Distributed Geospatial Data Access on the WWW

by

Lushu Li

M.E., Nanjing University of Sciences and Technology (China), 1993

PhD., Southeast University (China), 1998

A Thesis Submitted in Partial Fulfillment of
the Requirements for the Degree of

Master of Computer Science

in the Graduate Academic Unit of Computer Science

Supervisor: Bradford G. Nickerson, Ph.D., Computer Science

Examining Board:

Patricia Evans, Ph.D., Computer Science
Przemyslaw Pochec, Ph.D., Computer Science
Adam Chrzanowski, Ph.D., Geodesy and Geomatics Engineering

This thesis is accepted.

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

April, 2001

©Lushu Li, 2001

Dedication

To my seven-month-old son Anthony J. Li, whose smiles always cheer me up!

Abstract

This thesis investigates the design and implementation of a Web-based distributed geospatial data warehouse (WDGSDW) system which allows a user to query geographical information and access the geospatial data services across multiple servers over the Internet.

A multi-tiered client/server architecture was used to implement WDGSDW. The CORBA-based (for Java and C++), Java RMI-based and Java servlet-based implementations of the server-side components of DWGSDW are tested and compared for the contextual data service, which providing the user interface of WDGSDW. The comparison showed that the performance of servlets-based implementation is much better than those of other implementations. The servlets technique was chosen to implement an experimental catalog server and geospatial data servers. An integrated tool to visualize the Canada Land Inventory data (in Arc/Info Export .E00 format) and raster image data was also implemented in this research. The search engine, which is the kernel of WDGSDW, supports combined text search and geographical search with an adjustable match factor. The search engine was built using R-Tree and AVL-Tree indexes.

WDGSDW system was tested using test data sets containing 6979 CEONet metadata files, 1690 CLI vector data sets and 45 CCRS raster data sets. For the contextual data server, CORBA and RMI techniques are 2 to 2.5 time slower compared to the Java servlet and a performance of 85 bytes/ms was observed for the latter, on average. The keyword searches can take up to 4.9 seconds compared to bounding box searches times of less than 2.5 seconds on a catalogue containing 8188 entries. A combined keyword and bounding box search requires an average of 1.2 times more than the individual searches. For a fixed bounding box $[200, 350; 20, 84]$, the variation of match factor from 0.90 to 10^{-8} resulted in a change of the number of returned items from 4 to 5673. Search time per item found varied from 0.17 ms to 0.83 ms. The Fat-client via Thin-server architecture for CLI data service achieved the best performance of 349 bytes/ms, about 23 times as fast as the Thin-client via Fat-server architecture.

Acknowledgments

I would like to thank my supervisor, Dr. Bradford G. Nickerson, for his insight and knowledgeable guidance and great support throughout the process of working on this thesis.

Many thanks are due to Dr. Fuqun Zhou of the CCRS Department of Natural Resources Canada for his providing some of the test data.

Thanks to my wife Liqin Yong and my daughter Chen Li for their support, encouragement and consideration.

Contents

List of Tables

List of Figures

List of Abbreviations and Acronyms

API	Application Programming Interface
ASP	Active Server Page
AVHRR	Advanced Very High Resolution Radiometer
CCRS	Canada Centre for Remote Sensing
CEONet	Canada Earth Observation Network
CLI	Canada Land Inventory
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
CSDGM	Content Standards for Digital Geospatial Metadata
DBMS	Database Management System
DCOM	Distributed Component Object Model
DII	Dynamic Invocation Interface
DSS	Decision Support System
EJB	Enterprise Java Bean
FGDC	Federal Geographic Data Committee
GIS	Geospatial Information System
GML	Geography Markup Language
GMT	Generic Mapping Tools
GSD	Geospatial Data
GSHHS	Global, Self-consistent, Hierarchical, High-resolution Shoreline database
GUI	Graphic User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transport Protocol
IDL	Interface Definition Language
IIOP	Internet Inter–ORB Protocol
IS	Information System
ISO	International Standards Organization
JDBC	Java Database Connectivity
JPEG	Joint Photographic Experts Group

JSP	Java Server Page
NOAA	National Oceanic and Atmospheric Administration
OGC	Open GIS Consortium
OMG	Object Management Group
ORB	Object Request Broker
RMI	Remote Method Invocation
SGML	Standard Generalized Markup Language
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol/Internet Protocol
TIFF	Tag Image File Format
URL	Uniform Resource Locater
WDGSDW	Web-oriented Distributed Geospatial Data Warehouse
WWW	World Wide Web
XML	Extensible Markup Language

Chapter 1

Introduction

The Internet and especially the World Wide Web (WWW) provide a promising approach to make all kinds of data and information available publicly and privately (using an Intranet). There is a growing awareness that the WWW has changed forever the way data, information and knowledge is collected, analyzed and distributed. Using the WWW together with other elaborate distributed computing techniques, the general public will be able to access, retrieve, merge, and analyze complex sets of geospatial information and data [36]. This thesis investigates web-based distributed computing technologies and XML standards to see how best to provide access to geospatial data which are available around the world.

1.1 Geospatial Information Systems and OpenGIS

1.1.1 Geospatial Information System (GIS)

According to the International GIS Dictionary, a GIS is a "computer system for capturing, managing, integrating, manipulating, analyzing and displaying data which is spatially referenced to the Earth." [25]. What distinguishes a GIS from other forms of information systems, such as databases and spreadsheets, is that a GIS deals with geospatial information. A GIS has the capability to relate layers of data for the same points in space, combining, analyzing and, finally, mapping out the results. Geospatial information uses location, within a geodetic coordinate system, as its reference base. Most geospatial information and data have been stored in heterogeneous systems which are found within and across organizations. Modern solutions to storing and handling complex heterogeneous information in distributed systems include specialized tools for web-based geographical data integration. In the USA [39], GIS is a $2 billion dollar per year high technology industry that drives decision-making in the public sector and business. Nearly 80,000 government agencies are involved in the creation of geographic information using taxpayer funding to gather, compile, and store data. Agencies are now beginning to make these stores of data directly available to the publics through GIS applications using the Internet and the World Wide Web (e.g. Oakland Unified School District Map Center [29] and San Diego regional SanGIS [37]).

1.1.2 OpenGIS

Much geospatial data is available on the web and in off-line archives, but it is complex, heterogeneous, and incompatible. OpenGIS is a standardization effort by the Open GIS Consortium (OGC) [33] as transparent access to heterogeneous geo-data and geo-processing resources in a networked environment. Hence, the OpenGIS can be defined as “Open and interoperable geo-processing” or “The ability to share heterogeneous geo-data and geo-processing resources transparently in a networked environment". The goal of the OpenGIS is to promote the use of interoperable geo-processing throughout the Information Technology marketplace and provide a comprehensive suite of open interface specifications that enable developers to write interoperating components that provide these capabilities. So far, several OpenGIS implementation specifications have been completed. They include the Geography Markup Language (GML) v.1.0 [21], the OpenGIS Abstract Specification v.4.0 [30], the OpenGIS Simple Feature Specification for SQL rev. 1.1 [31] and the OpenGIS Web Map Server Interface Implementation Specification rev. 1.0.0 [32].

1.2 Data Warehousing and Metadata

1.2.1 Data Warehousing

Data warehousing has become very popular among organizations seeking competitive advantage by getting strategic information quickly and easily. Since William Inmon coined the phrase "data warehouse" in 1990, it began attracting the attention from IS managers and vendors. As defined by William Inmon [19], a data warehouse is a subject oriented, integrated, time-varying, non-volatile collection of data in support of management's decision making process. Basically data warehouses are large, special-purpose databases that contain data integrated from a number of independent sources, supporting clients who wish to analyze the data for trends and anomalies. The process of analysis is usually performed with queries that aggregate, filter, and group the data in a variety of ways. Processing the queries quickly is a critical issue in the data warehousing environment, since the queries are often complex and the warehouse database is often very large. The query throughput and response times are more important than transaction throughput in a data warehouse. Chaudhuri [7] has more details about the tools, utilities and services in a data warehouse.

1.2.2 Metadata

Metadata is “data about data” which contains data describing the operational environment. It plays a very important role in the data warehouse and is used as:

- a directory to help the DSS analyst locate the contents of the data warehouse, and
- a guide to the mapping of data as the data is transformed from the operational environment to the data warehouse environment.

In GIS, geospatial metadata are simply that type of descriptive information applied to a digital geospatial file. They are a set of common terms and definitions to use when documenting geospatial data. In essence, metadata answer who, what, when, where, why, and how about every facet of the data that are being documented. Metadata can be organized into several levels ranging from a simple listing of basic information about available data to detailed documentation about an individual data set.

Metadata may exist in forms other than ones compliant with the Content Standards for Digital Geospatial Metadata (CSDGM). CSDGM compliant digital metadata may be created, stored, and used in a variety of formats. The most basic is an ASCII text document. An ASCII document is easy to transfer to other users independent of the hardware/software platform they use. Another common format is Hypertext Markup Language (HTML). HTML provides an attractive way to view metadata using a browser such as Netscape Navigator, Mosaic, or Microsoft Internet Explorer. Recently, there has been strong interest in creating metadata in Standard Generalized Markup Language (SGML). SGML provides an effective way to tag metadata elements. The ISO geospatial metadata standard [23] uses XML (Extensible Markup Language) [46], a subset of SGML, as the metadata file format. This standard provides the starting point for indexing and searching metadata and provides a definition of how to exchange metadata between metadata users, metadata databases, and metadata tools.

1.3 Web-based Internet Technologies and Distributed Programming

1.3.1 Web-based Internet Technologies

A *network* is a collection of computers and other devices that are connected together by some medium and can send data to and receive data from each other. The Internet is made up of many separate but interconnected networks. The different devices on the Internet communicate with each other through a *protocol*, a precise set of rules that two or more computers must follow to exchange messages. There are many different kinds of protocols defining different aspects of network communication. The important communication protocols used in the Internet are the suite of protocols

TCP/IP, which are commonly run on TCP (Transmission Control Protocol) and IP (Internet Protocol).

The World Wide Web (Web or WWW, for short) is a hypermedia-based system that provides a simple “point and click” means of exploring the immense volume of information residing on the Internet. In web technology, a basic client-server architecture underlies all activities. Information is stored on computers designed as web servers in publicly accessible shared files encoded using HyperText Markup Language (HTML). Information on the web is organized according to a Uniform Resource Locator (URL) and exchanged using the HyperText Transport Protocol (HTTP) between the web servers and web browsers. Popular web browsers include Internet Explorer from Microsoft and Netscape Communicator.

Today’s technology has been moving rapidly from static to dynamic web pages. In the Internet world, the need to deliver dynamically generated content in a maintainable fashion is extremely important. Common Gateway Interface (CGI), a standard for external gateway programs to interface with information servers such as HTTP servers, was the first response to this need. This interface allows web servers to call scripts to obtain data from (or send data to) a database, documents, and other programs, and present that data to viewers via the web. However, CGI technology has a number of limitations. First, the code within a CGI script that accesses resources, such as a file system or database, must be specific to the server’s platform. This limits their utility in distributed environments where web applications may need to run on multiple platforms. Second, CGI scripts are interpreted (not compiled), which makes them resource intensive and slow and thus tend not to scale well. A new process must be created each time a CGI script is invoked. Finally, CGI applications are difficult to maintain because they combine content and display logic in one code base.

To overcome the limitations of CGI technology, Java servlets and JavaServer Pages (JSP) provide alternate solutions. Java servlets are a means of extending the functionality of a web server. Servlets can be viewed as applets that run on the server. Servlets are a portable, platform independent means of delivering dynamic content. A browser-based application that calls servlets need not support the Java programming language because a servlet's output can be HTML, XML, or any other content type. JSP technology was designed to provide a declarative, presentation-centric method of developing servlets. Along with all the benefits servlets offer, JSP technology offers the ability to rapidly develop servlets where content and display logic are separated, and to reuse code thorough a component-based architecture.

In summary, currently CGI scripts are widely used to provide dynamic content. Technologies such as servlets and JSP technology that are scalable and easy to write and maintain can be used instead of CGI scripts. This is driven by the need to provide dynamic content in a platform-independent, scalable way.

1.3.2 Distributed Programming

A distributed system is a collection of computers connected by a network; the network is equipped with distributed system software that enables computers to co-ordinate activities and to share the resources of the system. The resources can be hardware, software, or data. The wide-spread use of distributed systems is due to the price-performance revolution in microelectronics, the development of cost effective and efficient communication networks, the development of resource sharing software, and the increased user demands for communication, economical sharing of resources, and productivity.

Resources in a distributed system are physically encapsulated within some of the computers; other computers can only access them via communication networks. The resources are managed by a resource manager, which is an important component of a distributed system. In a distributed system, resource users communicate with resource managers to access the shared resources of the system. The WWW represents a good example of a distributed system. Many web servers run on various computers, and each server holds a wide range of documents and information in other media on diverse topics. These web servers act as resource managers.

The major types of application of distributed system are distributed computing and parallel computing. In distributed computing, a set of computers connected by a network are used collectively to accomplish a distributed job. On the other hand, in parallel computing, a solution to a large problem is divided into many small tasks. The tasks are distributed to and executed on multiple computers to achieve high performance. Parallel computing usually requires more interprocessor communication than distributed computing.

Distributed systems can be implemented using two models: The client/server model and the object-based model. In Chapter 2, we will discuss more details about these two models and distributed programming techniques.

1.4 Thesis Objectives

The primary goal of this thesis is to design and implement a Web-oriented Distributed Geospatial Data Warehouse (WDGSDW) system which allows a user to query globally distributed geographic information and to access geospatial data services running on multiple servers through the Internet using a web browser. A multi-tiered client-server architecture was used in the implementation, and the performance of different approaches (i.e. Java Servlet, Java RMI, CORBA) to implement the middleware (middle tier) were compared.

The objectives of this thesis include:

(a) Design and test the multi-tier client-server architecture to implement a web-oriented distributed geospatial data warehouse system over multiple servers.

(b) Implement and compare the performance of the various middleware techniques, such as Java Servlet, Java RMI, and CORBA (for Java and C++).

(c) Investigate if Java servlet, Java RMI can compete with CORBA (for Java and C++) in delivering contextual geospatial data to build the user interface of the WDGSDW system.

(d) Investigate how heterogeneous geospatial databases distributed across multiple servers can be effectively indexed, queried and updated from a web browser.

(e) Build search engine for searching distributed geospatial information in a geospatial data warehouse on the web, which supports the combined text queries and geographical queries with fuzzy match restriction.

(f) Implement some client-server applications which provides the actual geospatial data services across multiple servers over the Internet.

(g) Test and evaluate the implemented system quantitatively.

The thesis is organized as follows. In Chapter 2 we present an overview of the models and technologies of distributed computing, the foundation for the rest of the thesis. The FGDC CSDGM and ISO 19115 standards for XML geospatial metadata are introduced in Chapter 3. In Chapter 4, we describe the test data sets and construct a data structure for the CLI data. Chapter 5 shows the architectural overview of the indexing scheme and building the search engine using ISO standard geospatial metadata files. Chapter 6 and Chapter 7 are dedicated to the design, implementation and evaluation of the WDGSDW system. Finally, we close with conclusions and discussions of future extensions to DWGSDW in Chapter 8.

Chapter 2

Web-based Distributed Programming and Architectures

Distributed computing is one of the major types of application of distributed systems. It allows business logic and data to be reached from remote locations. In this chapter, we give an overview of the web-based distributed computing models, architectures and programming techniques.

2.1 Client/Server Model

Distributed systems can be implemented using two models: the client/server model and the object-based model. This section introduces the client/server model of distributed computing. The distributed object-based model is discussed in section 2.2.

2.1.1 Two-tier Client-Server Architecture

The most widely used model of distributed computing is currently the 'Client-Server' model. A client is defined as a requester of services and a server is defined as the provider of services. Client/server is an architecture for distributing the functions of an application across the most suitable combination of user workstations, networks and shared computers. It contains a set of server processes, each one acting as a resource manager for resources of a given type. It also contains a collection of client processes, each one performs a task that requires access to some shared hardware and software resources. Resource managers may themselves need to access resources managed by another process, so some processes are both client and server processes. However, in the client/server model, all shared resources are held and managed by server processes. The client communicates with the server for the purpose of exchanging information. Both the client and server usually speak the same language–a protocol that the client and server both understand–so they are able to communicate with each other.

Traditionally, most client/server models use a 2-tier software architecture. Two-tier software architectures consist of three components distributed in two layers: client (requester of services) and server (provider of services). The three components are:

- User System Interface (such as session, text input, dialog, and display management services).
- Processing Management (such as process development, process enactment, process monitoring, and process resource services).
- Database Management (such as data and file services).

The two tier design allocates the user system interface exclusively to the client. It places database management on the server and splits the processing management between client and server, creating two layers. Figure 2.1 depicts the 2-tier software architecture.

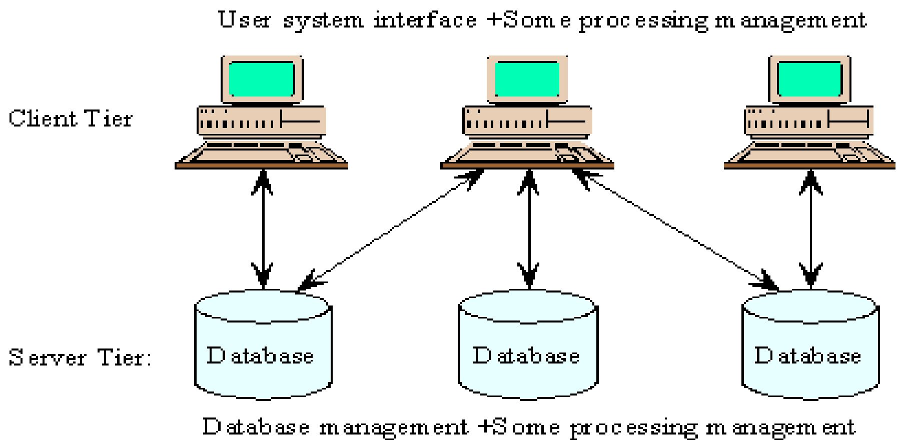

User system interface +Some processing management
Client Tier
Server Tier:
Database
Database
Database
Database management +Some processing management

Figure 2.1: 2-tier distributed client/server architecture.

Most modern network programming is also based on a 2-tier client/server architecture (see Figure 2.2). A client-server application typically stores large quantities of data on an expensive, high-powered server, while most of the program logic and the user-interface is handled by client software running on relatively inexpensive personal computers. In most cases, a server primarily sends data, while a client primarily receives it, but it is rare for one program to send or receive exclusively. A more reliable distinction is that a client initiates a conversation, while a server waits for clients to start conversations with it. Some servers process and analyze the data before sending the results to the client. Such servers are often referred to as *application servers*, to distinguish them from the more common file servers and database servers. The latter exist only to send out chunks of information, but do not do anything with that information. The most popular client/server system is the World Wide Web. Web servers respond to requests from web clients using the protocol HTTP. Data is stored on the web server and is sent out to the clients that request it. Aside from the initial request for a page, almost all data is transferred from the server to the client, not from the client to servers. Web servers that use CGI or servlet programs double as application and file servers.

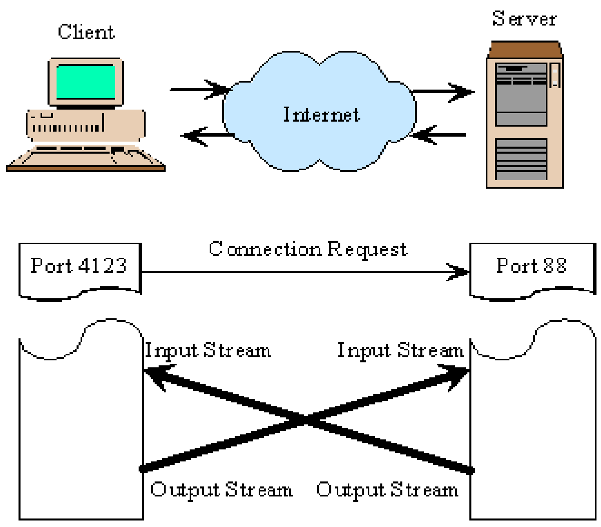


Figure 2.2: A WWW client/server model.

In a 2-tier client-server architecture, the data logic resides on the server and the presentation function on the client; the business logic runs both in the client and the server, with each processor doing an appropriate level of processing and exchanging inter-process communication messages across the network. Detailed readings on two tier architectures can be found in [38].

2.1.2 Three-tier Client-Server Architecture

Unfortunately, the 2-tier model shows striking weaknesses in the scalability, interoperability, system administration and configuration [38]. The limitations of the

2-tier model make the development and maintenance of distributed applications much more expensive. Three-tier and n-tier architectures endeavour to overcome these limitations. This goal is achieved primarily by moving the application logic from the client back to the server.

A 3-tier architecture is used when an effective distributed client/server design is needed that provides (when compared to the two tier) increased performance, flexibility, maintainability, reusability, and scalability, while hiding the complexity of distributed processing from the user. These characteristics have made three layer architectures a popular choice for Internet applications and net-centric information systems.

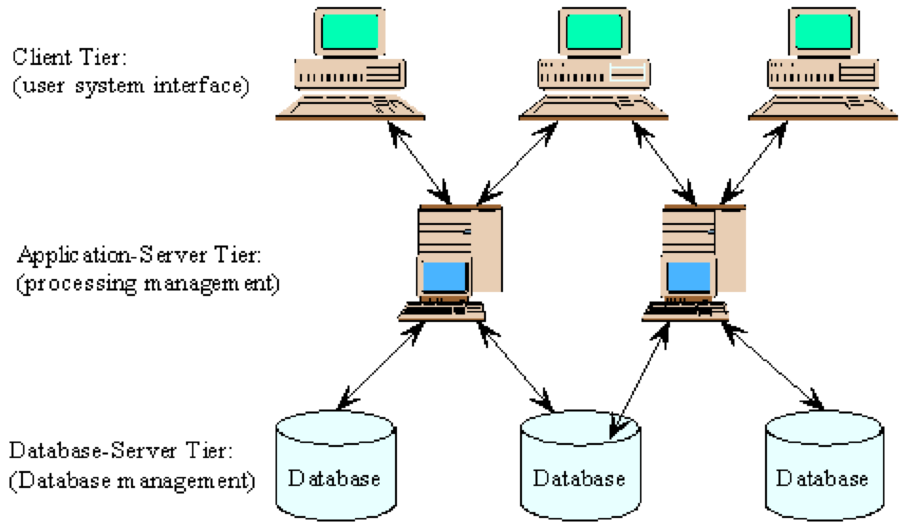


Figure 2.3: Three-tier distributed client/server architecture.

A three tier distributed client/server architecture (as shown in Figure 2.3) includes a Client-tier (user system interface), middle tier or Application-Server-tier (processing management) and Data-Server-tier (database management). The Client-tier is responsible for the presentation of data, receiving user events and controlling the user interface. The actual business logic has been moved to an application-server. The third tier or Data-Server-tier provides database management functionality and is dedicated to data and file services that can be optimized without using any proprietary database management system languages. The data management component ensures that the data is consistent throughout the distributed environment through the use of features such as data locking, consistency, and replication. It should be noted that connectivity between tiers can be dynamically changed depending upon the user's request for data and services.

The middle tier or Application-Server-tier provides process management services that are shared by multiple applications. Business-objects that implement the business rules "live" here, and are available to the client-tier. This level now forms the central key to solving 2-tier problems. This tier protects the data from direct access by the clients. The middle tier server improves performance, flexibility, maintainability, reusability, and scalability by centralizing process logic. Centralized process logic makes administration and change management easier by localizing system functionality so that changes must only be written once and placed on the middle tier server to be available throughout the systems. With other architectural designs, a change to a function (service) would need to be written into every application [38].

Sometimes, the middle tier is divided in two or more units with different functions; in these cases the architecture is often referred to as multi layer. This is the case, for example, of some Internet applications. These applications typically have light clients written in HTML and application servers written in C++ or Java. The gap between these two layers is too big to link them together. Instead, there is an intermediate layer (web server) implemented in a scripting language. This layer receives requests from the Internet clients and generates HTML pages using the services provided by the business layer. This additional layer provides further isolation between the application layout and the application logic.

It is important to note that boundaries between tiers are only logical. It is quite easily possible to run all three tiers on one and the same (physical) machine. The main importance is that the system is neatly structured, and that there is a well planned definition of the software boundaries between the different tiers.

2.2 Object-based Model

A more recent development in distributed computing is distributed object-based systems. A distributed object-based system is a collection of objects that isolates the requestors of services (client) from the providers of services (servers) by a well-defined encapsulating interface. In the object-based mode, a client sends a message to an object, which in turn interprets the message to decide what service to perform. This service selection can be performed by either the object or a broker. Distributed object technologies such as Java RMI, CORBA, and Microsoft's DCOM allow objects running on one machine to be used by client applications on different computers.

