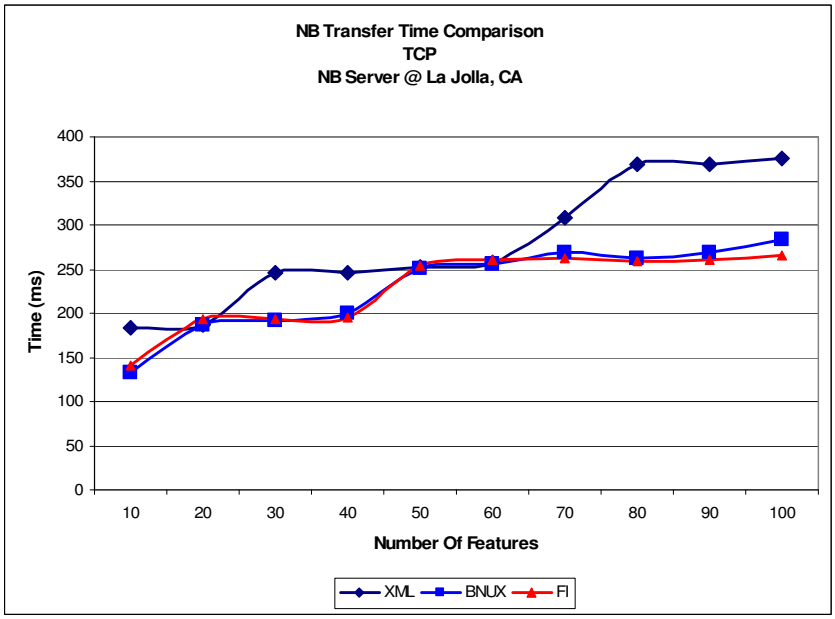


Figure 5-32 - Total processing times for different XML encodings, small files

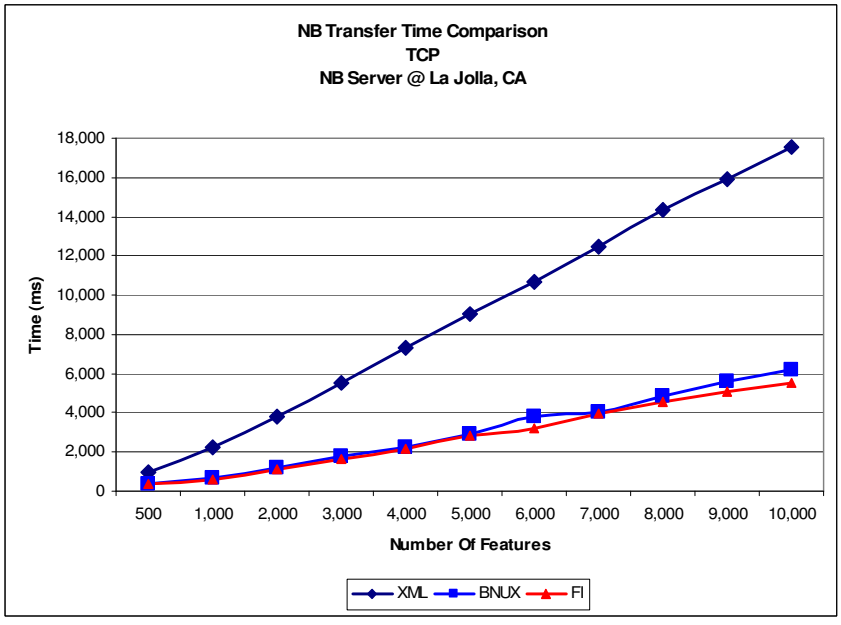


Figure 5-33 - Total processing times for different XML encodings, large files

The real performance improvement is seen in Figure 5-33; while the XML transfer times follow a linear steep increase, because of the linear increase in the payload sizes, the binary encoding keeps the increase rate at a lower level. So for the largest test file, which contains 10000 features Web Feature Service with binary XML integration requires around 6000ms, whereas the same number of features with textual XML encoding takes about 18000ms.

5.5 Summary

In this chapter we have discussed a scenario for testing the performance of the most important component in our GIS Data Grid architecture, the Streaming Web Feature Service. We show the performance of this service with and without using binary XML frameworks. The results show that there is not much gain with using these frameworks for the local network, and probably for short network distances. However the performance gain for the long distance data transfer is significant, over 60% with both BNUX and Fast Infoset integration cases. The results also imply that although the Binary XML frameworks can be used for better transfer times and bandwidth they should only be used when there is a performance gain. To make this decision a decision module might be implemented. This module can be designed based on Case Based Reasoning approach which can pick up best available binary encoding for similar queries and network distances that was recorded in the past.

Chapter 6

Real-Time GIS Data Grid

6.1 Introduction

Recent research has discussed that the sensors are changing the way we acquire data about various entities. Recent advancements in sensor technologies such as micro-circuitry, nano-technology and low-power electronics allowed sensors to be deployed in a wide variety of environments [21, 22, 143-146]. The trend in this field shows us that in the near future we will see thousands of sensor nodes being deployed either individually or as part of sensor networks in a large variety of application domains. Environmental monitoring, air pollution and water quality measurements, detection of the seismic events and understanding the long-term motions of the Earth crust are only a few areas where extent of the deployment of sensor networks can easily be seen. Extensive use of sensing devices and deployment of the networks of sensors that can communicate with each other to achieve a larger sensing task will fundamentally change information gathering and processing [147].

Our work on developing a common Grid infrastructure for Geographic Information Systems has led us to the conclusion that this new type of data source is capable of producing very large amounts of observational data which potentially may help us obtain detailed knowledge about the environment we live in. Although the most common type of geographic data are kept in various types of data stores, the real-time sensor measurements are becoming the dominant type of data sources with the capacity to produce tremendous amount of measurements, which might be more than the traditional systems can handle in normal operation. For instance Southern California Integrated GPS Network (SCIGN) [148] has deployed 250 continuously-operating GPS stations in Southern California whereas several hundred non-real time stations are operating elsewhere in the United States. The GPS Earth Observation Network System or GEONET in Japan consists of an array of 1200 GPS stations that cover the entire country with an average spacing of about 20 km [149]. These networks are capable of producing terabytes of measurements per year.

Table 6-1 shows approximate amount of data produced by the Southern California Integrated GPS Network (SCIGN) real-time GPS stations. The observations obtained from a proxy server are encoded in a binary format called RYO. The table shows the increase in size for different encodings of the same observations.

Table 6-1 –Approximate amount of data produced by SCIGN GPS Networks

		Message Format		
	Time	**RYO**	**ASCII**	**GML**
SDCRTN GPS Network (9 Stations)	1 second	1.5KB	4.03KB	48.7KB
	1 hour	5.31MB	14.18MB	171.31MB
	1 day	127.44MB	340.38MB	4.01GB
	1 month	3.8GB	9.97GB	123.3GB
	1 year	45.8GB	119.67GB	1.41TB
Entire SCIGN Network (250 stations)	1year	1.23TB	16.18TB	160TB

However rapid proliferation of sensors presents unique challenges different than the traditional computer network problems. Several studies have discussed the technological aspects of the challenges with the sensor devices, such as power consumption, wireless communication problems, autonomous operation, adaptability to the environmental conditions, load balancing etc [150] [151] [21]. In this chapter we describe a Service Oriented Architecture termed SensorGrid to support real-time information gathering and processing from sensors. The architecture supports real-time integration of sensors with scientific applications such as simulation, visualization or data mining software.

Scientific applications that require processing of huge data sets are increasing in number with the evolution of computing resources, network bandwidth, and storage capabilities etc. At the same time some of the applications are being designed to run on real-time data to provide near-real time results; such applications are gaining ground in systems like Crisis Management [152] or Early Warning Systems [153] [154] because they allow authorities to take action on time. Earthquake data assimilation tools are good examples of this group since they use data from Seismic or GPS sensors. However, most

of these tools currently consume data from repositories and they do not have access to real-time data due to several reasons.

6.2 Real-Time Data Grid Components

In this chapter we present a common Grid Infrastructure for coupling real-time data sources such as sensors to distributed science tools such as distributed data analysis or simulation applications. The Grid is built around streams created and consumed by filters and managed by a powerful Grid messaging substrate.

This architecture consists of three major components in addition to the actual sensor nodes; individual filters that process the real-time data streams, information services which provide the metadata about the filters or filter chains, and Grid messaging system, which provides supports in areas like fault tolerance, notification, recovery etc. The filters are run in specific order to achieve particular processing goals. This corresponds to different workflow [155] scenarios in different Grid domains. The filters are exposed as Web Services to allow remote composition of filter chains which also enables running different workflow scenarios. Although we did not study workflows during our research, we note that Grid workflow engines such as Taverna [156] can be used to create various workflow scenarios with our filter services.

Overall the Sensor Grid paradigm is similar for different scientific domains, the sensors, filter services, the metadata registry and the Grid messaging support make up the backbone of the system.

Here we turn our attention to the major parts of the system;

6.2.1 Filters

Filters in SensorGrid context are data processing applications, mostly operating in real-time. Similar to filters in electronics which accepts processes and outputs certain types of signals, a real-time software filter accepts transforms and presents messages. Depending on the task they are designed for filters may be small applications such as format converters or very complex applications like simulation software. They may be expected to run in real-time by immediately processing and presenting the results or in near-real time by accumulating certain amount of data and executing the processing afterwards.

Figure 6-1 – Simple Filter concept includes a signal generator unit, actual data processing filter unit and the output signal.

The filters in our real-time data Grid architecture have three major parts corresponding to the three components depicted in Figure 1. The first component is the data input unit which is responsible for obtaining the data from the sources. In real-time data Grid the data sources are sensors, or proxy servers which disseminate the real-time sensor measurements. The input unit must have the capability to access and present the data to the actual filter. The second component is the actual data processing unit. And the last component is a data output unit. Depending on the type of the Grid or client applications, the output unit may be implemented to support various data transfer protocols.

While designing real-time applications one obvious principle to remember is the importance of keeping the data flow from sources to destinations alive. Data processing in general requires multiple steps. For instance in a most simplistic case three steps would be required: accessing the data, converting them into application specific formats, and executing the actual processing. However in real world applications many more steps might be required to create a data processing flow. One must remember that any kind of interruption at some step of the flow will disrupt the entire process and possibly cause major breakdown because in the real-time systems the data are likely to be streamed continuously. For this reason it is wise to break down the data processing applications into as many small filters as possible and allow them to be accessed and controlled through standard interfaces. This approach helps creating robust real-time data flows because it allows distribution of the processing components and in turn integrating failsafe measures. For instance backup services could be used to replace any failed services thus allowing keeping the flow alive.

We adopt Web Service approach to create standard control interfaces for the filters. Every filter in the system implements the same Web Service interface. The Filter Web Service interface has capabilities like to start, stop and resume the filter operations. It should also provide a unique identifier for each filter, which can be useful for creating distributed chains. Another important feature of this service is to provide metadata descriptions of the filters.

6.2.2 Filter Metadata

Creating distributed systems require successful orchestration of multiple remote resources. To minimize human interference in this orchestration is also important for fast

an accurate operation. However to do this the resources are need to be well defined. There must be a standard way for resources to communicate for successful integration. Web Services present us a useful way for defining the service capabilities so that coupling the resources can be done automatically. In the filters case we also need additional metadata about each filter for creating filter chains. The metadata documents should contain unique properties of the filters such as its id, input –output requirements, dependencies and a short description of the processing it is responsible for.

In SensorGrid the filters are essentially deployed as Web Services and in most cases run in a workflow as part of complex processes. Each filter is designed to execute a particular task, which means it accepts and produces certain type of messages. This introduces dependencies between the filters. Defining these dependencies in a filter specific metadata document is useful for checking if these dependencies are satisfied at the time of the deployment. This way the users will see which other filters must be deployed as well.

Figure 6-2 shows the metadata schema for the SensorGrid filters. It should be noted that the OGC SensorML [75] specification provides schemas for describing the sensor metadata and it could have been used here. However because of the complexity of the SensorML schemas and our System's requirement for providing additional schema to describe filter chains we have decided to create a simple schema instead.

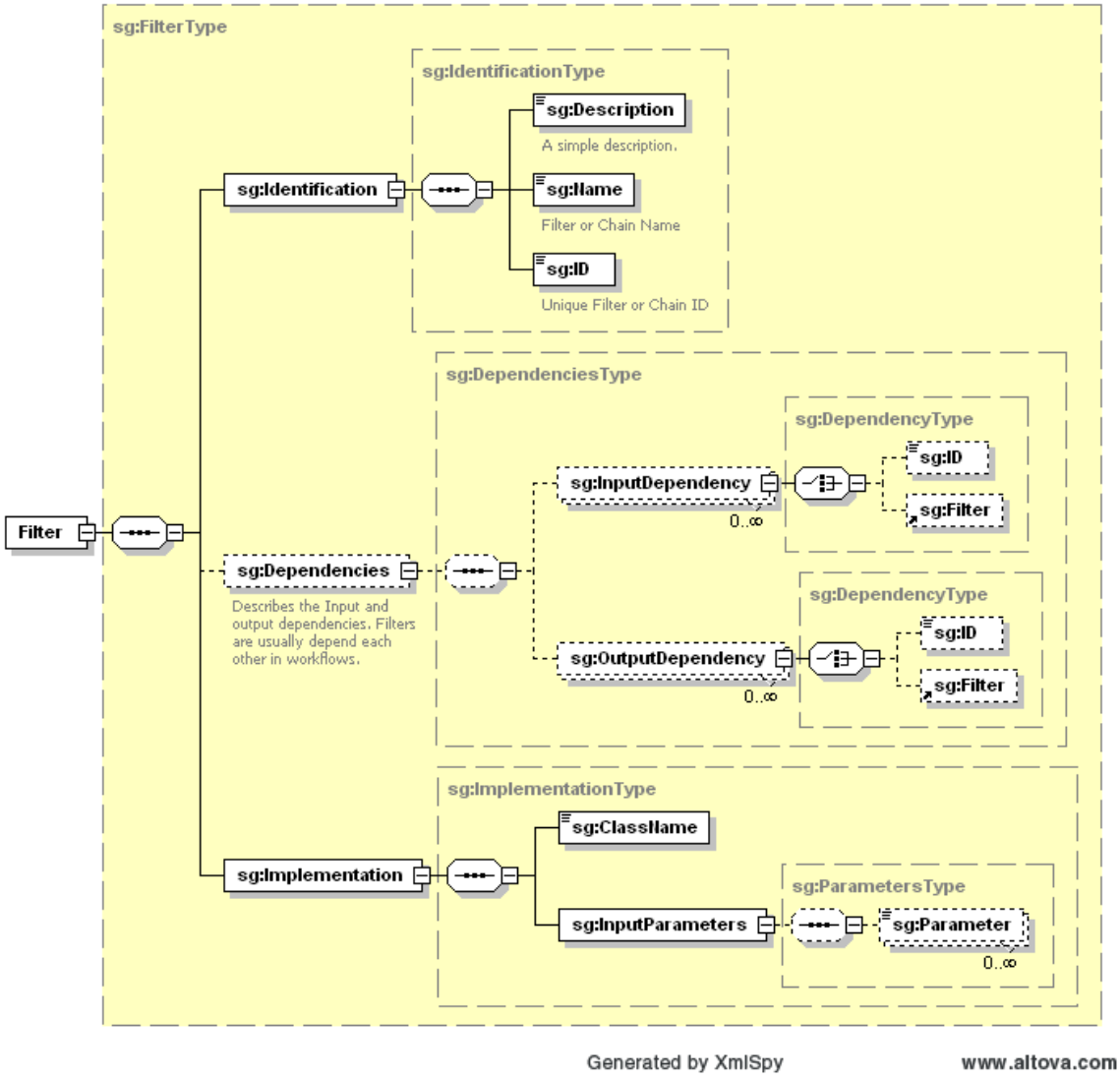


Figure 6-2 - XML Schema for the Filter Metadata

6.2.3 Filter Chains

As described above complex data processing tasks require multiple steps. In our architecture we use distributed filters for realizing such complex tasks. The standard Web Service interfaces for the filters allow remote creation and management of filter chains. We also provide base classes for creating new filters and filter Web Services. The filters

are deployed successively around a publish-subscribe messaging system which federates the distributed resources and allows hierarchical operation of the filters.

Depending on the type of the processing the filters may be chained in parallel or serial modes. If the input data can be processed by different filters at the same time and the results of them are merged after these independent processes are complete then the parallel operation is appropriate.

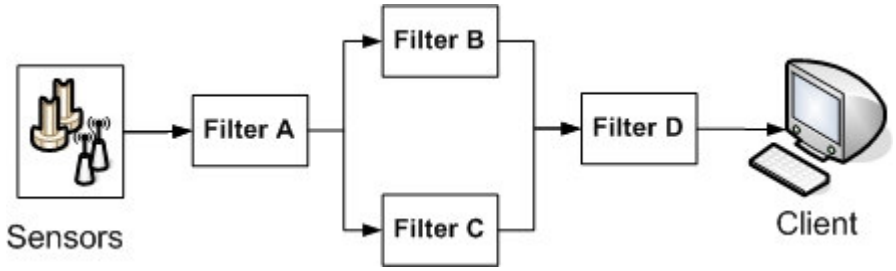


Figure 6-3 - Parallel operation of the filters

Figure 5-6 show the output of the Filter A is shared as input by Filters B and C, while the Filter D merges the outputs from both and does the final processing.

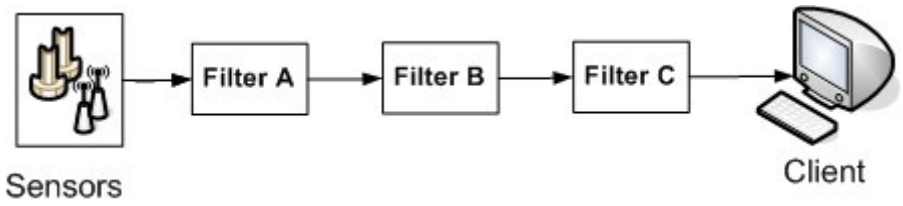


Figure 6-4 - Serial operation of the filters

However in the real-world applications serial operations are dominant. A filter requires output from another as input, which also provides its output to the successive one as input. Figure 5-6 shows three filters processing sensor messages in serial connection.

In the larger picture each domain specific Grid would have specific filter chains. It is desirable for the Grid services to have access to the filter-chains at a given time for

different reasons. For instance at the time of any server failures, it may be expected from the SensorGrid to restart all the filter services in some chains. To be able to do this we need to keep metadata about the running, or potentially useful chains. For this reason we have developed an XML Schema for describing filter chains.

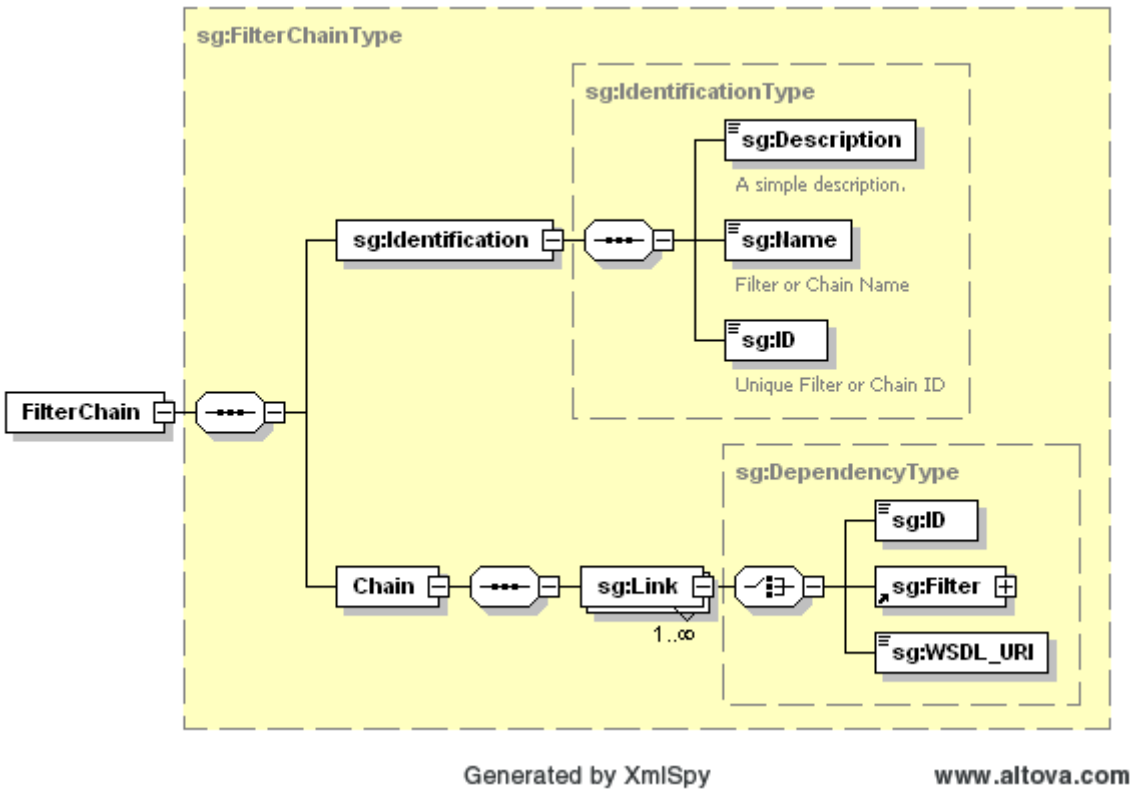


Figure 6-5 - XML Schema for the Filter Chains

Figure 5-6 shows elements of XML Schema document for describing the filter chains. This Schema imports the filter Schema described above. Each filter chain has metadata elements like simple description, name and ID. A chain is composed of multiple Links. Each link refers to a filter implementation, which can be invoked with its unique ID or by importing its metadata document for the local filters, and the WSDL URL for the remote filters.

6.2.4 Information Service

SensorGrid services are implemented as traditional WSDL-SOAP based Web Services. In any architecture where several services are expected to be used there is a need for some sort of a registry. The registry is usually responsible for keeping the endpoint addresses of the services for easy access and manipulation. In our real-time data Grid architecture we usually run several distributed filters consecutively. And because the services are distributed we have multiple WSDL endpoints for any particular real-time workflow. For these reasons we use a UDDI based Information Service to access the service addresses. More discussion about this Information Service can be found in following references [81-86].

6.2.5 Streaming Messaging Support

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 calls traditional Web Services has to offer. 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. In the next section we discuss NaradaBrokering a topic based publish-subscribe messaging system that can provide us several useful capabilities for managing and serving real-time streaming data.

It is obvious that in a real-time data Grid the top priority is to be able to provide access and process the continuously streaming data. Any kind of interruption in this process will result in data loss. To prevent data loss or similar problems the messages should be reliably transferred between the services. Traditional Web Services are implemented on top of the already existing Web infrastructure, which also makes them universal. However HTTP is not appropriate for high-rate data flow requirements. For instance if we want to disseminate real-time messages from GPS sensors which output measurements once per second, the latency associated with the HTTP would be larger than allowed, especially for long distances. This is also true for large scale data transfers. Therefore a better messaging solution is required for connecting the filters and the sources.

The real-time data Grid architecture is expected to provide multiple users with access to multiple streaming data sources. Therefore the system must support many-to-many interactions. Also since the sensors are always alive and continuously produce data the reliability and fault-tolerance of the messaging architecture is extremely important. For these reasons we have studied publish-subscribe based messaging systems as our messaging infrastructure. Our research showed that topic based publish-subscribe messaging systems can meet our requirements. One strong candidate in this area is NaradaBrokering, which was explained in 4.4.1.

6.2.6 Filter Web Services

We have created a base filter class that provides NaradaBrokering publisher and subscriber capabilities. By extending the base filter class new filters can easily be implemented. We have created several such filters to process real-time GPS messages.

The filters receive messages from the NaradaBrokering messaging middleware by ***onEvent*** function. Processed messages can be published to a NaradaBrokering topic by using ***publishData*** function. These are the only two functions required to connect a filter to the messaging system. By aggregating several such filters we can create an assembly line that takes the raw data and process it along the way.

The base filter class also provides two methods to start and stop individual filters. ***startFilter*** and ***stopFilter*** methods use Java Reflections API to invoke and stop any filter that extends the base filter class. New filters that extend the Filter class must implement a constructor with two arguments; the first argument is a String Array with 5 string elements. These arguments are used to initialize the NaradaBrokering subscriber and publishers. The second argument is another array of strings used to pass the filter specific arguments.

For instance, following are the arguments used to start the ***ryo2ascii*** filter:

```
String siteName = "GLRS";
String[] args_1 = ["niotcp","gf2.ucs.indiana.edu","3045",
"/SOPAC/GPS/OCRTN/ASCII","/SOPAC/GPS/OCRTN/ASCII/" + siteName];
String[] args_2 = [siteName];
```

The first set of arguments is NaradaBrokering specific: communication type, server address, port number, topic to subscribe and topic to publish. Depending on the type of the filter subscriber topic or publisher topic may be null. The second argument is filter specific, and in this case contains only one string, name of a GPS Station.

An advantage of creating new filters by extending a base class is the ability to keep track of all filters that are currently initialized and active. The base class generates a unique ID for every invocation it receives and keeps a hash table that contains

'uniqueID/filterType' couples. This unique ID is returned to the user by *startFilter* method. The user can pass this unique ID to *stopFilter* method as an argument to stop that particular filter instance.

There are two possible ways for creating web services for filters: first, we can expose individual filter classes as web services however this method may introduce several problems such as keeping track of the Web Service end point addresses/URLs for all filters. The second method is to create a proxy web service that is generic to all filters extending the base filter class.

By exposing the *startFilter* and *stopFilter* methods of the base Filter class we have created a proxy Web Service to start/stop filters. Instead of exposing individual filters as Web Services the client need only pass the filter name and required parameters to invoke a filter. This approach allows us to use only one service to control multiple filters in the system which can be useful in keeping track of the status of the overall system.

For instance using the above parameters we can start the ryo2ascii filter remotely as following

```
FilterWSClient f = new FilterWSClient();
String id = f.StartFilter(
            "cgl.sensorgrid.sopac.gps.filters.ryo2ascii",
          args_1,
            args_2,
            "http://mastar.ucs.indiana.edu:8080/sensorgrid-
        services/services/Filter");
```

Here the first argument is the full name of the filter to be initialized, second and third argument s are explained above, and the last argument is the filter web service URL.

The id returned by the *startFilter* method looks like the following string:

52ad94ea-a460-4e5d-b33a-6e9ce6e353ad

We use this ID to stop the filter as following:

```
f.StopFilter("cgl.sensorgrid.sopac.gps.filters.ryo2pos",
        "52ad94ea-a460-4e5d-b33a-6e9ce6e353ad",
         "http://mastar.ucs.indiana.edu:8080/sensorgrid-
services/services/Filter");
```

In addition to the control functions, the filter Web Service also provides another function *getFilterCapabilities,* which can be used to query a particular filter's metadata description. The filter metadata file, as described above contains information about the filters such as description, name, dependencies, implementation class etc. The user can query the filter capabilities by providing full class name, or the unique ID for active filters.

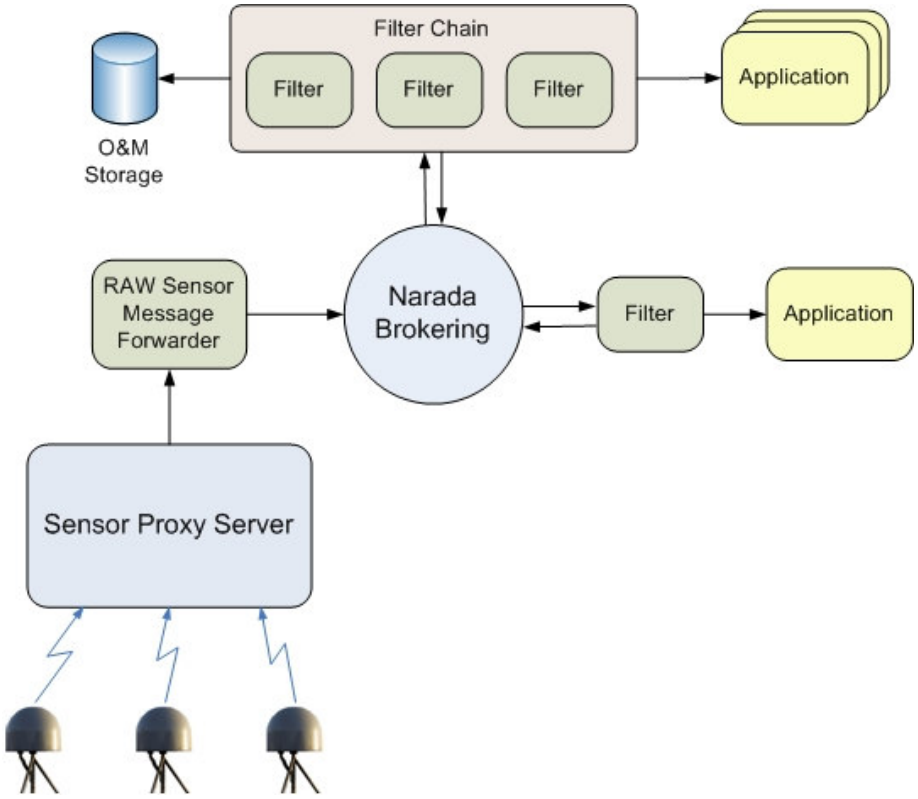


Figure 6-6 - Overall SensorGrid Architecture

Figure 5-6 shows the overall SensorGrid architecture which contains several filters for processing, converting or aggregating data streams, NaradaBrokering messaging system for message transfer and integrated applications.

6.3 Real time Data Grid Implementation for Global Positioning System Networks

To demonstrate the use of technologies discussed earlier we describe GPS Services developed for the Scripps Orbit and Permanent Array Center (SOPAC) GPS networks [157]. SOPAC's distributed GPS networks continuously provide publicly available data. Raw data from the GPS stations are collected by a Common Link proxy (RTD server) and archived in RINEX files. In this section we describe the implementation of the aforementioned technologies.

Figure 6-7 displays Plate Boundary Observatory (PBO) [158] GPS stations. As of August 2006 more than 400 GPS stations are operational (http://pboweb.unavco.org). Note that these stations are continuously operating and data are periodically being collected however they are not real-time stations i.e. they do not provide access to position measurements in real-time. The data are retrieved are made available through online archives (FTP sites).

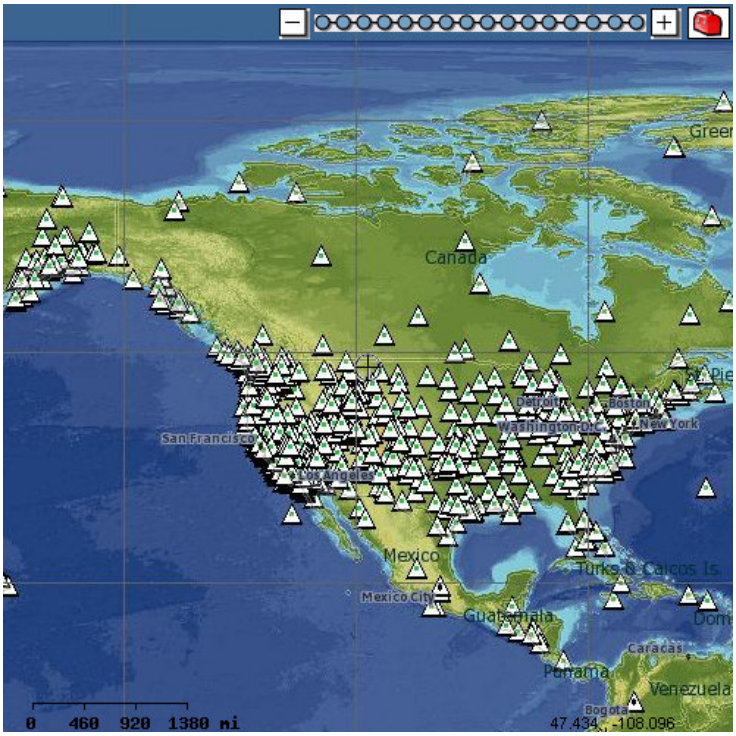


Figure 6-7 –Plate Boundary Observatory (PBO) GPS Stations in North America; Image is obtained from SOPAC GPS Explorer at http://sopac.ucsd.edu/maps.

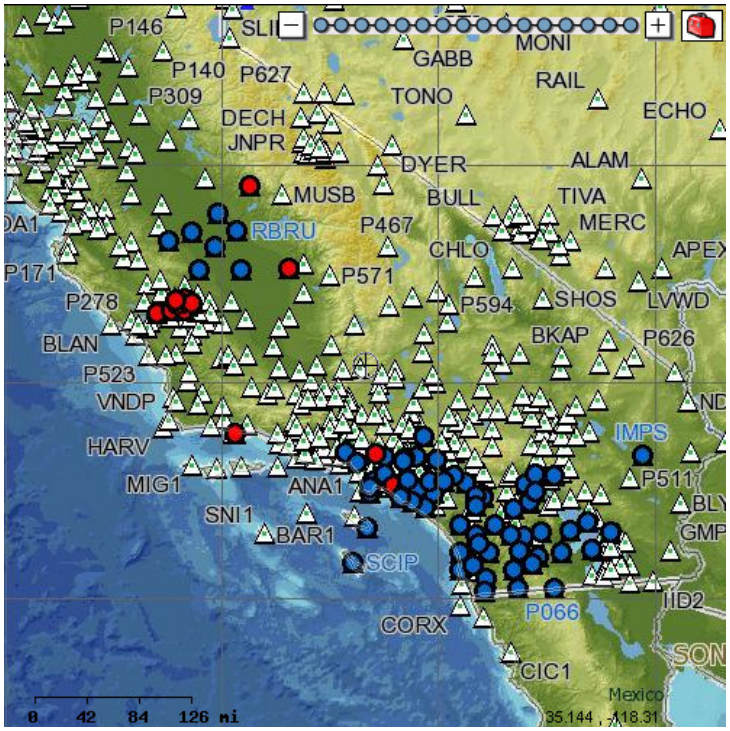


Figure 6-8 – California Real-Time GPS Network (CRTN). Note the Continuous GPS Stations (CGPS) are depicted as triangles while the Real-Time stations are represented as circles. Image is obtained from SOPAC GPS Explorer at http://sopac.ucsd.edu/maps.

Figure 6-8 displays Real-Time GPS networks in Southern California. The triangles represent the continuous GPS stations while the blue and red circles represent the real-time stations. The real-time stations are deployed by the Southern California Integrated GPS Network (SCIGN) [159] project.

6.3.1 Real-Time GPS Networks

Global Positioning System has been used in geodesy to identify long-term tectonic deformation and static displacements while Continuous GPS (CGPS) has proven very effective for measurement of the inter-seismic, co-seismic and post-seismic deformation. [160]. GPS Stations are effectively independent sensors that calculate and broadcast their instant geographic positions. They can run for long periods of times without need for frequent maintenance and can communicate with the data collection points using various connection types such as Wi-Fi, modems and phone lines or fiber-optic lines. Today networks of individual GPS Stations (monuments) are deployed along the active fault lines, and data from these are continuously being collected by several organizations. One of the first organizations to use GPS in detection of the seismic events and for scientific simulations is Southern California Integrated GPS Network (SCIGN) [159]. One of the collaborators in SCIGN is Scripps Orbit and Permanent Array Center (SOPAC) which maintains several GPS networks and archives high-precision GPS data, particularly for the study of earthquake hazards, tectonic plate motion, crustal deformation, and meteorology. Real time sub-networks maintained by SOPAC include Orange County, Riverside County (Metropolitan Water District), San Diego County, and Parkfield. These networks provide real-time position data (less than 1 sec latency) and operate at high rate (1 – 2 Hz). The raw measurements from the GPS sensors are continuously collected and

locally stored by a Common Link Proxy (RTD) Server and later made available to public via FTP sites. The GPS networks provide real-time position data (less than 1 sec latency) and operate at high rate (1 – 2 Hz). The RTD server also broadcasts real-time positions in a proprietary binary format called RYO. Each RYO message contains the positions of the stations that reported for that epoch.

The data collected from the GPS stations are served in various formats as following:

- **RAW**: For archiving and record purposes, not interesting for scientific applications, not available in real-time.
- **RTCM**: Published real-time and no records are kept. This is useful for RTCM capable GPS receivers as reference.
- **Positions**: Positions of the stations. Updated and presented every second. GPS Time Series can be produced using these positions and they can be in different epochs such as hourly, daily etc.

Perhaps the most interesting of these formats for scientists is position information which can be used in scientific calculations, simulation or visualization applications. The RTD server however outputs the position messages in a binary format called RYO. This introduces another level of complexity on the client side because the messages have to be converted from binary RYO format.

To receive station positions, clients are expected to open a socket connection to the RTD server. An obvious downside of this approach is the extensive load this might introduce to the server when multiple clients are connected.

After the RTD server receives raw data from the stations it applies some filters and for each network generates a message. This message contains a collection of position

information for every individual station from which the position data has been collected in that particular instant. In addition to the position information there are other measurements in a message such as quality of the measurement, variances etc.

For each GPS network, RTD server broadcasts one position message per second through a port in RYO format.

To make the position information available to the clients in a real-time streaming fashion we used NaradaBrokering. Additionally we developed applications to serve position messages in ASCII and GML formats.

6.3.2 Chain of Filters

To process GPS sensor streams in real-time we have developed several filters and Web Services to make real-time position messages available to scientific applications. In summary, the core of the system is to implement filter chains that convert or otherwise process the incoming data streams. These filters serve as both subscribers (data sinks) and publishers (data sources). NaradaBrokering topics are used to organize different data stream sources into hierarchies as shown in Table 6-3. Currently the filters are being used to support 8 networks with 85 GPS Stations maintained by SOPAC.

In our architecture filters are small applications designed to realize simple tasks such as transforming or aggregating messages. We have developed an abstract filter interface which can be extended to create new filters. A basic filter is consisted of three parts: a NaradaBrokering subscriber, a publisher and a data processing unit. The abstract filter interface provides subscriber and publisher capabilities. Typically a filter subscribes to a specified NaradaBrokering topic to receive streaming messages, process the received data and publishes the results to another topic. However outputs need not be always

published, for instance a Database Filter may only receive the station positions to insert into the database. Furthermore filters can be connected in parallel or serial for realizing more complicated tasks.

The first filters we have developed are format converters that present original binary messages in different formats since GIS applications require different representations of geographic data. Since the data provided by RTD server is in a binary format we developed filters to decode and present it in different formats. Once we receive the original binary data we immediately publish this to a NaradaBrokering topic (null filter), another filter that converts the binary message to ASCII subscribes to this topic and publishes the output message to another topic. We have developed a GML schema to describe the GPS position messages. Another filter application subscribes to ASCII message topic and publishes GML representation of the position messages to a different topic. This approach allows us to keep the original data intact and different formats of the messages accessible by multiple clients in a streaming fashion.

The GML Schema we wrote is based on RichObservation type which is an extended version of GML 3 Observation model [18]. This model supports Observation Array and Observation Collection types which are useful in describing SOPAC Position messages since they are collections of multiple individual station positions. We follow strong naming conventions for naming the elements to make the Schema more understandable to the clients.

We used Apache XML Beans [161] for data binding purposes and created an application that reads ASCII position messages and generate GML instances using the code generated by XML Beans. SOPAC GML Schema and sample instances are

available at: http://www.crisisgrid.org/schemas. The GML-OM Schema developed for GPS station messages and a sample XML output is given in the Appendix.

6.3.3 GPS Station Messages and Filters

As discussed above, station messages collected from GPS stations have several sub-sections. We have developed several filters that simplify or convert the messages since not all the parts of a position message are needed by most clients. Figure 6-9 shows the entire system including the GPS networks, proxy server, filters and the broker.

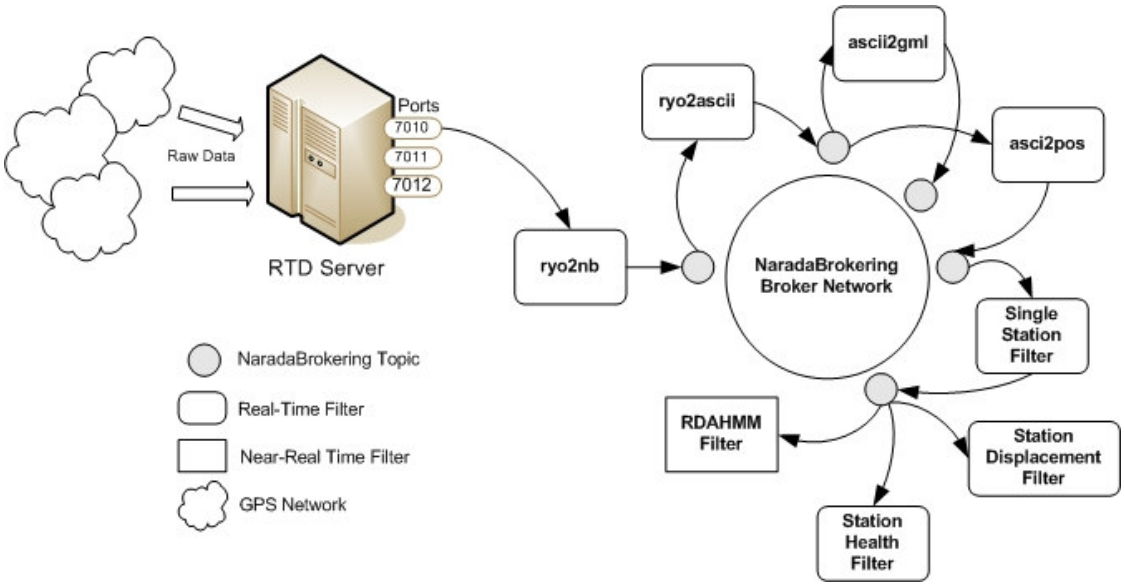


Figure 6-9 – Real-Time Filters for processing real-time GPS streams

Here we give sample output messages from different filters:

The first filter in our architecture is a null filter which forwards original RYO binary messages from RTD server to a NaradaBrokering topic. The output of this filter is unreadable binary messages.

6.3.3.1 Decoding RYO Messages

RYO Message Type 1 starts with a 5-byte Header which is followed by a 47-byte GPS Position message. Three types of optional blocks may follow the Position Message and a 2-byte checksum is located at the end of the message.

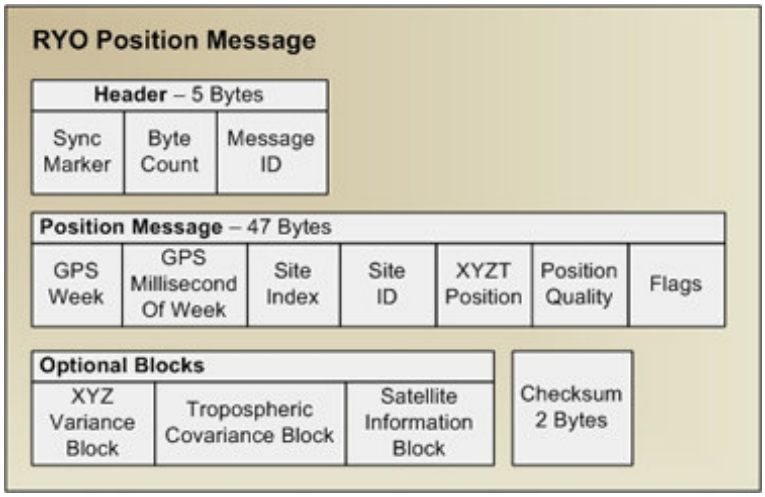


Figure 6-10 - RYO Message Parts

A non-blocking Java Socket connection is opened to RTP server to collect RYO messages. An RYO Decoder application which uses binary conversion tools to convert RYO messages into text messages is used to receive the raw GPS messages.

Furthermore since we do not expect clients to know about the GPS time format we convert GPSWeek and GPSmsOfWeek values to Gregorian calendar format (i.e. 2006-19-07/04:19:44PM-EST). Additionally since we anticipate some clients to expect position information in terms of Latitude and Longitude, we calculate Latitude, Longitude and Height values from XYZT Position.

The second filter is called ***ryo2ascii*** which converts RYO messages to ASCII and publishes to a NB topic. Following is the parts of a message generated by this filter:

```
2005-12-12  03:23:16PM-EST    LEMA  2     3
```

```
-2556508.624797094 -4467101.665687391 3754378.932770622
2.2950492465819603
36.29202035061081 -119.78237412236496 35.929088668037025
L1/L2 Phase XYZ Satellite
0.06901739184684381 0.0377138796649775 0.10830764487854985
0.08631783233709235 0.06057192662251049 -0.09281413763791896 -
0.05338606394765551
1 2 7 42 179 2 4 7 61 117 3 5 7 23 -54 4 7 7 50 45 5 9 7 55 -75 6 24 7
35 54 7 26 7 5 -142
```

The message contains following parts:

Message time

Date	Time
2005-12-12	03:23:16PM-EST

Station Metadata

Station Name	Station Number	Station Count
LEMA	2	3

XYZT Position

X	Y	Z
2556508.624797094	4467101.665687391	3754378.932770622

T
2.295049246581960

Latitude-Longitude-Height Position

Latitude	Longitude	Height
36.29202035061081	119.78237412236496	35.929088668037025

Position quality : L1/L2

Flags: Phase

Optional blocks present in this message: XYZ variance block + Satellite info block

XYZ variance block

Scale	Xvar	Yvar
0.06901739184684381	0.0377138796649775	0.10830764487854985

Zvar	**YXvar**	**YZvar**
0.08631783233709235	0.06057192662251049	-0.09281413763791896

ZXvar
-0.05338606394765551

Satellite Info Block

Satellite No	1	2	3	4	5	6	7
PRN	2	4	5	7	9	24	26
PRN Flags	7	7	7	7	7	7	7
Elevation	42	61	23	50	55	35	5
Azimuth	179	117	-54	45	-75	54	-142

Ryo2ascii filter converts the whole RYO message and does not filter out anything. However some of the information included in a position message is unnecessary for most clients. For instance we have developed a user interface to display the current positions of the stations on a map. For this particular application we only need station names and their positions in terms of latitude and longitude. For this client interface we have developed ryo2pos filter which converts RYO messages to simple position messages. Following is a sample output message from ***ryo2pos*** filter:

```
LEMA     2005-12-12  03:58:37PM-EST    36.29202028791537
-119.78237425030088  35.90217063464758
```

Here we only include Station name, date-time and latitude, longitude and height values in the message. This small application is an example of how individual filters can be chained using NaradaBrokering to achieve specific tasks. Another example application integrated using this approach is RDAHMM which only requires X, Y, Z or Lat, Lon Height values for a given station. We can easily write a filter to strip unwanted parts from the message and output only the position information.