The goal of most distributed object systems is to let any object reside anywhere on the network, and allow an application to interact with these objects exactly the same way as they do with a local object. Additional features found in some distributed object schemes are the ability to construct an object on one host and transmit it to another host, and the ability for an agent on one host to create a new object on another host. These features, plus some others, are illustrated in Figure 2.4 [9]. An object interface specification is used to generate a server implementation of a class of objects, an interface between the object implementation and the object manager, sometimes called an object skeleton, and a client interface for the class of objects, sometimes called an object stub. The skeleton will be used by the server to create new instances of the class of objects and to route remote method calls to the object implementation. The stub will be used by the client to route transactions (mostly method invocations) to the object on the server. On the server side, the class implementation is passed through a registration service, which registers the new class with a naming service and an object manager, and then stores the class in the server's storage for object skeletons.

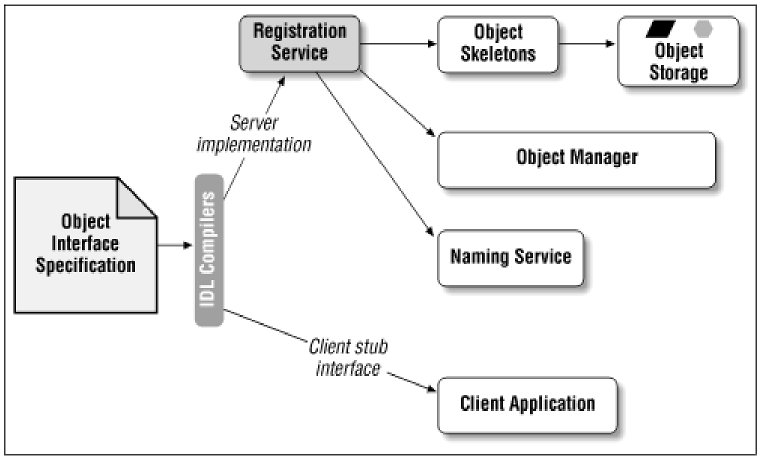


Figure 2.4: General architecture for distributed object systems (from [9]).

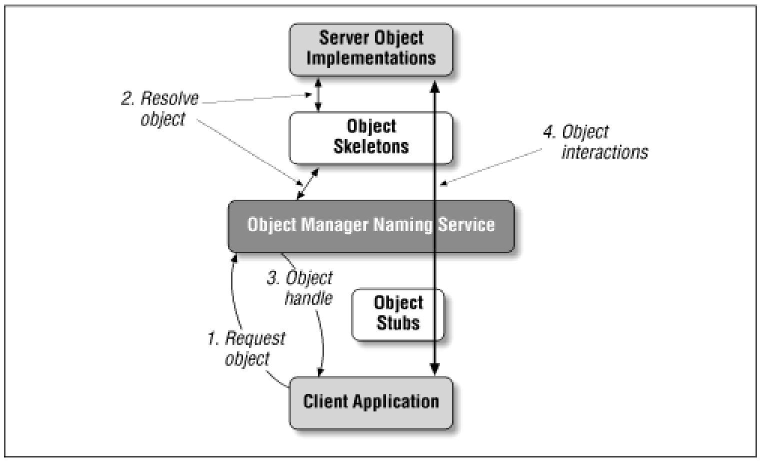


Figure 2.5: Remote object transactions at runtime (from [9]).

With an object fully registered with a server, the client can now request an instance of the class through the naming service. The runtime transactions involved in requesting and using a remote object are shown in Figure 2.5. The naming service routes the client's request to the server's object manager, which creates and initializes the new object using the stored object skeleton. The new object is stored in the server's object storage area, and an object handle is issued back to the client in the form of an object stub interface. This stub is used by the client to interact with the remote object. Refer to [9] for the details about each element of the general distributed object architecture illustrated in Figure 2.4.

2.3 Distributed Object Schemes: CORBA and Java RMI

There are several distributed object schemes that can be used to build distributed computing system. In this section we present a brief overview of CORBA and Java RMI technologies.

2.3.1 CORBA

CORBA, the Common Object Request Broker Architecture, is an industry distributed object standard developed by the Object Management Group (OMG). CORBA itself is simply a generic framework (specification) for building systems involving distributed objects. The framework is meant to be platform- and language-independent and it is implemented by CORBA-compliant products, such as Inprise's VisiBroker, Iona's OrbisWeb and Sun Microsystem's Java IDL. This standard allows CORBA objects to invoke one another without knowing where the objects they access reside or in what language the requested objects are implemented. The OMG-specified Interface Definition Language (IDL) is used to define the interfaces to CORBA objects. It is important to note that CORBA objects differ from typical programming objects in three ways:

1. CORBA objects can run on any platform.
2. CORBA objects can be located anywhere on the network.
3. CORBA objects can be written in any language that has an IDL mapping.

The CORBA framework for distributing objects consists of the following elements:

- An Object Request Broker (ORB), which provides clients and servers of distributed objects with the means to make and receive requests of each other. ORBs can also provide object services, such as a Naming Service that lets clients look-up objects by name, or Security Services that provide for secure inter-object communications.

- Methods for specifying the interfaces that objects in the system support. These interfaces specify the operations that can be requested of the object, and any data variables available on the object. CORBA offers two ways to specify object interfaces: an Interface Definition Language (IDL) for static interface definitions, and a Dynamic Invocation Interface (DII), which lets clients access interfaces as first-class objects from an Interface Repository. The DII is analogous in some ways to the Java Reflection API.

- A binary protocol for communication between ORBs, called the Internet Inter-ORB Protocol (IIOP).

A client program acting on an object is illustrated in Figure 2.6. The Object Request Broker (ORB) in Figure 2.6 connects a client application with the objects it wants to use. The client program does not need to know whether the object implementation it is in communication with resides on the same computer or is located on a remote computer somewhere on the network. The client program only needs to know the object's name and understand how to use the object's interface. The ORB takes care of the details of locating the object, routing the request, and returning the result.

CORBA ORBs usually communicate using the Internet Inter-ORB Protocol (IIOP). Other protocols for inter-ORB communication exist, but IIOP is fast becoming the most popular, first of all because it is the standard, and second because of the popu-

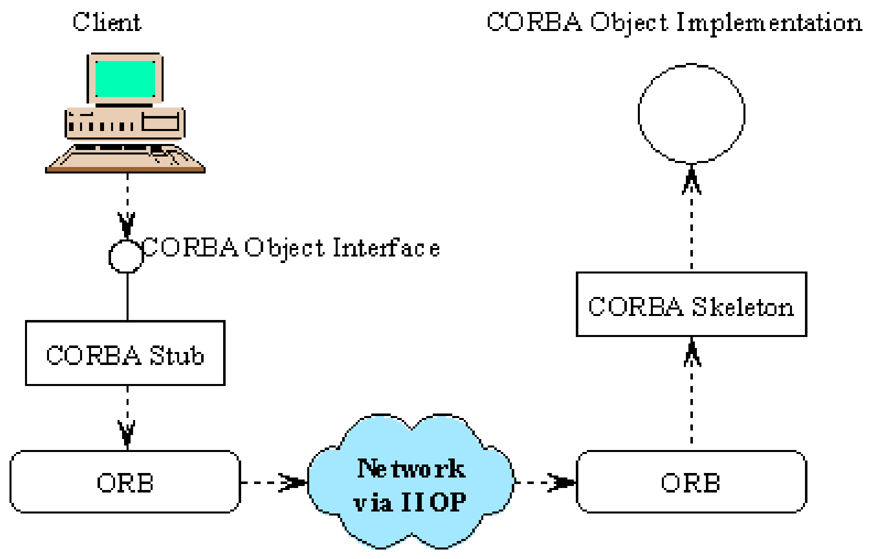


Figure 2.6: Client program acting on an object.

larity of TCP/IP (the networking protocols used by the Internet), a layer that IIOP sits on top of. CORBA is independent of networking protocols, however, and could (at least theoretically) run over any type of network protocol. Another protocol, Simple Object Access Protocol (SOAP) [5], is a competitor to IIOP.

Vinoski [44] and the OMG CORBA web site (http://www.corba.org/) both have more details on the CORBA architecture.

2.3.2 Java RMI

The Java Remote Method Invocation (RMI) [42] is an alternative approach for developing distributed applications. RMI is a Java-centric scheme for distributed objects that is now a part of the core Java API of the JDK1.1 and above. RMI offers some of the critical elements of a distributed object system for Java, plus some other features that are made possible by the fact that RMI is a Java-only system. RMI has object communication facilities that are analogous to CORBA's IIOP, and its object serialization system provides a way to transfer or request an object instance from one remote process to another. Instead of creating and instantiating objects on local machines, you create some of the objects on other remote machines, and you communicate with those objects as you normally would with local objects. Unlike CORBA, your objects can only communicate with one another if they are all implemented in Java.

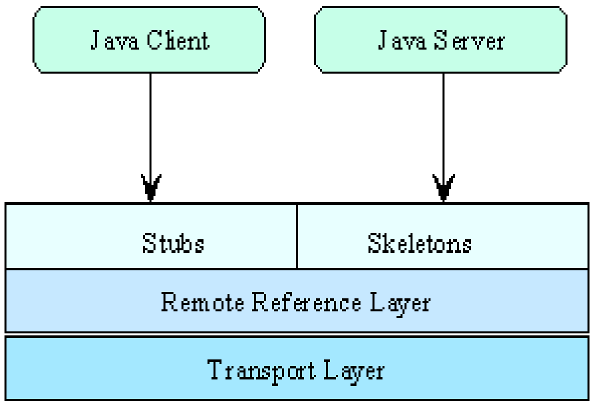


Figure 2.7: RMI system architecture.

The RMI system consists of three layers: the *stub/skeleton* layer, the *remote reference* layer and the *transport* layer, as shown in Figure 2.7. The boundary at each layer is defined by a specific interface and protocol, each layer is independent of the next, and can be replaced by an alternate implementation without affecting the other layers in the system.

RMI is a layer on top of the Java Virtual Machine which leverages the Java system's built-in garbage collection, security and class-loading mechanisms. The application layer sits on top of the RMI system. A Remote Method Invocation from a client to a remote server object travels down through the layers of the RMI system to the client-side transport. Next, the invocation is sent - potentially via network communication - to the server-side transport, where it then travels up through the transport to the server. A client invoking a method on a remote server object actually uses a stub or proxy as a conduit to the remote object.

A client-held reference to a remote object is a reference to a local stub, which is an implementation of the remote interfaces of the object and which forwards invocation requests to it via the remote reference layer.

The remote reference layer in the RMI system separates out the specific remote reference behavior from the client stub. Any call initiated by the stub is done directly through the reference layer, enabling appropriate reference semantics to be carried out.

Transparent transmission of objects from one address space to another is achieved through Object Serialization, a technique that supports the encoding of objects - and the objects they can reach - into a stream of bytes. Object Serialization also supports the complementary reconstruction of the object graph from the stream.

Another technique - known as dynamic stub loading - is used to support client-side stubs that implement the same set of remote interfaces as a remote object. When a stub of the exact type is not already available to the client, dynamic stub loading allows the client to use the Java Platform's built-in operators for casting and type-checking.

Sun [42] and other Sun RMI web sites contain more details on RMI techniques.

2.3.3 RMI vs. CORBA

We have seen many similarities between the two in terms of functionality. There are also some critical differences between the two technologies. In general, CORBA differs from RMI in the following areas:

- CORBA is designed to be language-independent. CORBA objects run in a heterogeneous environment. On other hand, RMI is a Java-centric distributed object system and it is designed for the Java language only. RMI objects run in a homogenous language environment.
- CORBA interfaces are defined in IDL, while RMI interfaces are defined in Java.
- CORBA objects are not garbage collected. Once it is created, a CORBA object exists until you get rid of it. RMI objects, on the other hand, are garbage collected automatically.
- Relatively speaking, RMI can be easier to master than CORBA.
- CORBA is a more mature standard than RMI, and has had time to gain richer implementations.

2.4 Java Applet and Java Servlet

2.4.1 Java Applets

Applets are Java-based GUI components that typically execute in a Web browser. Applets can provide a powerful user interface for Web-based distributed applications.

Applets have access to all the features and advantages of the Java platform technology. In a heterogeneous Web environment, it is especially important that client-side components be portable. For the protection of the client machine, it is important to be able to place security restrictions on these components and detect security violations. Java applets serve both these needs.

2.4.2 Java Servlets

Java servlets are a means of extending the functionality of a Web server. Servlets can be thought of as an applet that runs on the server. They provide a portable, component-based, platform and Web server independent means of delivering dynamic content. Servlets are written in the Java programming language. This allows servlets to be supported on any platform that has a Java virtual machine and a Web server that supports servlets. Servlets have access to the entire family of Java APIs, including the JDBC API to access enterprise databases. Servlets can also access a library of HTTP-specific calls, and have all the benefits of the mature Java language, including portability, performance, reusability, and crash protection. Servlets perform better than CGI (Common Gateway Interface) scripts. A servlet can be loaded into memory once and then called as many times as needed and scale well without requiring additional hardware. Once a servlet is loaded into memory it can run on a lightweight thread while CGI scripts must be loaded in a different process for every request. Another benefit of servlets is that, unlike a CGI script, a servlet can maintain and or pool connections to databases or other necessary Java objects which saves time in processing requests. Hence, servlets are a popular choice for building interactive web applications today. Hunters [18] has more detail about the servlet lifecycle and servlet–applet communication.

Chapter 3

Geospatial Metadata Standards and XML Metadata Files

Geospatial metadata support spatial data infrastructures by allowing users to locate, evaluate, extract, and employ geospatial data. It plays a special and very important role in a geospatial data warehouse. A geospatial metadata standard is simply a common set of terms and definitions that describe geospatial metadata. In this chapter, we discuss briefly the Federal Geographic Data Committee (FGDC) and the International Organization for Standardization (ISO) geospatial metadata standards.

3.1 FGDC Geospatial Metadata Standards

The Federal Geographic Data Committee approved the Content Standard for Digital Geospatial Metadata (CSDGM) (FGDC-STD-001-1998) in June 1998 [10]. The objectives of the standard are to provide a common set of terminology and definitions for the documentation of digital geospatial data. The standard establishes the names of data elements and compound elements (groups of data elements) to be used for these purposes, the definitions of these compound elements and data elements, and information about the values that are to be provided for the data elements [10].

The standard CSDGM is designed to describe a long list of geospatial data. There are 334 different elements in the CSDGM standard, 119 of which exist only to contain other elements. These compound elements are important because they describe the relationships among other elements. For example, a bibliographic reference is described by an element called Citation_Information which contains both a Title and a Publication_Date. You need to know which publication date belongs to a particular title; the hierarchical relationship described by Citation_Information makes this clear. CSDGM uses both SGML and XML for structuring information.

A typical CSDGM metadata file contains the following sections:

Metadata (Mandatory) *CONTAINS*

1. Identification Information (Mandatory) *AND*
2. Data Quality Information (Mandatory if Applicable) *AND*
3. Spatial Data Organization Information (Mandatory if Applicable) *AND*
4. Spatial Reference Information (Mandatory if Applicable) *AND*
5. Entity and Attribute Information (Mandatory if Applicable) *AND*
6. Distribution Information (Mandatory if Applicable) *AND*
7. Metadata Reference Information (Mandatory).

FGDC [1998] has more details about the FGDC geospatial metadata standards and the tools to create metadata and check the structure of metadata.

3.2 ISO Geospatial Metadata Standards

The International Organization for Standardization Technical Committee for Global GIS Standards (ISO/TC 211) is developing an integrated suite of standards to promote global interoperability. ISO/TC 211 issued a third Committee Draft, ISO/CD 19115.3, for review on metadata in June 2000 [22] and the final text, ISO/DIS 19115, for submission to ISO as Draft International Standard in December 2000 [23]. The objective of the standard is to provide a clear procedure for the description of digital geospatial datasets so that users will be able to locate geographic data, to determine whether the data in a holding will be of use to them, and how to access the data. By establishing a common set of metadata terminology, definitions, and extension procedures, the standard will promote the proper use and effective retrieval of geospatial data. Supplementary benefits of this standard for metadata are to facilitate the organization and management of geospatial data, and to provide information about an organization's database to others. This standard for the implementation and documentation of metadata furnishes those unfamiliar with geospatial data the appropriate information to characterize their geospatial data and it makes possible dataset cataloguing which enables data discovery, retrieval and reuse.

This International Standard defines metadata elements, provides a schema and establishes a common set of metadata terminology, definitions, and extension procedures. Following ISO Directives, the standard is divided into two major parts: a normative section which users must follow to be compliant with the standard; and an informative section which helps guide them in its uses and provides examples to aid in understanding. Due to the diversity of geographic data, no single set of metadata elements will satisfy all requirements. For this reason the ISO metadata standard provides a standard way for users to extend their metadata and still ensure interoperability. By using standard methods other users will be able to understand and use this extended metadata. In the normative section the standard defines the schema required for describing geographic information. It defines the mandatory metadata elements and the characteristics required to provide information about the identification, the extent, the quality, the spatial and temporal schema, spatial reference, and distribution of digital geospatial data. The standard applies to all geographic data, it is applicable to datasets in series, datasets, individual geographic features, and their attributes. It also defines and standardizes a comprehensive set of optional metadata elements and characteristics, necessary to fully and extensively document a dataset.

For ease of understanding these metadata elements are divided into 11 sections as shown in Table 3.1. Among these metadata entity sections, the Identification entity is the only mandatory one. It contains mandatory, conditional, and optional elements to uniquely identify the data. The Identification entity includes information about the citation for the resource, an abstract, the purpose, credit, the status and points of contact. Table 3.2 summarizes the Identification information. It may be specified (subclassed) as DataIdentification when used to identify data and as ServiceIdentification when used to identify a service. The ServiceIdentification provides a high level description of a service. For further information see [23].

The other entity sections also can contain some optional sub-entities. The UML diagrams illustrating all of the metadata schemas are given in Appendix A.

Table 3.1: Summary of the ISO/TC 211 Metadata entity set information.

Entity Name	**Description**	**Max. No.**
MD_Metadata	Aggregate of the entities below	
MD_Identification	Information to identify the data	N
MD_Constraints	Constraints placed on the data	N
MD_DataQuality	Overall quality of the data	N
MD_MaintenanceInformation	Information about the frequency and scope of metadata updates	1
MD_ReferenceSystem	Description about the reference systems used in the dataset	N
MD_SpatialRepresentation	Digital representation of spatial information	N
MD_PortrayalCatalogueReference	Information about the catalogue for the portrayal of a resource	N
MD_Distribution	Information about the distributor of the data	1
MD_MetadataExtensionInformation	information describing metadata extensions	N
MD_ApplicationSchemaInformation	Information about the conceptual schema of a dataset	N
MD_ContentDescription	Information about the catalogue for image data characteristics	N

Table 3.2: Summary of the Identification information.

Entity Name	**Description**	**Maximum Occurrence**
MD_Identification	Aggregate of the entities below	N
MD_Format	Format of the data	N
MD_BrowseGraphic	Graphic overview of the data	N
MD_Usage	Specific uses of the data	1
MD_Constraints	Constraints placed on the resource	N
MD_Keywords	Keywords describing the resource	N
MD_Maintenance	Frequency and scope of the data updates	N

3.3 XML Metadata files for ISO 19115 Standards

XML (Extensible Markup Language) [46] is a subset of SGML (Standard Generalized Markup Language), the international standard for defining descriptions of the structure and content of different types of electronic document on the World Wide Web. XML is not a single, predefined markup language (such as HTML, which defines a way to describe web pages). XML is a metalanguage – a language for describing other languages. XML lets you define your own customized markup languages for limitless different document classes.

XML shows great promise for the definition, exchange, and processing of structured information on the WWW, including geospatial data in general. Fortunately, geospatial metadata have been encoded in SGML, the parent form of XML, so the availability of free and commercial general-purpose XML tools to process metadata shows great benefit for the GIS community without the need to create and maintain special-purpose parsers and presentation schemes. Teng [43] investigated the use of XML for query processing in Web-based geospatial data warehouses. It is evident that XML is a powerful tool to define and exchange geospatial metadata, since it is a platform and vendor independent, software-independent, extensible, reliable document exchange. XML metadata files for ISO 19115 standard can be easily created using the short names of the metadata entities (elements) provided in the standard. A sample of the XML ISO 19115 metadata file is given in Appendix B.

Chapter 4

Data Sets for Testing Distributed Geospatial Systems

This chapter describes the data sets used for testing the distributed geospatial systems developed in this research. The data sets include GSHHS and GMT contextual data, CEONet geospatial metadata, CCRS raster data and CLI vector data.

4.1 GSHHS and GMT Contextual Data for User Interface of Catalog Server

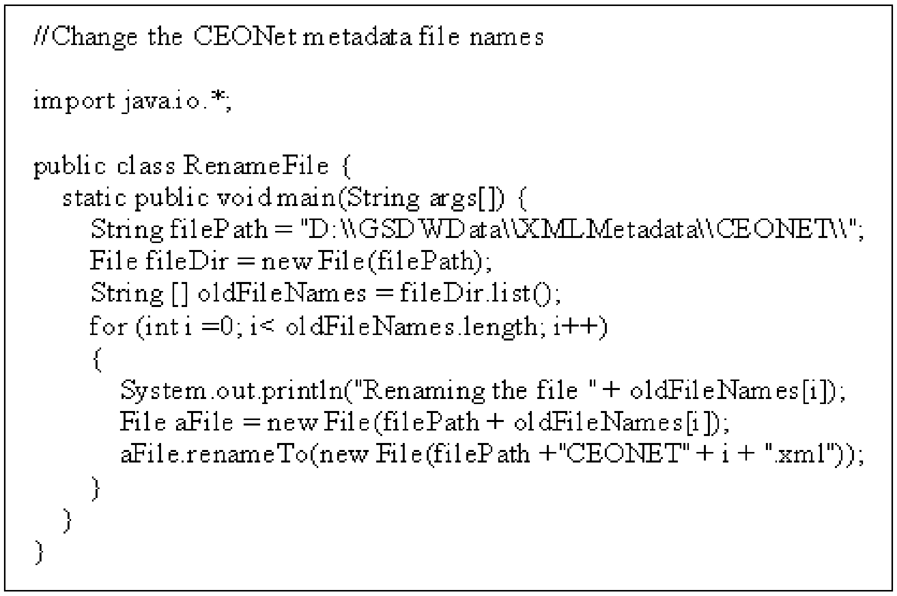
In Xiao's work [49], two kinds of contextual data were used for building the user interface of a catalog server: polygon data and polyline data. The polygon data include shorelines, lakes, and islands in lakes, which comes from the Global, Self-consistent, Hierarchical, High-resolution Shoreline database (GSHHS)[47]. The polyline data include political borders and rivers, which comes from the Generic Mapping Tools (GMT) [48]. Both of them are used in this research for the same purpose, i.e. to build the user interface of the catalog server.

In this research, the GSHHS database provides the polygon data of shorelines, lakes, and islands in lakes for the entire world to build the user interface of catalog server. The details about processing and assembly of the GSHHS data are described in [47]. The GMT database is a free, open source collection of approximately 60 UNIX tools that allow users to manipulate (x, y) and (x, y, z) data sets (including filtering, trend fitting, projecting, etc.) and produce Encapsulated PostScript File (EPS) illustrations ranging from simple x-y plots through contour maps to artificially illuminated surfaces and 3-D perspective views in black and white, grey tone, hachure patterns, and 24-bit colour. GMT supports 25 common map projections plus linear, log, and power scaling, and comes with support data such as coastlines, rivers, and political boundaries. GMT can also dump ASCII polyline data into a file. In this research, the GMT database provided the polyline data of political borders and rivers for the entire world to build the user interface for the catalog server.

4.2 CEONet XML Metadata for Metadata Service

As we discussed in Chapter 3, metadata is the "data about data" describing the content, quality, condition, and other characteristics of data and there are many different standards. Teng [43] dealt with the FGDC CSDGM XML metadata set from the Canada Earth Observation Network (CEONet). There are a total of 6979 metadata files in this metadata set and they require about 100 MB of disk space. Teng [43] transformed these FGDC CSDGM standard XML metadata files to ISO/CD 19115.2 standard XML metadata files. In this research, we use these ISO/CD 19115.2 standard XML metadata files as the test data set to perform metadata query processing on the catalog.

The names of the original metadata files are very long and it is not convenient to manipulate them. For example, *isr==United _Nations _Environment _Programme _@FSL@Global _Resource _Information _Database _@HYP@ _SIOUX _FALLS==United _Nations _Environment _Programme _@FSL@ _Global _Resource _Information _Database _@HYP@ _SIOUX _FALLS.xml* is one of the file names. Some of them even have more than 255 characters which causes difficulty in the Windows operating system environment. We wrote a small Java program called *RenameFile.java* using the API *File* in the package Java *java.io* to change the metadata file names into shorter names, such as *CEONET7777.xml*. The source code of this Java program is shown in Figure 4.1.



```
//Change the CEONet metadata file names

import java.io.*;

public class RenameFile {
   static public void main(String args[]) {
      String filePath = "D:\\GSDWData\\XMLMetadata\\CEONET\\";
      File fileDir = new File(filePath);
      String [] oldFileNames = fileDir.list();
      for (int i =0; i< oldFileNames.length; i++)
      {
         System.out.println("Renaming the file " + oldFileNames[i]);
         File aFile = new File(filePath + oldFileNames[i]);
         aFile.renameTo(new File(filePath +"CEONET" + i + ".xml"));
      }
   }
}
```

Figure 4.1: The source code of Java program *RenameFile.java*.

4.3 CCRS Imagery Data for Geospatial Data Services

In this research we used some of the Canada Centre for Remote Sensing (CCRS) imagery data as test data set for actual raster data services of our system. We downloaded 40 JPEG image files of Canada with a total size of 3MB from the CCRS website [6] which show some of the images of Canada from different space sensors and satellites. These image data allow one to look at the mountains of Kluane National Park in 3-D, see ice in the waters off Ranklin Inlet in July, and to visit Quebec City, Summerside, Trepassey, Calgary, and Fredericton.

We downloaded the National Oceanic and Atmospheric Administration (NOAA) Advanced Very High Resolution Radiometer (AVHRR) composite image data *Canada_Noaa* and the Radarsat (mosaic) image data *Canada_Mosaic* from the CCRS Geogratis website [13]. The NOAA AVHRR composite of Canada is in JPEG format and contains 3 channels of 5700 pixels by 4800 lines, with a size of $2,475$ KB. Research has been conducted at the CCRS by the Environmental Monitoring Section to study the use of the NOAA image for various projects such as: estimating Net Primary Productivity, measuring the albedo at the top of the atmosphere, and detecting forest fires. The Radarsat images of Canada (mosaic) cover the whole Canada and have multiple resolutions (pixel sizes of 250m, 500m, 750m, and 1000m). They are in geotiff format with a total size of $609,410$ KB. Table 4.1 summarizes the CCRS imagery data files.

Table 4.1: Contents of the CCRS Imagery Data Files.

Raster Data File Name	**Coverage**	**No. of Files**	**File Size (KB)**
Images of Canada	Places of Canada	40	3,100
Canada_NOAA	Whole Canada	1	2,475
Canada_Mosaic1000	Whole Canada	1	26,757
Canada_Mosaic750	Whole Canada	1	47,551
Canada_Mosaic500	Whole Canada	1	106,951
Canada_Mosaic250	Whole Canada	1	427,651
Total		45	612,510

4.4 Canada Land Inventory Data for Geospatial Data Services

In this research we used the Canada Land Inventory (CLI) data as test data sets for the vector data services of our system.

4.4.1 Introduction to CLI data

The Canada Land Inventory (CLI) is a comprehensive survey of land capability and use designed to provide a basis for resource and land use planning. It includes assessments of land capability for agriculture, forestry, recreation, wildlife, and land use planning projects in each province.

The CLI covers the settled portions of rural Canada and adjoining areas which affect the income and employment opportunities of rural residents. Thus it covers the area of Canada where questions of alternative use of land have a strong bearing on sustainable rural development.

The broad objectives of the CLI are to classify lands as to their capabilities, to obtain a firm estimate of the extent and location of each land class, and to encourage use of CLI data in planning. Lands are classified according to their physical capability for use in agriculture, forestry, recreation, and wildlife, and their present use.

We downloaded 1:250,000 scale Canada Land Inventory data in Arc/Info Export ('.E00') format free of charge from the Geogratis website [13]. This data set contains 1,232 zipped ASCII Arc/Info Export files covering the thematic layers of Soil Capability for Agriculture, Soil Capability for Forestry, Land-Use Capability, Land Capability for Recreation, Land Capability for Ungulate Wildlife, Land Capability for Waterfowl Wildlife plus 458 zipped ASCII Arc/Info Export files for the Roads and place Names. Table 4.2 summarizes the CLI ASCII data file contents.

Table 4.2: Summary of the CLI Arc/Info Export(E00) data files.

No.	**Theme**	**Number of Files**	**Size (MB)**
1	Agriculture	193	860
2	Forestry	180	785
3	Land-Use	202	1,040
4	Recreation	224	941
5	Ungulate	227	527
6	Waterfowl	206	644
7	Roads and Place Names	458	12.2
Total		1690	4809.2

4.4.2 Format Analysis for CLI Arc/Info Export (E00) Data

The CLI data we downloaded are ASCII files in Arc/Info Export ('.E00') format.

The Geographic Information System designed by Environmental Systems Research Institute (ESRI) considers the Arc/Info export/import file format to be proprietary. Here we give a brief format analysis for the Arc/Info Export (E00) file [27].

Overall Organization

The export file begins with a line with three fields: 1- an initial 'EXP'; 2- what appears to be a constant of '0' for uncompressed files, and 1 for compressed files (FULL or PARTIAL); 3- the pathname for the creation of the export file. The export file ends with a line beginning with 'EOS'.

The ARC sections are included first, in the following order (note that all these sections are not always present): ARC (arc coordinates and topology), CNT (Polygon Centroid Coordinates), LAB (Label Points), PAL (Polygon Topology), TOL (Tolerance Type), TXT (Annotations), SIN (Spatial Index), LOG (Coverage History), PRJ (Projection Parameters), RXP (Specific to Regions), RPL (Region Coverages). These are followed by the INFO sections, which contains descriptive information about the arcs and polygons described in the ARC sections.

The beginning of each ARC section is indicated by the section name (a three-character identifier) followed by '2' for single- precision or '3' for double-precision. Floating point values carry 8 digits (e.g. -1.0000000E+02) in single-precision coverage, and 15 digits (e.g. -1.19299887000023E+02) in double-precision coverage.

Each ARC section ends with a line of seven numbers beginning with a -1 and followed by six zeros, except the SIN, LOG, and PRJ sections which end in 'EOX', ‘EOL’, and ‘EOP’, respectively. The LAB section uses a slight variation of this -1 ending line (see below). The format for each ARC section is specific to that type of section.

The beginning of the INFO sections is indicated by ‘IFO 2’, and its end is indicated by ‘EOI’. Each INFO section begins with the file name. For example, the polygon attribute table would begin with ‘A021G.PAT’ on a line by itself. The format is the same for every INFO section (see The INFO Section Formats below). A sample of parts of the A021G.E00 CLI Arc/Info file is given in Appendix D.

The ARC Section Formats

Formats are given for the four most common ARC sections, i.e. ARC, CNT , LAB and PAL.

ARC: The ARC (arc coordinates and topology) section consists of repeating sets of arc information. The first line of each set has seven numbers: 1. coverage#; 2. coverage-ID; 3. from node; 4. to node; 5. left polygon; 6. right polygon; 7. number of coordinates. The subsequent lines of a set are the coordinates with two x-y pairs per line, if the coverage is single-precision. If there are an odd number of coordinates, the last line will have only one x-y pair. Double-precision puts one coordinate pair on each line.

CNT: The CNT (Polygon Centroid Coordinates) section contains the centroid of each polygon in the coverage. It has sets of centroid information with an initial coordinate line and, if there are labels, the label ids will follow, with up to 8 label ids per line. The coordinate line has three fields: 1. number of labels in polygon; 2. centroid x; 3. centroid y.

LAB: The LAB (Label Points) section consists of repeating sets of label point information. The first line of each set has four numbers: 1. coverage-ID; 2. Polygon ID; 3. x coordinate; 4. y coordinate. The second and final line of the set gives the label box window. This information contains repetitions of the x and y coordinates. Note that the LAB section ends with a different '-1' line than the other sections.

PAL: The PAL (Polygon Topology) section consists of repeating sets of polygon information. For single-precision, the first line of each set has five numbers: 1. number of arcs in polygon; 2. x min of polygon; 3. y min of polygon; 4. x max of polygon; 5. y max of polygon. The subsequent lines of a set give information on the arcs which comprise the polygon. There are three numbers per arc with information for two arcs per line: 1. Arc_Id (negative if reversed); 2. From_Node_Id (if arc is reversed, then this is the arc's To_Node_Id); 3. Adjacent_Polygon_Id (Id of the polygon that shares this arc with the current polygon). The first polygon given is the universal polygon, i.e. the polygon containing the whole data set.

The information in the ARC and PAL sections is extremely important for every application since it provides the basic geospatial coordinate data for all polyline and polygon objects.

The INFO Section Formats

Formats are given for the three most common INFO sections, i.e. AAT (Arc

Attribute Table), BND (Coverage Min/Max Coordinates), and PAT (Polygon or Point Attribute Table).

AAT: The AAT (Arc Attribute Table) contains seven basic fields whose attribute names are as follows: FNODE_# (start-node number); TNODE_# (end-node number); LPOLY_# (left-polygon number); RPOLY_# (right-polygon number); LENGTH (arc length); COVER_# (arc number); COVER_ID (arc ID). Additional attributes may be added as desired, after the COVER_ID attribute. For example, in the CLI data for the thematic layer Roads, there is an additional attribute LEGEND which takes one of the values of ROADS, RAILROAD or TRAIL for each arc. This information is very useful in our research since it gives us the type of the road represented by this arc when we visualize the CLI data.

BND: The BND (Coverage Min/Max Coordinates) table contains four fields whose attribute names are self-explanatory: XMIN; YMIN; XMAX; YMAX. The information in the BND table is very useful in our research since it gives us the geospatial BoundingBox of each data set. We use it to create the entity of geographicBox in the XML metadata for the purpose of boundary search.

PAT: The PAT (Polygon or Point Attribute Table) contains four basic fields whose attribute names are as follows: AREA (polygon area); PERIMETER (polygon perimeter); COVER_# (polygon number); and COVER_ID (polygon ID). Additional attributes may be added as desired, after the COVER_ID attribute. In the CLI data for the thematic layer Agriculture, Forestry, Land-use, Recreation, Ungulate and Waterfowl, there are several additional attributes which characterize the classification of the physical capability for each polygon area. The first 4 additional attributes are: CLASS_A (The primary and/or dominant CLI class), PERCENT_A (The proportion (% base 10) of the polygon in Class_A), SUBCLAS_A1 (The primary limitation for the proportion of the polygon in Class_A), and SUBCLAS_A2 (The secondary limitation for the proportion of the polygon in Class_A). We use these attributes to determine the fill-colour of the polygon when we visualize the CLI data. In the CLI data for the thematic layer PlaceNames, there is an additional attribute *PPPTNAME* which gives the place name for the area represented by this polygon.

4.4.3 CLI Data Structures and Pre-processing

In this research, we provide a service of CLI data visualization as requested by a client. In order to process the CLI E00 data efficiently, we built a data structure to represent the CLI data called the CLI Data Structure. The CLI data structure is shown with a UML diagram in Figure 4.2.

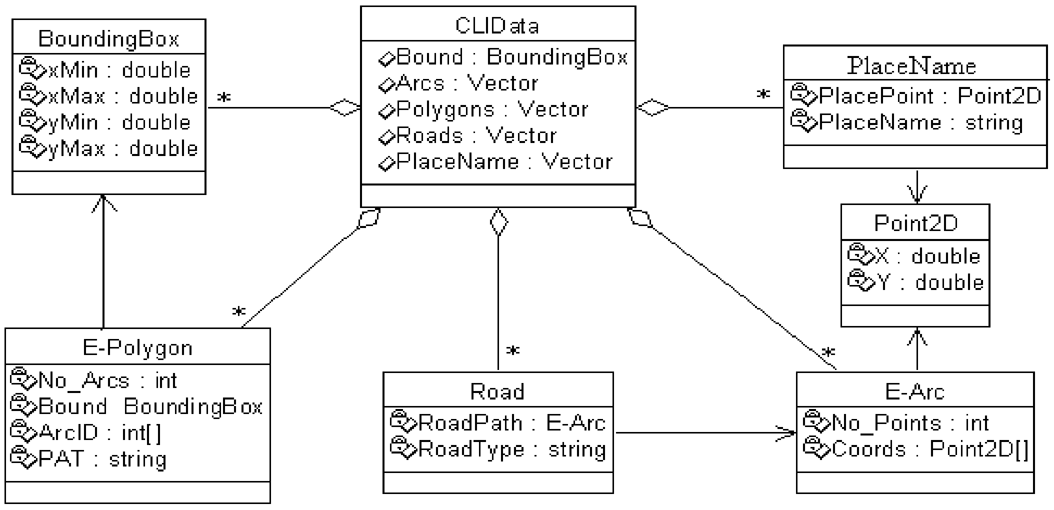
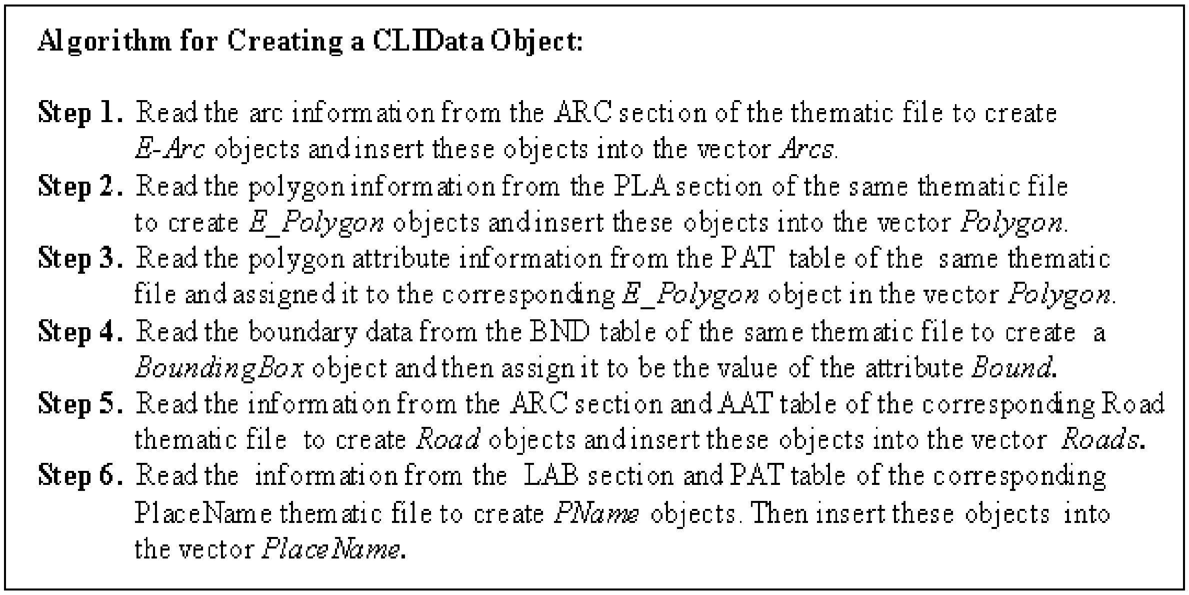


Figure 4.2: A UML diagram of the CLI data structure.

Each CLI data file for thematic layer Agriculture, Forestry, Land-use, Recreation, Ungulate and Waterfowl corresponds to a CLIData object. The attribute values of each CLIData object are assigned using the algorithm shown in Figure 4.3.



Algorithm for Creating a CLIData Object:

Step 1. Read the arc information from the ARC section of the thematic file to create *E-Arc* objects and insert these objects into the vector *Arcs*.
Step 2. Read the polygon information from the PLA section of the same thematic file to create *E_Polygon* objects and insert these objects into the vector *Polygon*.
Step 3. Read the polygon attribute information from the PAT table of the same thematic file and assigned it to the corresponding *E_Polygon* object in the vector *Polygon*.
Step 4. Read the boundary data from the BND table of the same thematic file to create a *BoundingBox* object and then assign it to be the value of the attribute *Bound*.
Step 5. Read the information from the ARC section and AAT table of the corresponding Road thematic file to create *Road* objects and insert these objects into the vector *Roads*.
Step 6. Read the information from the LAB section and PAT table of the corresponding PlaceName thematic file to create *PName* objects. Then insert these objects into the vector *PlaceName*.

Figure 4.3: The algorithm for creating a CLIData object.

To create a CLIData object, one needs three CLI thematic files: one for the thematic layers of Agriculture, Forestry, Land-use, Recreation, Ungulate and Waterfowl; one for the corresponding thematic layer of Road and the other for the corresponding thematic layer of PlaceName.

Chapter 5

Geospatial Data Index Structures and Search Engine

In this research, we employ the R-Tree index structure for spatial objects and AVL-Tree index structure for string objects initially developed by Teng [43] to build the Geospatial Data (GSD) search engine. Correspondingly, there are two search types that have been implemented in this research: geospatial coordinates search and string (keyword) search. In this chapter, we discuss the approaches implemented that use geospatial metadata to catalog, query and browse geospatial data.

5.1 AVL-Tree and R-Tree Index Structures

In this research we implement the combined geospatial coordinates and strings (keywords) indexing and search scheme. The geospatial coordinates represent the geographic areal domain of the data set: western-most/eastern-most longitudes and northern-most/southern-most latitudes of the limit of the dataset expressed in decimal degrees. Strings represent the theme and place name keywords which are the common-use words or phases used to describe the theme or place names involved in the dataset.

The strings (keywords) are indexed with the AVL-Tree data structure [45] and the geospatial coordinates are indexed with the R-Tree data structure [14]. Based on their well-known characteristics, these two indexing structures give fast search performance.

5.1.1 AVL-Tree Index Structure

Binary search techniques require time $O(\log N)$ to search a balanced binary tree, where N is the number of elements in the tree. If data are inserted in order using a naive algorithm, binary search degenerates to sequential search if the binary tree is not balanced. This problem can be solved by rebalancing the tree after each insertion or deletion. In rebalancing, nodes are rearranged via transformations called rotations using an algorithm that tends to minimize the tree's height. There are several schemes for rebalancing binary trees. A common type of balanced binary search tree is the AVL-Tree. An AVL-Tree [45], invented by Russian mathematicians G. M. Adelson-Velskii and E. M. Landis [1], ensures that the time for a search, insertion or deletion is $O(\log N)$.

In this research, we implemented the AVL-Tree data structure using Java. In our implementation of the AVL-Tree data structure, each binaryNode is a quadplex of (*Keyword*, *Vec_file_Name*, *Left*, *Right*), where *Keyword* is a string object and acts as the key of AVL-Tree, *Vec_file_Name* is an object of Java class Vector which holds all the names of the ISO XML metadata files (will be discussed below) that own the same keyword, *Left* and *Right* are the left and right children of this binaryNode. The insertion algorithm implemented ensures that if several metadata files have the same keyword, then all of these metadata file names will be put in the same node of the AVL-Tree. The search algorithm of the AVL-Tree is the typical search algorithm of binary search trees and requires an exact match. The returned search result will be a collection of metadata file names.

5.1.2 R-Tree Index Structure

The R-Tree [14] is an object hierarchy which is applicable to arbitrary spatial objects. It is a multi-dimensional generalization of the B-Tree, that preserves height-balance, and can be used to store multi-dimensional geometric objects, such as points, lines, polygons and polyhedrons.

In this research, we implemented the R-tree index structures using Java. In our R-Tree implementation, each data object is a duplex of (*Vec_file_Name*, *Bound*), where *Bound* is an object of the class BoundingBox which has four attribute members of *xMin, xMax, yMin, yMax* representing the minimum bounding rectangle of the data object, *Vec_file_Name* is an object of Java class Vector which holds all the names of the ISO XML metadata files (will be discussed below) that own the same minimum bounding rectangle *Bound*. We used the linear node splitting method for R-tree insertion. That is, we pick up the two bounding boxes with the greatest normalized separation along both axes as the splitting seeds when splitting the node [14]. It is notable that we improved the insertion algorithm by using an object *Vec_file_Name* of the Java class Vector to hold the names of metadata files with the same bounding rectangle. That is, if there are several metadata files have the same bounding rectangle, then only one data object is created and all of their file names are put in the vector *Vec_file_Name* of the data object.

Traditionally, the R-Tree supports both range search and point search. In order to locate all objects which intersect a query rectangle, the search algorithm descends the tree from the root. The algorithm recursively traverses down the subtrees of bounding rectangles that intersect the query rectangle. When a leaf node is reached, bounding rectangles are tested against the query rectangle and their objects are fetched for testing if they intersect the query rectangle. For the GSD system, this traditional R-Tree search algorithm usually returns too many search results, especially for a very large number of datasets being catalogued. This situation happens when the size of the overlap of the query rectangle and the dataset bounding rectangle is quite small relative to that of query rectangle or dataset bounding rectangle.

We improved the R-Tree search algorithm by introducing a match factor k which takes a real value in the range of $(0, 1]$. With the match factor k, the improved R-Tree range search algorithm only returns data objects satisfying the condition

$$\min\left\{\frac{S(O)}{S(D)}, \frac{S(O)}{S(Q)}\right\} \geq k, \tag{5.1}$$

where $S(D)$ is the size (area) of the minimum bounding rectangle of data object, $S(Q)$ is the size (area) of the query rectangle and $S(O)$ is the size (area) of the overlap of the query rectangle and the minimum bounding rectangle of the data object. Figure 5.1 illustrates how k controls which data sets are returned from a search that uses the match factor k. Suppose the sizes (areas) of the query rectangle Q, the minimum

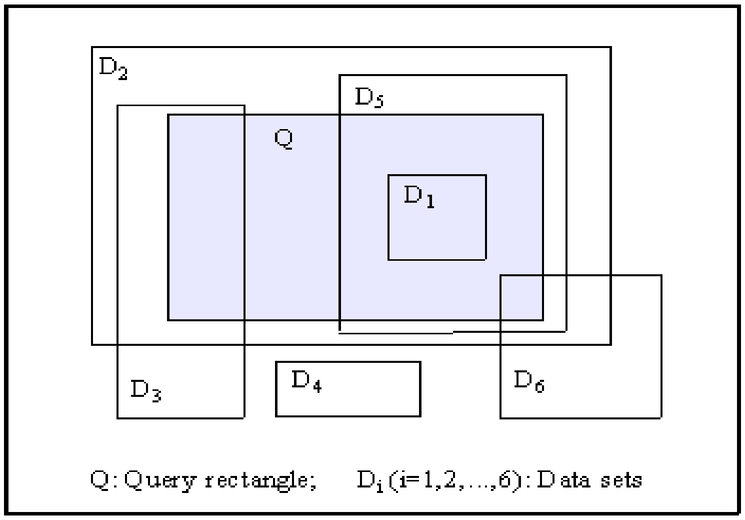


Figure 5.1: The effect of match factor on the search result.

bounding rectangle of data sets D_i and their overlap $O_i = D_i \bigcap Q \ \ (i = 1, 2, ..., 6)$ are given in Table 5.1. The relation between the match factor k and the returned data sets can be shown in Table 5.2.

Table 5.1: The areas of the rectangles in Figure 5.1.

$(O_i = D_i \bigcap Q)$

i	$S(Q)$	$S(D_i)$	$S(O_i)$	$S(O_i)/S(Q)$	$S(O_i)/S(D_i)$	$\min\{S(O_i)/S(D_i), S(O_i)/S(Q)\}$
1	10	1	1	0.1	1	0.1
2	10	16	10	1	0.625	0.625
3	10	4	2	0.2	0.5	0.2
4	10	1	0	0	0	0
5	10	7	5	0.5	0.714	0.5
6	10	3	0.25	0.025	0.083	0.025

Obviously, the smaller the match factor k is, the larger the number of found items returned. The end-user (GSD Client) can control the number of found items

Table 5.2: The relation between the match factor k and the returned data sets.

Range of k	Returned Data Sets
$0 \leq k \leq 0.025$	D_1, D_2, D_3, D_5, D_6
$0.025 < k \leq 0.1$	D_1, D_2, D_3, D_5
$0.1 < k \leq 0.2$	D_2, D_3, D_5
$0.2 < k \leq 0.5$	D_2, D_5
$0.5 < k \leq 0.625$	D_2
$k > 0.625$	null

returned by setting the match factor k in the range $(0, 1]$. The returned search result is a collection of metadata file names.

5.2 Hierarchy Indexing Scheme for Heterogeneous Geospatial Databases

We developed a hierarchy search engine for our GSD system using the AVL-Tree and R-Tree index structures described above. This section introduces this indexing scheme.