Following table shows information about these networks:

Table 6-2 – Real-Time GPS Networks, individual stations and RTD server information

Network Name	RTD Server Address	Stations
LACRTN	132.239.154.69:5014	vtis, hbco, cvhs, lors, tabl, ucsb, azu1, csdh, dyhs, vdcy, uclp, cit1, lapc
PARKFIELD	n/a	hogs, pomm, mida, crbt, carh, land, mnmc, lows, rnch, cand, masw, tblp, hunt
OCRTN	132.239.154.69:5010	oeoc, cat2, whyt, trak, sacy, mjpk, scms, sbcc, fvpk, blsa
SDCRTN	132.239.154.69:5013	p486, monp, raap, mvfd, p472, sio5, dvlw, pmob, p480, dsme, oghs
IMPERIAL	132.239.154.69:5012	slms, crrs, usgc, dhlg, glrs
DVLRTN	132.239.152.72:8001	dvle, dvne, dvsw, dvse, esrw, dvls, dvnw, ese2
CVSRN	132.239.154.69:5015	coma, rbru, lema
RCRTN	132.239.154.69:5011	pin2, widc, kyvw, psap, cotd, pin1, mlfp, cnpp, bill, ewpp, azry

Following table shows the NaradaBrokering topic names for several filters:

Table 6-3 NaradaBrokering topics for GPS streams

Network Name	RYO Topic (null filter Publishes to)	ASCII topic (ryo2ascii filter Publishes to)
LACRTN	/SOPAC/GPS/LACRTN/RYO	/SOPAC/GPS/LACRTN/ASCII
PARKFIELD	/SOPAC/GPS/PARKFIELD/RYO	/SOPAC/GPS/PARKFIELD/ASCII
OCRTN	/SOPAC/GPS/OCRTN/RYO	/SOPAC/GPS/OCRTN/ASCII
SDCRTN	/SOPAC/GPS/SDCRTN/RYO	/SOPAC/GPS/SDCRTN/ASCII
IMPERIAL	/SOPAC/GPS/IMPERIAL/RYO	/SOPAC/GPS/IMPERIAL/ASCII
DVLRTN	/SOPAC/GPS/DVLRTN/RYO	/SOPAC/GPS/DVLRTN/ASCII
CVSRN	/SOPAC/GPS/CVSRN/RYO	/SOPAC/GPS/CVSRN/ASCII
RCRTN	/SOPAC/GPS/RCRTN/RYO	/SOPAC/GPS/RCRTN/ASCII

Similarly the ryo2pos filter subscribes to the appropriate RYO topic and publishes to for instance /SOPAC/GPS/LACRTN/POS topic.

Here we give brief overview for some of the filters we have developed for SensorGrid architecture:

ryo2nb Filter: This is a simple message forwarding application that opens a TCP socket connection to the RTD server to receive the RYO messages and publishes to a NaradaBrokering topic (i.e. "/RYO").

ryo2ascii filter: Subscribes to the RYO topic to receive binary messages, converts them to simple ASCII format and publishes to another topic (i.e. "/ASCII").

ascii2gml filter: Geography Markup Language is perhaps today's most popular geographic data format produced by OGC. We have developed a GML Schema conformant with the latest Observations and Measurements [23] extension to describe GPS station messages. This filter converts the ASCII position messages into GML and publishes to a new topic (i.e "/GML"). We expect that in the near future most GIS applications will be developed to conform to OGC standards and presenting GPS messages in GML will help us easily integrate scientific applications.

ascii2pos filter: The RYO message type contains several sub parts other than physical position of the station such as position quality and several optional blocks. However most of this extra information is not required by the applications. This filter eliminates optional blocks and unnecessary information from the ASCII messages to create concise position messages which only include a time stamp, station id and position measurements.

Station Displacement Filter: One of the use cases of GPS stations is to detect seismic activities. We have developed a simple filter that analyzes position information of a GPS Station and outputs its real-time physical displacement. The filter allows displacements to be calculated based on different time intervals, i.e. actual displacement of the station in last hour or in last 24 hours.

Station Health Filter: One advantage of dealing with the real-time measurements is that we can instantly see if any of the sensors in a network is not publishing position information. We have developed this filter which logs the down times of the stations and (potentially) alerts administrator if a threshold value is reached. For instance it can be tolerable for a GPS station to be down for a few minutes due to network problems etc. but if a station has not been publishing position values for over an hour a maintenance call may be required.

Single Station Filter: As mentioned above the original messages imported from the RTD server contains position information for multiple stations. However some applications may be required to analyze data for a particular station. For this reason we have developed this filter to pull measurements from a particular station.

6.4 Application integration Use Case: Coupling RDAHMM with Streaming Data

The Regularized Deterministic Annealing Hidden Markov Model (RDAHMM) [58] [60], is a data analysis program that employs Hidden Markov Models to identify different modes of the system and their probabilistic descriptions. RDAHMM has successfully been used to identify mode changes in GPS time series data. With the development of our real-time GPS data support architecture a new version of RDAHMM has been under development to analyze streaming data. Current version operates in two phases: Training and Evaluation. In our test case first the application is trained on a set of data for a particular station. Then it can be run continuously on accumulated data once a

pre-determined time window is reached. Although this version is not completely real-time we can run it near-real time by keeping the time window relatively small.

To integrate RDAHMM with real-time data we have tested two different approaches. The first one is based on scripting service management by using HPSearch. The second method is based on more traditional filter method as depicted in Figure 8, by treating RDAHMM application as another filter. Both of these methods are explained here.

6.4.1 RDAHMM Integration using HPSearch

HPSearch [118-120] is a scripting based management interface used to manage publish/subscribe systems. HPSearch also provides tools to wrap existing codes as Web Services and provides a scripting based workflow management interface to connect different components of the workflow. HPSearch uses NaradaBrokering's publish/subscribe based messaging architecture to stream data between various services. Ref. [25] describes an initial version of RDAHMM using HPSearch. Figure 6-11 illustrates the architecture for RDAHMM integration. As shown in the figure, the system consists of 3 Web Services, a NaradaBrokering server and an HPSearch node.

The Web Services in this architecture are as follows:

1- **DataTransfer Service**: This service transfers position messages accumulated by the RDAHMM Filter via NaradaBrokering to the server where RDAHMM actually runs.

2- **RDAHMMRunner Service**: Invokes RDAHMM to run on the transferred data set.

3- **GraphPlotter Service**: Runs Matlab to plot RDAHMM results as TIFF files and copies figures to a Web accessible folder.

Additionally HPSearch kernel also has a WSDL interface which is used by RDAHMM Filter to start the flow.

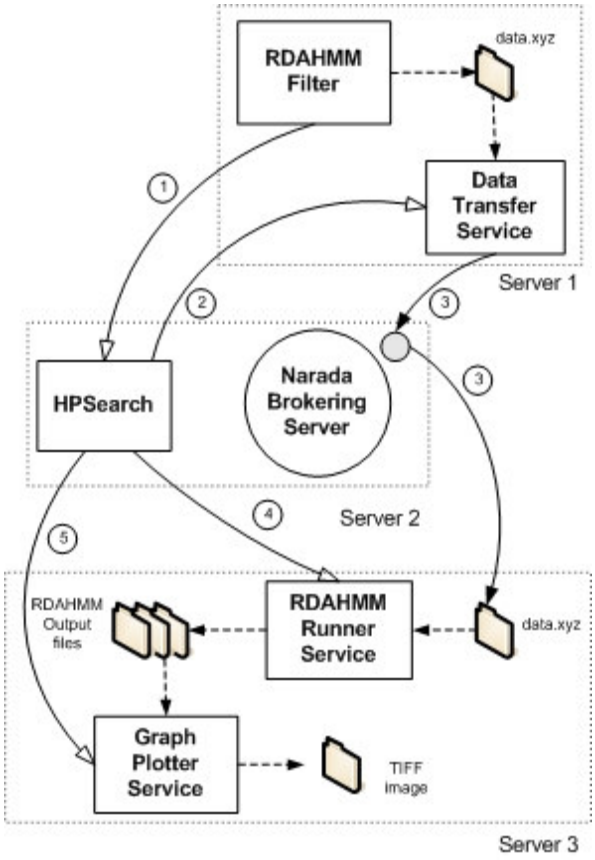


Figure 6-11 – Filter Services and RDAHMM Integration

The system components are distributed over three servers. RDAHMM Filter and Data Transfer Service runs on Server-1. HPSearch kernel and NaradaBrokering server are installed on Server-2, whereas RDAHMM application, RDAHMM Runner Service and Graph Plotter Service run on Server-3. We also run an Apache Tomcat Web Server on Server-3 to present the generated TIFF images online.

The system uses following real-time filters described above: ryo2nb, ryo2ascii, ascii2pos and Single Station Filter. Additionally the RDAHMM Filter subscribes to a single station topic to save that station's position information.

The experimental system works as follows: The RDAHMM Filter is a part of the architecture discussed previously and shown in Figure 6-11. It accumulates the position messages of a particular station in a file (data.xyz) for a certain amount of time (i.e. 10 minutes for 600 lines, or 30 minutes for 3600 lines). Once the time threshold is reached it invokes HPSearch to start the process. HPSearch starts executing the script that defines the service locations and the order of the services to be executed. It first invokes the DataTransfer Service to start transferring the data.xyz file created by RDAHMM Filter to Server-3. Once this transfer is completed HPSearch engine invokes RDAHMMRunner Service and waits until it finishes the evaluation. Then it invokes GraphPlotter Service to read the RDAHMM outputs and plot the resulting graphic. This cycle is repeated every time the RDAHMM Filter reaches the time threshold.

For this system we have created a simple application that acts as the RTD server to publish RYO messages once per second. We used an RYO data set collected by 13 Parkfield GPS Network sensors for a 24-hour period between 09-27-2004, 06:59:47 PM and 09-28-2004, 06:59:46 PM. The latest major Parkfield earthquake occurred on 09-28-2004 at 10:15:24 AM.

The RDAHMM outputs tell us the number of different states detected in the input and information useful for plotting these states. Previous versions of RDAHMM were used to analyze archived GPS daily time-series and successfully detected state changes in the inputs which correspond to seismic events.

Our tests show that the real-time filters used in this architecture do not introduce any overhead. Since the GPS messages are received every second it is expected from the real-time filters to complete processing under one second not to skip the next message.

According to our timing measurements all of the four real-time filters finish message processing under 100ms. We have tested RDAHMM using two different methods. First we used a sliding window method and run RDAHMM for every 1000, 3000, 5000 etc. lines of data. Next we applied an increasing window method by transferring every 1000 new measurements to the RDAHMM server and appending this to previous data file. Thus RDAHMM was run on increasing data sizes.

6.4.2 RDAHMM Integration as a Filter

The second method we used to couple RDAHMM with real-time data is by using RDAHMM as another filter. This method requires the RDAHMM Filter to be deployed on the same server as the actual RDAHMM application. Then the filter is expected to listen to the ASCII position topic and accumulate certain number of messages. As soon as the limit number is reached the RDAHMM Filter invokes the actual RDAHMM application for the accumulated data. Figure 6-12 depicts the NaradaBrokering topic hierarchy for this particular application integration case.

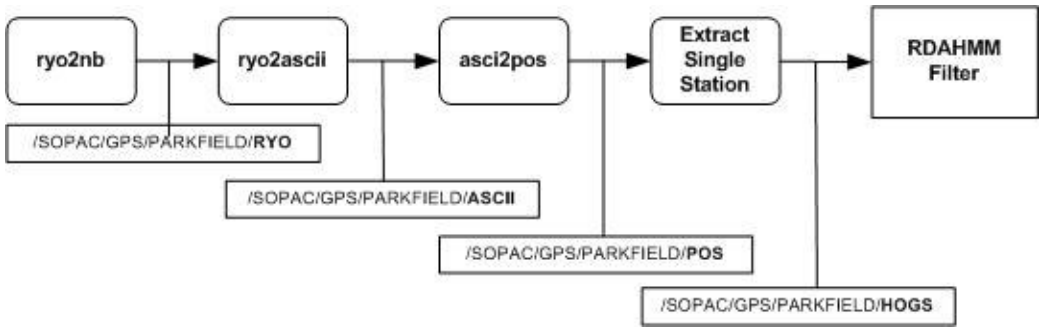


Figure 6-12 - NaradaBrokering topics can be arranged in a hierarchical order.

6.5 Real-Time display of the GPS station positions on Google Maps using AJAX methods

To demonstrate the technologies discussed earlier we have developed several JSP based client interfaces leveraging AJAX [162] techniques. The user interfaces we discuss here demonstrate use of topic based publish-subscribe messaging for managing and serving real-time data coupled with several real-time data filters.

AJAX or Asynchronous JavaScript and XML is a relatively new web development technique for creating highly interactive web interfaces. AJAX is not a technology by itself rather a name for using a collection of several well-known technologies together. It employs XHTML or HTML, JavaScript, DOM and XMLHttpRequest. XMLHttpRequest is originally developed by Microsoft and available since Internet Explorer 5.0. This object is used to exchange data with the server asynchronously.

Traditionally user's every action generates an HTTP request; in the case of AJAX these requests are JavaScript calls to the server side which allows only the related portion of the web pages to refresh instead of whole page to be submitted to the server and refreshed. This technique allows creation of powerful user interfaces and uninterrupted browsing experience for the users.

Creating AJAX compatible pages is relatively simple. Here we summarize common JavaScript techniques:

- Creating an XMLHttpRequest Object

  For any browser, except IE `var requester = new XMLHttpRequest();`

  For IE `var requester = new ActiveXObject("Microsoft.XMLHTTP");`

- Transporting Data using an XMLHttpRequest Object

  To retrieve data from the server we use two methods:

  `open()` to initialize the connection,

  `send()` to activate the connection and make the request. i.e.

```
requester.open("GET", "getFaultNames.jsp?State=CA");
requester.send(null);
```

- To find out if the data retrieval is done we check the status of the **`readyState`** variable. Object's status may be any of the following:

```
0 - Uninitialised
1 - Loading
2 - Loaded
3 - Interactive
4 - Completed
```

`requester.onreadystatechange` can be used to monitor the **`readyState`** variables status.

```
if (requester.readyState == 4){
   if (requester.status == 200){
   success();
 }
 else{
fail ();
}
 }
```

- After a successful request `XMLHttpRequest` object may hold data in one of the two properties: `responseXML` or `responseText`.

  `responseXML` stores a DOM-structured XML data, such as:

```
<Fault>
  <Name>San Andreas</Name>
</Fault>
```

- We use JavaScript XML parsing methods such as `getElementsByTagName(), childNodes[], parentNode…`

```
var faultNameNode = requester.responseXML.
getElementsByTagName("Name")[0];
var faultName = faultNameNode.childNodes[0].nodeValue;
```

- We can then use Google Map JavaScript functions to create the browser display.
- responseText stores the data as one complete string in case the content type of the data supplied by the server was text/plain or text/html.

Most of the AJAX compatible interfaces that invoke JAVA classes on the server side use Java Servlets. However since our user interfaces are based on JSP we have developed a novel method for making AJAX calls from Java Server Pages.

In the 1st JSP page we have a JavaScript method that creates an XMLHTTPRequest and sends it to a second JSP page:

```
function checkForMessage() {
  var url = "relay.jsp";
  initRequest(url);
  req.onreadystatechange = processReqChange;
  req.open("GET", url, true);
  req.send(null);
 }
```

The initRequest method creates the actual request object:

```
function initRequest(url) {
  if (window.XMLHttpRequest) {
    req = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    isIE = true;
    req = new ActiveXObject("Microsoft.XMLHTTP");
  }
}
```

In the 2nd JSP page (relay.jsp) we only invoke the server side Java class with the JSP response parameter.

```
<%  Bean.getNames(response); %>
```

On the server side when the request arrives, the Java class checks for new messages from the NaradaBrokering server and saves these position messages in XML format in the response object as follows:

```
<message>
      <pos>
            <name>DSME</name>
            <lat>33.03647257927002</lat>
            <lon>-117.24952051832685</lon>
      </pos>
      <pos>
            <name>OGHS</name>
            <lat>33.13060260841207</lat>
            <lon>-117.04175378543312</lon>
      </pos>
      <pos>
            <name>PMOB</name>
            <lat>33.357234902933584</lat>
            <lon>-116.85953161093065</lon>
      </pos>
      <pos>
            <name>MVFD</name>
            <lat>33.21086802863064</lat>
            <lon>-116.52529897245469</lon>
      </pos>
      <pos>
            <name>P486</name>
            <lat>33.260186243838994</lat>
            <lon>-116.3222711652632</lon>
      </pos>
      <pos>
            <name>P482</name>
            <lat>33.24017400862219</lat>
            <lon>-116.67139746579954</lon>
      </pos>
</message>
```

Once the response object is returned `processReqChange` method parses the response and extracts the elements from the XML document.

Figure 6-13 depicts the integration of Real-Time GPS messages, NaradaBrokering and AJAX based user interfaces.

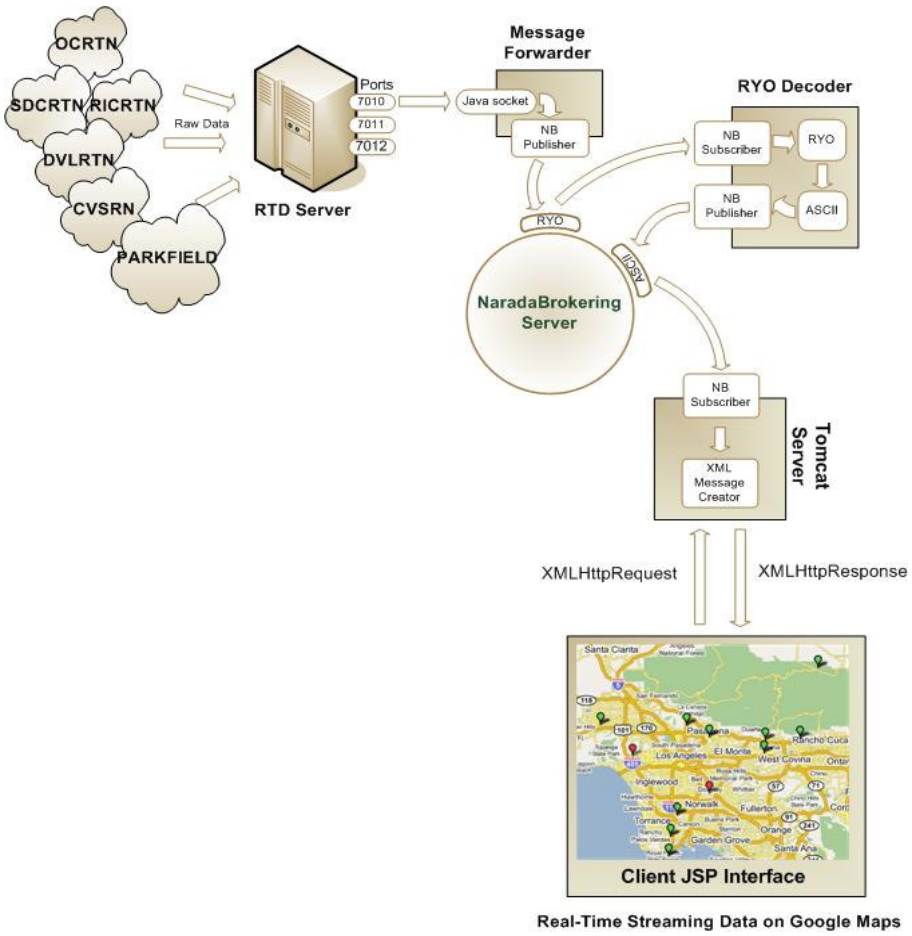


Figure 6-13 - Architectural diagram for Real-Time GPS messages and AJAX integration

For this demo architecture we use an online XML document (RSS feed) provided by SOPAC to retrieve up-to-date list of available GPS stations. This document contains several properties of each station such as the station name, the network it belongs to, latitude and longitude values, and the RTD server IP address and the port number for receiving the binary positions. Following is a segment from this file:

```
<station>
 <network>
  <name>LACRTN</name>
  <ip>132.239.154.69</ip>
  <port>5014</port>
 </network>
 <id>vtis</id>
 <longitude>-118.294</longitude>
 <latitude>33.713</latitude>
 <status>up</status>
</station>
```

Figure 6-14 shows all of the GPS stations managed by SOPAC. Each GPS network has a distinct color.

Figure 6-14 – Real-Time GPS networks in Southern California displayed on Google Maps. A list of the Real-Time networks and corresponding symbols are given at the right side of the figure.

Our user interfaces first retrieves the XML document from SOPAC and creates a HTML form for user to select a network as displayed in Figure 6-15.

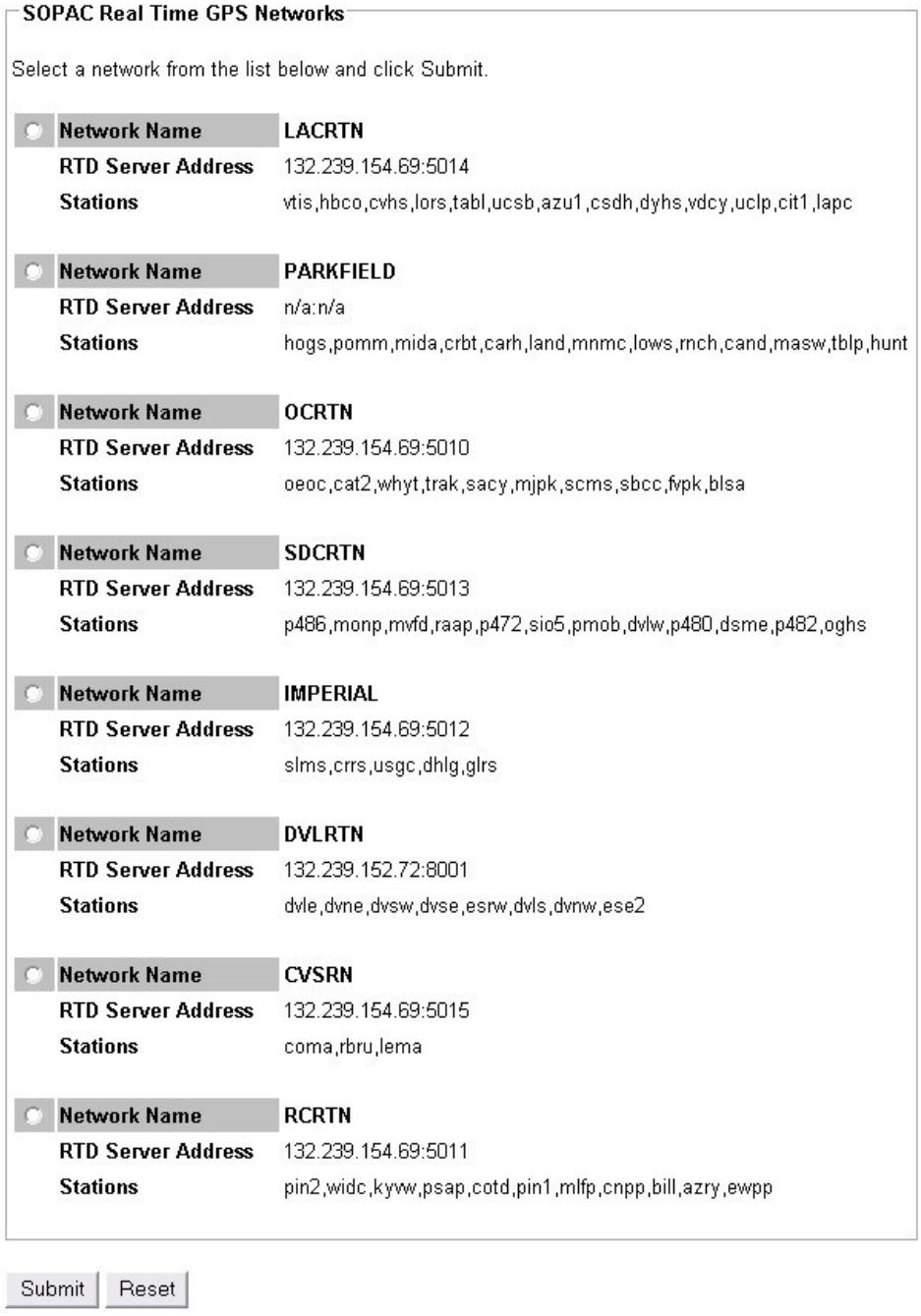
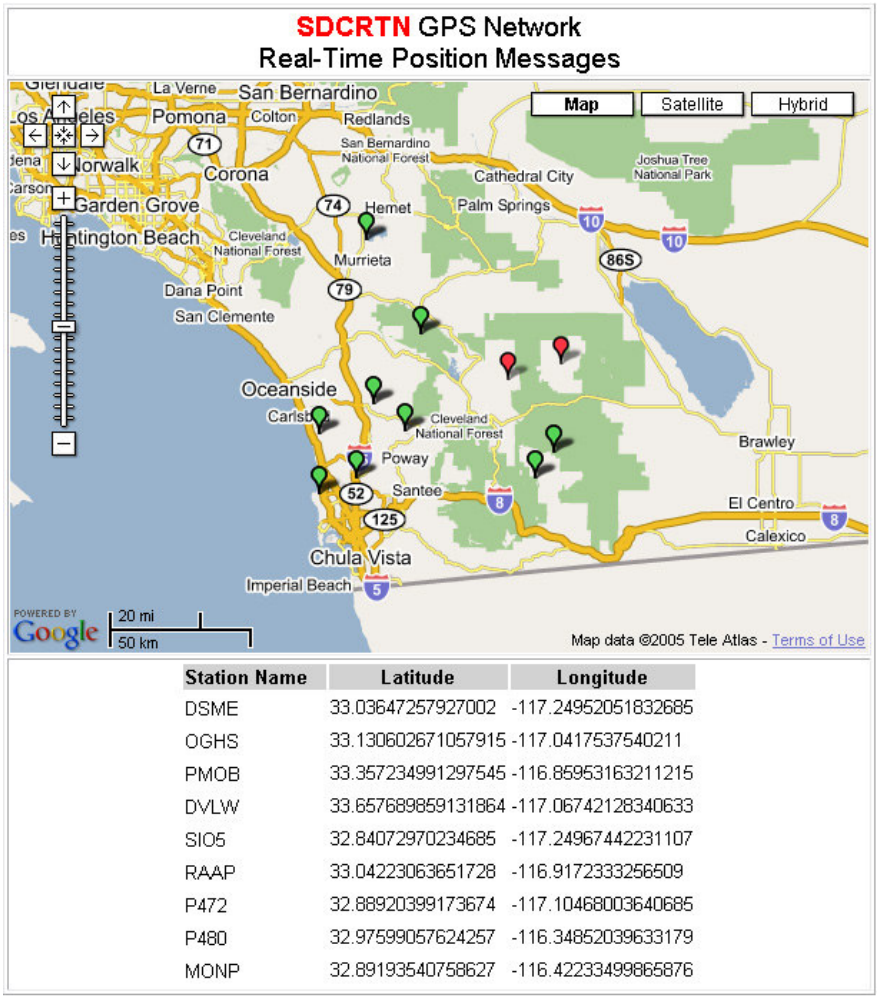


Figure 6-15 - Network selection page for AJAX and Google Maps Demo

Once the user selects a network and clicks the Submit button a server side Java Bean subscribes to the appropriate NaradaBrokering topic and starts receiving position messages at the same time user is forwarded to the second JSP page which contains a Google Map.