5.2.1 Building Phases of the GSD Index

The building of the hierarchical search engine for heterogeneous geospatial databases in this research involves the three phases shown in Figure 5.2.

Phase 1: Creating ISO XML Metadata files

In order to index and search the heterogeneous geospatial databases effectively, we use metadata tools to create the corresponding geospatial metadata for each type of the geospatial data. The ISO standard (see Chapter 3 for the ISO XML metadata

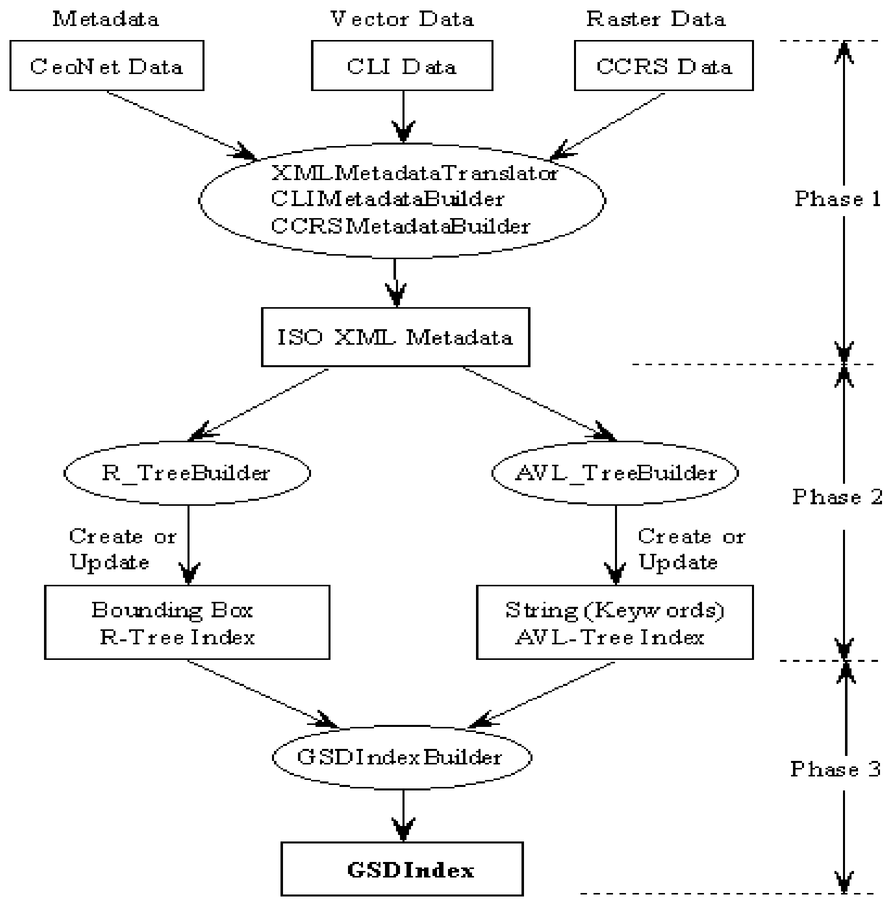


Figure 5.2: Architectural overview of the GSD index scheme.

standards) is necessary to meet "international" requirements since numerous organizations plan to use the ISO metadata standard once it has been approved by the ISO Standards Committee. In Teng's work [43], a metadata tool called *XMLMetadata-Translator* was developed to translate from CEONet FGDC XML metadata files to ISO XML metadata files (ISO/TC 211 CD 2). In this research, we developed two metadata tools called *CLIMetadataBuilder* and *CCRSMetadataBuilder* using Java to create ISO XML metadata files (ISO/TC 211 CD 3) for the Canada Land Inventory (CLI) vector data and the Canada Centre for Remote Sensing (CCRS) raster data.

For other types of geospatial data, a metadata tool to create the corresponding ISO XML metadata files is required. For the purpose of indexing and searching the GSD system, when creating the ISO XML metadata files, we emphasized the following attribute entities of the ISO XML metadata files:

- **geographicBox** in the Identification Information section of ISO XML metadata files, and surrounded by the XML tag pair of <geoBox> and </geoBox> which are nested within the XML tag pair of <DataIdent> and </DataIdent> (see Appendices B and C). This attribute entity gives the minimum bounding rectangle within which data is available. The information given in this entity is used to build the R-Tree index of the geospatial objects.

- **keyword** in the Identification Information section of ISO XML metadata files, and surrounded by the XML tag pair of <Keywords> and </Keywords> (see Appendices B and C). This attribute entity gives the commonly used word(s) or formalised word(s) or phrase(s) used to describe the subject of the data. The information given in this entity is used to build the AVL-Tree index of the string objects (keywords).

- **onLine** in the Distribution Information section of ISO XML metadata files, and surrounded by the XML tag pair of <onLineRes> and </onLineRes> which are nested within the XML tag pair of <Distrib> and </Distrib> (see Appendices B and C). This attribute entity provides information about online sources from which the resource can be obtained. The information given in this entity is used to return search results to the client about the online linkage of the found data resource.

For all other types of geospatial data service of the GSD system, one should provide corresponding ISO XML metadata files which define the above three attribute entities.

Phase 2: Creating or Updating R-Tree Index and AVL-Tree Index

After being created with appropriate metadata tools, the ISO XML metadata files are used to build or update the bounding box R-Tree index with the information given in the attribute entity *geographicBox* and the keywords AVL-Tree index with the information given in the attribute entity *keyword*. To extract the required information from the entity sections of ISO XML metadata files, one can use the XML parser packages (e.g. Microsoft MSXML Parser 3.0 [26]) to parse entity values of XML metadata file or read the corresponding lines from the ASCII XML metadata file.

In this research, we developed two Java programs called *R-TreeBuilder* and *AVL-TreeBuilder* to create or update the BoundingBox R-Tree index and keywords AVL-Tree index. First, for simplicity, we read the lines of <westBL> ... </westBL>, <eastBL> ... </eastBL>, <southBL> ... </southBL> and <northBL> ... </northBL> from the attribute entity section *geographicBox* of an ASCII ISO XML metadata file, which represents a geospatial data set, and used a string tokenizer "<>" to parse the values of the bounding rectangle of the data set. Using these values of bounding rectangle and the name of the metadata file, we created an object of the class *DataObject* and then inserted this object into the R-Tree. If several metadata files have the same bounding rectangle, as mentioned above, then only one object of the class *DataObject* is created and therefore there is only one R-Tree leaf-node corresponding to these metadata files. Secondly, we read all the lines of <keyword> ... </keyword>'s from the attribute entity section *keyword* of a ASCII ISO XML metadata file, and used a string tokenizer "<>" to parse the string values of keywords (including place names) of the data set. Using these string values and the name of the metadata file, we created an object of the class *binaryNode* and then inserted this object into the object of AVL-Tree. Finally, the objects of R-Tree index and AVL-Tree index created from all the currently available metadata files are saved as persistent files using Java object serialization techniques.

Phase 3: Composing the GSD Index

Using the objects of R-Tree index and AVL-Tree index created in Phase 2, we developed a Java program called *GSDIndexBuilder* to compose the GSD index. The *GSDIndexBuilder* consists of a GSDIndex class definition and a main program which loads objects of the R-Tree index and AVL-Tree index from the persistent files, and creates a GSDIndex object. The GSDIndex object then is also saved as a persistent file using Java object serialization techniques. This allows it to be loaded by the query servlet in the catalog server. The GSDIndex class implements the combined search algorithm which supports the string keywords query and/or boundary BoundingBox query with a match factor k. The details about the combined search algorithm are described in the next section.

5.2.2 Architectural Overview and UML Diagram of the GSD Index Scheme

The UML diagram of the Java classes structure for the hierarchy indexing scheme is illustrated in Figure 5.3.

It is worth pointing out that this hierarchical GSDIndex is created in incremental fashion. That is, if new geospatial data are available and expected to be provided as the data service of the GSD system, we simply load the current objects of R-Tree index and AVL-Tree index and enlarge them by inserting the corresponding bounding rectangles and keywords of the new data following the steps in Phase 2. The updated R-Tree index and AVL-Tree index are then used to create a new GSDIndex. This incremental update capability enables the GSD system to add new geospatial data sets automatically.

5.3 GSD Search Engine

In this research, we implemented the GSD search engine using Java servlet techniques. Figure 5.4 below gives an architectural overview of the implementation of the GSD search engine.

First, the GSD client sends a query request through the user interface of GSD (initially developed by Xiao [49]) by using mouse selection and/or filling in the corresponding text fields. The query request consists of a query rectangle (western-most/eastern-most longitudes and northern-most/southern-most latitudes in decimal degrees) with a match factor $k \in (0, 1]$ or a string of keywords to search or both.

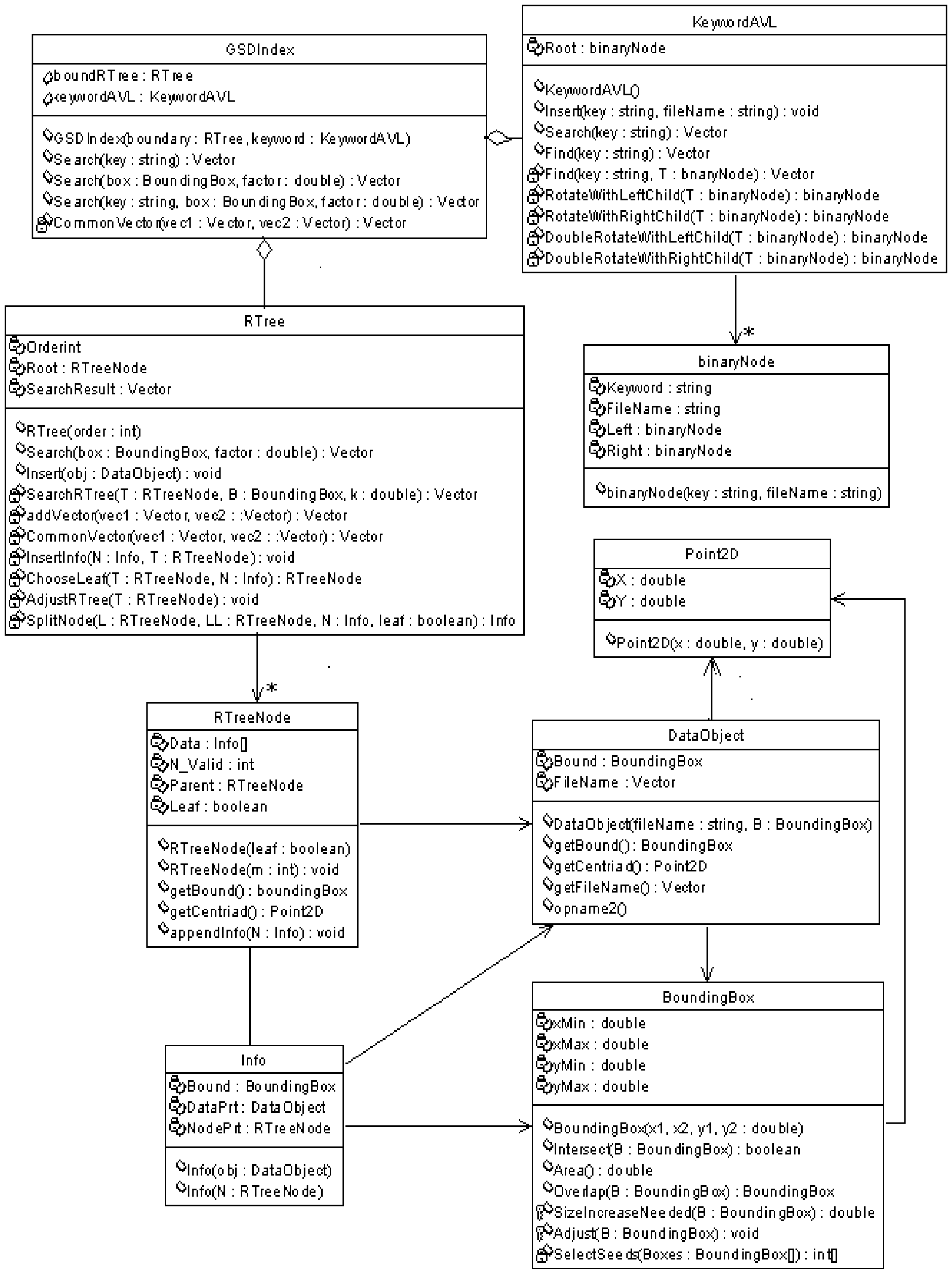

GSDIndex
boundRTree : RTree
keywordAVL : KeywordAVL
GSDIndex(boundary : RTree, keyword : KeywordAVL)
Search(key : string) : Vector
Search(box : BoundingBox, factor : double) : Vector
Search(key : string, box : BoundingBox, factor : double) : Vector
CommonVector(vec1 : Vector, vec2 : Vector) : Vector
KeywordAVL
Root : binaryNode
KeywordAVL()
Insert(key : string, fileName : string) : void
Search(key : string) : Vector
Find(key : string) : Vector
Find(key : string, T : bnaryNode) : Vector
RotateWithLeftChild(T : binaryNode) : binaryNode
RotateWithRightChild(T : binaryNode) : binaryNode
DoubleRotateWithLeftChild(T : binaryNode) : binaryNode
DoubleRotateWithRightChild(T : binaryNode) : binaryNode
RTree
Orderint
Root : RTreeNode
SearchResult : Vector
RTree(order : int)
Search(box : BoundingBox, factor : double) : Vector
Insert(obj : DataObject) : void
SearchRTree(T : RTreeNode, B : BoundingBox, k : double) : Vector
addVector(vec1 : Vector, vec2 : :Vector) : Vector
CommonVector(vec1 : Vector, vec2 : :Vector) : Vector
InsertInfo(N : Info, T : RTreeNode) : void
ChooseLeaf(T : RTreeNode, N : Info) : RTreeNode
AdjustRTree(T : RTreeNode) : void
SplitNode(L : RTreeNode, LL : RTreeNode, N : Info, leaf : boolean) : Info
*
binaryNode
Keyword : string
FileName : string
Left : binaryNode
Right : binaryNode
binaryNode(key : string, fileName : string)
Point2D
X : double
Y : double
Point2D(x : double, y : double)
*
RTreeNode
Data : Info[]
N_Valid : int
Parent : RTreeNode
Leaf : boolean
RTreeNode(leaf : boolean)
RTreeNode(m : int) : void
getBound() : boundingBox
getCentriad() : Point2D
appendInfo(N : Info) : void
DataObject
Bound : BoundingBox
FileName : Vector
DataObject(fileName : string, B : BoundingBox)
getBound() : BoundingBox
getCentriad() : Point2D
getFileName() : Vector
opname2()
BoundingBox
xMin : double
xMax : double
yMin : double
yMax : double
BoundingBox(x1, x2, y1, y2 : double)
Intersect(B : BoundingBox) : boolean
Area() : double
Overlap(B : BoundingBox) : BoundingBox
SizeIncreaseNeeded(B : BoundingBox) : double
Adjust(B : BoundingBox) : void
SelectSeeds(Boxes : BoundingBox[]) : int[]
Info
Bound : BoundingBox
DataPrt : DataObject
NodePrt : RTreeNode
Info(obj : DataObject)
Info(N : RTreeNode)

Figure 5.3: A UML class diagram of the GSD index scheme.

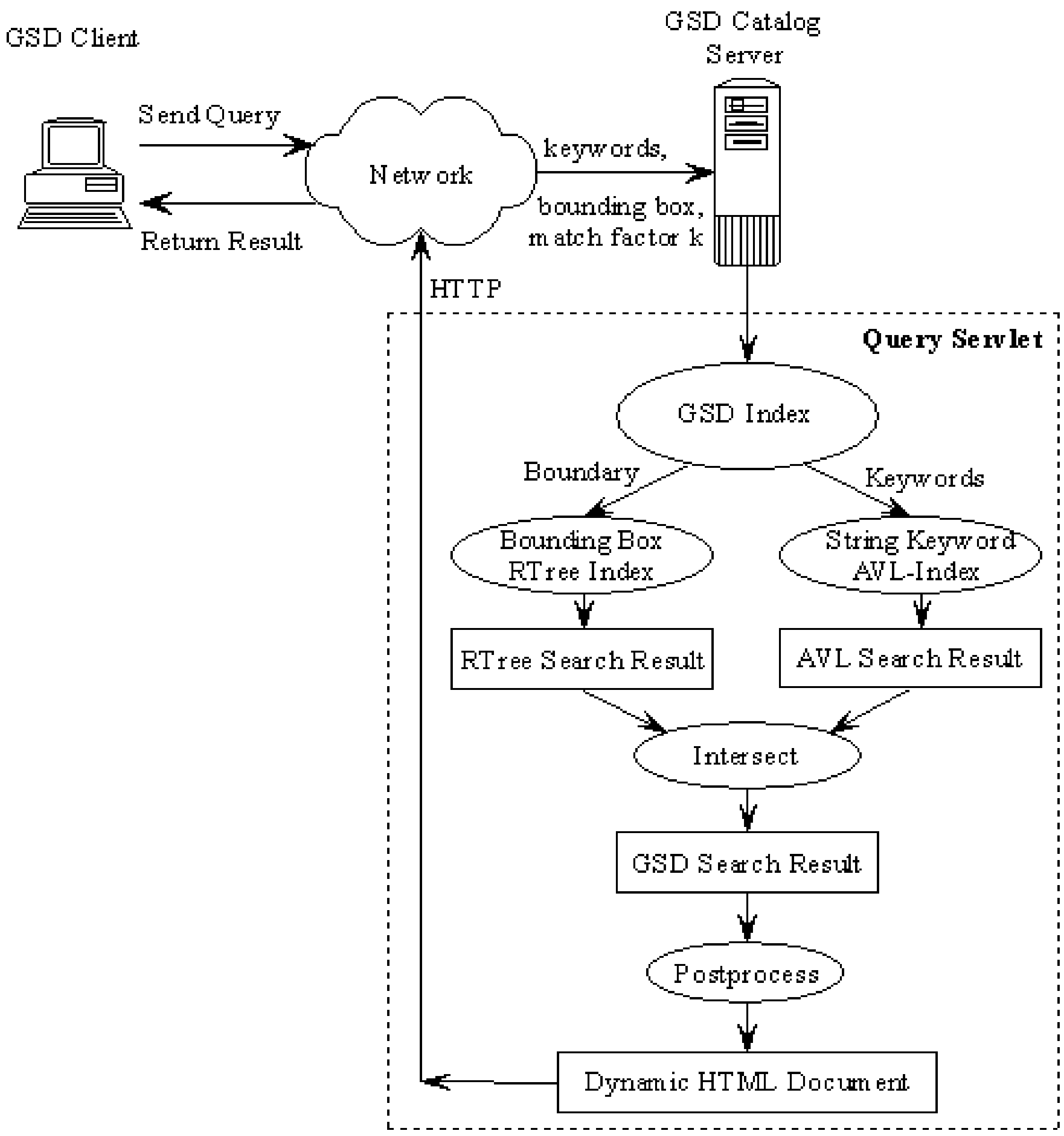
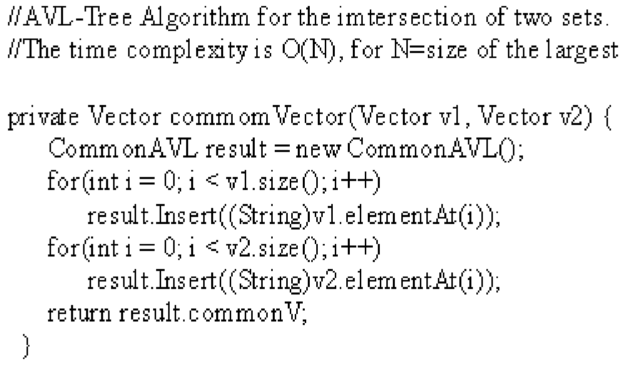


Figure 5.4: GSD Search Architecture.

By default, the match factor k has the value of 0.5. The string of keywords search supports the logical *AND* function to reduce the number of returned items. If there is more than one keywords to search, the addition sign "+" is used to logical combine the keywords in a string. For example, if one wants to find data about the Canada land inventory in the Fredericton area, he or she just fills in the keywords text field with "Canada land inventory + Fredericton". The string search is not case-sensitive. The "Begin Search" button is pressed to send the query request to the GSD search engine which resides in the GSD Catalog server as a Java servlet called *QueryServer*.

Secondly, the servlet QueryServer receives the query request and decides if it is a boundary search or a string search or a combination search. If it is the first time for QueryServer to receive a query request, then the Query Servlet will load the GSDIndex from a persistent file located at the GSD catalog server (servlet initialization). Otherwise, the GSDIndex already resides in the memory of the catalog server. Then the QueryServer passes the search parameters (query rectangle with match factor and keywords string) to the boundary R-Tree index and/or keywords AVL-Tree index to perform the corresponding search and obtain the search results (vectors containing metadata file names as their elements), respectively.

Thirdly, if both the R-Tree search result and AVL-Tree search result are non-empty, then the intersection (the common elements) of these two results is taken as the GSD search result. If either of the R-Tree search result or AVL-Tree search result is empty, the non-empty set is taken as the GSD search result. In this research, a modified insertion method of AVL-Tree [43] is used to implement the intersection algorithm (see Figure 5.5). The modified AVL-Tree insertion method puts the duplicates in the AVL-Tree's class attribute member *CommonV* (a vector). When we insert all the items of the two separate search results into an AVL-Tree, the intersection of the two search results is stored in the class attribute member *CommonV*. The time complexity of this algorithm is $O(N)$, where N is the size of the largest search result set.



```
//AVL-Tree Algorithm for the imtersection of two sets.
//The time complexity is O(N), for N=size of the largest set.

private Vector commomVector(Vector v1, Vector v2) {
    CommonAVL result = new CommonAVL();
    for(int i = 0; i < v1.size(); i++)
        result.Insert((String)v1.elementAt(i));
    for(int i = 0; i < v2.size(); i++)
        result.Insert((String)v2.elementAt(i));
    return result.commonV;
 }
```

Figure 5.5: The AVL-Tree algorithm for GSDIndex intersection (from [43]).

Finally, if the GSD search result is empty, the QueryServer sends back to the client a dynamic HTML document which shows a message of "No Match". Otherwise, the QueryServer will post-process the GSD search result before sending the client the query result. For each item of the GSD search result (a name of a metadata file), the post-processing performs the following tasks:

(a) Read the corresponding ISO XML metadata file;

(b) Extract the information about *Data Type*, *The Title of Data Set*, *On-Line Linkage* (if applicable), *Data Service Linkage* (if applicable) using the XML metadata tags from the metadata file;

(c) Using the above information, create a record for the dynamic HTML document.

Sometimes the number of found items in the GSD search result is quite large. For example, if we search the keyword "Canada", there are 716 items found. The post-processing of all found items and the transfer of the entire dynamic HTML document takes time and sometimes is not necessary. To increase the efficiency of the query process and to reduce the network traffic, the post-processing is designed in batch fashion, like other web search engines. That is, if the number of found items in the GSD search result is larger than a certain predetermined integer, say 20, then currently the QueryServer only post-processes the first 20 found items, sends the corresponding created dynamic HTML document to the client, informs the client how many items are found, and asks the client if he/she wants to see the next or previous 20 items. Upon receiving the client's further request of seeing the next or previous 20 items, the QueryServer then post-processes the next or previous 20 items and sends the result back to the client.

If the client issues a new query request, the QueryServer clears the vectors of the R-Tree search result and the AVL-Tree search result, and performs a new query process as described above.

Chapter 6

System Design and Implementation

The Web-oriented Distributed Geospatial Data Warehouse system (WDGSDW) developed in this research is a web-based information system and allows a user to query geographical information of the world and access to the geospatial data services the system provides using a web browser. This chapter describes the details of the design and implementation of WDGSDW.

6.1 Architetural Overview of WDGSDW

This research aims to develop a web-based multi-tier client/server architectural solution for the geographical data in distributed systems. The servers are distributed around the world and provide individual geospatial data services by connecting to some geospatial databases. The geospatial metadata is indexed by a catalog server. The users can then issue queries to this catalog server to look for the geospatial data services and/or resources using web browsers. These geospatial queries are fundamentally different from traditional string queries issued by web browsers on a standard search engine such as Lycos, Google or Yahoo [28]. Besides the traditional string querying function, geospatial data queries usually include a spatial context, such as "find all collections of geospatial data intersecting query window $[(\phi, \lambda)_{SW}, (\phi, \lambda)_{NE}]$", where ϕ = latitude, λ = longitude and SW and NE refer to the southwest and northeast corners of the query window, and we call them *geographical queries* or *boundary queries*. For geospatial information, the geographical queries usually find many results. To increase the search accuracy and the network traffic efficiency, some query restrictions can be imposed to further limit the number of search results (e.g. by date when the data was collected, by specific agency that produced the data, or by a specific data type (e.g. satellite imagery)).

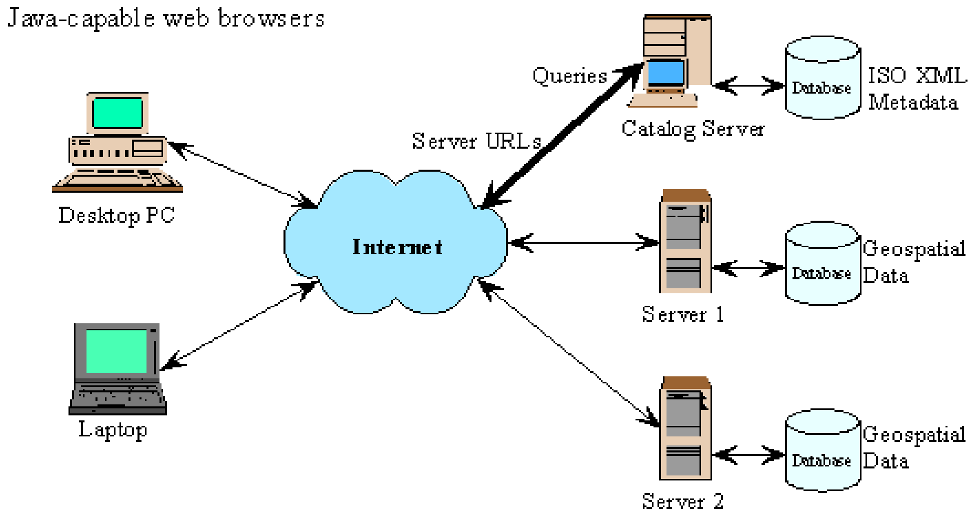


Figure 6.1: Architectural overview of WDGSDW.

Figure 6.1 illustrates the three-tier client/server overview architecture of our web-based geospatial data integration system and Figure 6.2 shows the three-tier implementations of the geospatial query and data services.

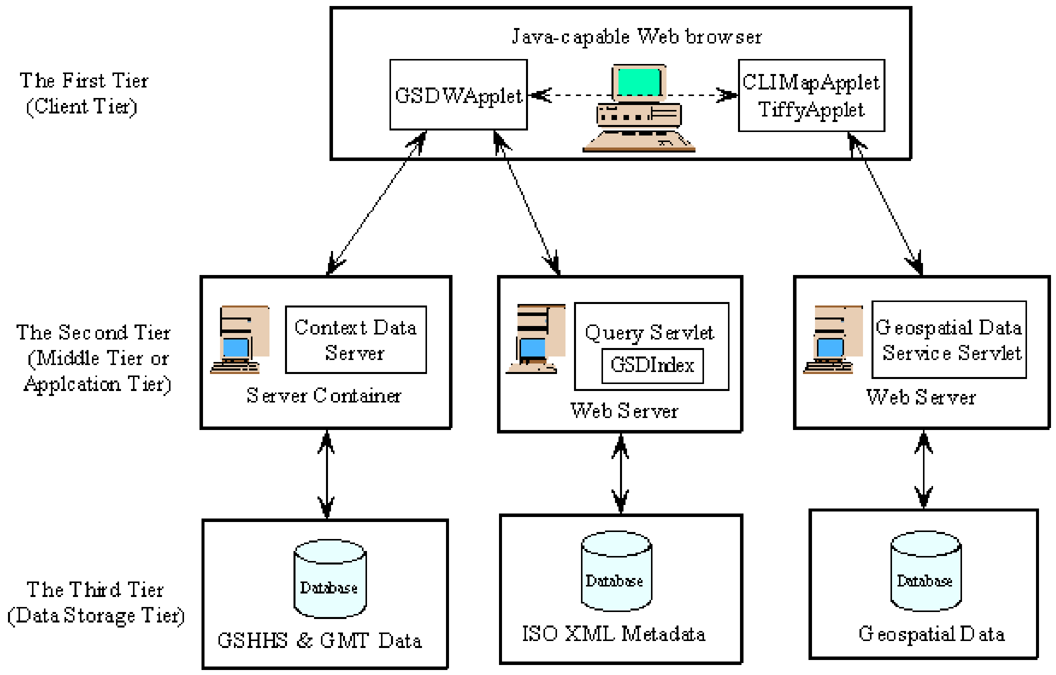


Figure 6.2: Three-tier implementation of the geospatial query and data services.

6.2 The Client-tier Components

A user's perception of the WDGSDW application is tied to the behaviour of the application's client tier. A client request is made to the server, and presents the outcome of those requests to the user. Therefore, it is important to choose a client configuration that best addresses the requirements of the application and empowers the user with a rich interface. Web clients use HTTP or HTTPS as the transport protocol and usually run inside a browser and use the services of the browser to render content provided by the web server. A stand-alone web browser is the web client of choice on the Internet. It is widely available, users are familiar with using it and there are no issues with deployment. Applets are GUI components that typically execute in a web browser and they are used to enhance the browsing interface so that the web clients can use any Java-capable web browser to run the client Java-based components.

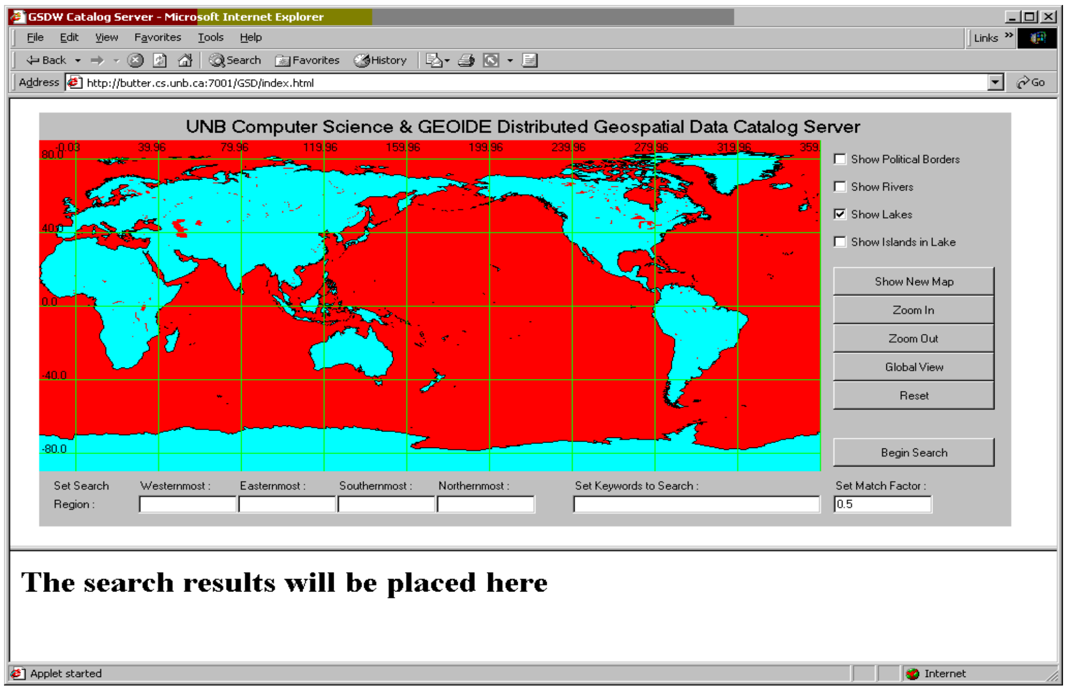
In this research, we choose the web client configuration by combining stand-alone web browsers and applets.

6.2.1 GSDWApplet for User Interface of Geospatial Queries

Xiao [49, Chapter 5] presents some requirements and implements the client applet, *GSDWApplet*, to provide the user interface for the geospatial query user interface in the client tier. We adapted Xiao's implementation of the *GSDWApplet* by imposing some further requirements on the applet to enhance the user interface.

Figure 6.3 shows the implementation of the GSDWApplet. Two subsystems are designed and implemented in this research. The first subsystem, originally implemented by Xiao [49], is an applet user interface which is designed as an information system of the world that provides a user the information about the shorelines, political borders, rivers, lakes, and islands in a lake of the whole world. This subsystem also helps a user to locate the data sets that they are interested in and helps them to define a geospatial query in the second subsystem. It enables a user to create and submit a geospatial query combining latitude, longitude boundaries, match factor and searching keyword(s). This subsystem provides the following services:

1. Shows a map on a view window.



Address: http://butter.cs.unb.ca:7001/GSD/index.html

- [ ] Show Political Borders
- [ ] Show Rivers
- [x] Show Lakes
- [ ] Show Islands in Lake

Set Match Factor : 0.5

The search results will be placed here

Figure 6.3: User interface of the geospatial query and data services.

2. Allow a user to select different thematic layers of the map information, and overlay any thematic layers.

3. Allow a user to zoom in the map shown in the view window by using mouse drag or to zoom out on the map.

4. Provide a global view to help a user find the location on the world map of the area which they select to zoom in or zoom out.

5. Allow a user to set a search region by filling-in the text fields of *Westernmost, Easternmost, Southernmost* and *Northernmost* or dragging the mouse on the map screen. The region search supports the query restriction by user's choosing an appropriate match factor $k \in (0, 1]$, which represents a degree to which the

query region area overlaps the data boundaries. The larger the match factor is, the more the search result returns. By default, the match factor is set as 0.5.

6. Allow a user to set the keyword(s) to search. The logical conjunction operation *AND* for keyword search is supported to narrow the search results. The logical *AND* operation of the keywords are represented as a addition sign "+". For example, if a user is interested in the datasets with keywords "Canada" and "Land use", he/she can type the keywords as "Canada + Land use" in the text field.

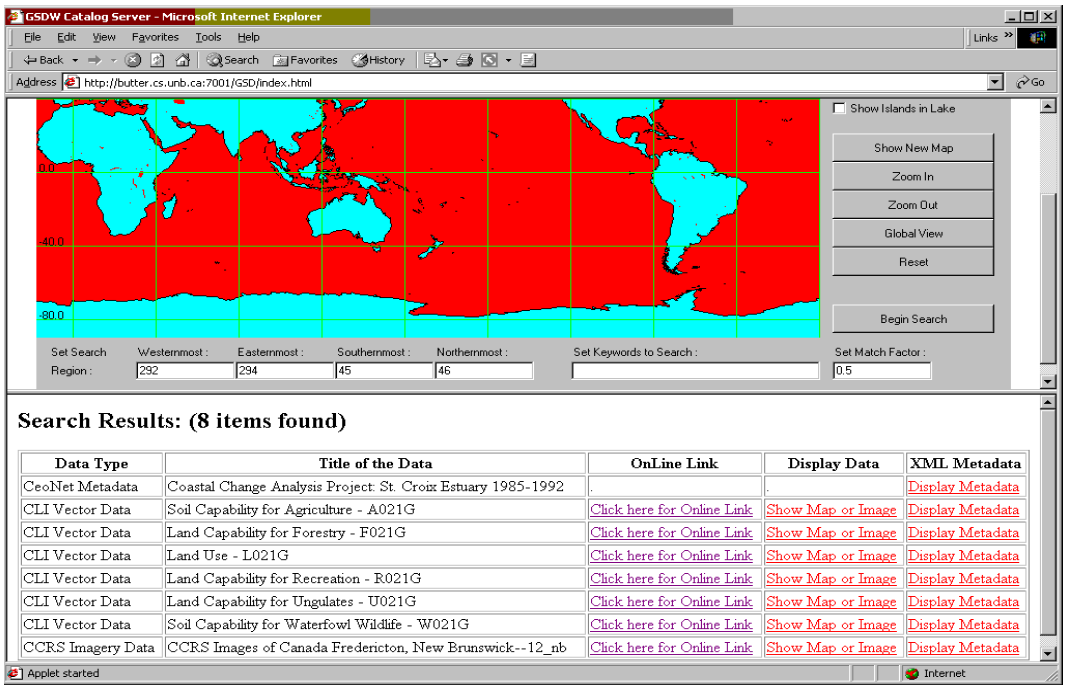
7. Provide a reset service to allow a user to reset all geospatial query criteria. All the text fields of *Set Search Region* and *Set Keyword to Search* will be set to empty and the text field of the *Set Match Factor* will be set the default value, 0.5.

The services 1 – 4 above are implemented by Xiao [49] and Teng [43]. Services 5 and 6 allow specification of a combined geospatial query.

The second subsystem of GSDWApplet displays the geospatial query results and allows a user to access the corresponding geospatial data service servers which provide the services the user is interested in. This subsystem provides the following services:

1. Provide the user a command button "Begin Search" to start the query process once a geospatial query is defined.

2. The datasets that meet the query criteria are returned and automatically displayed in a second frame within the same web page where the user interface



http://butter.cs.unb.ca:7001/GSD/index.html

Set Search Region :	Westernmost :	Easternmost :	Southernmost :	Northernmost :	Set Keywords to Search :	Set Match Factor :
	292	294	45	46		0.5

Search Results: (8 items found)

Data Type	Title of the Data	OnLine Link	Display Data	XML Metadata
CeoNet Metadata	Coastal Change Analysis Project: St. Croix Estuary 1985-1992	.	.	Display Metadata
CLI Vector Data	Soil Capability for Agriculture - A021G	Click here for Online Link	Show Map or Image	Display Metadata
CLI Vector Data	Land Capability for Forestry - F021G	Click here for Online Link	Show Map or Image	Display Metadata
CLI Vector Data	Land Use - L021G	Click here for Online Link	Show Map or Image	Display Metadata
CLI Vector Data	Land Capability for Recreation - R021G	Click here for Online Link	Show Map or Image	Display Metadata
CLI Vector Data	Land Capability for Ungulates - U021G	Click here for Online Link	Show Map or Image	Display Metadata
CLI Vector Data	Soil Capability for Waterfowl Wildlife - W021G	Click here for Online Link	Show Map or Image	Display Metadata
CCRS Imagery Data	CCRS Images of Canada Fredericton, New Brunswick--12_nb	Click here for Online Link	Show Map or Image	Display Metadata

Figure 6.4: A sample of the GSDW search result frames.

applet is embedded. "No Match" is displayed in the frame if there is no geospatial data set satisfying the query criteria.

3. Each record of the query result contains information about the type, title, on-line service linkages (if applicable), and on-line linkage to the XML metadata file of the geospatial data. A user can access the data services by clicking the corresponding link.

4. The query result is displayed in a *small-batch* fashion. That is, if the number of the records of the query result is larger than a predetermined integer, say 20, only display the current 20 records of the result. Then user can be able to choose to display the next or previous 20 records if it is applicable.

Figure 6.4 shows a sample of the GSDW search result frames.

6.2.2 CLIMapApplet for CLI Data Visualization

In this research we use the Canada Land Inventory (CLI) data as one collection of test data sets for actual vector data services of our system (refer to Chapter 4). When a user sends a query about the CLI data service and gets the query result (e.g., see Figure 6.4), there is an on-line link called *Show Map or Image*. If the user clicks a *Show Map or Image* link, the user is able to access the CLI data visualization service automatically. On the client side, the actual vector data service is implemented via an applet called *CLIMapApplet*.

Figure 6.5 and Figure 6.6 show the function of the applet which is invoked by clicking the *Show Map or Image* link for a data set with title "Land Capability for Forestry - F021G" and "Land Use - L021G", respectively.

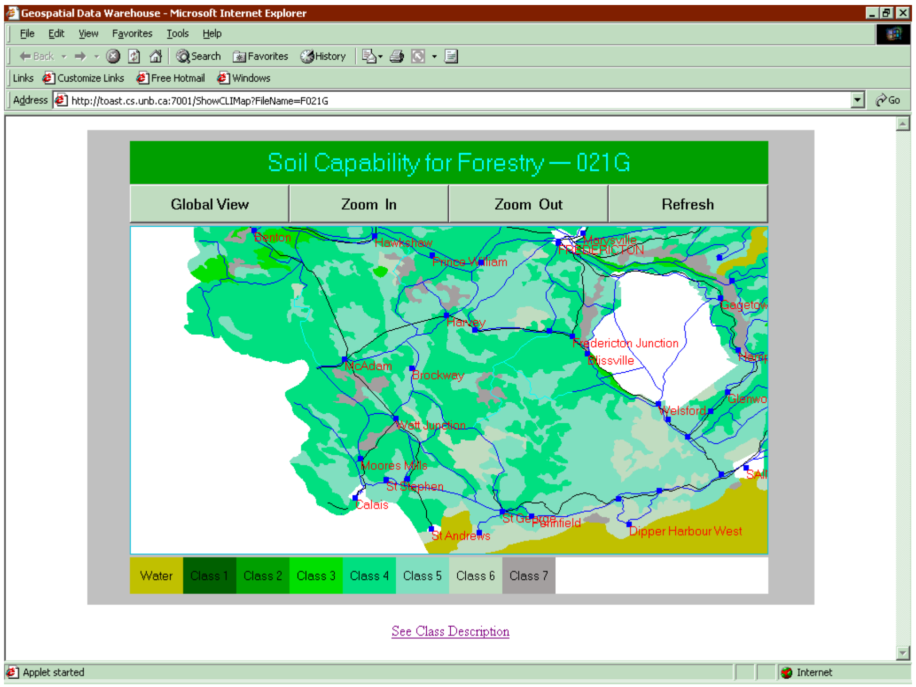


Figure 6.5: Visualization of the CLI data set "Land Capability for Forestry - F021G".

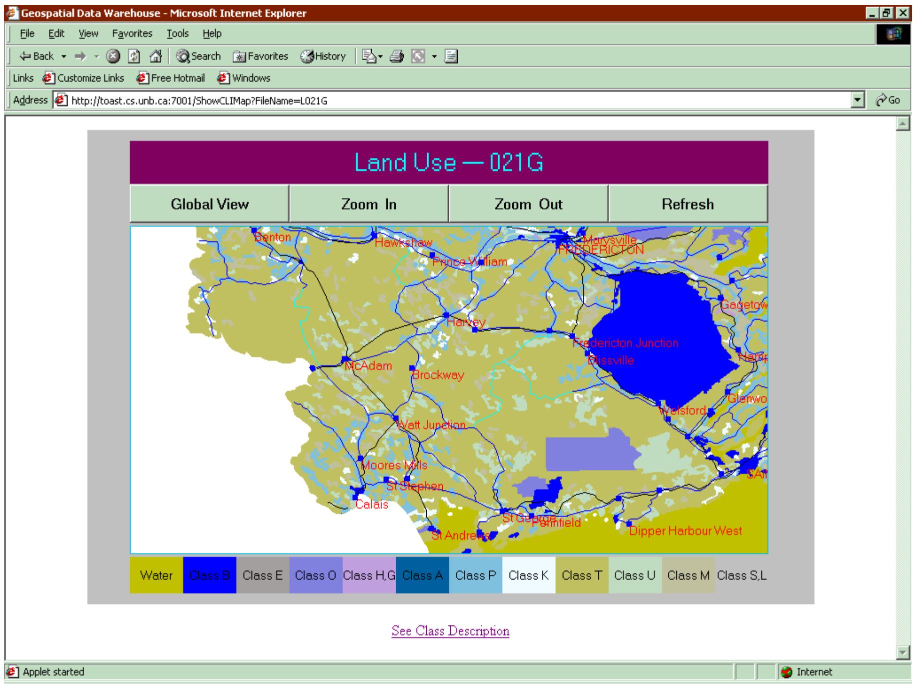


Figure 6.6: Visualization of the CLI data set "Land Use - L021G".

The CLIMapApplet receives a file name of the CLI data set as a parameter. Then it communicates with the CLI data server and gets the corresponding CLI data. The data transmission between applet and server can be implemented in two ways: raw data transmission and serialized object transmission. In raw data transmission, the server sends the raw data (ASCII Arc/Info file) through the network upon the client's request. The applet creates a CLIData object using the received data and then renders the CLIData object. In serialized object transmission, the server reads the CLI raw data from the CLI database upon the client's request, creates a CLIData object using that raw data and then sends the serialized object through the network. The applet renders the received CLIData object directly. In this research, we used the raw data transmission approach to achieve a higher data transmission rate, since the experimental result in Chapter 7 shows that the raw data transmission can achieve the best performance of 349 bytes/ms, about 23 times as fast as the serialized object transmission.

The CLIMapApplet allows a user to perform the *Zoom In, Zoom Out, Global View* and *Refresh* operations. Figure 6.7 shows the "Zoom In" result of the Figure 6.5.

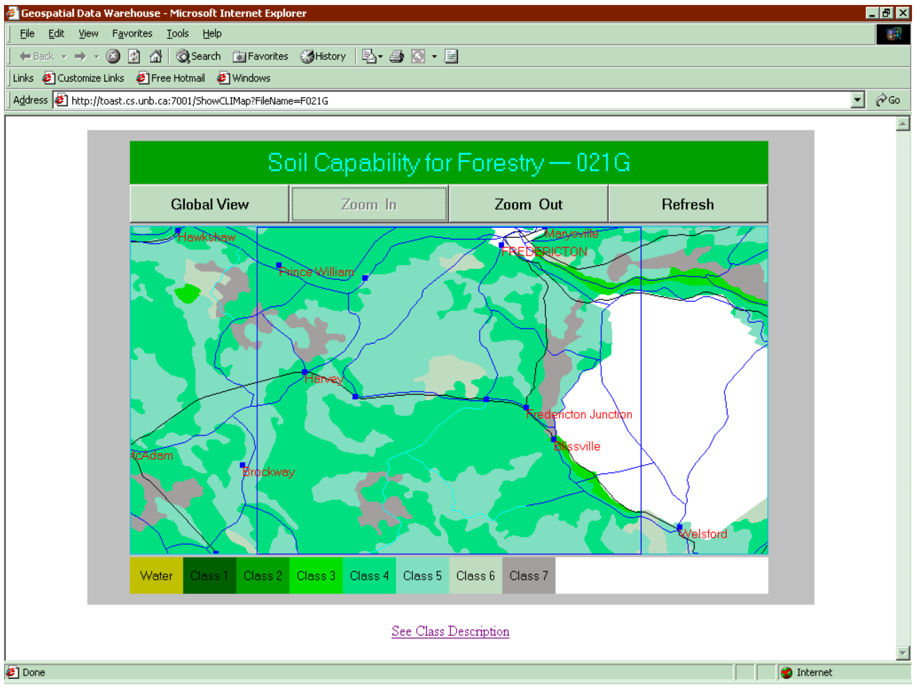
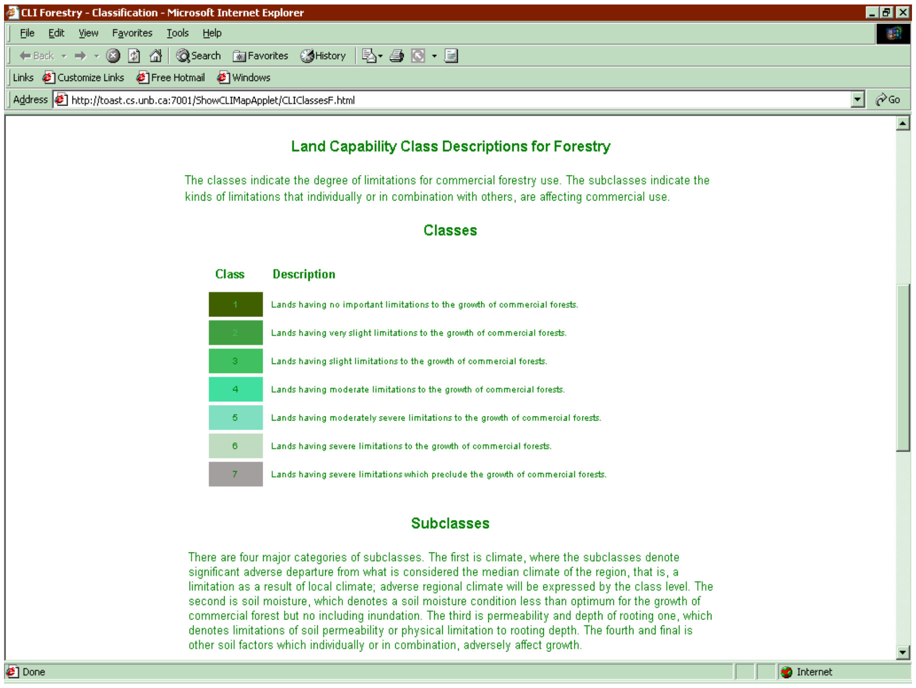

Geospatial Data Warehouse - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Links Customize Links Free Hotmail Windows
Address http://toast.cs.unb.ca:7001/ShowCLIMap?FileName=F021G
Go
Soil Capability for Forestry — 021G
Global View
Zoom In
Zoom Out
Refresh
Hawkshaw
Marysville
FREDERICTON
Prince William
Harvey
Fredericton Junction
Blissville
McAdam
Brockway
Welsford
Water
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6
Class 7
See Class Description
Done
Internet

Figure 6.7: Zoom In result of Figure 6.5.