Station Name	Latitude	Longitude
DSME	33.03647257927002	-117.24952051832685
OGHS	33.130602671057915	-117.0417537540211
PMOB	33.357234991297545	-116.85953163211215
DVLW	33.657689859131864	-117.06742128340633
SIO5	32.84072970234685	-117.24967442231107
RAAP	33.04223063651728	-116.9172333256509
P472	32.88920399173674	-117.10468003640685
P480	32.97599057624257	-116.34852039633179
MONP	32.89193540758627	-116.42233499865876

Figure 6-16 – Real-Time Data Display on Google Maps. This figure shows the real-time GPS stations belonging to SDCRTN network on the map and the actual latitude and longitude position at the bottom.

Figure 6-16 shows real-time GPS stations from San Diego County Real-time Network (SDCRTN) on the Google maps. The values shown here are the actual real-time latitude-longitude values of the stations obtained from the RTD server. GPS stations

which did not publish a position message in the previous epoch are represented with red markers while online stations are shown with green markers.

6.5.1 Near Real-Time Data Analysis Display on Google Maps

One of the most significant implications of real-time or near real-time data analysis is the potential capability this gives us to evaluate the results as the event being investigated is actually happening right now. This is perhaps the most important feature the sensors can offer us. Whatever the entity the sensors are measuring the ability to evaluate the data on the fly gives authorities to take action on time. Crisis Management systems may greatly benefit from this capability.

As we have discussed above in RDAHMM integration scenario SensorGrid architecture enables seamless integration of the sensor streams with the data analysis applications. Scientific visualization tools are important group of software for scientists to see and demonstrate the result of data analysis. In this section we discuss two examples which demonstrate the successful coupling of simple visualization applications with real-time data streams.

6.5.1.1 GPS Station Position Changes on Google Maps

One of the cases we can use the GPS station messages is to visualize the movements of the stations. This can be especially helpful to understand the long-term tectonic movements. At the same time observing the status changes of a group of stations in pre-seismic periods may help us correlate small activities with earthquakes.

For these reasons we have developed a simple visualization application to display the position changes of the stations. We have integrated this application with the

SensorGrid as in RDAHMM scenario. A data accumulation filter was deployed to listen to the ASCII position topic of each network and pick up the individual station positions. This filter was set to collect as many as 600 data points (roughly equal to 10 minutes worth of data) and trigger the visualization software to visualize the accumulated data.

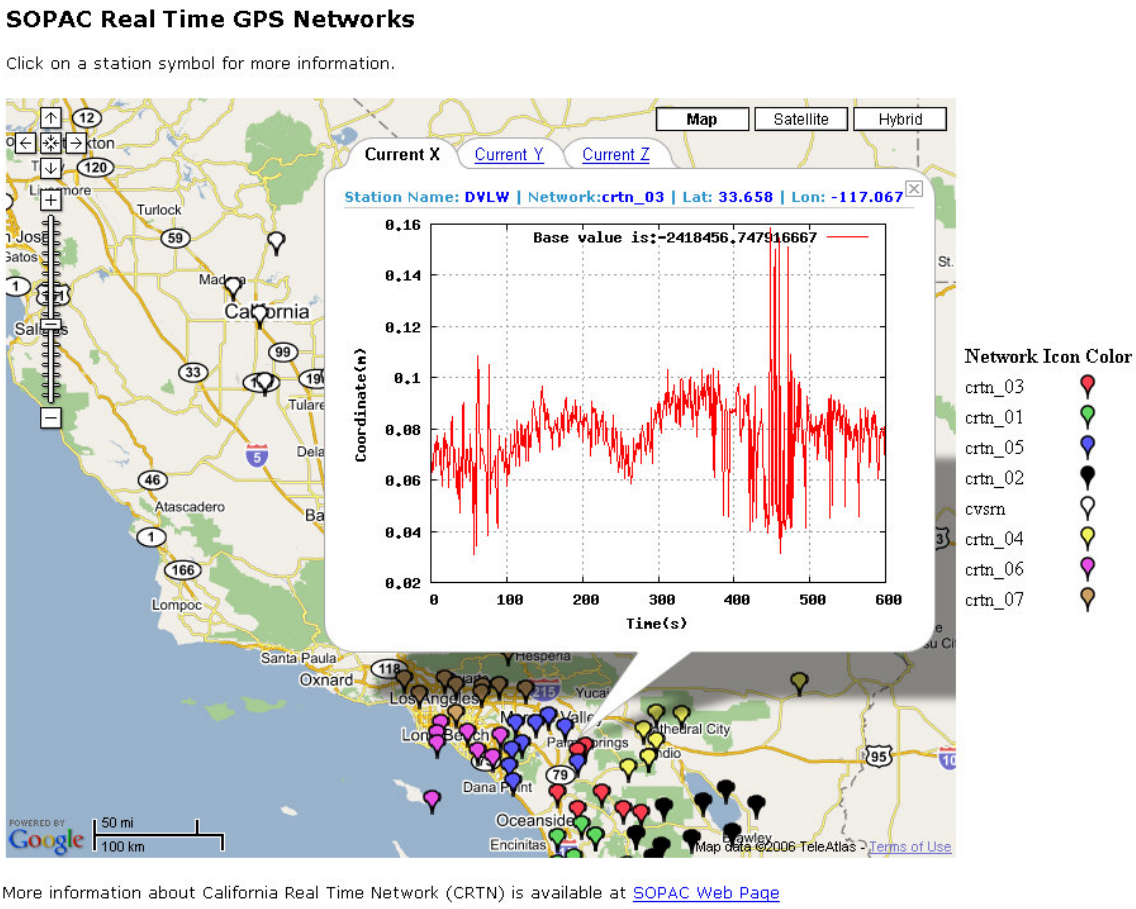


Figure 6-17 – GPS Station Position Changes are displayed on Google Maps. The pop-up window displays the relative position change of the DVLW station for the last 600 measurements or approximately 10 minutes.

We have used Google maps to display the visualization results as shown in Figure 6-17. The data visualization was performed for X, Y and Z positions of each station. The picture shows the result for X axis position change for the last 10 minutes. User can see the Y and Z axis position changes by clicking to the corresponding tab.

6.5.1.2 RDAHMM Analysis Results on Google Maps

The second visualization example is display of RDAHMM data analysis results on Google Maps. As discussed in the second RDAHMM integration scenario the application runs on certain number of data points and creates several output files. We have developed a visualization filter to display the RDAHMM results. For a given time series data RDAHMM detects unique states, and marks them in the output files. The visualization filter is triggered after the RDAHMM analysis was complete after every 600 data points.

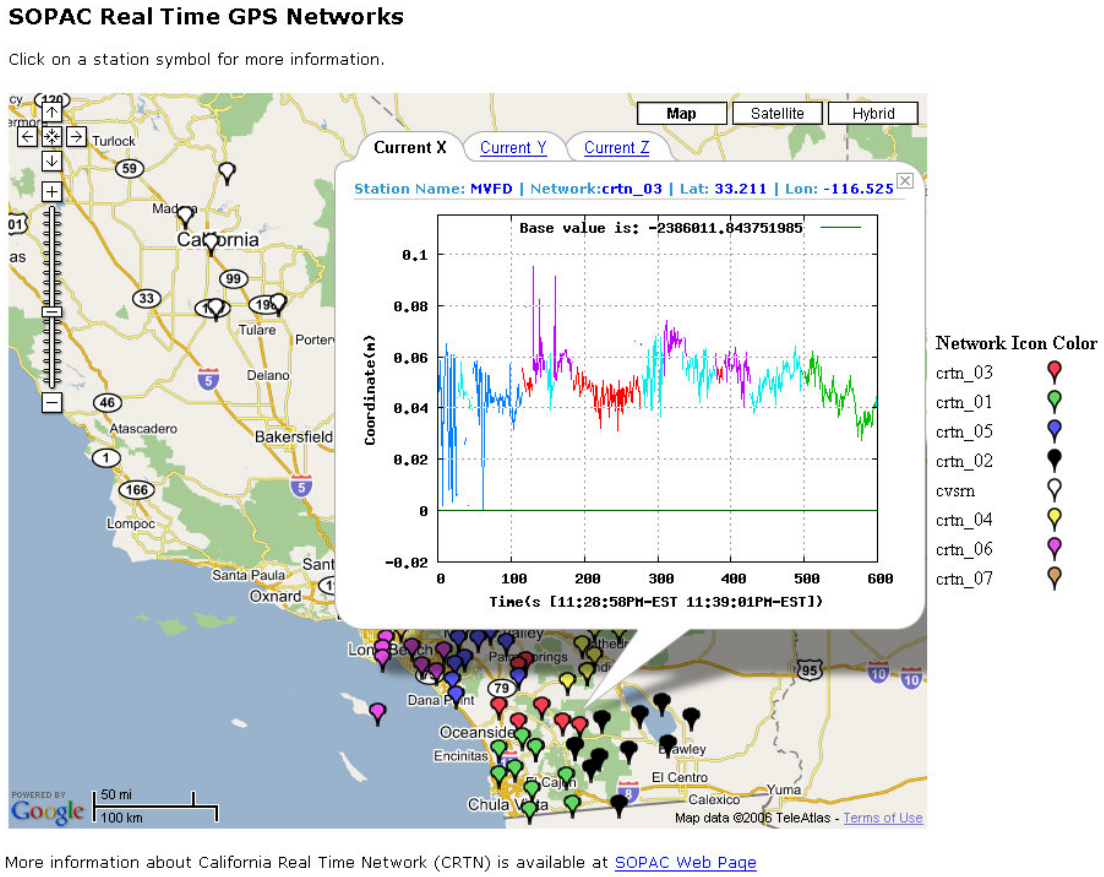


Figure 6-18 – RDAHMM Analysis Results Displayed on Google Maps

A sample result of RDAHMM output visualization is shown in Figure 6-18. As it can be seen from the figure unique states are represented in different colors to help

understand when exactly a state change has occurred. Similar to the previous example the visualization filter is run for X, Y and Z coordinates separately.

6.6 Summary

In this chapter we have discussed a Grid Architecture for coupling real-time data sources with Web Services using a publish-subscribe messaging system. The architecture is based on sequentially deployed filters around a publish-subscribe messaging system to receive and process real-time data streams. This architecture has potential uses for crisis management and rapid response systems. We have also demonstrated successful implementation of our approach for GPS data streams. The system described in this chapter is currently actively being used in several projects by Scripps Institution of Oceanography and NASA scientists.

Chapter 7

Performance and Scalability of the Real-Time Data Grid

7.1 Introduction

In Chapter 6 we have discussed Web Service based filter architecture to couple real-time data sources with applications and presented an implementation of this architecture for permanent Global Positioning System networks. The SensorGrid implementation has so far been used in several projects for archival and real-time data access. In this chapter we discuss Real-Time Data Grid performance and scalability tests. The main goal of these tests is to find out if our filter architecture is scalable for use with large numbers of real time data providers and clients.

Real-time data sources such as sensors are generally used by small number of experts and specific applications. However because the pervasive nature of internet helped diverse user groups to have access to various types of data, sensors can be thought

of the next generation data sources which could supply Web with real-time measurements. The SensorGrid architecture may help make the sensors Web accessible, however this requires the system to support many number of data producers and clients to be served simultaneously. Continuous operation is crucial for the sensor data processing applications which might be used in decision making such as in disaster management, or rapid response cases. Therefore the SensorGrid system should be evaluated carefully both from performance and scalability aspects. The system should be stable enough to work indefinitely and should not be affected by addition or removal of new clients or data sources.

As discussed in the previous chapter our real-time data Grid implementation is based on three types of major components; filter Web Services, Information Services and publish/subscribe messaging system. Both the filters and the messaging system are required components for real-time, continuous and streaming operation. On the other hand the Information Service is usually only used at the beginning of a session for gathering information about the location of the filter Web Services thus is not part of the continuous operation. Therefore the performance of the system is directly affected by the deployed filters and the messaging system. For these reasons we focus the performance tests on the integration of the filters and the messaging substrate.

Perhaps the most obvious unique characteristic of a real-time sensor data Grid from more traditional Grid frameworks is the need for providing support for continuous operation. There should be no interruptions on the data flow, and the message delivery system should be able to handle this type of operation. The filters should be tested for any

type of memory leaks, which could result in interruption of the sensor message processing.

7.2 Testing the Real-Time GPS Data Grid Implementation

In Chapter 6 we have described the implementation of the Real-Time Data Grid architecture for managing GPS data streams. The implementation is built using topic based publish-subscribe messaging system and filter Web Services. In this application domain the GPS streams are made available to the users through a series of filters connected by the NaradaBrokering messaging substrate. The specific application area for the Grid consists of 8 GPS networks each of which contains as many as 10 individual permanent stations. The GPS stations publish their positions regularly once per second. Currently the system supports real-time access to 85 stations maintained by SOPAC. The total number of real-time stations deployed by Southern California Integrated GPS Network (SCIGN) is 250. The trend in this specific field shows that more real-time stations will be deployed around the globe in the near future. This fact is in line with the trends in sensors related developments which shows that increasing numbers of sensors are being used globally. Therefore in the near-future we should expect to have large numbers of sensors producing data and clients plugging into the real-time streams. To support large numbers of GPS networks and customers for prolonged time periods it is important to identify the limits of the system components.

In the simplest setup the system will comprise of a broker and several filters. So the system performance will be mainly affected by the performance of the broker since the

filters will mostly be deployed on different servers. However in some cases where large numbers of filters are run on the same server performance degradation can be expected.

Our tests so far show that we can use a broker for several networks, with running at least 3 filters for each network. We run this setup for over 3 months without any problems. However since the system is supposed to be expandable to support hundreds of clients and tens of GPS networks, we should find the thresholds where the system performance starts to degrade.

The messaging broker in the system is responsible for routing the real-time streams from sources to the subscribers. Since the data is continuously flowing in 1Hz frequency we want the messages to be delivered in less than 1sec before the next message is received, and we do not want any kind of queuing to delay the message delivery. Any queuing lasting more than one second or temporary storage of the messages will be extremely risky since new messages will arrive continuously and the queue will continue to grow, causing delivery failures.

Therefore the performance tests should focus on finding out the maximum number of real-time providers and clients a single broker can support without introducing additional overhead or become unresponsive. There exist limits for the broker in terms of the supportable numbers of publishers or subscribers as well as a maximum data rate.

7.3 Test Methodology

To test the performance of the system we have created a basic setup which consists of several filters and a single broker. In this setup we have three filters: A message forwarding filter to route GPS messages from the RTD server to the NB server, a RYO to

ASCII converter and a simple client filter. In the normal operation we plug into the SOPAC RTD server to receive the GPS messages however for the performance tests we record the raw GPS messages for 24 hours and replay them using another filter.

We wrote two filters for recording and replaying the binary RYO messages: RYO Recorder filter and RYO Publisher filter. The first filter subscribes to a RYO topic and creates daily GPS records by saving incoming messages into files. It creates a new file after midnight, and names it to reflect of which GPS network it holds the records and for which date. For instance a file named `CRTN_01-09_11_2006-12_00_00_AM.ryo` has RYO records of the CRTN_01 network for the date 09/11/2006 and the first sample was collected at 12:00:00 AM.

In our performance tests we use RYO Publisher filter to publish the binary messages in these files to a broker topic. This way we are replacing the actual RTD server with a filter, which allows us to create as many GPS network as we want. The RYO Publisher filter also provides capability to change the message frequency. Currently the actual RTD server publishes network messages at 1Hz frequency, i.e. one message per second is published for each network. By changing a filter parameter we can change this frequency and hence the data flow rate. Considering the fact that in the near future the GPS stations are expected to work on 2Hz frequency, i.e. send their positions twice in a second, this capability of the RYO Publisher filter allows us to simulate future GPS networks.

Overall performance of the system can be estimated by measuring several characteristics:

1- The stability of the system for continuous operation

2- Ability to support multiple data sources

3- Ability to support multiple clients

4- End-to-end message delivery times

5- Ability to preserve the order of the incoming messages

Figure 7-1 depicts the basic system configuration for the performance tests. The test system consists of three filters and a NaradaBrokering server. This is the simplest filter configuration that allows clients to access the GPS messages in human readable format.

The first filter is the RYO Publisher which replaces the RTD server used in real-world operation. The RYO Publisher filter reads a daily RYO archive file and publishes the GPS position messages to a broker topic in 1Hz frequency. The RYO to ASCII Converter filter converts the binary messages into ASCII format and publishes to a new topic; finally the Simple Filter subscribes to this topic and receives them.

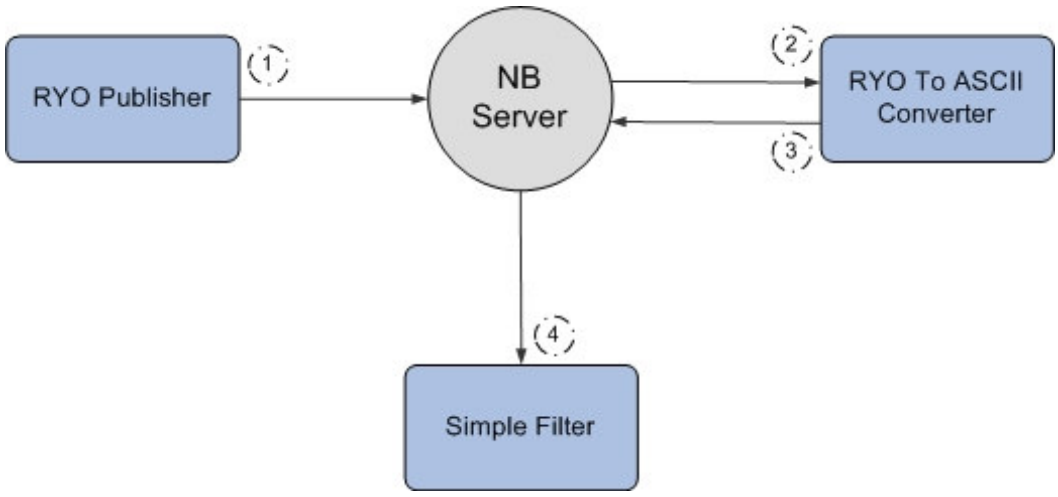


Figure 7-1 – SensorGrid Performance Test Setup includes three real-time filters and a broker

To measure the mean end-to-end delivery time for messages we take measurements at 4 points:

1- Before the message is published by the RYO Publisher,

2- As soon as it is received by the RYO to ASCII converter filter

3- After the format conversion and right before publishing to another topic

4- When it is received by the Simple Filter

The configuration in Figure 7-1 has a complete network path from 1 to 4, but it also includes RYO to ASCII conversion between 3 and 4. So in order to find the actual wire transfer times we subtract the format conversion cost from the total time:

$T_{transfer} = (T_2 - T_1) + (T_4 - T_3)$

Other than the network delay, we also test if the messages are delivered in the correct order. To do this the RYO Publisher marks the outgoing messages in increasing order. It also records the message size, which may affect the transfer time.

We use NaradaBrokering event properties to pass the timestamps and other information from one filter to another. To do this the RYO Publisher creates a string with three values and inserts it as MSGSTAMP property into the NB Event it is about to publish; The first value is the message number, the second number is the size of the message in bytes and the last value is the time stamp in milliseconds. When the subsequent filters receive the NB Event they first extract the MSGSTAMP property and append the current time stamp. This way all publish and subscribe operations in the filter chain will be marked in the MSGSTAMP property. When the final filter receives a

message it just extracts the string then appends its timestamp and saves it in a file for further analysis. Two message stamp samples are given here:

Message Number	Message Size (Bytes)	Time Stamp 1	Time Stamp 2	Time Stamp 3	Time Stamp 4
1	175	1159247077749	1159247078314	1159247078349	1159247078472
2	1561	1159247079030	1159247079034	1159247079058	1159247079063

Table 7-1 – Time Stamps Created for Performance Tests

To measure the five characteristics of the system as described above we identified following test cases:

1- Stability of the system for continuous operation

2- Number of GPS networks that can be supported by a single broker

3- Number of clients that can be supported by a single broker

4- Number of topics that can be supported by a single broker

To eliminate the outliers in the final measurements we recursively apply a Z-filter. Given a number of measurements the Z-filter finds if any particular value is an outlier by using its standard deviation value and the average value of all the entries. For a measurement (x) the formula for the z-filter can be expressed as:

$$Z\_value = abs[t - average] / t_{standard\_deviation}$$

Then the calculated value is compared to a cutoff value, which is usually set to 2.5. If the z-value is greater than the cutoff value then it is considered an outlier and removed from the measurements.

7.4 Test Results

In this section we discuss the test results.

7.4.1 System Stability Test

The first test is to run the system shown in Figure 7-1, for 24 hours and record the timings. At the end of the test we first measure the average message delivery times, and then by dividing the timings into segments, figure out if continuous operation degrades the system performance. We also want to see if the messages will be delivered in the incoming order.