In CLI data services, different data sets have different classifications and different colouring schemes. For example, the CLI data sets of "Land Capability for Forestry" have 7 classes using 8 colours, while the CLI data sets of "Land Use" have 13 classes using 12 colours. The applet receives a file name of a CLI data set as a parameter, then goes to the CLI data server to get the vector data (in Arc/Export (.E00) format), create an object of CLIData on-the-fly, and finally use this CLIData object to visualize the CLI data by selecting the colour scheme dynamically. The user can view the corresponding class descriptions in a new web browser window by clicking the on-line link "See Class Description" (see Figure 6.8).


CLI Forestry - Classification - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Links Customize Links Free Hotmail Windows
Address http://toast.cs.unb.ca:7001/ShowCLIMapApplet/CLIClassesF.html
Go
Done
Internet

Land Capability Class Descriptions for Forestry

The classes indicate the degree of limitations for commercial forestry use. The subclasses indicate the kinds of limitations that individually or in combination with others, are affecting commercial use.

Classes

Class	Description
1	Lands having no important limitations to the growth of commercial forests.
2	Lands having very slight limitations to the growth of commercial forests.
3	Lands having slight limitations to the growth of commercial forests.
4	Lands having moderate limitations to the growth of commercial forests.
5	Lands having moderately severe limitations to the growth of commercial forests.
6	Lands having severe limitations to the growth of commercial forests.
7	Lands having severe limitations which preclude the growth of commercial forests.

Subclasses

There are four major categories of subclasses. The first is climate, where the subclasses denote significant adverse departure from what is considered the median climate of the region, that is, a limitation as a result of local climate; adverse regional climate will be expressed by the class level. The second is soil moisture, which denotes a soil moisture condition less than optimum for the growth of commercial forest but no including inundation. The third is permeability and depth of rooting one, which denotes limitations of soil permeability or physical limitation to rooting depth. The fourth and final is other soil factors which individually or in combination, adversely affect growth.

Figure 6.8: The window of class descriptions
obtained by clicking on "See Class Description" link in Figure 6.7.

6.2.3 TiffyApplet for Viewing CCRS Imagery Data

In this research we also use Canada Centre for Remote Sensing (CCRS) imagery data as test data sets for actual imagery data services of our system. On the client side, the imagery data service is implemented via applet *TiffyApplet*.

TiffyApplet is a demo applet within the TIFFY Toolkit for Java 2.0.2 that was produced by a Germany company Art&Computer Hackbarth. The TIFFY Toolkit allows Java developers to develop their own intra-/internet applets and applications with a compact (only about 115KB code) component for fast viewing, image processing and printing capabilities for images. The TiffyApplet of the TIFFY Toolkit for Java demonstrates how to implement a modern, platform-independent viewer with printing capabilities for the following formats: TIFF, BMP, GIF, JPG and optionally PNG. It can be used on any system for which a Java-capable browser and/or a Java Virtual Machine is available (e.g. Windows 95/98/NT, MacOS, OS/2, Solaris, LINUX). We downloaded the free TIFFY Toolkit for Java 2.0.2 from [2], and embedded the TiffyApplet into our system as one of the client components to visualize the CCRS imagery data. Since it supports multiple image formats, the TiffyApplet can also be used to provide other geospatial imagery raster data services.

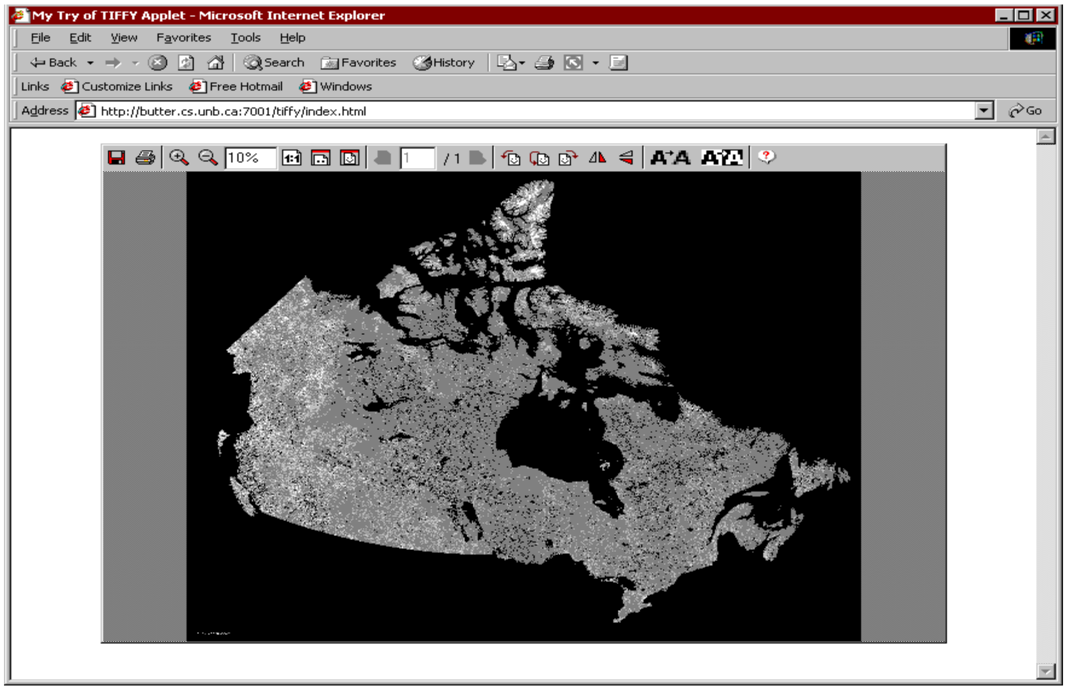

My Try of TIFFY Applet - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Links Customize Links Free Hotmail Windows
Address http://butter.cs.unb.ca:7001/tiffy/index.html
Go
10%
1
/ 1

Figure 6.9: Visualization of the Radarsat (mosaic) imagery data *Canada_Momosaic*.

When a user issues a query about the CCRS imagery data service and gets the

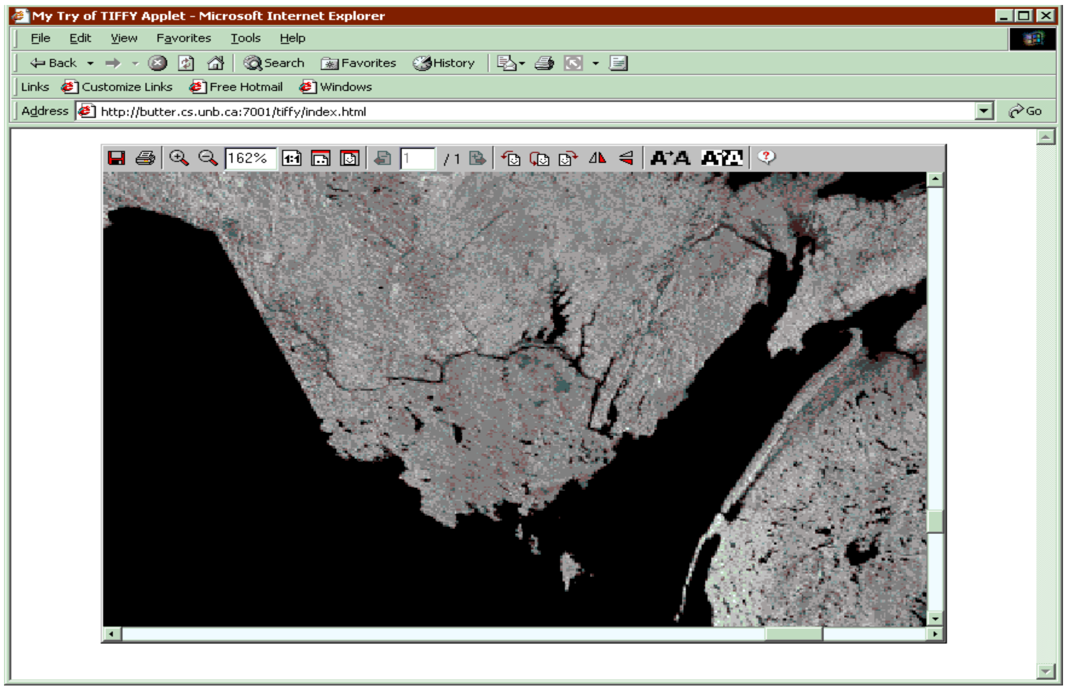

My Try of TIFFY Applet - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Links Customize Links Free Hotmail Windows
Address http://butter.cs.unb.ca:7001/tiffy/index.html
Go
162%
/ 1

Figure 6.10: Zoom In result of the Figure 6.9.

query result, there is an on-line link called *Show Map or Image*. If the user clicks this link, then the user is able to access the CCRS imagery data service by invoking the TiffyApplet and passing to it the corresponding name of the data set. The server transfers the imagery data to the applet and then the applet can view the imagery data and perform some operations. Figure 6.9 shows the function of the TiffyApplet for the Radarsat (mosaic) imagery data *Canada_Momosaic* (GeoTiff format). Figure 6.10 and Figure 6.11 show the results of two *Zoom In* operations for Figure 6.9.

6.2.4 Communications between Client-tier Components

As illustrated in Figure 6.2, there exist communications between the pair of applets: GSDWApplet–CLIMapApplet and GSDWApplet–TiffyApplet. Usually, applets can communicate with each other except for the following security restriction:

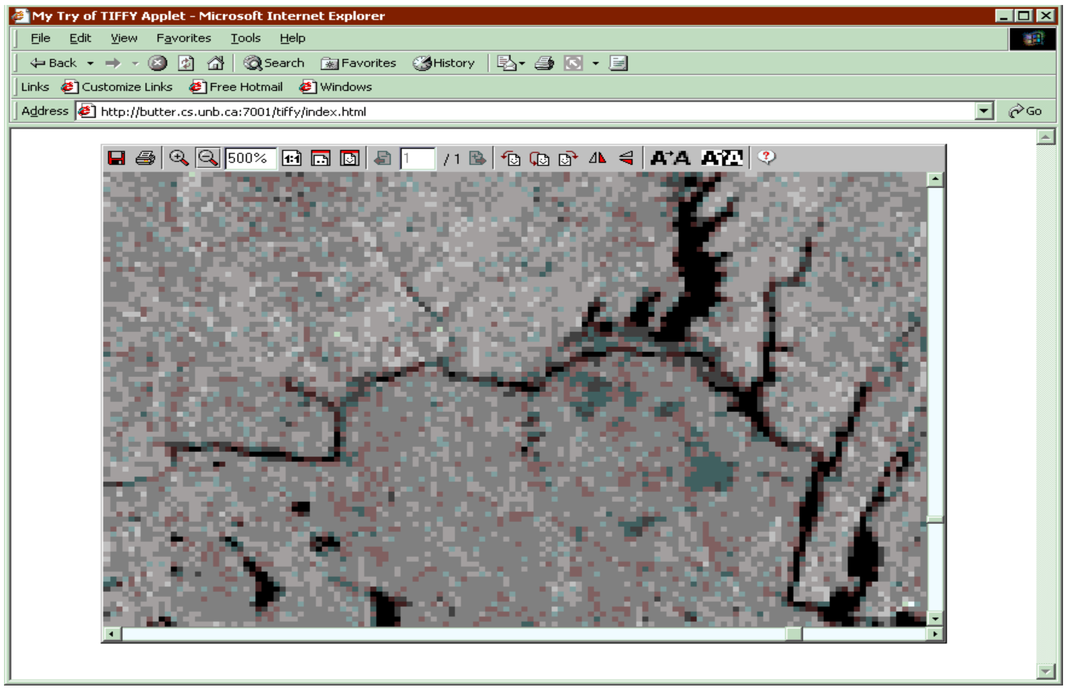


Figure 6.11: Zoom In result of the Figure 6.10.

many browsers require that the applets originate from the same server. Many browsers further require that the applets originate from the same directory on the server (the same code base). The Java API requires that the applets be running on the same page, in the same browser window.

To overcome these limitations, we used a *middle-bridge* to implement the communications between GSDWApplet–CLIMapApplet and GSDWApplet–TiffyApplet. The two middle-bridges used in this research are two servlets – ShowCLIMap and ShowCCRSMap. The GSDWApplet sends a geospatial query to the query server and gets the result back. The result contains an entry which gives the filename of the data set interested. The GSDWApplet then issues a request to the servlet Show-CLIMap or ShowCCRSMap with the filename as a parameter. Upon receiving the request, the servlet ShowCLIMap or ShowCCRSMap creates a dynamic HTML document in which the CLIMapApplet or TiffyApplet is embedded with the filename as a parameter. Figure 6.12 illustrates this process.

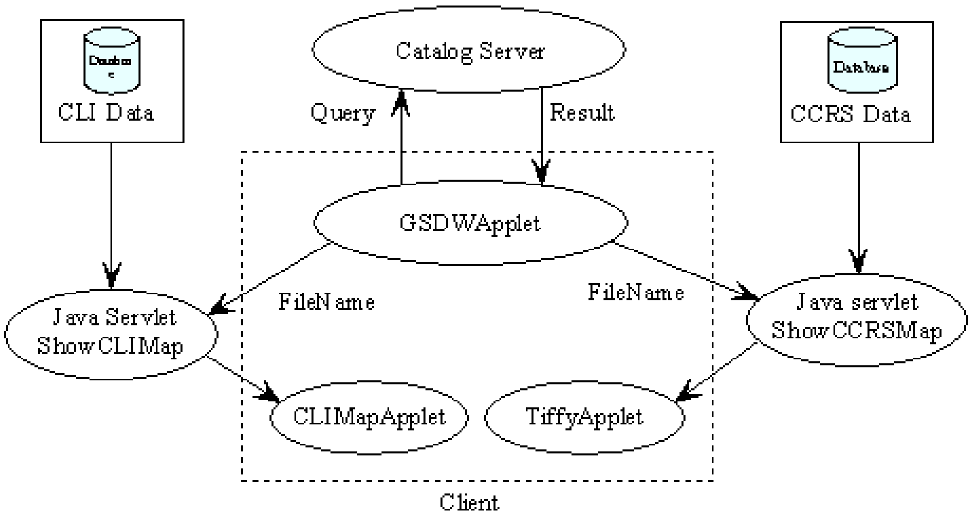


Figure 6.12: Communications between Client-tier Components.

6.3 Contextual Data Server

The middle tier serves as the application server or web server. The components of the middle tier are the kernel of the system, since they provide all the business logic of the system. In this section and the following sections, we describe the design and implementation of the middle-tier components: Contextual Data Server, Query Server, CLI Data Server and CCRS Data Server.

As illustrated in Figure 6.2, the Contextual Data Server communicates with the GSDWApplet and handles the proper display of the map which is made of the GSHHS and GMT contextual data. The Contextual Data Server works with GSDWApplet together to provide a user interface for creating or defining geospatial queries.

The Contextual Data Server can be implemented using different distributed computing architectures and techniques. For the purpose of comparison, three types of implementations are considered in this research. They are described below.

6.3.1 Servlet-based Contextual Data Server

Xiao [49] implemented the Contextual Data Server based on the Java servlet techniques. In this research, we used Xiao's Thin-client via Fat-server architectural implementation as that of the Servlet-based Contextual Data Server. This architectural implementation is illustrated in Figure 6.13. Xiao [49] has the full details of the implementation.

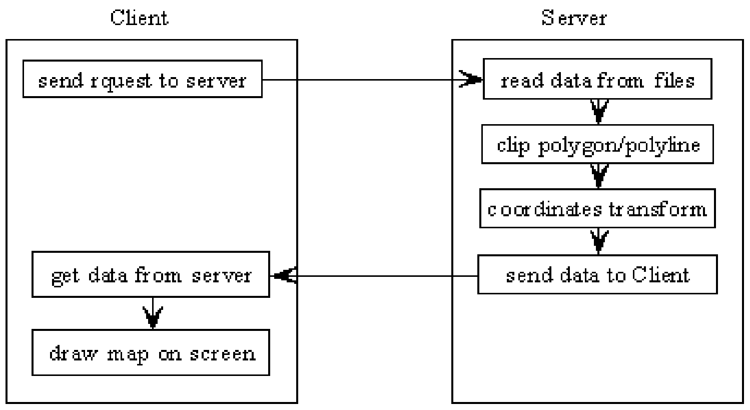


Figure 6.13: Architecture of the Contextual Data Server (from [49]).

6.3.2 CORBA-based Contextual Data Server

Xiao [49] implemented the Contextual Data Server based on the CORBA techniques using VisiBroker for Java 3.0 and investigated whether the CORBA performance will be improved or not if the server side program is written in C++. In this research, we adapted Xiao's implementation using VisiBroker for Java 4.1 and re-implemented it using VisiBroker for C++ 4.1.

VisiBroker for Java and C++ [20] provides a complete CORBA 2.3 ORB runtime and supporting development environment for building, deploying, and managing distributed Java applications that are open, flexible, and inter-operable. Objects built with VisiBroker for Java are easily accessed by Web-based applications that communicate using OMG's Internet Inter-ORB Protocol (IIOP) standard for communication between distributed objects through the Internet or through local intranets.

There are a number of steps for developing a CORBA-based application using VisiBroker for Java or C++. Figure 6.14 illustrates the CORBA application development paradigms for Java or C++.

Defining Object Interfaces

The first step in creating an application with VisiBroker is to specify all of the objects and their interfaces using the OMG's Interface Definition Language (IDL). Then the IDL can be mapped to Java and C++ programming languages in this research. Figure 6.15 shows the IDL interface for the distributed object called *DataProvider* [49].

Generating Client Stubs and Server Servants

Once we finish defining the IDL interface, we are ready to compile it. VisiBroker for Java (C++) comes with an IDL compiler, *idl2java* (*idl2cpp*), which is used to map IDL definitions into Java (C++) declarations and statements. From the DataProvider.idl file, the *idl2java* compiler generates 7 .java files and the *idl2cpp* compiler

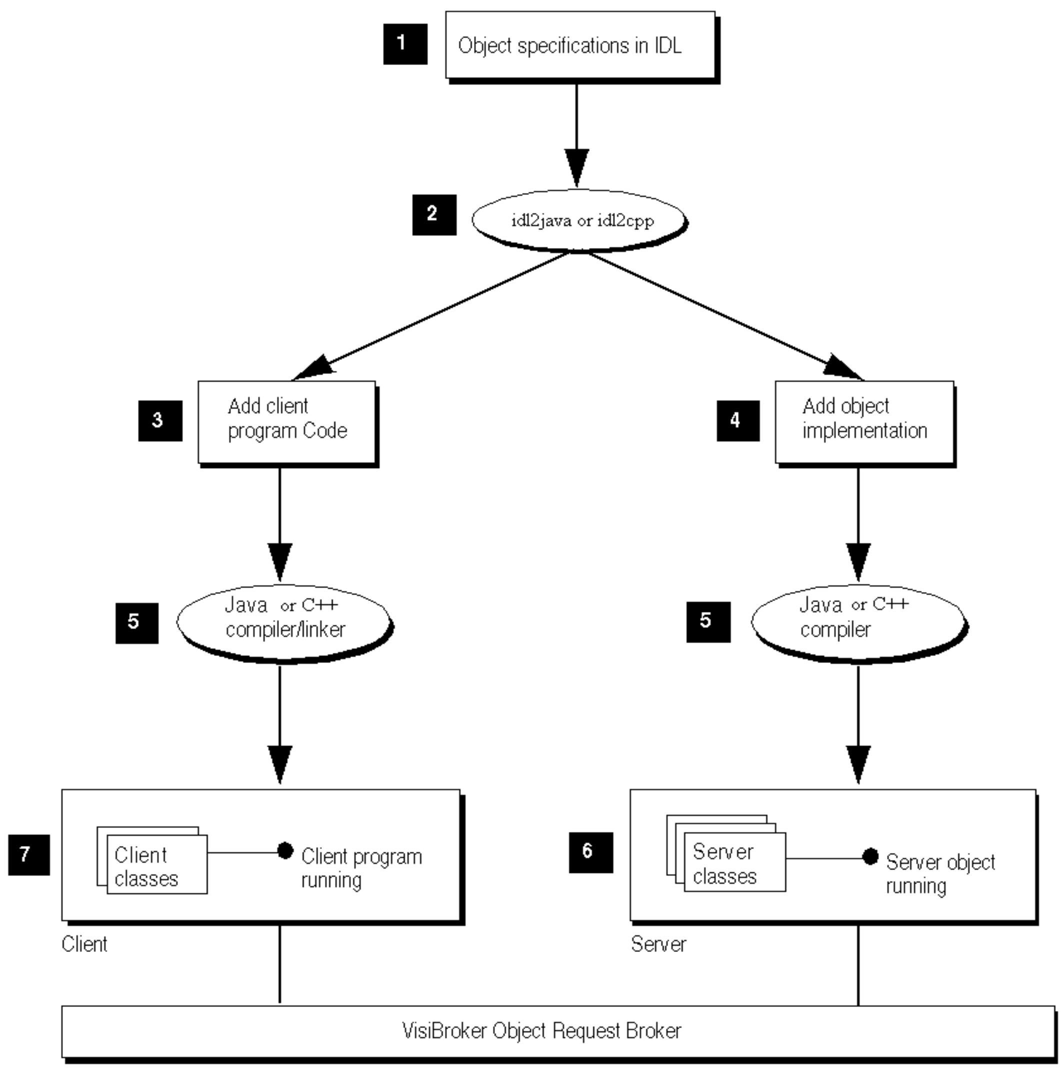


Figure 6.14: Development process with VisiBroker (from [20]).

```
//DataProvider.idl

module GSDWCORBA {
    interface DataProvider {
         string GenerateData(  in string Border,  in string River, in string Lake, in string Island,
                               in string West,  in string East,  in string South,  in string North,
                               in string EnlargeFactor, in string OrientX, in string OrientY );
         string GetData();
     };
};
```

Figure 6.15: The IDL interface for the distributed object DataProvider.

Table 6.1: Generated .java Files by idl2java Compiler.

Filename	**Description**
DataProvider.java	The DataProvider interface declaration.
DataProviderHelper.java	Declares a class defining helpful utility methods.
_DataProviderStub.java	Stub code for the object on client side.
DataProviderHolder.java	Declares a class providing a holder for the object.
DataProviderOperation.java	Declares the method signatures.
DataProviderPOA.java	POA servant code (implementation-base code).
DataProviderPOATie.java	Class implementing the object using tie mechanism.

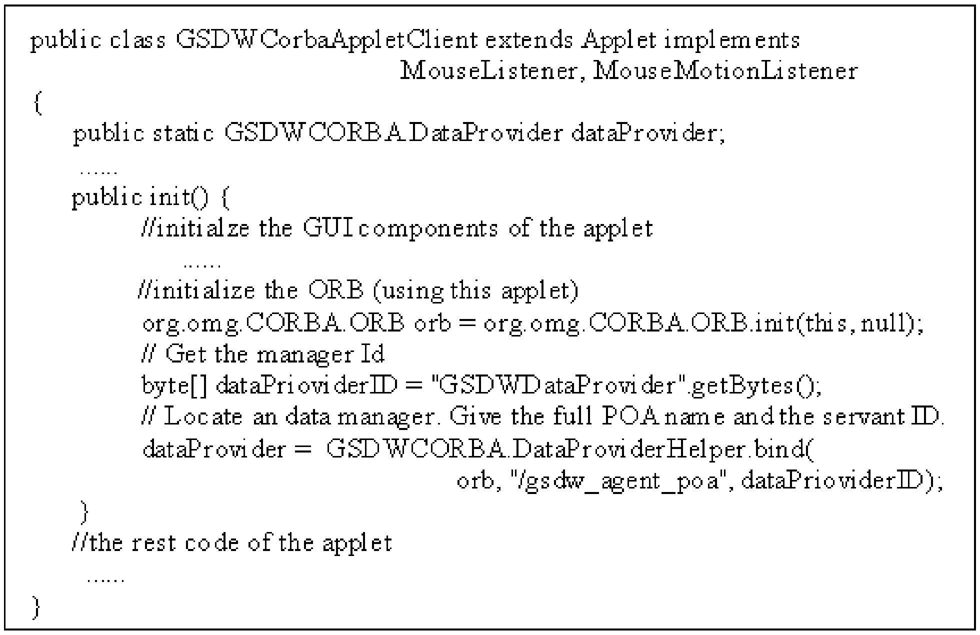
Table 6.2: Generated C++ Files by idl2cpp Compiler.