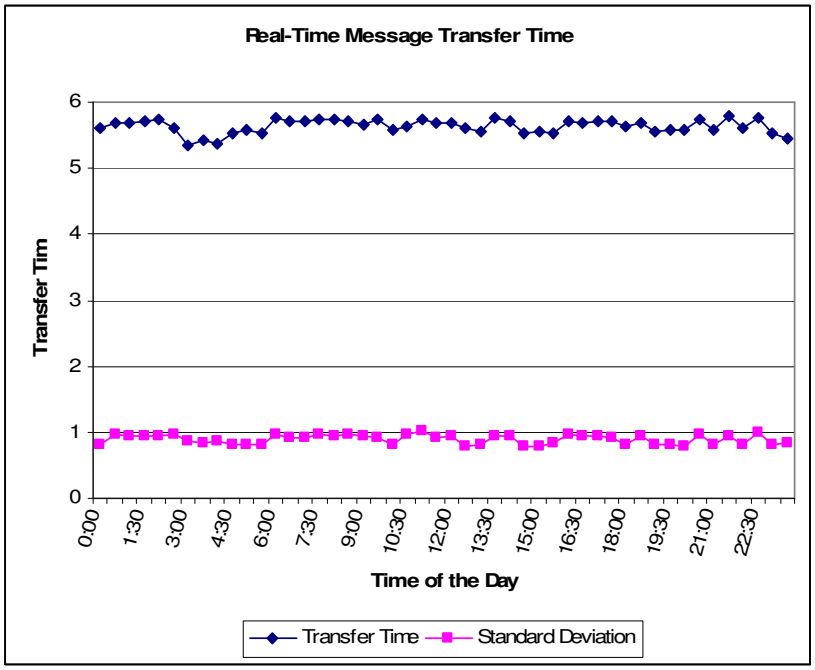


Figure 7-2 – System Stability Test Results for 24-hour operation of the sample test setup.

Figure 7-2 illustrates the results of the first test. The test was run for 24 hours. The last filter in the test setup described in Figure 1 records timings from all steps for each message published every second. We apply a Z-filter to clean the outlier values, and calculate averages for each half hour. Each point in the graph corresponds to 1800 measurements or roughly equal to 30 minutes worth of data. The results show that the

transfer time is stable for average around 5.6ms. Overall the test shows that the continuous operation does not degrade the system performance.

7.4.2 Maximum number of GPS networks a single broker can support

Another important feature the system should provide is to be able to serve multiple publishers simultaneously. This is important for managing GPS streams because there are multiple GPS networks we need to support simultaneously.

For this test we keep the original configuration described in Figure 7-1 intact, and increase the number of the sensor data sources by adding new RYO Publishers. Thus we simulate publishing messages from multiple GPS networks. The result of this test allows us to specify the number of networks one single broker should be used in the real world applications.

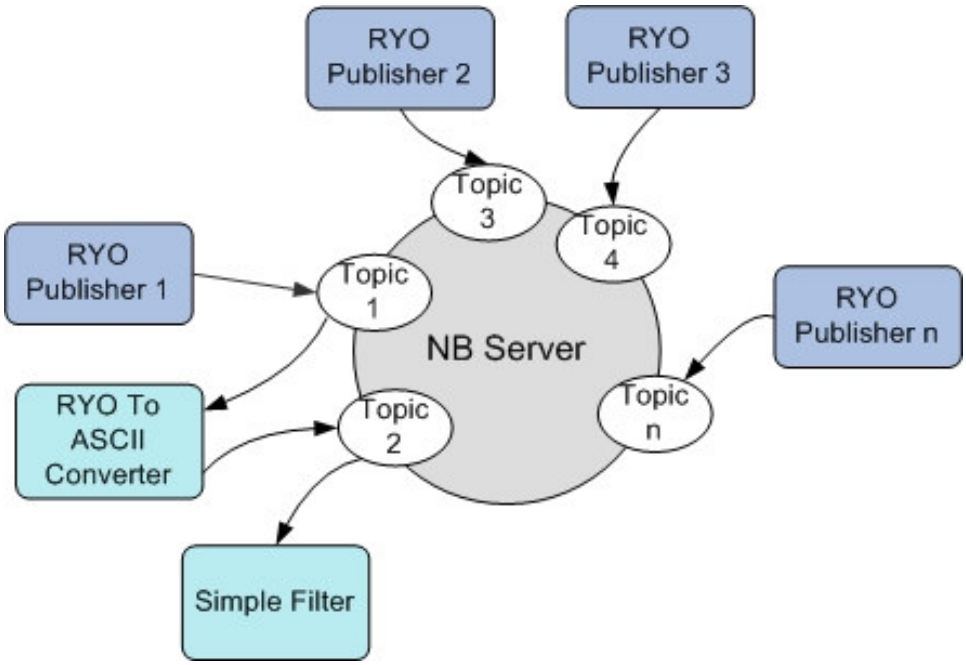


Figure 7-3 – Multi Publisher Test Architecture

Figure 7-3 shows the test architecture described in Figure 7-1 with additional RYO Publishers. Note the NaradaBrokering topics are used to connect filters with each other. The RYO Publisher 1 publishes the raw data to the Topic 1; the RYO to ASCII converter subscribes to this topic and receives the binary messages, which in turn publishes the converted messages to Topic 2. Subsequently the Simple Filter receives the ASCII messages from the Topic 2. For this particular test, we then start new RYO Publisher filters as shown in Figure 7-3. Each of the new publisher filters publishes binary data to a new topic.

We have run this test for two consecutive days. Initially the system consisted of the components shown in Figure 7-1, but after every 30 minutes we started 50 new publishers. The maximum number of the publishers we were able to reach was 1000 which is due to the fact that the maximum number of open-file descriptors the operating system allowed is 1024. At the end of the test, we removed the outliers from the results and divided the results into 1800-entry segments which is roughly equal to 30 minutes of GPS data stream.

As Figure 7-4 and Figure 7-5 show the message delivery time is always stable at around 5ms. We don't observe any unexpected increase or decreases. This shows that even with maximum number of publishers allowed by the broker, the system supports GPS message delivery without any problems.

We also confirmed that the message order is preserved during the test run.

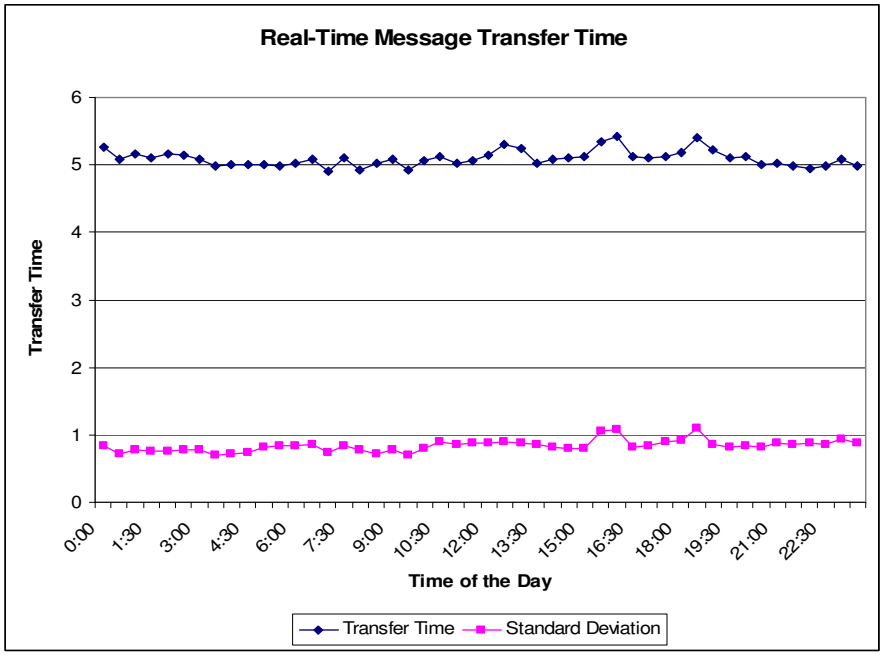


Figure 7-4 – Multiple publisher test results for the first 24 hour

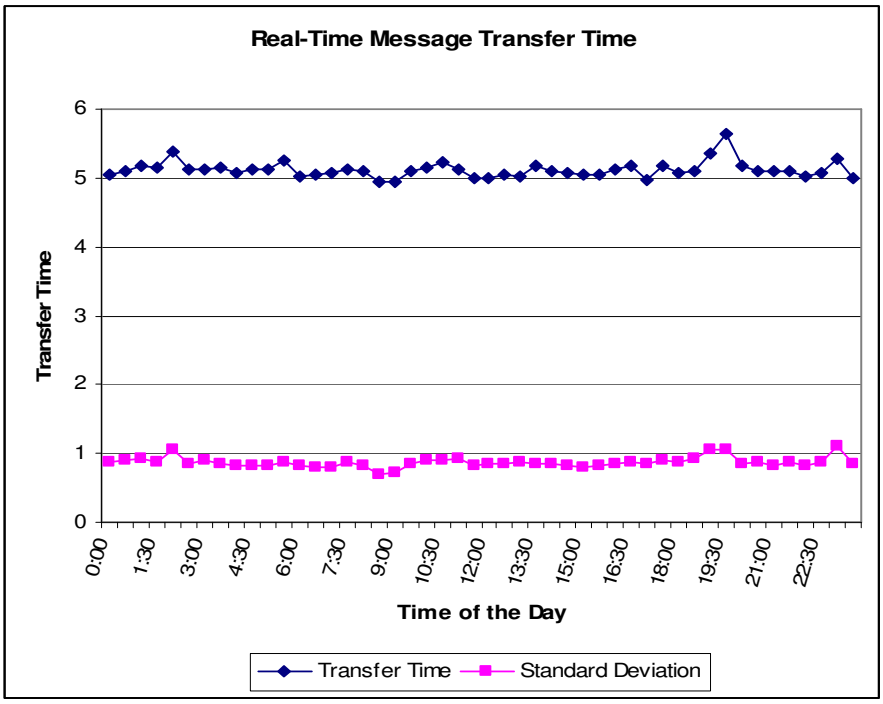


Figure 7-5 – Multiple publisher test for the second 24 hour with 1000 active publishers

7.4.3 Maximum number of clients a single broker can support

In the second test we explored the limits of the system from the data provider side. In the third type of the tests we look at the system from the client's side and try to find the number of clients the basic system described in Figure 7-1 can support. Determining this limit is important for providing uninterrupted real-time data access to large number of clients.

In these tests we will have only one GPS Network publishing the data and increase the number of Simple Filters to simulate the real-time data clients. The result of these tests will allow us to decide when to deploy a new broker if large number of customers plug into the real-time streams. For this test the number of clients can be increased exponentially.

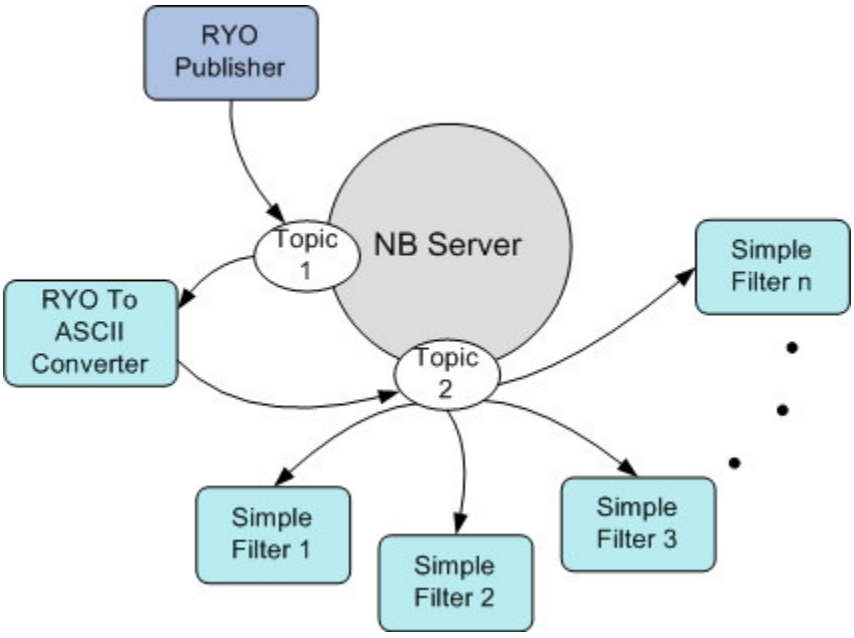


Figure 7-6 – Multi Client Test Architecture

Figure 7-6 shows the system architecture for test 3. Note the RYO Publisher and the RYO to ASCII Converter filters are connected the same way as in test 2, however in this particular test we run multiple Simple Filters which are all connected to the Topic 2. This allows us to see if the system can support distribution of real-time messages to large number of clients without introducing additional overhead. For our GPS applications the message frequency is 1Hz, therefore we expect messages delivered to the clients under 1000ms.

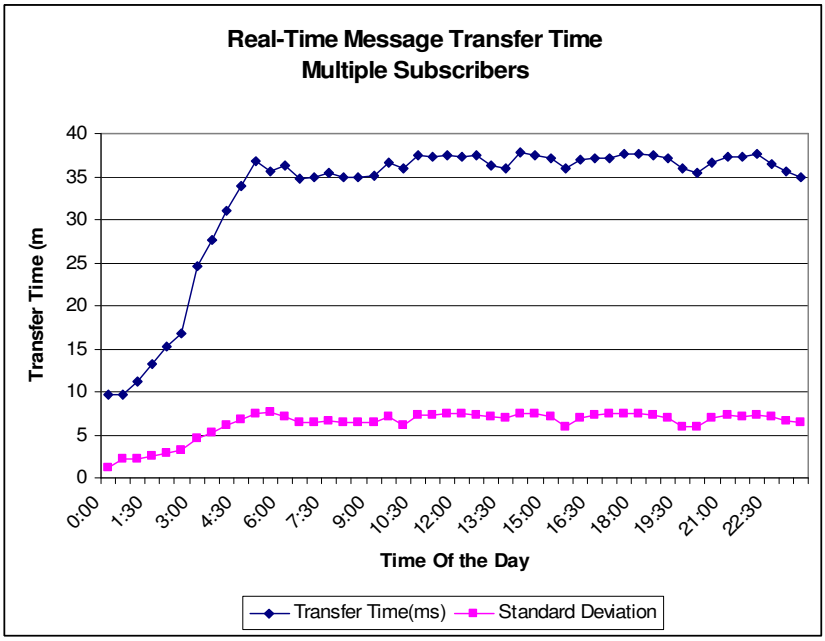


Figure 7-7 – Multiple Subscribers Test Results

Figure 7-7 shows the results of the test 3. The test starts exactly as depicted in Figure 1, but after every 30 minutes we add 100 Simple Filter clients to the system. As a result at the end of the 5th hour there are 1000 subscribers. As explained in test 2 the broker allows only 1024 socket connections to be open simultaneously. Therefore after the 5th hour the system works with 1000 clients for another 19 hours. We run this test for

two consecutive days. The results shown in Figure 7-7 are the average of these two successive tests conducted for two consecutive days.

The test results tell us that the behavior of the broker for multiple clients is different than with multi publishers. In the multi publishers case the average transfer time for GPS messages is almost always around 5ms which is same as the single publisher and single subscriber case described in test 1. Thus we can say that the number of the publishers in the system does not affect the overall performance, as long as the number of the clients do not exceed a certain threshold. On the other hand increasing the number of clients has an obvious effect on the average message distribution time. This is due to the fact that the broker needs to forward messages to many receivers, which takes more time.

Figure 7-7 shows that for every 100 clients subscribing to the same topic in 30 minutes period, the average message delivery time increase a few milliseconds. The total number of clients reaches to 1000 after 5 hours and the average delivery time from the publisher to the final client filter increases to 35ms. Although this is several times higher than the average time measured in tests 1 and 2 it is still acceptable, since the delay between successive GPS messages is 1 second.

To sum up, this test shows that the system with a single broker scales up to a thousand clients with an acceptable transfer delay.

7.4.4 Multiple Broker Test

The test system described in Figure 7-1 is the most basic configuration to provide human readable GPS messages for our application use case. The first three test cases explained in this chapter show that this system is stable for the continuous operation and scales up to the maximum number of file descriptors allowed by the server's operating

system. Both in multiple-publisher and multiple-client tests we identified the limits of the system as 1024 concurrent clients or publishers. Although this is the upper limit of the broker without changing the operating system variables for increasing open file descriptors, the system might need to support many more clients or producers. Therefore we have investigated an alternate approach for increasing the number of the clients to support. This approach is based on creating distributed NaradaBrokering broker network. NaradaBrokering allows creating broker clusters for extending message delivery

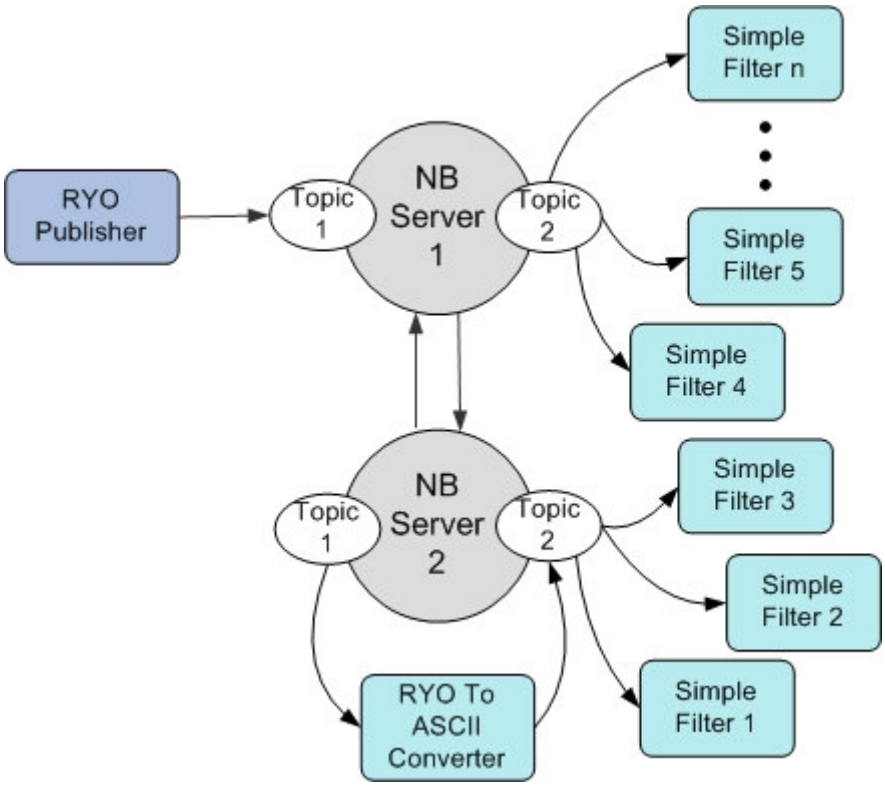


Figure 7-8 – Multiple Broker Test

Figure 7-8 displays the setup for multiple-broker test. For this test we first linked two NaradaBrokering servers with each other utilizing broker's networking capability. This allows brokers to exchange messages and provide access to streams on the same

topics. For instance any message published to Topic-A on the first broker can be retrieved from both Broker-1 and Broker-2 from Topic-A.

This configuration potentially can solve the limits we faced in the previous tests because now we can have as many as 1024 connections on each broker. To prove that we can actually overcome the limits set by the operating system we executed this test by connecting 750 clients to each broker. Thus in total the system was run with 1500 clients for 24-hour period.

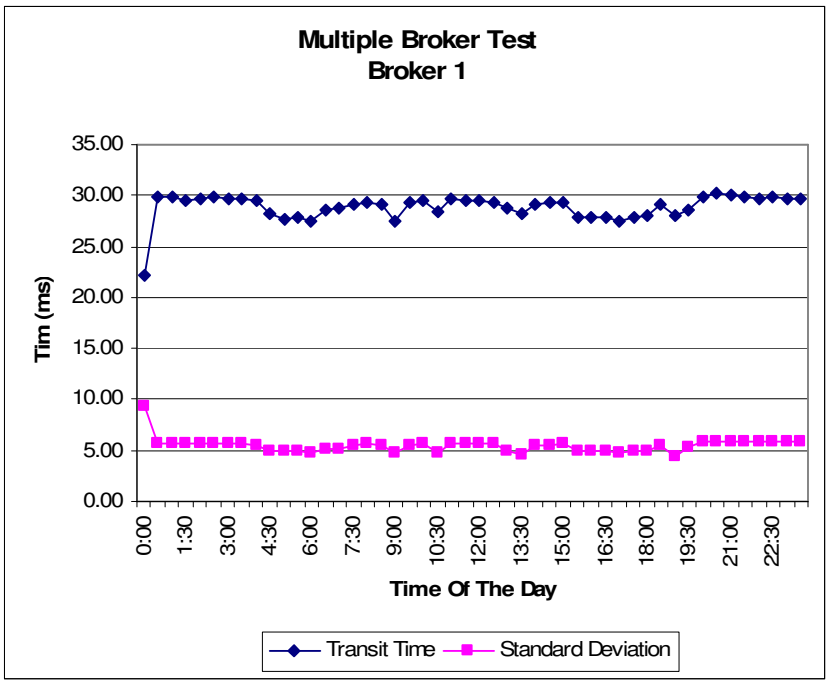


Figure 7-9 – Total transit times for the first broker; Note that in the first 30 minutes we increase the number of the clients to 750 and the average transmit time reaches to 30ms.

To show that multiple-broker configuration is feasible and does not introduce unacceptable overheads we took timing measurements for each broker. Figure 7-9 shows the timings from the Broker-1 which demonstrates similar behavior with the previous test shown in Figure 7-7. Here we see that continuous operation does not degrade the performance.

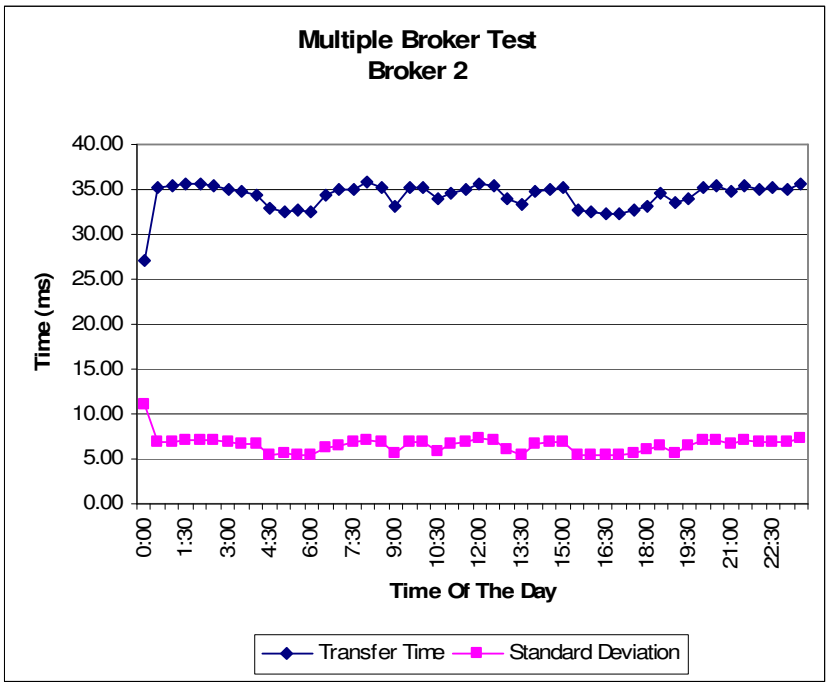


Figure 7-10 – Total transfer times for the second broker; We start 750 clients in the first 30 minutes and the average transfer times reach to 35ms.

Figure 7-10 shows the test results for the second broker. Overall behavior of the Broker-2 is very similar to the Broker-1 except there is a 5ms addition in this case. This 5ms overhead is due to the fact that the messages published to Broker-1 is processed in this broker and sent to Broker-2. Even with this extra overhead the system performance is acceptable and the 35ms transfer time is acceptable since the continuing GPS messages arrive in 1sec or 1000ms intervals.

This test shows that the system can be scaled up by creating NaradaBrokering networks. This potentially means that there is no hard limit on the number of sensors or clients to support.

It should also be noted that for all tests described above we have also checked and confirmed that the message ordering was preserved.

7.5 Summary

In this chapter we have discussed the performance and the stability aspects of our real-time data Grid architecture. We have designed a basic filter system which represents the conventional use of our system to support real-time GPS streams. We have executed several tests to find the limits of the system in terms of the number of sensors and clients that can be supported. The test results have shown that the system can be used up to 1024 clients or sensor streams due to the operating system restrictions. To overcome this limit we designed a multiple-broker test and shown that this setup can support 1500 clients for continuous operation. The last test also shows that the system can be expanded to support as many clients or sensors as required by creating broker networks.

Chapter 8

Conclusion and Future Research Directions

8.1 Thesis Summary

In this thesis we presented a Service Oriented Architecture to provide seamless access to both archival and real-time geographic data. It provides a unified framework with common interfaces for accessing and manipulating distributed geospatial data sources. The architecture is based on Web Services and Grid Messaging paradigms and provides high rate, high performance data transfer options for scientific applications.

The SensorGrid architecture is an example of the Grids of Grids [GOG] paradigm [29]. Grids of Grids are built around Service Oriented Architecture principles by employing family of services and integrating them with common messaging substrates and information services. Our implementation includes Data Grid services for supporting distributed access to stored geospatial data and filter services for accessing real-time

sensor measurements. We used NaradaBrokering publish-subscribe system to implement streaming services for both archival and real-time data access and demonstrated the performance of these services.

We have developed Streaming Web Feature Service for the GIS Data Grid which provides high performance and high rate data transfer options. This service utilizes NaradaBrokering to deliver geographic features to the clients which provides performance improvement over traditional Web Services. Further performance improvements gained by incorporating Binary XML frameworks with this service. Integration of the Binary XML frameworks proved to have improved the performance for greater network distances by a significant margin. We conducted extensive performance tests to analyze the performance of the streaming WFS and determined that it can be used in scenarios where high rate data access is required.

To support Real-Time Data Grids we developed the notion of Sensor Filter Grids. Sensor data sources have unique characteristics which distinguish them from the offline, archival data access. Sensors can be used to understand a geophysical entity in great detail. The measurements obtained from sensors can be evaluated immediately after receiving in real-time, or they can be stored for delayed access. However the trend in GIS world today is moving towards on-the-fly data evaluation which requires stable, flexible and scalable frameworks to support real-time coupling of sensors with computational services. We developed real-time filters for this purpose. We used NaradaBrokering to provide inter-filter communication and hierarchical arrangement of the filters via topics.

We conducted extensive performance and scalability tests to determine the limits of the Real-Time Data Grid. Since one of the most important requirements for a real-time system is stability, we tested the system for this requirement. The tests revealed that the architecture works reliably and addition of new data sources or clients does not significantly affect the performance. The scalability tests demonstrated that the maximum number of data publishers and clients is determined by the system configuration and without any modifications to the operating system parameters it can support as many as 1000 sensor sources or clients simultaneously. We demonstrated that by creating NaradaBrokering networks the system can scale to even larger number of data providers or clients.

8.2 Answers to Research Questions

In this section we give brief answers to the research questions outlined in Chapter 1.

1. **Can we implement unified data-centric Grid architecture to provide common interfaces for accessing real-time and archival geospatial data sources?**

The major contribution SensorGrid architecture makes to the GIS community is creating a scalable, stable and flexible framework for supporting archival and real-time data in data-centric Grids. SensorGrid architecture is developed to fill an important gap existing in GIS community: Lack of a software framework that bridges offline and online data sources. Traditionally Geographic Information Systems have been used to analyze data obtained from spatial databases; however with the sensors propagating in many GIS related fields the need for a real-time data support system is being felt.

In chapter 3 we described the overall architecture of a Grid system that can provide unified access to both types of geospatial data. The viability of this system has been

discussed and proved in subsequent chapters with extensive testing. Several application use cases and real-world applications have shown that the system has been successfully implemented.

2. How can we incorporate widely accepted geospatial industry standards with Web Services?

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 scientific workflows which also proved the usability of these services in real scientific world. This is described in detail in Chapter 4.

3. Are the performance of the Web Services acceptable for Geographic Information Systems and how can we make performance improvements? Can we build services for supporting scientific GIS applications that demand high-performance and high-rate data transfers?

The initial Web Services we created for serving geographic data proved to be useful in use cases where the amount of the requested data is not very large and in such cases where transfer rate is not an issue. However with large data queries the response time of

the Web Services are not satisfactory. Also the limitations on the amount of data that can be served by the Web Service implementations are another issue we have faced.

To create high performance Web Services we have created streaming version of the Web Feature Service. We used the traditional SOAP messaging for initial geospatial query and utilized NaradaBrokering to stream query results. This method improved the performance of the system significantly, also removed any limitations on the size of the data that can be requested. Further improvements made by incorporating popular Binary XML notion. We tested the system with two major Binary XML frameworks and demonstrated that the XML transport performance can be improved significantly.

4. **How can we build a Grid architecture to couple real-time sources with scientific applications that also provides high interactivity and performance?**

Serving real-time sensor measurements using traditional Web Services is not possible since the overhead associated with creating and transferring SOAP messages is larger than the time interval between continuous messages. This is especially true for large network distances. For these reasons we have created real-time filters and Web Service interfaces to control them. The filters we have created are deployed around NaradaBrokering messaging system. We have created XML Schemas to provide metadata descriptions for filters and filter chains. The filters are also controlled via Web Service interfaces, which make it possible to create workflows. We also incorporated information system for users to search and access filters and their capabilities.

5. **What is the way for managing real-time data filters using Web Services?**

Web Services provide standard interfaces for exposing computational resources. We utilized Web Services to provide control interfaces for our real-time filters. The

filters can be remotely deployed, started, stopped or organized through these interfaces. The filter Web Services can also be used by the clients to request the filter metadata files which describe the information such as capabilities of particular filters.

6. **Can we organize and manage real-time sensor data products using publish-subscribe systems? Are the mechanisms of topic based publish/subscribe systems appropriate?**

Considering the sheer number of sensors and sensor networks it is wise to look for a way to organize real-time measurements. We suggested using NaradaBrokering topics to logically organize sensor data which also helps their discovery. We implemented this in GPS data streams and it has been used by field scientists for various applications and demonstrations. We have shown that the topic based publish/subscribe system is appropriate for real-time, continuous data exchange.

7. **Is the performance of the Real-Time Data Grid acceptable for uninterrupted, continuous operation?**

Chapter 7 explains the extensive scalability and performance tests we conducted for understanding the limits of the Real-Time Data Grid implementation. The tests show that the system is stable for continuous operation. Apart from the tests we have been running real-time filters for over 4 months for all of the 8 GPS networks supported by SOPAC. Our experience shows that the SensorGrid implementation can work indefinitely provided no network or hardware related problems occur. The SensorGrid services have also been deployed and run by GPS scientist in SOPAC for the past several months.

8. **Will the Real-Time Data Grid implementation scale for large number of data providers such as sensors and clients?**

Scalability is another issue with the real-time sensor services because of two reasons: First, the number of sensors and sensor networks are already very large and growing rapidly. Hence the system should be able to support addition of large numbers of sensors. Second, the system should be able to serve the measurements in real-time to many clients.

For these reasons we have conducted scalability tests for the real-time filters and explained these tests in Chapter 7. The tests show that the system can scale to as many as 1000 clients or publishers with a single broker. For supporting more numbers we create NaradaBrokering networks, which allow the system to be enlarged infinitely.

8.3 Directions for Future Research

In this thesis we have outlined our initial research and implementations to build a geophysical Grid architecture. We addressed several issues related to archival and real-time data access and processing. At the end of this dissertation we discuss possible future directions for this research.

Although there are several related projects developed or still under development, we think that more research is needed in GIS Grids area. Our work is an example especially aimed towards earthquake science, and 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 sub-domains in the GIS community, and if any improvements are needed in such standards.

Our research for improving the service performance is focused on two fronts: better transport mechanisms, and decreasing the message payload size. We were able to gain

performance improvements in both cases, by integrating publish/subscribe messaging, and binary XML Frameworks. One interesting issue to discover is the second generation Axis Web Service Container (Axis2) which can provide different capabilities to integrate alternative transport protocols. Another issue that should be studied is that although the Binary XML frameworks help in certain cases the current system is based on static, pre-determined selections. Either the client or the server decides which binary format the output GML is to be encoded. A case based reasoning approach, which makes the selection based on the previous events, may help the system choose more appropriate binary encodings and transport mechanisms.

We have also built filter based Grid architecture for real-time data access to be used with sensor streams. The current system is being used in several projects related to GPS sensor streams. The developments in these projects will also shape the SensorGrid research directions. One interesting issue to study is to use Grid workflow tools such as Taverna to create scientific workflows with real-time data streams. Research in this area may help us better understand real-time data analysis requirements.

The scalability tests for our real-time Grid system has shown that by creating NaradaBrokering networks we can support as many sensors and clients as necessary. Therefore it will be useful to have a way for the system to automatically create/deploy required number of brokers and update the system registry on-the-fly.

Appendix A

Sample GML Schemas

A.1 Fault Schema

Following GML 2.1 conformant XML Schema is developed for describing California Faults. The schema is based on the Quake Tables Database as described in [97]

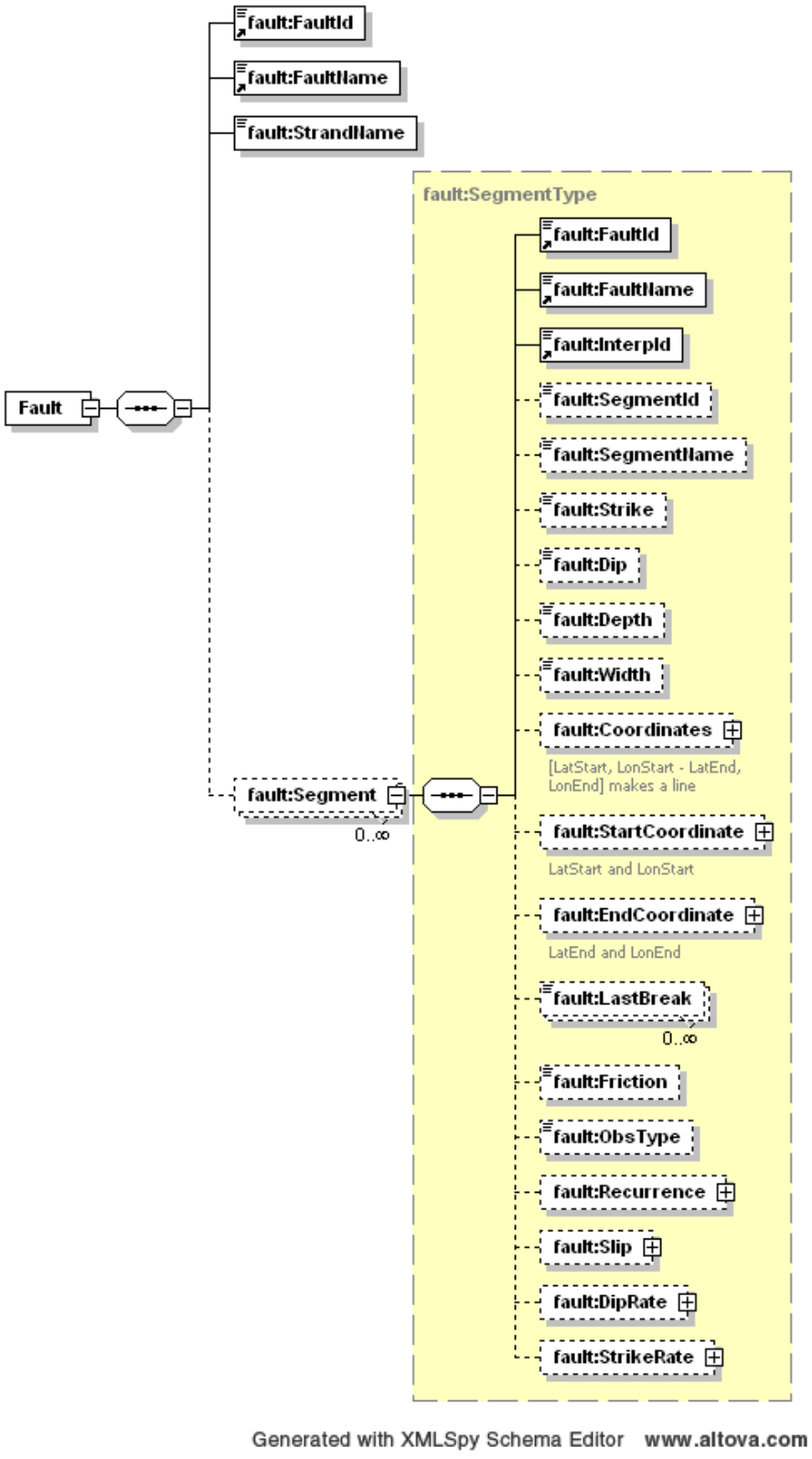


Figure A-1 Fault Schema

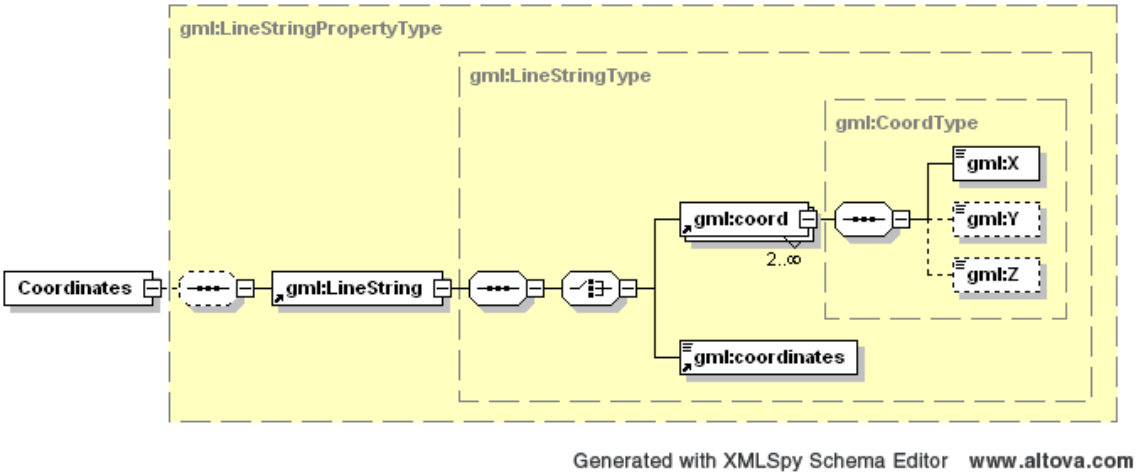


Figure A-2 - Coordinates elements in the Fault Schema is derived from the GML LineString construct

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2006 (http://www.xmlspy.com) by Galip Aydin-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:fault="http://mastar.ucs.indiana.edu/fault"
xmlns:wfs="http://complexity.ucs.indiana.edu/~gaydin/wfs"
xmlns:gml="http://www.opengis.net/gml"
targetNamespace="http://mastar.ucs.indiana.edu/fault"
elementFormDefault="qualified" version="0.1">
	<xs:import namespace="http://www.opengis.net/gml"
schemaLocation="http://complexity.ucs.indiana.edu/~gaydin/ogc/original/gml/2.1.
2/feature.xsd"/>
	<xs:element name="Fault">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="fault:FaultId"/>
				<xs:element ref="fault:FaultName"/>
				<xs:element name="StrandName" type="xs:string"/>
				<xs:element name="Segment" type="fault:SegmentType"
minOccurs="0" maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:complexType name="FaultType">
		<xs:sequence>
			<xs:element ref="fault:FaultId"/>
			<xs:element ref="fault:FaultName"/>
			<xs:element name="StrandName" type="xs:string"/>
			<xs:element name="Segment" type="fault:SegmentType"
minOccurs="0" maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="Segment" type="fault:SegmentType"/>
	<xs:complexType name="SegmentType">
		<xs:sequence>
			<xs:element ref="fault:FaultId"/>
			<xs:element ref="fault:FaultName"/>
			<xs:element ref="fault:InterpId"/>
			<xs:element name="SegmentId" type="xs:int" minOccurs="0"/>
```

```
                        <xs:element name="SegmentName" type="xs:string"
minOccurs="0"/>
                        <xs:element name="Strike" type="xs:float" minOccurs="0"/>
                        <xs:element name="Dip" type="xs:float" minOccurs="0"/>
                        <xs:element name="Depth" type="xs:float" minOccurs="0"/>
                        <xs:element name="Width" type="xs:float" minOccurs="0"/>
                        <xs:element name="Coordinates"
type="gml:LineStringPropertyType" minOccurs="0">
                              <xs:annotation>
                                    <xs:documentation>[LatStart, LonStart -
LatEnd, LonEnd] makes a line</xs:documentation>
                              </xs:annotation>
                        </xs:element>
                        <xs:element name="StartCoordinate"
type="gml:PointPropertyType" minOccurs="0">
                              <xs:annotation>
                                    <xs:documentation>LatStart and LonStart
</xs:documentation>
                              </xs:annotation>
                        </xs:element>
                        <xs:element name="EndCoordinate"
type="gml:PointPropertyType" minOccurs="0">
                              <xs:annotation>
                                    <xs:documentation>LatEnd and
LonEnd</xs:documentation>
                              </xs:annotation>
                        </xs:element>
                        <xs:element name="LastBreak" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
                        <xs:element name="Friction" type="xs:float" minOccurs="0"/>
                        <xs:element name="ObsType" type="xs:int" minOccurs="0"/>
                        <xs:element name="Recurrence" type="fault:RecurrenceType"
minOccurs="0"/>
                        <xs:element name="Slip" type="fault:SlipType"
minOccurs="0"/>
                        <xs:element name="DipRate" type="fault:DipRateType"
minOccurs="0"/>
                        <xs:element name="StrikeRate" type="fault:StrikeRateType"
minOccurs="0"/>
                  </xs:sequence>
            </xs:complexType>
            <xs:element name="Reference" type="fault:RefType"/>
            <xs:element name="LReference" type="fault:RefType"/>
            <xs:complexType name="RefType">
                  <xs:sequence>
                        <xs:element ref="fault:InterpId"/>
                        <xs:element name="Author" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
                        <xs:element name="Publication" type="xs:string"
minOccurs="0"/>
                        <xs:element name="Year" type="xs:string" minOccurs="0"/>
                        <xs:element name="Title" type="xs:string" minOccurs="0"/>
                        <xs:element name="Volume" type="xs:string" minOccurs="0"/>
                        <xs:element name="Number" type="xs:string" minOccurs="0"/>
                        <xs:element name="Pages" type="xs:string" minOccurs="0"/>
                        <xs:element name="Comment" type="xs:string" minOccurs="0"/>
                  </xs:sequence>
            </xs:complexType>
            <xs:annotation>
                  <xs:documentation>global elements</xs:documentation>
            </xs:annotation>
            <xs:element name="FaultName" type="xs:string"/>
            <xs:element name="FaultId" type="xs:int"/>
```

```
	<xs:element name="InterpId" type="xs:int"/>
	<!--============================================================-->
	<!--======= General Magnitude Types - Max Min and Average ============-->
	<xs:complexType name="RateType">
		<xs:sequence minOccurs="0">
			<xs:element name="Max" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
			<xs:element name="Min" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
			<xs:element name="Average" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
	<!--========================= Recurrence ==============================-
->
	<xs:complexType name="RecurrenceType">
		<xs:complexContent>
			<xs:extension base="fault:RateType"/>
		</xs:complexContent>
	</xs:complexType>
	<!--========================= Slip ================================-->
	<xs:complexType name="SlipType">
		<xs:sequence>
			<xs:element name="SlipRate" type="fault:SlipRateType"
minOccurs="0"/>
			<xs:element name="SlipType" type="fault:SlipTypeType"
minOccurs="0"/>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="SlipRateType">
		<xs:complexContent>
			<xs:extension base="fault:RateType"/>
		</xs:complexContent>
	</xs:complexType>
	<xs:complexType name="SlipTypeType">
		<xs:choice minOccurs="0">
			<xs:element name="StrikeSlip" type="fault:RateType"
minOccurs="0"/>
			<xs:element name="DipSlip" type="fault:RateType"
minOccurs="0"/>
		</xs:choice>
	</xs:complexType>
	<!--========================= Dip ================================-->
	<xs:complexType name="DipRateType">
		<xs:complexContent>
			<xs:extension base="fault:RateType"/>
		</xs:complexContent>
	</xs:complexType>
	<!--========================= Strike ==============================--
>
	<xs:complexType name="StrikeRateType">
		<xs:complexContent>
			<xs:extension base="fault:RateType"/>
		</xs:complexContent>
	</xs:complexType>
	<xs:element name="Layer" type="fault:LayerType"/>
	<xs:complexType name="LayerType">
		<xs:sequence>
			<xs:element ref="fault:InterpId"/>
			<xs:element name="LayerId" type="xs:int" minOccurs="0"/>
			<xs:element name="LayerName" type="xs:string"
minOccurs="0"/>
```