Filename	**Description**
DataProvider_c.hh	The definition for the DataProvider class.
DataProvider_c.cpp	The internal stub routines used by the client.
DataProvider_s.hh	The definition for the DataProviderPOA servant classes.
DataProvider_s.cpp	The internal routines used by the server.

generates 4 C++ files. These generated files are listed in Table 6.2 and Table 6.1, respectively.

Implementing the Client

Many of the classes used in implementing the CORBA version applet client (GSDWCorbaAppletClient.java) are contained in the GSDWCORBA package generated by the idl2java compiler. Figure 6.16 shows part of the java code of the GSDWCorbaAppletClient class.



```
public class GSDWCorbaAppletClient extends Applet implements
                                    MouseListener, MouseMotionListener
{
    public static GSDWCORBA.DataProvider dataProvider;
    ......
    public init() {
           //initialze the GUI components of the applet
               ......
           //initialize the ORB (using this applet)
           org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(this, null);
           // Get the manager Id
           byte[] dataPrioviderID = "GSDWDataProvider".getBytes();
           // Locate an data manager. Give the full POA name and the servant ID.
           dataProvider =  GSDWCORBA.DataProviderHelper.bind(
                                       orb, "/gsdw_agent_poa", dataPrioviderID);
     }
    //the rest code of the applet
     ......
}
```

Figure 6.16: Part of the Java code of GSDWCorbaAppletClient class.

Implementing the Distributed Object

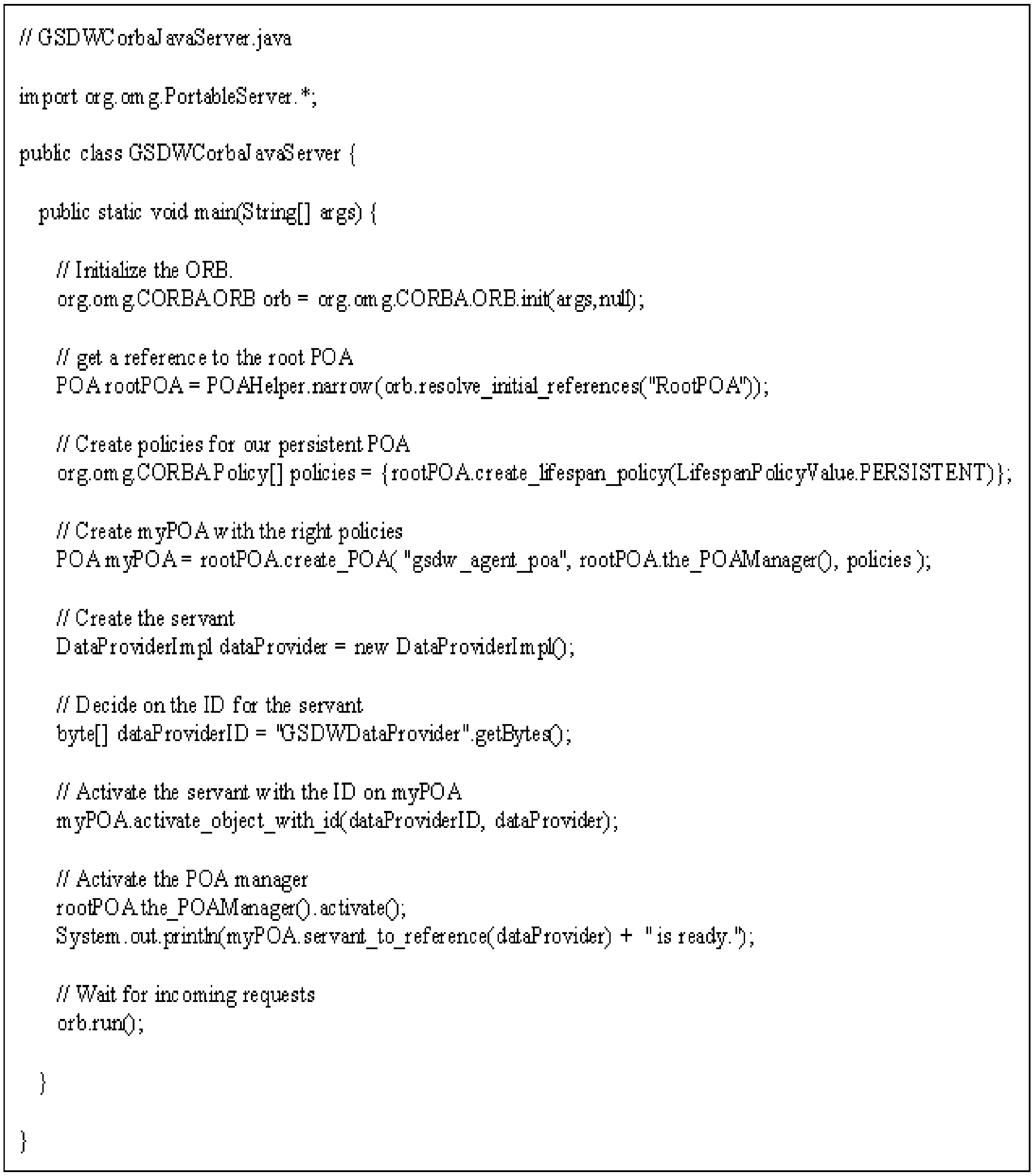
The implementation class DataProviderImpl of the distributed object DataProvider should inherit from the server skeleton class, i.e. the POA (Portable Object Adapter) servant class DataProviderPOA.java for the Java version or include the header file DataProvider_s.hh for the C++ version. We also need to write code to implement the methods defined in the IDL file DataProvider.idl.

Implementing the Server

Just as with the client, many of the classes used in implementing the GSDW server are generated by the idl2java or idl2cpp compiler. Figure 6.17 and Figure 6.18 show the java code and C++ code for the GSDW Server class, respectively.

Starting the Server

Once we finish implementing the client and server programs, we can compile them



```
// GSDWCorbaJavaServer.java

import org.omg.PortableServer.*;

public class GSDWCorbaJavaServer {

  public static void main(String[] args) {

    // Initialize the ORB.
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);

    // get a reference to the root POA
    POA rootPOA = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));

    // Create policies for our persistent POA
    org.omg.CORBA.Policy[] policies = {rootPOA.create_lifespan_policy(LifespanPolicyValue.PERSISTENT)};

    // Create myPOA with the right policies
    POA myPOA = rootPOA.create_POA( "gsdw_agent_poa", rootPOA.the_POAManager(), policies );

    // Create the servant
    DataProviderImpl dataProvider = new DataProviderImpl();

    // Decide on the ID for the servant
    byte[] dataProviderID = "GSDWDataProvider".getBytes();

    // Activate the servant with the ID on myPOA
    myPOA.activate_object_with_id(dataProviderID, dataProvider);

    // Activate the POA manager
    rootPOA.the_POAManager().activate();
    System.out.println(myPOA.servant_to_reference(dataProvider) + " is ready.");

    // Wait for incoming requests
    orb.run();

  }

}
```

Figure 6.17: Java code for the GSDW server class.

```
//GSDWCorbaCPPServer.cpp

#include "DataProviderImpl.h"

// USE_STD_NS is a define setup by VisiBroker to use the std namespace
USE_STD_NS

int main(int argc, char* const* argv)
{
  // Initialize the ORB.
   CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

   // get a reference to the root POA
   CORBA::Object_var obj = orb->resolve_initial_references("RootPOA");
   PortableServer::POA_var rootPOA = PortableServer::POA::_narrow(obj);

   CORBA::PolicyList policies;  policies.length(1);
   policies[(CORBA::ULong)0] = rootPOA->create_lifespan_policy(PortableServer::PERSISTENT);

  // get the POA Manager and Create myPOA with the right policies
   PortableServer::POAManager_var poa_manager = rootPOA->the_POAManager();
   PortableServer::POA_var myPOA = rootPOA->create_POA("gsdw_agent_poa", poa_manager, policies);

   // Create the servant and  decide on the ID for the servant
   DataProviderImpl dataProviderServant;
   PortableServer::ObjectId_var dataProviderId =  PortableServer::string_to_ObjectId("GSDWDataProvider");

   // Activate the servant with the ID on myPOA  and  activate the POA Manager
   myPOA->activate_object_with_id(dataProviderId, &dataProviderServant);
   poa_manager->activate();

   CORBA::Object_var reference = myPOA->servant_to_reference(&dataProviderServant);
   cout << reference << endl;  cout << "The GSDW Data Provider ( C++ ) is ready...!" << endl;

   // Wait for incoming requests
   orb->run();

   return 0;
}
```

Figure 6.18: C++ code for the GSDW server class.

to create the class files or executable.

Before attempting to run VisiBroker client programs or server implementations, you must first start the Smart Agent, which provides a fault-tolerant object location service and runtime licensing of VisiBroker applications, on at least one host in your local network using the command

`prompt> osagent` (For MS Windows) or `prompt> osagent` & (for Unix)

Once the Smart Agent is running, we can start the server program as a background process. To start the Java version server, we use the *vbj* command, which invokes the JVM and offers other special services such as setting paths:

`prompt> start vbj CORBAJavaServer` (For MS Windows) or

`prompt> vbj CORBAJavaServer`& (for Unix).

To start the C++ version server, we use the command

`prompt> start CORBACPPServer` (For MS Windows) or

`prompt> CORBACPPServer` & (for Unix).

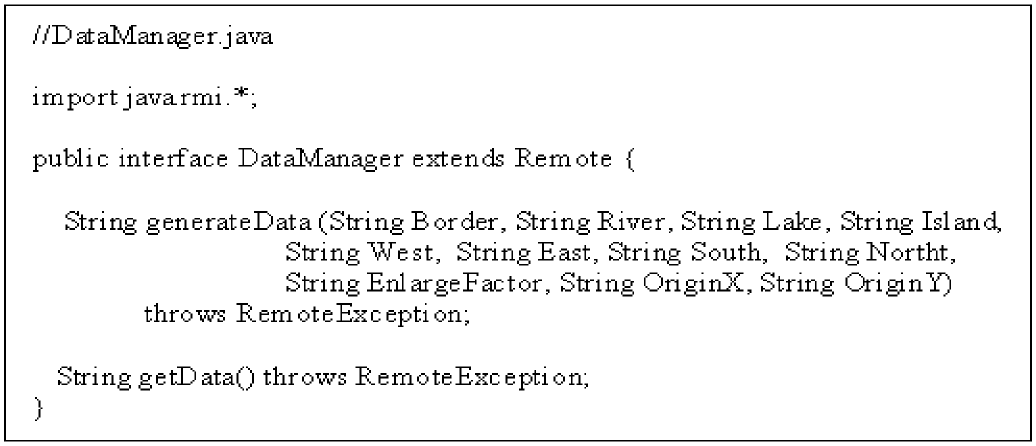
6.3.3 RMI-based Contextual Data Server

Like any other application, a distributed application built using Java RMI is made up of interfaces and classes. The interfaces define methods, and the classes implement the methods defined in the interfaces. Building a RMI-based application involves the following steps.

6.3.3.1 Defining the remote interfaces

A remote interface specifies the methods that can be invoked remotely by a client.

It provides the connection between the client and the server. Objects that have methods that can be called across virtual machines are remote objects. The remote interface for the GSDW server we are using is shown in Figure 6.19. It must extend the interface *java.rmi.Remote* and be declared *public*. The interface DataManager contains two methods: generateData and getData; both return a string and must declare *java.rmi.RemoteException* in its throws clause.



```
//DataManager.java

import javarmi.*;

public interface DataManager extends Remote {

   String generateData (String Border, String River, String Lake, String Island,
                        String West,  String East, String South,  String Northt,
                        String EnlargeFactor, String OriginX, String OriginY)
          throws RemoteException;

  String getData() throws RemoteException;
}
```

Figure 6.19: The Remote interface DataManager

6.3.3.2 Implementing the remote interfaces

In general the implementation class of a remote interface should at least

(a) Declare the remote interfaces being implemented:

```
public class DataManagerImpl  extends UnicastRemoteObject
                              implements DataManager
```

This declaration states that the DataManagerImpl class implements the DataManager remote interface (and therefore defines a remote object) and extends the class *UnicastRemoteObject* in the package *java.rmi.server*.

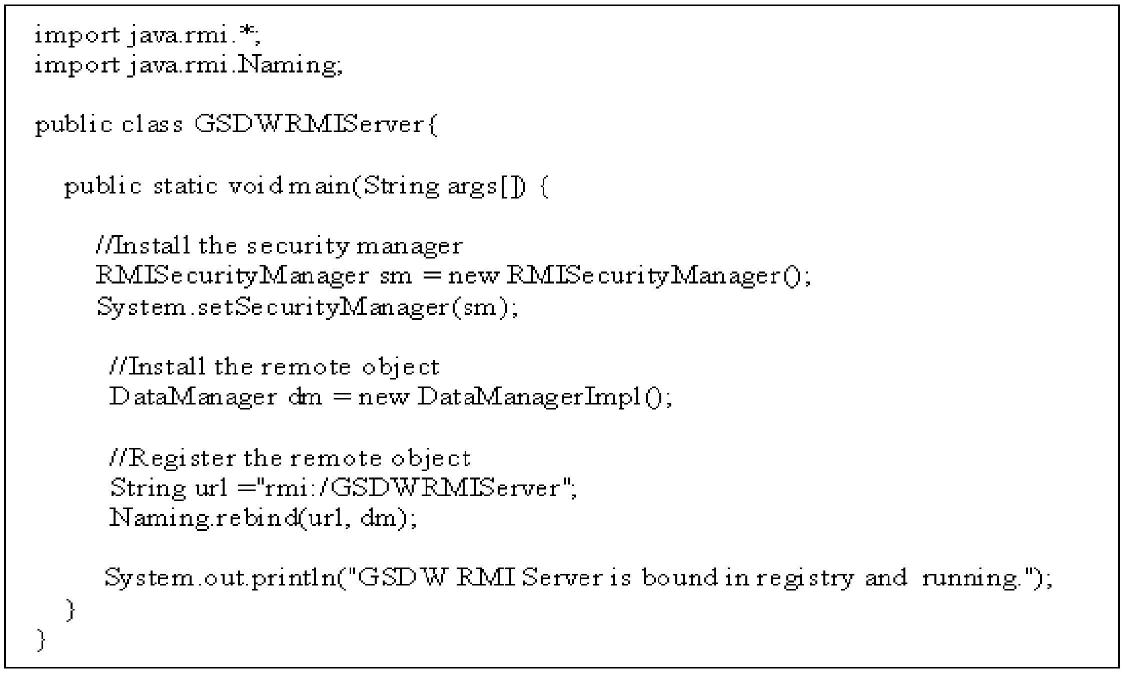
(b) Define the constructor for the remote object:

`public The DataManagerImpl() throws RemoteException { super(); }`

(c) Provide an implementation for each remote method in the remote interfaces. There are two remote methods called *generateData* and *getData* specified in the remote interfaces DataManager. These two methods are implemented in the class DataManagerImpl.

6.3.3.3 Implementing the server

The server needs to create and to install the remote objects. This procedure is included in a separate class *GSDWRMIServer*, which is shown in Figure 6.20.



```
import java.rmi.*;
import java.rmi.Naming;

public class GSDWRMIServer{

   public static void main(String args[]) {

      //Install the security manager
      RMISecurityManager sm = new RMISecurityManager();
      System.setSecurityManager(sm);

       //Install the remote object
       DataManager dm = new DataManagerImpl();

       //Register the remote object
       String url ="rmi:/GSDWRMIServer";
       Naming.rebind(url, dm);

      System.out.println("GSDW RMI Server is bound in registry and running.");
   }
}
```

Figure 6.20: The java code for the GSDWRMIServer class.

(a) Create and install a security manager. The security manager determines whether downloaded code has access to the local file system or can perform any other privileged operations.

(b) Create one instances of a remote object:

```
DataManager dm = new DataManagerImpl();
```

Note that the type of the variable dm is DataManager, not DataManagerImpl. This declaration emphasizes that the interface available to clients is the DataManager interface and its methods, not the DataManagerImpl class and its methods.

(c) Register the remote objects with the RMI remote object registry. The java.rmi. Naming interface is used as a front-end API for binding, or registering, and looking up remote objects in the registry. The registry can be shared by all servers running on a host, or an individual server process may create and use its own registry, if desired.

The implementations can be complied using the javac compiler to create the class files DataManagerImpl.class and GSDWRMIServer.class.

6.3.3.4 Implementing the client

The client GSDWRMIApplet program performs these steps:

(a) Install an RMI security manager.

(b) Look up a DataManager remote object.

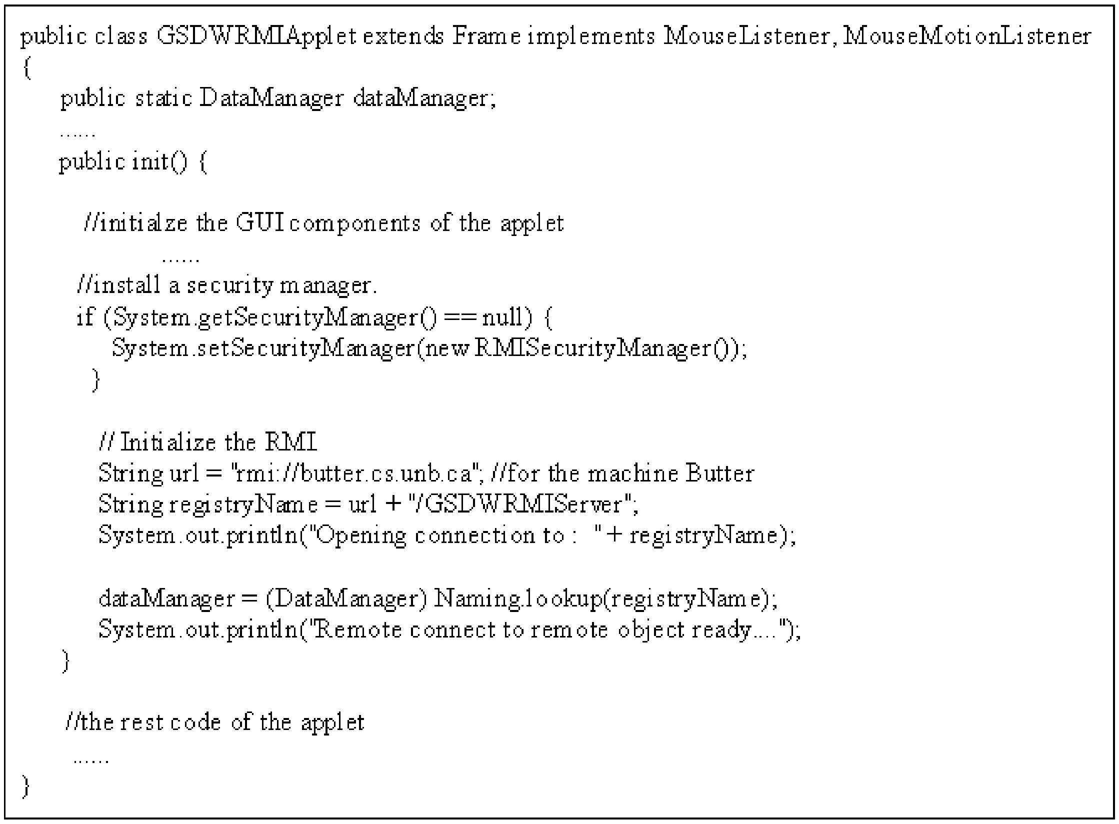
(c) Obtains data using the object.

(d) Draw the map on the screen.

Figure 6.21 shows part of the java code of the GSDWRMIApplet class.

6.3.3.5 Generating stubs and skeletons:

Once all the code is written, we are ready to generate the stubs and skeletons. A skeleton for a remote object is a server-side entity containing a method that dispatches



```
public class GSDWRMIApplet extends Frame implements MouseListener, MouseMotionListener
{
    public static DataManager dataManager;
    ......
    public init() {

        //initialze the GUI components of the applet
                ......
        //install a security manager.
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }

        // Initialize the RMI
        String url = "rmi://butter.cs.unb.ca"; //for the machine Butter
        String registryName = url + "/GSDWRMIServer";
        System.out.println("Opening connection to :  " + registryName);

        dataManager = (DataManager) Naming.lookup(registryName);
        System.out.println("Remote connect to remote object ready....");
    }

    //the rest code of the applet
     ......
}
```

Figure 6.21: Part of the Java code of GSDWCorbaAppletClient class

calls to the remote object implementation. A stub is a proxy for a remote object that is responsible for forwarding method invocations on remote objects to the server where the actual remote object implementation resides. The stubs and skeletons are used to connect the client and server together.

RMI stubs and skeletons are generated easily using the rmic compiler which takes the class name DataManagerImpl as an argument:

```
prompt> rmic DataManagerImpl
```

This command will generate the two files DataManagerImpl_Skel.class and DataManagerImpl_Stub.class. The DataManagerImpl_Skel.class file is a server skeleton, and the DataManagerImpl_Stub.class file is a client proxy or stub.

6.3.3.6 Starting the RMI registry and running the server

The RMI registry is a naming service that acts as a manager for RMI remote object reference and allows a client to obtain a reference to a remote object. Therefore, before the server and the client are run, the RMI registry must be started. The RMI registry can be started in a Window environment as follows:

```
prompt> start rmiregistry
```

This will, by default, run the RMI registry and start it listening on the port 1099. Once the RMI registry is running and everything else is in place, we can fire up our server as follows:

```
prompt> start java GSDWRMIServer
```

6.4 Query Server

In the Internet world, the need to deliver dynamically generated content in a maintainable fashion is extremely important. Common Gateway Interface (CGI), Servlets and JavaServer Page (JSP) are the three most commonly used techniques. In this research, we choose Servlets to implement the middle-tier components. The Query Server, CLI Data Server and CCRS Data Server are all servlets. Servlets can handle complex logic processing, give access to enterprise data and are a portable (platform and Web server independent) means of delivering dynamic content.

6.4.1 Implementation Scheme

Figure 6.22 and Figure 6.23 show an activity diagram and a sequence diagram to illustrate how the query server works.

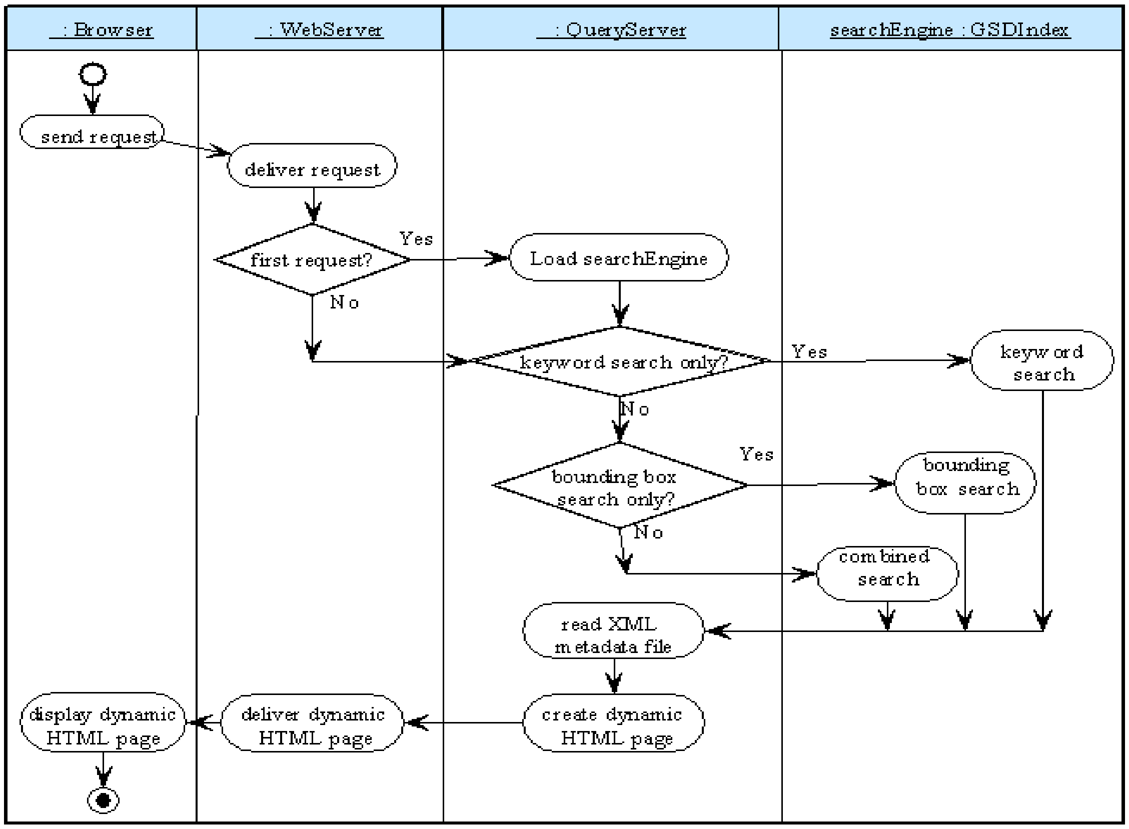


Figure 6.22: The activity diagram of the query server.