```
                    <xs:element name="LatOrigin" type="gml:PointPropertyType"
minOccurs="0"/>
                    <xs:element name="LonOrigin" type="gml:PointPropertyType"
minOccurs="0"/>
                    <xs:element name="Datum" type="xs:string" minOccurs="0"/>
                    <xs:element name="Origin" type="gml:PointPropertyType"
minOccurs="0"/>
                    <xs:element name="Length" type="xs:float" minOccurs="0"/>
                    <xs:element name="Width" type="xs:float" minOccurs="0"/>
                    <xs:element name="Depth" type="xs:float" minOccurs="0"/>
                    <xs:element name="LameLambda" type="xs:float"
minOccurs="0"/>
                    <xs:element name="LameLambdaUnits" type="xs:string"
minOccurs="0"/>
                    <xs:element name="LameMu" type="xs:float" minOccurs="0"/>
                    <xs:element name="LameMuUnits" type="xs:string"
minOccurs="0"/>
                    <xs:element name="Viscosity" type="xs:float"
minOccurs="0"/>
                    <xs:element name="ViscosityUnits" type="xs:string"
minOccurs="0"/>
                    <xs:element name="ViscosityExponent" type="xs:float"
minOccurs="0"/>
                </xs:sequence>
        </xs:complexType>
        <xs:element name="Coordinates" type="gml:LineStringPropertyType"/>
</xs:schema>
```

A.2 GPS Station Schema

This XML Schema imports GML 2.1 schemas to describe metadata about the GPS Stations.

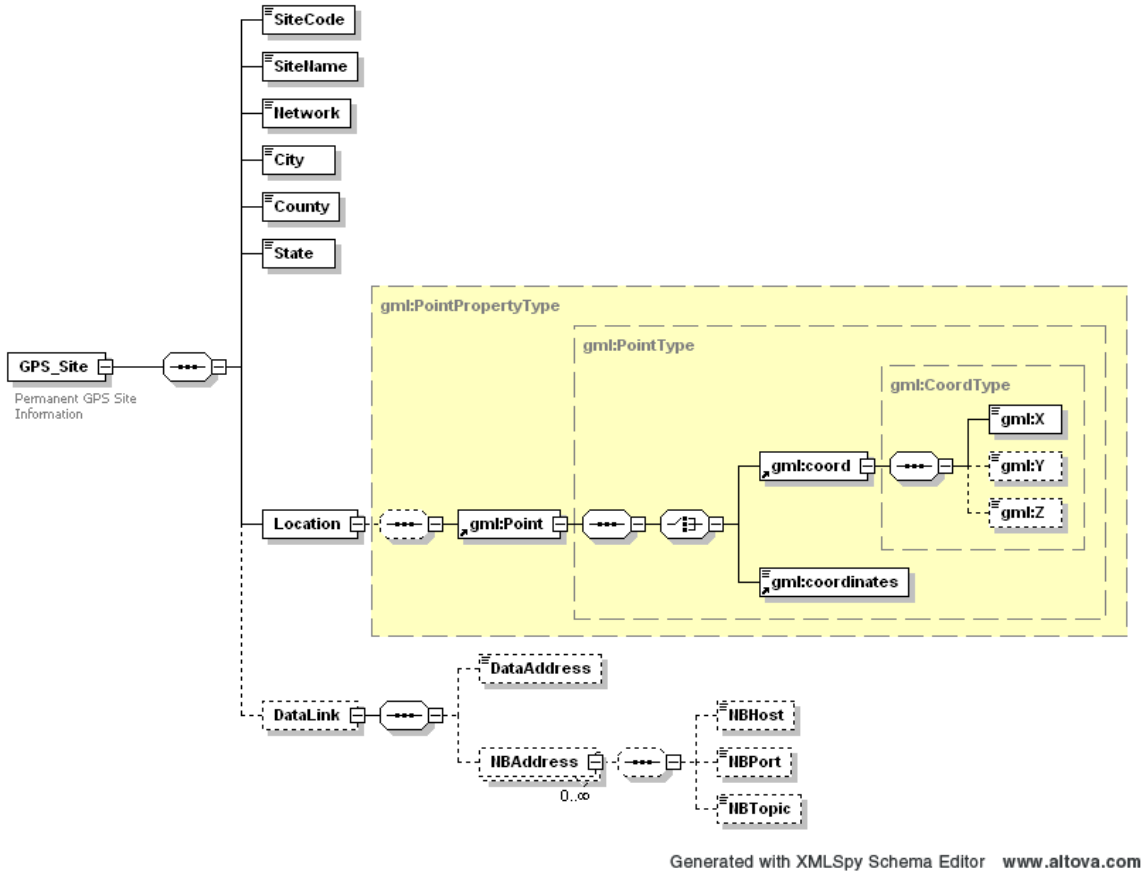


Figure A-3 GPS Station Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:gml="http://www.opengis.net/gml" elementFormDefault="qualified"
attributeFormDefault="unqualified">
<xs:import namespace="http://www.opengis.net/gml" schemaLocation="F:\
schemas\ogc_schemas\gml\2.1.2\feature.xsd"/>
<xs:import namespace="http://www.opengis.net/gml" schemaLocation="F:\
schemas\ogc_schemas\gml\2.1.2\geometry.xsd"/>
<xs:element name="GPS_Site">
       <xs:annotation>
              <xs:documentation>Permanent GPS Site
Information</xs:documentation>
       </xs:annotation>
       <xs:complexType>
              <xs:sequence>
       <xs:element name="SiteCode" type="xs:string"/>
       <xs:element name="SiteName" type="xs:string"/>
       <xs:element name="Network" type="xs:string"/>
       <xs:element name="City" type="xs:string"/>
```

```
        <xs:element name="County" type="xs:string"/>
        <xs:element name="State" type="xs:string"/>
        <xs:element name="Location" type="gml:PointPropertyType"/>
        <xs:element name="DataLink" minOccurs="0">
         <xs:complexType>
               <xs:sequence>
                           <xs:element name="DataAddress" minOccurs="0">
                                 <xs:complexType>
                                       <xs:simpleContent>
                                             <xs:extension base="xs:string">
                                                   <xs:attribute name="type"
type="xs:string"/>
                                             </xs:extension>
                                       </xs:simpleContent>
                                 </xs:complexType>
                           </xs:element>
                           <xs:element name="NBAddress" minOccurs="0"
maxOccurs="unbounded">
                                 <xs:complexType>
                                       <xs:sequence minOccurs="0">
                                             <xs:element name="NBHost"
type="xs:string" minOccurs="0"/>
                                             <xs:element name="NBPort"
type="xs:string" minOccurs="0"/>
                                             <xs:element name="NBTopic"
type="xs:string" minOccurs="0"/>
                                       </xs:sequence>
                                       <xs:attribute name="type"
type="xs:string"/>
                                 </xs:complexType>
                           </xs:element>
                     </xs:sequence>
               </xs:complexType>
        </xs:element>
</xs:sequence>
        </xs:complexType>
</xs:element>
</xs:schema>
```

A.3 Seismicity Schema

Following schema describes the metadata about the seismic events based on several formats. Note the location elements are constructed from GML types.

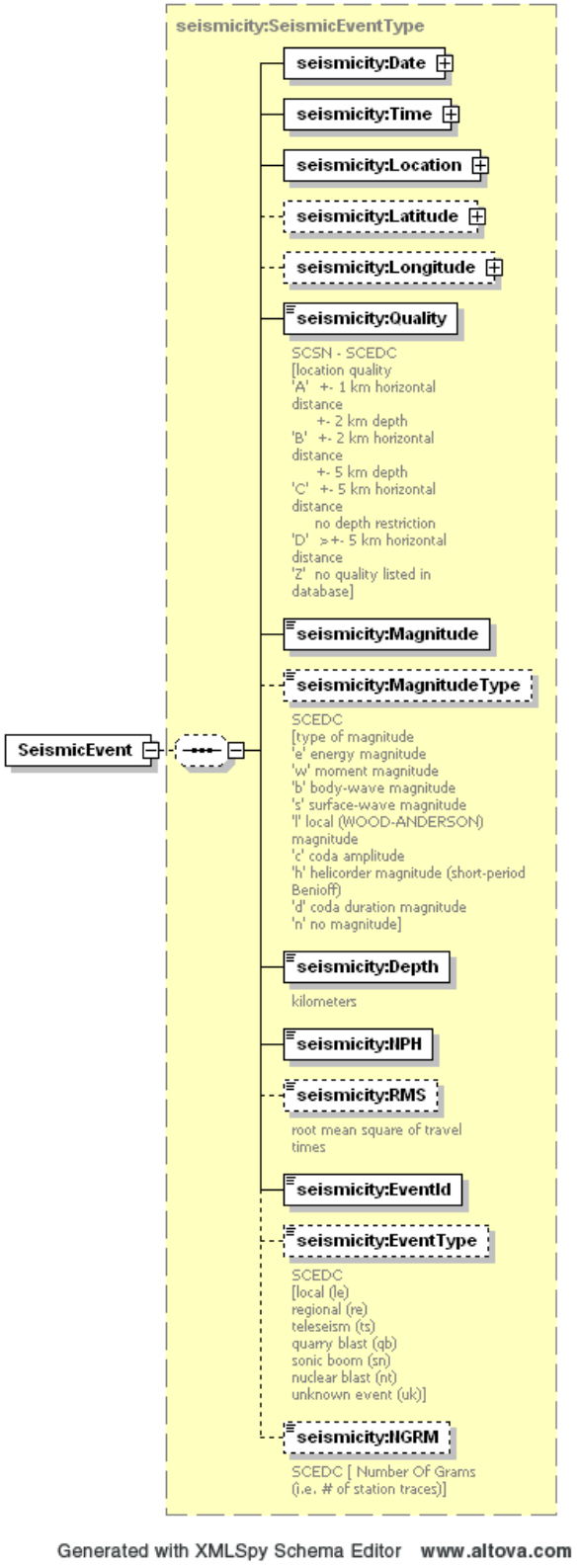


Figure A-4 – Seismicity schema for describing earthquakes and related metadata

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 2 U (http://www.xmlspy.com) by Galip Aydin -->
<schema xmlns:seismicity="http://mastar.ucs.indiana.edu/seismicity"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:gml="http://www.opengis.net/gml" xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://mastar.ucs.indiana.edu/seismicity"
elementFormDefault="qualified" version="3.0">
      <!-- import GML 2.1.2 feature and geometry schemas -->
      <import namespace="http://www.opengis.net/gml"
schemaLocation="F:\phd\schemas\ogc_schemas\gml\2.1.2\geometry.xsd"/>
      <import namespace="http://www.opengis.net/gml"
schemaLocation="F:\phd\schemas\ogc_schemas\gml\2.1.2\feature.xsd"/>
      <element name="SeismicEvent" type="seismicity:SeismicEventType"/>
      <complexType name="SeismicEventType">
            <sequence minOccurs="0">
                  <element name="Date">
                        <complexType>
                              <sequence minOccurs="0">
                                    <element name="Year" minOccurs="0"/>
                                    <element name="Month" minOccurs="0"/>
                                    <element name="Day" minOccurs="0"/>
                              </sequence>
                              <attribute name="DateContent"/>
                        </complexType>
                  </element>
                  <element name="Time">
                        <complexType>
                              <sequence minOccurs="0">
                                    <element name="Hour" minOccurs="0"/>
                                    <element name="Minute" minOccurs="0"/>
                                    <element name="Second" minOccurs="0"/>
                              </sequence>
                              <attribute name="TimeContent"/>
                        </complexType>
                  </element>
                  <element name="Location" type="gml:PointPropertyType"/>
                  <element name="Latitude" type="gml:PointPropertyType"
minOccurs="0"/>
                  <element name="Longitude" type="gml:PointPropertyType"
minOccurs="0"/>
                  <element name="Quality">
                        <annotation>
                              <documentation>SCSN - SCEDC
[location quality
'A'  +- 1 km horizontal distance
    +- 2 km depth
'B'  +- 2 km horizontal distance
    +- 5 km depth
'C'  +- 5 km horizontal distance
    no depth restriction
'D'  >+- 5 km horizontal distance
'Z'  no quality listed in database]</documentation>
                        </annotation>
                        <simpleType>
                              <restriction base="string">
                                    <enumeration value="A"/>
                                    <enumeration value="B"/>
                                    <enumeration value="C"/>
                                    <enumeration value="D"/>
                                    <enumeration value="Z"/>
                                    <enumeration value="E"/>
                                    <enumeration value="P"/>
                              </restriction>
```

```
                        </simpleType>
                    </element>
                    <element name="Magnitude" type="string"/>
                    <element name="MagnitudeType" minOccurs="0">
                        <annotation>
                            <documentation>SCEDC
[type of magnitude
'e' energy magnitude
'w' moment magnitude
'b' body-wave magnitude
's' surface-wave magnitude
'l' local (WOOD-ANDERSON) magnitude
'c' coda amplitude
'h' helicorder magnitude (short-period Benioff)
'd' coda duration magnitude
'n' no magnitude]</documentation>
                        </annotation>
                        <simpleType>
                            <restriction base="string">
                                <enumeration value="e"/>
                                <enumeration value="w"/>
                                <enumeration value="b"/>
                                <enumeration value="s"/>
                                <enumeration value="l"/>
                                <enumeration value="c"/>
                                <enumeration value="h"/>
                                <enumeration value="d"/>
                                <enumeration value="n"/>
                            </restriction>
                        </simpleType>
                    </element>
                    <element name="Depth" type="string">
                        <annotation>
                            <documentation>kilometers</documentation>
                        </annotation>
                    </element>
                    <element name="NPH">
                        <complexType>
                            <simpleContent>
                                <extension base="int"/>
                            </simpleContent>
                        </complexType>
                    </element>
                    <element name="RMS" type="float" minOccurs="0">
                        <annotation>
                            <documentation>root mean square of travel
times</documentation>
                        </annotation>
                    </element>
                    <element name="EventId" type="int"/>
                    <element name="EventType" minOccurs="0">
                        <annotation>
                            <documentation>SCEDC
[local (le)(re)(ts)blast (qb)boom (sn)blast (nt)event (uk)]</documentation>
                        </annotation>
                        <simpleType>
                            <restriction base="string">
                                <enumeration value="le"/>
                                <enumeration value="re"/>
                                <enumeration value="ts"/>
                                <enumeration value="qb"/>
                                <enumeration value="sn"/>
                                <enumeration value="nt"/>
```

```
                                        <enumeration value="uk"/>
                                    </restriction>
                                </simpleType>
                            </element>
                            <element name="NGRM" type="int" minOccurs="0">
                                <annotation>
                                    <documentation>SCEDC [ Number Of Grams (i.e. #
of station traces)]</documentation>
                                </annotation>
                            </element>
                        </sequence>
                </complexType>
                <element name="Catalog">
                        <complexType>
                            <sequence minOccurs="0">
                                <element ref="seismicity:SeismicEvent" minOccurs="0"
maxOccurs="unbounded"/>
                            </sequence>
                        </complexType>
                </element>
                <!--This Schema is based on GML 2.1.2 and supports following formats-->
                <!--SCSN (SOUTHERN CALIFORNIA SEISMOGRAPHIC NETWORK FORMAT) -->
                <!--SCEDC (SOUTHERN CALIFORNIA EARTHQUAKE DATA CENTER CATALOG FORMAT)-->
</schema>
```

Appendix B

XML Files

In this section we give several sample XML files used or generated by WFS.

B.1 Sample GetFeature Request

```
<?xml version="1.0" encoding="iso-8859-1"?>
<wfs:GetFeature
outputFormat="GML2":gml="http://www.opengis.net/gml":wfs="http://www.op
engis.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>
```

B.2 Sample GetFeature Response

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:FeatureCollection xmlns:wfs="http://www.opengis.net/wfs"
xmlns:gml="http://www.opengis.net/gml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://complexity.ucs.indiana.edu/~gaydin/wfs
C:/Projects/WFS/xml/schemas/fault_new.xsd
http://complexity.ucs.indiana.edu/~gaydin/ogc/original/wfs/1.0.0/WFS-
basic.xsd">
 <gml:boundedBy>
   <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#27354">
     <gml:coordinates decimal="." cs="," ts=" ">>-119.31,35 -
118,38</gml:coordinates>
   </gml:Box>
 </gml:boundedBy>
 <gml:featureMember>
   <fault>
     <name>White Wolf</name>
     <segment>5.0</segment>
     <author>Rundle J. B.</author>
     <gml:lineStringProperty>
       <gml:LineString srsName="null">
         <gml:coordinates>-118.65,35.26 -118.56,35.31</gml:coordinates>
```

```
      </gml:LineString>
    </gml:lineStringProperty>
  </fault>
</gml:featureMember>
<gml:featureMember>
  <fault>
    <name>White Wolf</name>
    <segment>4.0</segment>
    <author>Rundle J. B.</author>
    <gml:lineStringProperty>
      <gml:LineString srsName="null">
        <gml:coordinates>-118.73,35.21 -118.65,35.26</gml:coordinates>
      </gml:LineString>
    </gml:lineStringProperty>
  </fault>
</gml:featureMember>
<gml:featureMember>
  <fault>
    <name>White Wolf</name>
    <segment>3.0</segment>
    <author>Rundle J. B.</author>
    <gml:lineStringProperty>
      <gml:LineString srsName="null">
        <gml:coordinates>-118.82,35.15 -118.73,35.21</gml:coordinates>
      </gml:LineString>
    </gml:lineStringProperty>
  </fault>
</gml:featureMember>
<gml:featureMember>
  <fault>
    <name>White Wolf</name>
    <segment>2.0</segment>
    <author>Rundle J. B.</author>
    <gml:lineStringProperty>
      <gml:LineString srsName="null">
        <gml:coordinates>-118.9,35.1 -118.82,35.15</gml:coordinates>
      </gml:LineString>
    </gml:lineStringProperty>
  </fault>
</gml:featureMember>
<gml:featureMember>
  <fault>
    <name>White Wolf</name>
    <segment>1.0</segment>
    <author>Rundle J. B.</author>
    <gml:lineStringProperty>
      <gml:LineString srsName="null">
        <gml:coordinates>-118.99,35.05 -118.9,35.1</gml:coordinates>
      </gml:LineString>
    </gml:lineStringProperty>
  </fault>
</gml:featureMember>
<gml:featureMember>
  <fault>
    <name>White Mountains</name>
    <segment>10.0</segment>
    <author>Rundle J. B.</author>
```

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

B.3 GetCapabilities Request

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<GetCapabilities version="1.0.0"/>
```

B.4 Sample GetCapabilities Response

```
<?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>boundary_lines</Name>
      <Title>California State Boundary Lines</Title>
      <Abstract/>
      <Keywords>California.Boundary</Keywords>
      <SRS>EPSG:4321</SRS>
      <Operations>
        <Query/>
      </Operations>
      <LatLongBoundingBox minx="-124.376663" miny="-124.376663" maxx="-
118.074822" maxy="38.088043"/>
    </FeatureType>
    <FeatureType>
      <Name>city</Name>
      <Title>USA City Locations</Title>
      <Abstract/>
      <Keywords/>
      <SRS>NULL</SRS>
      <Operations>
        <Query/>
      </Operations>
      <LatLongBoundingBox minx="-157.82343908739" miny="21.305784962263" maxx="-
71.089115" maxy="61.1919"/>
    </FeatureType>
    <FeatureType>
      <Name>tsunami_clusters</Name>
      <Title>Tsunami Hotspot Clusters</Title>
      <Abstract/>
      <Keywords/>
      <SRS>SRS</SRS>
      <Operations>
        <Query/>
      </Operations>
      <LatLongBoundingBox minx="-180" miny="-60.5" maxx="179" maxy="62"/>
    </FeatureType>
    <FeatureType>
      <Name>LANL DEMO</Name>
      <Title>LANL</Title>
      <Abstract>Earthquake data</Abstract>
      <Keywords>LANL</Keywords>
      <SRS>null</SRS>
      <Operations>
        <Query/>
      </Operations>
      <LatLongBoundingBox minx="-85.758003234863" miny="25.363000869751"
maxx="30.700000762939" maxy="30.700000762939"/>
    </FeatureType>
    <FeatureType>
      <Name>LANL DEMO</Name>
      <Title>LANL</Title>
      <Abstract>IEISS Inout Data</Abstract>
      <Keywords>LANL</Keywords>
      <SRS>null</SRS>
      <Operations>
        <Query/>
      </Operations>
      <LatLongBoundingBox minx="-83.932197570801" miny="25.363000869751" maxx="-
80.165802001953" maxy="30.700000762939"/>
    </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>
```

B.5 Sample Mapping File

Following XML mapping file is used by the WFS to map MySQL query results into GML documents.

```
<?xml version="1.0" encoding="UTF-8"?>
<MapElements xmlns:xsi="http:/SeismicEvent/www.w3.org/2001/XMLSchema-instance">
       <MapElement No="0" XSDNodeXPath="/SeismicEvent/Date/Year"
DBColumnName="YEAR"></MapElement>
       <MapElement No="1" XSDNodeXPath="/SeismicEvent/Date/Month"
DBColumnName="MONTH"></MapElement>
       <MapElement No="2" XSDNodeXPath="/SeismicEvent/Date/Day"
DBColumnName="DAY"></MapElement>
       <MapElement No="3" XSDNodeXPath="/SeismicEvent/Time/Hour"
DBColumnName="HOUR"></MapElement>
       <MapElement No="4" XSDNodeXPath="/SeismicEvent/Time/Minute"
DBColumnName="MINUTE"></MapElement>
       <MapElement No="5" XSDNodeXPath="/SeismicEvent/Time/Second"
DBColumnName="SECOND"></MapElement>
       <MapElement No="5" XSDNodeXPath="/SeismicEvent/EventType"
DBColumnName="ET"></MapElement>
       <MapElement No="6" XSDNodeXPath="/SeismicEvent/Magnitude"
DBColumnName="MAGNITUDE"></MapElement>
       <MapElement No="7" XSDNodeXPath="/SeismicEvent/MagnitudeType"
DBColumnName="MAGNITUDE_TYPE"></MapElement>
       <MapElement No="8"
XSDNodeXPath="/SeismicEvent/Location/gml:Point/gml:coord/gml:X"
DBColumnName="LATITUDE"></MapElement>
       <MapElement No="9"
XSDNodeXPath="/SeismicEvent/Location/gml:Point/gml:coord/gml:Y"
DBColumnName="LONGITUDE"></MapElement>
       <MapElement No="10" XSDNodeXPath="/SeismicEvent/Depth"
DBColumnName="DEPTH"></MapElement>
       <MapElement No="11" XSDNodeXPath="/SeismicEvent/Quality"
DBColumnName="QUALITY"></MapElement>
       <MapElement No="12" XSDNodeXPath="/SeismicEvent/EventId"
DBColumnName="EVID"></MapElement>
```

```
       <MapElement No="13" XSDNodeXPath="/SeismicEvent/NPH"
DBColumnName="NPH"></MapElement>
       <MapElement No="14" XSDNodeXPath="/SeismicEvent/NGRM"
DBColumnName="NGRM"></MapElement>
</MapElements>
```

B.6 Sample Feature Configuration File

Following configuration file is used by the WFS for obtaining database, schema and other information about a feature.

```
<?xml version="1.0" encoding="UTF-8"?>
<feature>
 <db>
   <type>mySQL</type>
   <serveraddress>gf8.ucs.indiana.edu</serveraddress>
   <dbname>cce</dbname>
   <tablename>scedc</tablename>
   <driver>com.mysql.jdbc.Driver</driver>
   <username>galip</username>
   <password>password</password>
 </db>
 <xml_instance>
   <localaddress>C:/projects/newprojects/wfs-
streaming/xml/galip/seismicity/seismic_instance.xml</localaddress>
 </xml_instance>
 <map_file>
   <localaddress>C:/projects/newprojects/wfs-
streaming/xml/galip/seismicity/scedc_mapping.xml</localaddress>
 </map_file>
 <xmlschema>
   <localaddress>C:/projects/newprojects/wfs-
streaming/xml/schemas/seismicity.xsd</localaddress>
 </xmlschema>
 <maxmin_column_names>
   <minx>LONGITUDE</minx>
   <miny>LATITUDE</miny>
   <maxx>LONGITUDE</maxx>
   <maxy>LATITUDE</maxy>
 </maxmin_column_names>
 <Metadata>
   <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>
     <Operation type="Query"/>
   </Operations>
   <MetadataURL>http://www.crisisgrid.org</MetadataURL>
 </Metadata>
</feature>
```

B.7 Sample GML instance

Following file is used by the WFS to map MySQL query results into GML documents.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<GPS_Site xmlns:gml="http://www.opengis.net/gml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\projects\wfs\schemas\GPS_Site.xsd">
      <SiteCode></SiteCode>
      <SiteName></SiteName>
      <Network></Network>
      <City></City>
      <County></County>
      <State></State>
      <Location>
            <gml:Point srsName="WGS84">
                  <gml:coord>
                        <gml:X></gml:X>
                        <gml:Y></gml:Y>
                  </gml:coord>
            </gml:Point>
      </Location>
      <DataLink>
            <DataAddress type="Streaming"></DataAddress>
            <NBAddress>
                  <NBHost></NBHost>
                  <NBPort></NBPort>
                  <NBTopic></NBTopic>
            </NBAddress>
      </DataLink>
</GPS_Site>
```

Bibliography

1. [AHD], *geography. (n.d.). The American Heritage® Dictionary of the English Language, Fourth Edition. Retrieved November 05, 2006, from Dictionary.com website: http://dictionary.reference.com/browse/geography.*
2. Peng, Z.R. and M. Tsou, *Internet GIS: Distributed Geographic Information Services for the Internet and Wireless Networks*. 2003: Wiley.
3. FGDC. *The Federal Geographic Data Committee, Data & Services: http://www.fgdc.gov/dataandservices*. [cited.
4. Smith, T.R., *A digital library for geographically referenced materials.* Computer, 1996. **29**(5): p. 54.
5. Zao Liu, Marlon Pierce, and G. Fox, *Concurrent Web Map Cache Server, Community Grids Lab Presentation, Available from http://grids.ucs.indiana.edu/ptliupages/presentations/CacheServer.ppt*. 2006.
6. ESRI, *ArcIMS, 9 Architecture and Functionality, J-8694. ESRI White Paper, http://downloads.esri.com/support/whitepapers/ims_/arcims9-architecture.pdf*. 2004.
7. Autodesk. *MapGuide http://usa.autodesk.com*. [cited.
8. MapServer, W. *http://www.wthengineering.com/GIS/web_gis.htm*. [cited.
9. Inc, E., *ESRI Shapefile Technical Description, An ESRI White Paper—July 1998*. 1998, URL: http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf.
10. Cox, S., et al., *Geography Markup Language (GML) 2.0, OpenGIS® Implementation Specification, 20 February 2001, OGC Document Number: 01-029*. 2001.
11. GeoCommunity. *Web Site: http://www.geocomm.com/, Last accessed on 12/17/2006*. [cited.
12. 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.
13. Fox, G. and M. Pierce. *Web Service Grids for iSERVO*. in *International Workshop http://www.eps.s.u-tokyo.ac.jp/jp/COE21/events/20041014.pdf on Geodynamics: Observation, Modeling and Computer Simulation University of Tokyo Japan October 14 2004*. 2004.
14. de La Beaujardiere, J., *Web Map Service, OGC project document reference number OGC 04-024.* 2004.
15. Vretanos, P. (2002) *Web Feature Service Implementation Specification, OpenGIS project document: OGC 02-058, version 1.0.0.* **Volume**,
16. Cox, S., et al. (2003) *OpenGIS Geography Markup Language (GML) Implementation Specification, OpenGIS project document reference number OGC 02-023r4, Version 3.0.* **Volume**,
17. Sayar, A., M. Pierce, and G. Fox, *OGC Compatible Geographical Information Services*, in *Indiana Computer Science Report TR610*. 2005.
18. Cox, S. (2003) *Observations and Measurements*. **Volume**, DOI: OGC 03-022r3
19. Butler, D., *2020 computing: Everything, everywhere.* Nature, 2006. **440**: p. 402-405.

20. Gibbons, P.B., et al., *IrisNet: an architecture for a worldwide sensor Web.* IEEE, Pervasive Computing, , 2003. **2**(4): p. 22-33.
21. Akyildiz, I.F., et al., *A Survey on Sensor Networks.* IEEE Communications Magazine, 2002.
22. Akyildiz, I.F., et al., *Wireless sensor networks: a survey*. 2002, Elsevier. p. 393-422.
23. Estrin, D., et al. *Next Century Challenges: Scalable Coordination in Sensor Networks*. in *Fifth Annual International Conference on Mobile Computing and Networks (MobiCOM '99)*. 1999. Seattle, Washington.
24. Mainwaring, A., et al., *Wireless sensor networks for habitat monitoring*. 2002, ACM Press New York, NY, USA. p. 88-97.
25. Mainwaring, A., et al. *Wireless sensor networks for habitat monitoring*. in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications* 2002: ACM Press New York, NY, USA.
26. Pottie, G.J. and W.J. Kaiser, *Wireless integrated network sensors.* Communications of the ACM 2000. **43**(5): p. 51-58.
27. Zerger, A. and D.I. Smith, *Impediments to using GIS for real-time disaster decision support.* Computers, Environment and Urban Systems, 2003. **27**(2): p. 123-141.
28. Reichardt, M., *Sensor Web Enablement: An OGC White Paper*. 2005, Open Geospatial Consortium (OCG), Inc.
29. Fox, G., *Grids of Grids of Simple Services.* Computing in Science and Engg., 2004. **6**(4): p. 84-87.
30. 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.
31. Beth Plale, et al., *CASA and LEAD: Adaptive Cyberinfrastructure for Real-Time Multiscale Weather Forecasting* IEEE Computer, 2006. **39**(11): p. 56-64.
32. 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.
33. Beth Plale, D.G., Jay Alameda, Bob Wilhelmson, Shawn Hampton, Al Rossi, and Kelvin Droegemeier *Active Management of Scientific Data.* IEEE Internet Computing, special issue on Internet Access to Scientific Data, 2005. **9**(1): p. 27-34.
34. Kelvin K. Droegemeier, et al., *Linked environments for atmospheric discovery (LEAD): Architecture, Technology Roadmap and Deployment Strategy,* , in *21st Conference on Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology,* . 2005.
35. Sgouros, T. (2004) *A DODS Quick Start Guide Version 1.5, available from* http://www.opendap.org/user/quick-html/quick.html. **Volume**,
36. Cornillon, P., J. Gallagher, and T. Sgouros, *OPeNDAP: Accessing data in a distributed, heterogeneous environment.* Data Science Journal, 2003. **2**: p. 164-174.

37. Sgouros, T. (2004) *OPeNDAP User Guide Version 1.14, available from http://www.opendap.org/user/guide-html/guide.html*. **Volume**,
38. Braun, H.W., et al., *Distributed Data Management Architecture for Embedded Computing*, in *6th Workshop on High Performance Embedded Computing*. 2002: MIT Lincoln Laboratory.
39. Harvey, D., et al., *ORB: A New Real-Time Data Exchange and Seismic Processing System.* Seis. Res. Lett. , 1998. **69**.
40. Rajasekar, A., et al., *Virtual Object Ring Buffer: A Framework for Real-time Data Grid.*, in *HDPC Conference 2004*. 2004.
41. Bustamante, F.E., *The Active Streams Approach to Adaptive Distributed Applications and Services*, in *Computer Science*. 2001, Georgia Institute of Technology. p. 112.
42. Bustamante, F.E., G. Eisenhauer, and K. Schwan. *The Active Streams Approach to Adaptive Distrubuted Systems*. in *10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10 '01), 2001*. 2001.
43. Eisenhauer, G., K. Schwan, and F.E. Bustamante, *Publish-subscribe for high-performance computing.* IEEE Internet Computing, 2006. **10**(1): p. 40-47.
44. Isert, C. and K. Schwan. *ACDS: Adapting computational data streams for high performance*. in *14th International Parallel & Distributed Processing Symposium (IPDPS'00),* . 2000. Cancun, Mexico: IEEE Computer Society 2000
45. Plale, B. and K. Schwan, *dQUOB: Managing Large Data Flows Using Dynamic Embedded Queries*, in *Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC-9 '00)*. 2000. p. 263.
46. Vijayakumar, N. and B. Plale, *dQUOBEC event channel communication system*. 2005, Indiana University, Computer Science, Technical Report TR614.
47. Plale, B. and K. Schwan, *Dynamic querying of streaming data with the dQUOB system.* IEEE Transactions on Parallel and Distributed Systems, 2003. **14**(4): p. 422-432.
48. Bustamante, F., et al. *Efficient wire formats for high performance computing*. in *ACM/IEEE SC 2000 Conference (SC'00)*. 2000.
49. Liu, Y., N.N. Vijayakumar, and B. Plale. *Stream Processing in Data-Driven Computational Science*. in *7th IEEE/ACM Int'l Conference Grid Computing, Grid'06*. 2006. Barcelona, S.
50. Vijayakumar, N., Y. Liu, and B. Plale. *Calder Query Grid Service: Insights and Experimental Evaluation*. in *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*. 2006.
51. Golab, L. and M.T. Özsu, *Issues in data stream management.* ACM SIGMOD Record, 2003. **32**(2): p. 5-14.
52. Chandrasekaran, S., et al. *TelegraphCQ: continuous dataflow processing*. in *2003 ACM SIGMOD international conference on Management of data* 2003. San Diego, California ACM Press New York, NY, USA.
53. Chandrasekaran, S., et al. *TelegraphCQ: Continuous dataflow processing for an uncertain world*. in *First Biennial Conference on Innovative Data Systems Research (CIDR)*. 2003. Asilomar, CA, USA.
54. Avnur, R. and J.M. Hellerstein, *Eddies: continuously adaptive query processing.* ACM SIGMOD Record 2000. **29**(2): p. 261-272.

55. Babcock, B., et al. *Models and issues in data stream systems*. in *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* 2002: ACM Press New York, NY, USA.
56. Plale, B., *Using global snapshots to access data streams on the grid.* 2004. **3165**(3165): p. 191-201.
57. Plale, B., *Framework for bringing data streams to the grid.* Scientific Programming 2004. **12**(4): p. 213-223.
58. Granat, R., *Regularized Deterministic Annealing EM for Hidden Markov Models*, in *University of California Los Angeles*. 2004.
59. 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.
60. Granat, R.A., *A method of hidden Markov model optimization for use with geophysical data sets.* Comp. Sci., 2003(2659): p. 892–901.
61. Granat, R.A., *Statistical Analysis of Geodetic Networks for Detecting Regional Events*, in *4th International ACES Workshop*. 2004: Beijing, China 2004.
62. Holliday, J.R., et al., *A RELM earthquake forecast based on pattern informatics*, in *AGU Fall Meeting;*. 2005: San Francisco, California,.
63. 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).
64. Rundle, J.B., et al., *Self-organization in leaky threshold systems: The influence of near-mean field dynamics and its implications for earthquakes, neurobiology, and forecasting.* . Proc. Natl. Acad. Sci. U. S. A. , 2002. **99**: p. 2514-2521.
65. 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
66. 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.
67. 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.
68. Sayar, A., M. Pierce, and G.C. Fox, *DEVELOPING GIS VISUALIZATION WEB SERVICES FOR GEOPHYSICAL APPLICATIONS* in *ISPRS International Society for Photogrammetry and Remote Sensing Workshop Commission II WG/2* 2005: METU, Ankara, Turkey.
69. Sayar, A., et al., *Developing a Web Service-Compatible Map Server for Geophysical Applications available from* http://grids.ucs.indiana.edu/ptliupages/publications/acm-gis-sayar.pdf. 2005.
70. 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.
71. Thomas W. Meyer, et al., *The Los Alamos Center for Homeland Security.* LOS ALAMOS SCIENCE, 2003. **28**.

72. Hashimi, S., *Service-Oriented Architecture Explained*. 2004, O'Reilly http://dev2dev. bea. com/technologies/soa/articles/soa_hashimi, jsp, Apr.
73. Eugster, P.T.H., et al., *The Many Faces of Publish/Subscribe*. 2003. p. 114-131.
74. Cox, S., *Observations and Measurements*. 2002. p. 02-027.
75. Botts, M., *Sensor model language (SensorML) for in-situ and remote sensors specification*. 2002, discussion paper 02-026r4, Open GIS Consortium. http://www. opengis. org/techno/discussions/02-026r4. pdf.
76. OGC, *The Open Geospatial Consortium, Inc, http://www.opengeospatial.org/.*
77. 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.
78. Christensen, E., et al., *Web Services Description Language (WSDL) 1.1*. 2001, March.
79. Belwood, T., L. Clement, and C. von Riegen, *UDDI Version 3.0.1: UDDI Spec Technical Committee Specification. Available from http://uddi.org/pubs/uddi-v3.0.1-20031014.htm*. 2003.
80. CGL. *Community Grids Lab Web Site http://communitygrids.iu.edu/index.php*. [cited.
81. Aktas, M., et al., *Information Services for Grid/Web Service Oriented Architecture (SOA) Based Geospatial Applications*.
82. Aktas, M., et al., *Web Service Information Systems and Applications* in *GGF-16 Global Grid Forum Semantic Grid Workshop* 2006: Athens, Greece.
83. Aktas, M., G. Fox, and M. Pierce. *Managing Dynamic Metadata as Context*. in *Istanbul International Computational Science and Engineering Conference (ICCSE2005 http://www.iccse.org/) June 2005*. 2005.
84. Aktas, M., G. Fox, and M. Pierce, *An Architecture for Supporting Information in Dynamically Assembled Semantic Grids*. 2005.
85. Aktas, M., G. Fox, and M. Pierce. *Information Services for Dynamically Assembled Semantic Grids*. in *Proceedings of 1st International Conference http://kg.ict.ac.cn/SKG2005/ on SKG2005 Semantics, Knowledge and Grid Beijing China November 27-29 2005*. 2005.
86. Aktas, M.S., G.C. Fox, and M. Pierce, *Fault tolerant high performance Information Services for dynamic collections of Grid and Web services.* Future Generation Computer Systems, 2007. **23**(3): p. 317-337.
87. MySql, A.B., *MySQL Database Server, WWW page, at URL: http://www. mysql. com, last accessed on 12/17/2006*.
88. Kreger, H., *Web Services Conceptual Architecture (WSCA 1.0)*. 2001. p. 6-7.
89. 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.
91. Kirtland, M., *A Platform for Web Services*. 2001, Jan.
92. Box, D., et al., *Simple Object Access Protocol (SOAP) 1.1. W3C Note 08 May 2000.*
93. ISO/TC211. *Web Site http://www.isotc211.org*. [cited.
94. Fallside, D.C. and P. Walmsley, *XML Schema W3C Recommendation 28 October 2004, http://www.w3.org/TR/xmlschema-0/*. 2001. p. 2002-10.

95. Vretanos, P.A., *Filter Encoding Implementation Specification. OGC 02-059. Ver 1.0. 0*. 2001. p. 02-059.
96. Rao, A.P., et al., *Overview of the OGC catalog interface specification*. 2000.
97. Grant, L.B., et al., *QuakeTables: The Fault Database for QuakeSim*. 2004.
98. 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 http://www.geo-prose.com/computational_SES.html edited by Ron Cohen and started at May 2004 workshop on Computational Geoinformatics*.
99. Donnellan, A., et al., *Numerical simulations for active tectonic processes: increasing interoperability and performance*.
100. JPL. *GPS Data Files available from ftp://sideshow.jpl.nasa.gov/pub/mbh, last visited on 12/17/2006*. [cited.
101. SOPAC, *GPS Time Series available from ftp://garner.ucsd.edu/pub/timeseries, last visited 12/17/2003.*
102. USGS, *GPS Time Series available from http://pasadena.wr.usgs.gov/scign/Analysis/plotdata/, last visited 12/17/2006.*
103. SCSN, *format seismic records, available from http://www.data.scec.org/ftp/catalogs/SCSN/, last visited 12/17/2006.*
104. SCEDC, *format seismic records, available from http://www.data.scec.org/ftp/catalogs/SCEC_DC, last visited 12/17/2006.*
105. Dinger-Shearer, *format seismic records, avilable from http://www.data.scec.org/ftp/catalogs/dinger-shearer/, last visited 12/17/2006.*
106. Haukkson, *format seismic records available from http://www.data.scec.org/ftp/catalogs/hauksson/, last visited 12/17/2006.*
107. Appel, V.L. and R.W. Clayton, *The Southern California Earthquake Data Center (SCEDC): Update for 2004*. 2004.
108. Clark, J. and S. DeRose, *XML Path Language (XPath) Version 1.0*. 1999. p. 1999.
109. OGC *OGC Filter Encoding Implementation Specification, OGC document number 04-095* **Volume**,
110. Pallickara, S. and G. Fox. *NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids*. in *Lecture Notes in Computer Science*. 2003: Springer-Verlag.
111. Fox, G. *Global multimedia collaboration system*. in *Collaborative Technologies and Systems, 2005. Proceedings of the 2005 International Symposium on*. 2005.
112. Uyar, A., et al. *Service-Oriented Architecture for a Scalable Videoconferencing System*. in *Proceedings of IEEE International Conference on Pervasive Services 2005 (ICPS'05) http://icps2005.cs.ucr.edu/ July 2005, Santorini, Greece*. 2005.
113. Uyar, A., *Scalable service oriented architecture for audio/video conferencing*. 2005, Syracuse.
114. Bush, B.W., *NISAC Interdependent Energy Infrastructure Simulation System, Report LA-UR-04-7700,*. 2004, Los Alamos National Laboratory.
115. OnEarth, *NASA OnEarth Web Map Service for global satellite images, available at http://onearth.jpl.nasa.gov/.*
116. Gaia, *GIS Viewer, http://www.thecarbonportal.net/.*

117. Gadgil, H., et al., *Management of Data Streams for a Real Time Flood Simulation*. 2004.
118. Gadgil, H., G. Fox, and S. Pallickara. *HPSearch for Managing Distributed Services*. in *Work in Progress session at IEEE/ACM Cluster Computing and Grid 2005 Conference (CCGrid 2005 http://www.cs.cf.ac.uk/ccgrid2005/). Cardiff, UK May 2005*. 2005.
119. Gadgil, H., et al. *A Scripting based Architecture for Management of Streams and Services in Real-time Grid Applications*. in *Proceedings of the IEEE/ACM Cluster Computing and Grid 2005 Conference (CCGrid 2005). Cardiff, UK May 2005*. 2005.
120. Gadgil, H., et al., *HPSearch: Service Management & Administration Tool*, in *Abstract for VLAB Meeting Minnesota July 21-23 2005*. 2005.
121. Davis, D. and M.P. Parashar, *Latency Performance of SOAP Implementations*, in *Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on*. 2002. p. 407-407.
122. Kohlhoff, C. and R. Steele, *Evaluating SOAP for High Performance Business Applications: Real-Time Trading Systems*, in *In proceedings of the 2003 International WWW Conference,*. 2003: Budapest, Hungary. p. 03-2002.
123. van Engelen, R. *Constructing Finite State Automata for High-Performance XML Web Services*. in *International Symposium on Web Services and Applications (ISWS) 2004*. 2004.
124. van Engelen, R.A. *Pushing the SOAP envelope with Web services for scientific computing*. in *In proceedings of the International Conference on Web Services (ICWS),*. 2003. Las Vegas, 2003.
125. Chiu, K., M. Govindaraju, and R. Bramley. *Investigating the limits of SOAP performance for scientific computing*. in *High Performance Distributed Computing, 2002. HPDC-11 2002. .* 2002: IEEE
126. Goldman, O., *XML Binary Characterization, W3C Working Group Note, Mar, 2005* 2005.
127. Gudgin, M., et al., *SOAP Message Transmission Optimization Mechanism, W3C Proposed Recommendation, Nov, 2004*. 2004.
128. Gudgin, M., et al., *XML-binary Optimized Packaging, W3C Recommendation, Jan, 2005*. 2005.
129. Liefke, H. and D. Suciu. *XMill: an efficient compressor for XML data*. in *ACM SIGMOD international conference on Management of data, 2000* 2000: ACM Press New York, NY, USA.
130. Oh, S., et al. *Optimized communication using the SOAP infoset for mobile multimedia collaboration applications*. in *Collaborative Technologies and Systems, 2005. Proceedings of the 2005 International Symposium on*. 2005.
131. Oh, S. and G. Fox, *HHFR: A new architecture for Mobile Web Services: Principles and Implementations*. 2005.
132. Pericas-Geertsen, S. *Binary interchange of XML infosets*. in *XML Conference & Exposition 2003*. 2003. Pennsylvania Conventin Center, Philadelphia, PA, USA.
133. Sandoz, P. and K.K. Santiago Pericas-Geertsen, Marc Hadley, and Eduardo Pelegri-Llopart,, *Fast Web Service, available from*

http://java.sun.com/developer/technicalArticles/WebServices/fastWS/ *last visited 12/17/2006*.
134. Sandoz, P., A. Triglia, and S. Pericas-Geertsen. *Fast Infoset, article available from* http://java.sun.com/developer/technicalArticles/xml/fastinfoset/. 2004 [cited.
135. Govindaraju, M., et al. *Requirements for and Evaluation of RMI Protocols for Scientific Computing*. in *Proceedings of the IEEE/ACM SC2000 Conference (SC'00)*. 2000. Dallas, TX, 2000
136. Lim, S., et al. *GridFTP and Parallel TCP Support in NaradaBrokering*. in *Proceedings of 6th International Conference on Algorithms and Architectures for Parallel Processing ICA3PP 2005* http://www3.it.deakin.edu.au/ica3pp2005/index.php?id=Home *Melbourne Australia. October 2-5 2005*. 2005: Springer-Verlag.
137. Bayardo, R.J., et al. *An evaluation of binary xml encoding optimizations for fast stream based xml processing*. in *WWW2004, May 17–22, 2004, New York, New York, USA*. 2004: ACM Press New York, NY, USA.
138. W3C, *Report From the W3C Workshop on Binary Interchange of XML Information Item Sets, 24th, 25th and 26th September, 2003, Santa Clara, California, USA. Available from* http://www.w3.org/2003/08/binary-interchange-workshop/Report.html.
139. W3C. *XML Binary Characterization Working Group, Web Page* http://www.w3.org/XML/Binary/. [cited.
140. Chiu, K. *XBS: A Streaming Binary Serializer for High Performance Computing*. in *In Proceedings of the High Performance Computing Symposium 2004, April 2004*. 2004.
141. Hoschek, W., *A Quantitative Comparison of Binary XML Encodings, Presentation at GridWorld / Global Grid Forum 15, Boston, Oct 200, available from* http://dsd.lbl.gov/DSDlocal/DSDMeetings/ggf15-binaryXML.pdf. 2005.
142. NUX. *Web Site* http://dsd.lbl.gov/nux/. [cited.
143. Chong, C.Y., S.P. Kumar, and B.A. Hamilton, *Sensor networks: evolution, opportunities, and challenges.* Proceedings of the IEEE, 2003. **91**(8): p. 1247-1256.
144. Delin, K.A., *The Sensor Web: A Macro-Instrument for Coordinated Sensing.* Sensors, 2002 2002. **2**(1): p. 270–285.
145. Delin, K.A. and S.P. Jackson, *The Sensor Web: A New Instrument Concept*. 2001. p. 20-26.
146. Martinez, K., J.K. Hart, and R. Ong, *Environmental sensor networks*. 2004. p. 50-56.
147. Estrin, D., et al., *Next century challenges: scalable coordination in sensor networks*. 1999, ACM Press New York, NY, USA. p. 263-270.
148. Hudnut, K.W., et al., *The Southern California Integrated GPS Network (SCIGN).* 2002. p. 167–189.
149. Yamagiwa, A., Y. Bock, and J. Genrich. *Real-time monitoring of crustal deformation using large GPS geodetic networks-Japanese GEONET's potential as a natural hazards mitigation system*. in *American Geophysical Union, Fall Meeting 2004, abstract #SF53A-0724*. 2004.

150. Cardell-Oliver, R., et al., *A Reactive Soil Moisture Sensor Network: Design and Field Evaluation.* International Journal of Distributed Sensor Networks, 2005 2005. **1**(2): p. 149-162.
151. Raicu, I. *Efficient Even Distribution of Power Consumption in Wireless Sensor Networks*. in *ISCA 18th International Conference on Computers and Their Applications, CATA 2003, 4 pages*. 2004. Honolulu, Hawaii, USA.
152. Berfield, A., P.K. Chrysanthis, and A. Labrinidis. *Automated Service Integration for Crisis Management*. in *First Workshop on Databases In Virtual Organizations (DIVO 2004), available from http://dais.cs.uiuc.edu/divo2004/proceedings/divo04-berfield.pdf*. 2004.
153. Goldammer, J.G. *Early warning systems for the prediction of an appropriate response to wildfires and related environmental hazards*. in *Health Guidelines for Vegetation Fire Events,*. 1998. Lima, Peru, .
154. Allen, R.M. and H. Kanamori, *The Potential for Earthquake Early Warning in Southern California.* Science 2003. **300**(5620): p. 786-789.
155. Fox, G.C. and D. Gannon, *Workflow in Grid Systems.* Concurrency and Computation: Practice and Experience, 2006. **18**(10): p. 1009-1019.
156. Oinn, T., et al., *Taverna: Lessons in creating a workflow environment for the life sciences.* Concurrency and Computation: Practice and Experience, 2006. **Volume 18**(10): p. 1067 - 1100.
157. Bock, Y., et al., *Scripps Orbit and Permanent Array Center (SOPAC) and Southern Californian Permanent GPS Geodetic Array (PGGA)*. 1997, National Academy Press. p. 55–61.
158. PBO, *THE PLATE BOUNDARY OBSERVATORY. Creating a Four-Dimensional Image of the Deformation of Western North America. A PBO White Paper, available from http://www.unavco.org/pubs_reports/proposals/PBOwhitepaper.pdf*.
159. Hudnut, K.W., et al. *THE SOUTHERN CALIFORNIA INTEGRATED GPS NETWORK (SCIGN)*. in *The 10th FIG International Symposium on Deformation Measurements*. 2001. Orange, California, USA.
160. Bock, Y., L. Prawirodirdjo, and T.I. Melbourne, *Detection of arbitrarily large dynamic ground motions with a dense high-rate GPS network.* GEOPHYSICAL RESEARCH LETTERS, 2004. **31**.
161. Apache, X.M.L., *Beans Project, http://xmlbeans.apache.org/*. 2003.
162. Paulson, L.D., *Building rich web applications with Ajax.* IEEE Computer, 2005. **38**(10): p. 14-17.