A browser sends a search request to the web server. When the web server receives the first search request, the query servlet is activated through an *init* call. During the servlet's initialization, the object *SearchEngine* of GSDIndex is loaded from a persistent file. The SearchEngine has three overloaded search methods: `search(keyword: String)`, `search(box:BoundingBox, k:double)` and `search(keyword:String, box:BoundingBox, k:double)`. They can be used to perform the keyword search

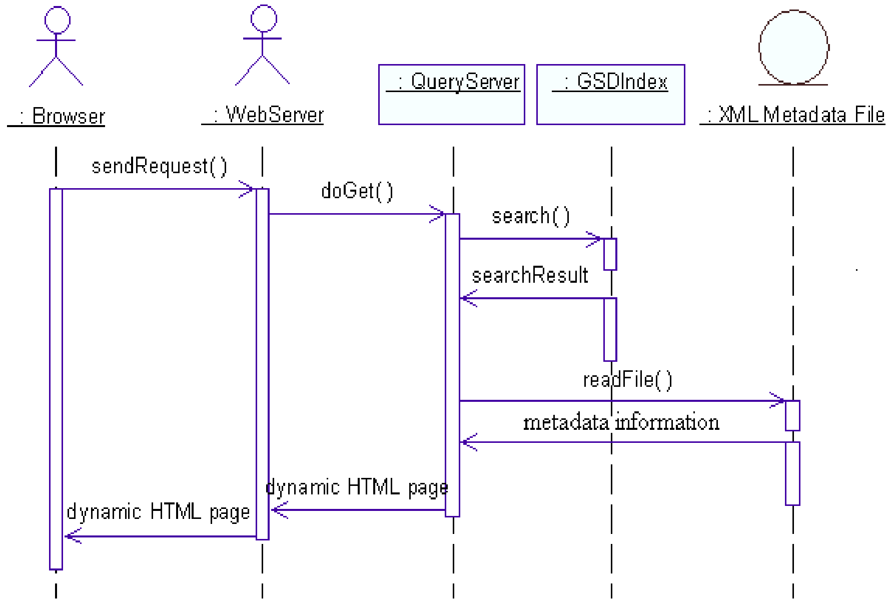


Figure 6.23: The sequence diagram of the query server.

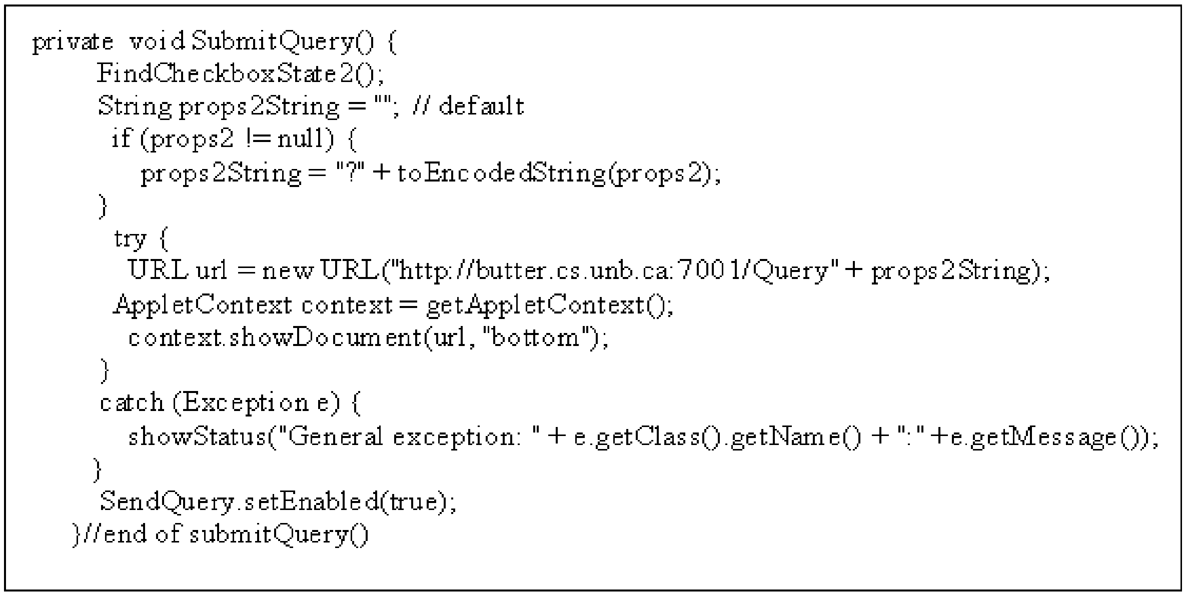
only, bounding box search only and combined search, respectively. After the web server delivers the search request to the servlet, the servlets begins to process the request. The public synchronized servlet method *doGet* is overridden to handle the request. First, the servlet determines the search type of the request and asks the SearchEngine to perform a corresponding search operation. The search result is a collection of XML metadata file names. Next, for each entity of the search result (a name of XML metadata file), the servlet reads the XML metadata file and extracts some information about the geospatial data set, such as the data resource, distribution and linkages of on-line services. Finally, the servlet creates a dynamic HTML page using the extracted information from the XML metadata file and hands it to the web server. The web server then delivers the dynamic HTML page to the browser through the network.

6.4.2 Applet–Servlet Communication

In this implementation, we have the GSDWApplet establish an HTTP connection to the query servlet on the web server machine. The HTTP connection allows the applet to take advantage of the java.net.URL and java.net.URLConnection classes to manage the communication channel. The GSDWApplet provides query information using an URL-encoded query string, which is formed as a list of *key/value* pairs, e.g. *name=John*. In the query string, *key/value* pairs are separated by "&" characters, spaces are converted to "+" characters, and special characters are converted to their hexadecimal equivalents. For example, if a user asks for the image data in a region with bounding box [West, East; South, North] = $[229, 227; 45, 46]$, the query string looks like "Keyword=image+data&West=229&East=227&South=45&North=46". In this research, the URL-encoded query string is created by calling method *toEncoded-String()* defined in the GSDWApplet class. This method takes an object of the Java class *java.util.Properties* as an argument and converts the properties list to a URL-encoded query string. To get the response from the query servlet, the GSDWApplet calls the *getAppletContext()* and *showDocument()* methods. The java code for the Applet–Servlet communication is shown in Figure 6.24.

6.5 Geospatial Data Servers

In this section we describe the implementation schemes for the two actual geospatial data servers – CLI data server and CCRS data server.



```
private void SubmitQuery() {
      FindCheckboxState2();
      String props2String = ""; // default
        if (props2 != null) {
            props2String = "?" + toEncodedString(props2);
      }
        try {
          URL url = new URL("http://butter.cs.unb.ca:7001/Query" + props2String);
        AppletContext context = getAppletContext();
          context.showDocument(url, "bottom");
      }
      catch (Exception e) {
          showStatus("General exception: " + e.getClass().getName() + ":" +e.getMessage());
      }
      SendQuery.setEnabled(true);
   }//end of submitQuery()
```

Figure 6.24: The java code for the Applet–Servlet communication.

6.5.1 ShowCLIMap Servlet and CLI Data Server

As described in section 6.2, the communications between GSDWApplet and CLIMapApplet is implemented using a servlet. ShowCLIMap and the CLIMapApplet communicates with the CLIDataServer to get the CLI data. The ShowCLIMap servlet is a middle-bridge and acts as a distributed CLI data manager. Figure 6.25 illustrates the implementation scheme for ShowCLIMap Servlet and CLIDataServer.

The GSDWApplet sends a request with a parameter "`fileName`", the file name of the CLI data set, to the ShowCLIMap servlet. The ShowCLIMap servlet determines which distributed CLI data server provides the service for that CLI data set and creates a dynamic HTML document which contains the CLIMapApplet. The parameter "`fileName`" is passed to the embedded applet CLIMapApplet. Then the applet CLIMapApplet sends a request to the servlet CLIDataServer with the query

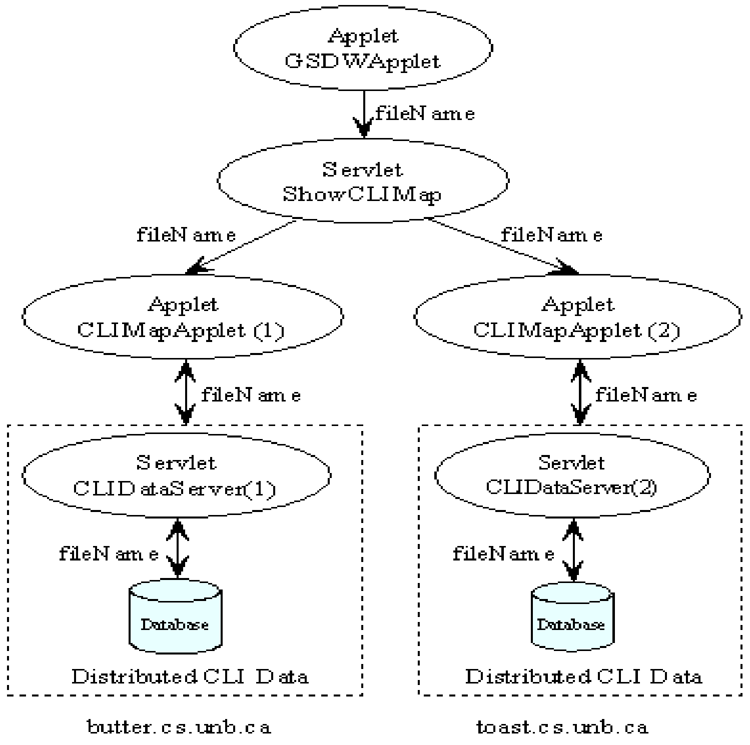


Figure 6.25: The implementation scheme of the CLI data server.

string of filename. Upon receiving the request from the embedded applet CLIMapApplet, the servlet CLIDataServer accesses the distributed CLI database to retrieve the CLI data.

There are two possible architectures to implement the applet CLIMapApplet and the servlet CLIDataServer: Fat-Client via Thin-Server and Thin-Client via Fat-Server. In the Thin-Client via Fat-Server architecture, the servlet CLIDataServer gets the raw CLI data from the CLI database and pre-processes it to create a Java serialized object of CLIData (See Chapter 4 for the CLI data structure) and then transfers the Java object to the CLIMapApplet over the network. In the Fat-Client via Thin-Server architecture, the servlet CLIDataServer simply reads the raw CLI data from the CLI database and sends it to the client. It is the CLIMapApplet's responsibility to create the CLIData object which is used to display the CLI map. The Fat-Client via Thin-Server architecture is employed in this research, since testing showed that the access times for the Thin-Client via Fat-Server architecture is significantly slower (see Chapter 7).

6.5.2 CCRS Data Server

The implementation scheme of the CCRS data server is similar to that of CLI data server and is illustrated in Figure 6.26. The ShowCCRSMap servlet also acts as a distributed CCRS data manager. The implementation details are omitted.

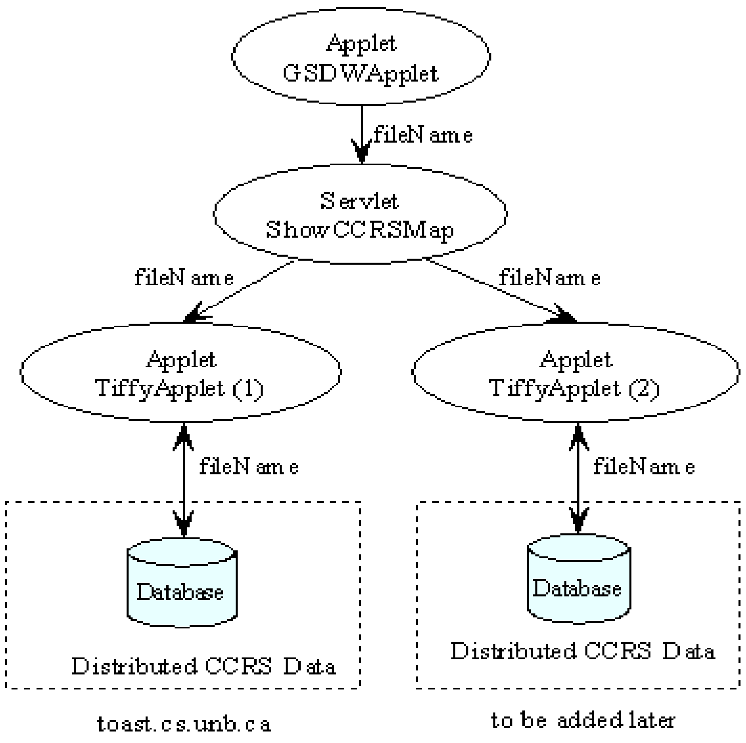


Figure 6.26: The implementation scheme of the CCRS data server.

Chapter 7

Testing and Evaluation

In this chapter, an explanation is provided for the tests designed and conducted to evaluate the implementation of the Web-oriented Distributed Geospatial Data system.

7.1 The Test Environment

7.1.1 Hardware and Software Settings

The Web-oriented Distributed Geospatial Data System was tested by using a Catalog Server, three geospatial data servers and a browser-based client (with three applets) in the Internet Computing Lab at UNB. The hardware settings of the test environment on the server side are listed in Table 7.1 and the software settings for testing our system are summarized in Table 7.2.

In the test, we chose Microsoft Internet Explorer 5.0 as the main client browser since it can display XML files. We also tested Netscape Navigator 4.6 and found it worked well except for displaying XML files. No browser plug-ins are required for the test.

Although many web server vendors support Java Servlets, BEA WebLogic Server

Table 7.1: The hardware settings of the test environment.

Host Name	**Role Name**	**Make/Model**	**CPU**	**Memory /LAN**	**Operating System**
butter	Catalog Server	Dell XPS T750r	PIII 750 MHz	512 MB 100Mbps	Windows 2000 Professional
butter	CLI Data Server(west)	Dell XPS T750r	PIII 750 MHz	512 MB 100Mbps	Windows 2000 Professional
toast	CLI Data Server(east)	Dell XPS T750r	PIII 750 MHz	512 MB 100Mbps	Windows NT 4.0 Workstation
toast	CCRS Data Server	Dell XPS T750r	PIII 750 MHz	512 MB 100Mbps	Windows NT 4.0 Workstation

Table 7.2: The software settings of the test environment.

Role Name	**Software Used**
CORBA-based Contextual Data Server	VisiBroker 4.1 for Java & C++
RMI-based Contextual Data Server	JDK1.2.2
Servlet-based Contextual Data Server	BEA WebLogic Server 6.0
Query, CLIData and CCRSData Server	BEA WebLogic Server 6.0
Client Browser	Microsoft Internet Explorer 5.0

6.0 (free download trial version) was chosen as the web server in this research. The BEA WebLogic Server is a popular application server for hosting E-business applications. It is powerful, reliable, flexible and integrated. BEA [2001a, 2001b] has more details about the features of BEA WebLogic Server 6.0 and its configuration for web service and servlets. In this research, the BEA WebLogic Server 6.0 was installed in two machines *butter* and *toast* in the Internet Computing Lab (ITB213) at UNB .

7.1.2 Distributing the Test Data Sets

Four types of data sets including GSHHS and GMT contextual data, CEONet XML metadata, CLI vector data (Arc/Info Export (.E00) format), and CCRS imagery data (GEOTIFF and JEPG format) were used for testing (see Chapter 4). The latter three types of data sets are used to create a total of 8188 ISO XML metadata files (total size of 42.12 MB). All of these ISO metadata files, together with the GSHHS and GMT contextual data files, are stored in the same machine hosting the catalog server, i.e. the machine *butter* in Internet Lab. The CLI vector data sets contains a total of 1690 files. They are divided into two parts, as shown in Figure 7.1. The Eastern_Canada data set contains 648 files that are stored in on the machine *toast* in the Internet Lab. The Western_Canada data set contains 942 files that are stored on machine *butter* in the Internet Lab. The CCRS imagery data are stored on machine *toast* in the Internet Lab.

7.2 Performance Test Results and Analysis

This section is dedicated to test results and analysis of the system. The performance evaluation is based on the criteria of process times.

7.2.1 Test Results of Building and Loading the GSDIndex

The GSDIndex (Search Engine) plays a key role in the implementation of our system. The process of building the GSDIndex involves two steps: (a) creating boundary and string AVL-Tree from ISO XML metadata files, and (b) building the GSDIndex.

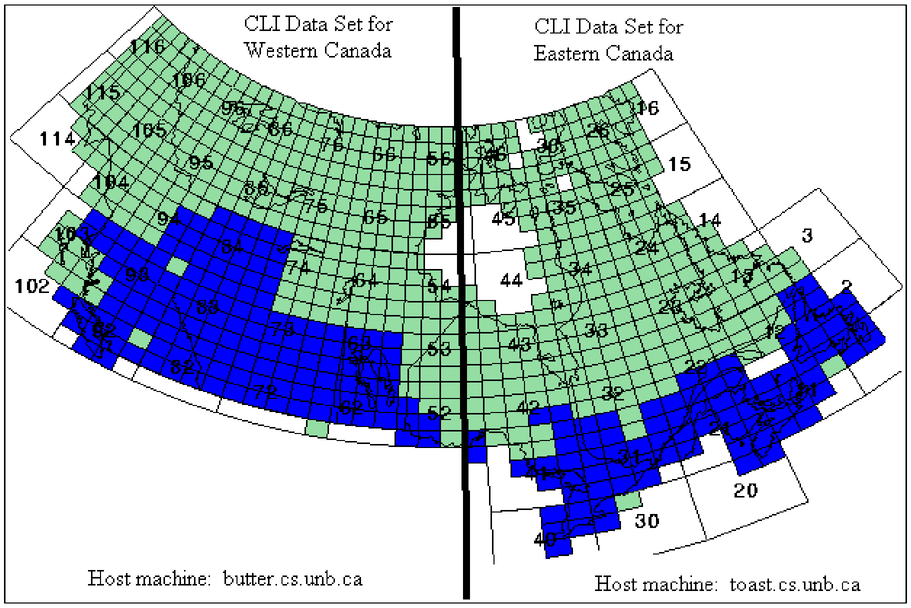

CLI Data Set for Western Canada
CLI Data Set for Eastern Canada
Host machine: butter.cs.unb.ca
Host machine: toast.cs.unb.ca

Figure 7.1: Distribution of the CLI data sets.

The test results of building the GSDIndex from the 8163 ISO XML metadata files for CEONet metadata and CLI vector data are listed in Tables 7.3 to Table 7.5.

Table 7.3: The test results for building the R-Tree and AVL-Tree for CEONet data.

	Building Time (ms)	**Saving Time** (ms)	**Total Size** (KB)	**Height**
R-Tree (M=50)	143,597	320	284	3
AVL-Tree	207,811	2,914	3,211	16

First, we build a new R-Tree (order M = 50) and a new AVL-Tree using the ISO XML metadata files for CEONet data and save them. Then, we load the created R-Tree and AVL-Tree, and insert bounding boxes and keywords from the ISO XML metadata files for CLI data and save them. The tests were conducted on *butter* with JDK1.2.2. The time (in ms) is the average time obtained from 10 time tests.

Table 7.4: The test results for building the R-Tree and AVL-Tree for CLI data.

	Loading Time (ms)	**Building Time** (ms)	**Saving Time** (ms)	**Size** (KB)	**Height**
R-Tree (M=50)	1,042	13,089	320	322	3
AVL-Tree	13,389	25,277	5,178	3,600	16

Table 7.5: The test results (time in ms) for building the GSDIndex.

R-Tree Load Time	**AVL-Tree Load Time**	**Build and Save Time**	**Total Size** (KB)	**Servlet Load Time**
1,012	16,554	6,399	3,921	7,710

In Table 7.5, the *Servlet Loading Time* (in ms) is the time that the Query servlet requires to load the GSDIndex (saved as a serialized Java object in a persistent file) during the servlet initialization. This may slow down the first query when the Query servlet is initialized by the BEA WebLogic Server.

7.2.2 Test Results of the Contextual Data Server

The performance of the four kinds of contextual data servers (servlet-based, CORBA-based (Java), CORBA-based (C++) and RMI-based) were tested in the same experimental environment. Both client and server programs were run in the same machine *butter*. A total of 16 crude resolution contextual data sets were generated by overlaying different thematic layers and used as the test data. Table 7.6 summarizes the selected test data sets.

Table 7.6: The summary of the selected contextual test data sets.

No.	Thematic Layers	Size (KB)	No.	Thematic Layers	Size (KB)
1	SL	217	9	SL+RV+LK	436
2	SL+PB	265	10	SL+RV+IL	260
3	SL+RV	255	11	SL+LK+IL	403
4	SL+LK	398	12	SL+PB+RV+LK	484
5	SL+IL	222	13	SL+PB+RV+IL	308
6	SL+PB+RV	303	14	SL+PB+LK+IL	451
7	SL+PB+LK	446	15	SL+RV+LK+IL	441
8	SL+PB+IL	270	16	SL+PB+RV+LK+IL	489

(SL: ShoreLines; PB: Political Borders; RV: Rivers; LK: Lakes; IL: Islands in Lakes)

During the test, the running time, which started from the client sending a request to the server to the end of drawing a map, was recorded in milliseconds by the client program. Table 7.7 summarizes the average times obtained from 10 tests for the four kinds of contextual data servers. Figure 7.2 shows the curve of the average transaction time changing with the size of data set in the 4 types of contextual data servers. The ratio of the pair-comparison is given in Table 7.8.

It is shown in Figure 7.2 and Table 7.8 that the performance of the servlet-based server is better than the others by a factor of approximately 2 to 2.5, and the others have almost the same performance. On average, the servlet-based contextual data server achieves a data transmission rate of 85 bytes/ms . That is because Java Servlet uses a simpler architecture to transfer data from the server to the client. When the Java Applet needs to communicate with the Java Servlet, it's the programmer's responsibility to find what data communication protocol should be used and at what

Table 7.7: The test results (time in ms) for the contextual data servers.

Dataset No.	**CORBA (Java)**	**CORBA (C++)**	**RMI**	**Servlet**	**Time/KB for Servlet**
1	5,027	4,676	4,897	2,464	11
2	7,110	6,630	6,994	3,344	13
3	6,780	6,239	6,199	3,198	13
4	10,245	8,692	9,324	4,456	11
5	5,508	4,737	7107	2,623	12
6	9,223	8,212	8,272	3,866	13
7	12,608	11,567	10,779	5,176	12
8	7,781	6,669	4,717	3,334	12
9	12,659	11,116	9,832	4,887	11
10	7,210	6,690	6,379	3,075	12
11	10,386	9,254	9,012	4,208	11
12	15,312	13,300	12,929	5,789	12
13	9,554	8,242	7,712	4,146	13
14	13,459	11,226	10,836	5,088	11
15	12,468	11,076	10,945	5,017	11
16	15,672	13,419	13,011	5,868	12

Table 7.8: The ratio of the pair-comparison.

Dataset No.	**J/C**	**J/R**	**C/R**	**J/S**	**C/S**	**R/S**
1	1.08	1.03	0.95	2.04	1.90	1.99
2	1.16	1.17	1.01	2.10	1.81	1.80
3	1.09	1.09	1.01	2.12	1.95	1.94
4	1.08	1.13	1.05	2.34	2.18	2.07
5	1.07	1.02	0.95	2.13	1.98	2.09
6	1.17	1.09	0.94	2.33	2.00	2.13
7	1.12	1.11	0.99	2.39	2.12	2.14
8	1.16	1.24	1.07	2.30	1.99	1.86
9	1.18	1.10	0.93	2.30	1.95	2.09
10	1.12	1.15	1.03	2.47	2.20	2.14
11	1.14	1.29	1.13	2.59	2.27	2.01
12	1.13	1.14	1.01	2.49	2.21	2.18
13	1.09	1.17	1.07	2.44	2.23	2.08
14	1.20	1.24	1.04	2.65	2.21	2.13
15	1.15	1.18	1.03	2.65	2.30	2.23
16	1.17	1.20	1.03	2.67	2.29	2.22
Average	1.14	1.15	1.01	2.36	2.15	2.07

(J: CORBA(Java); C: CORBA(C++); R: RMI; S: Servlet)

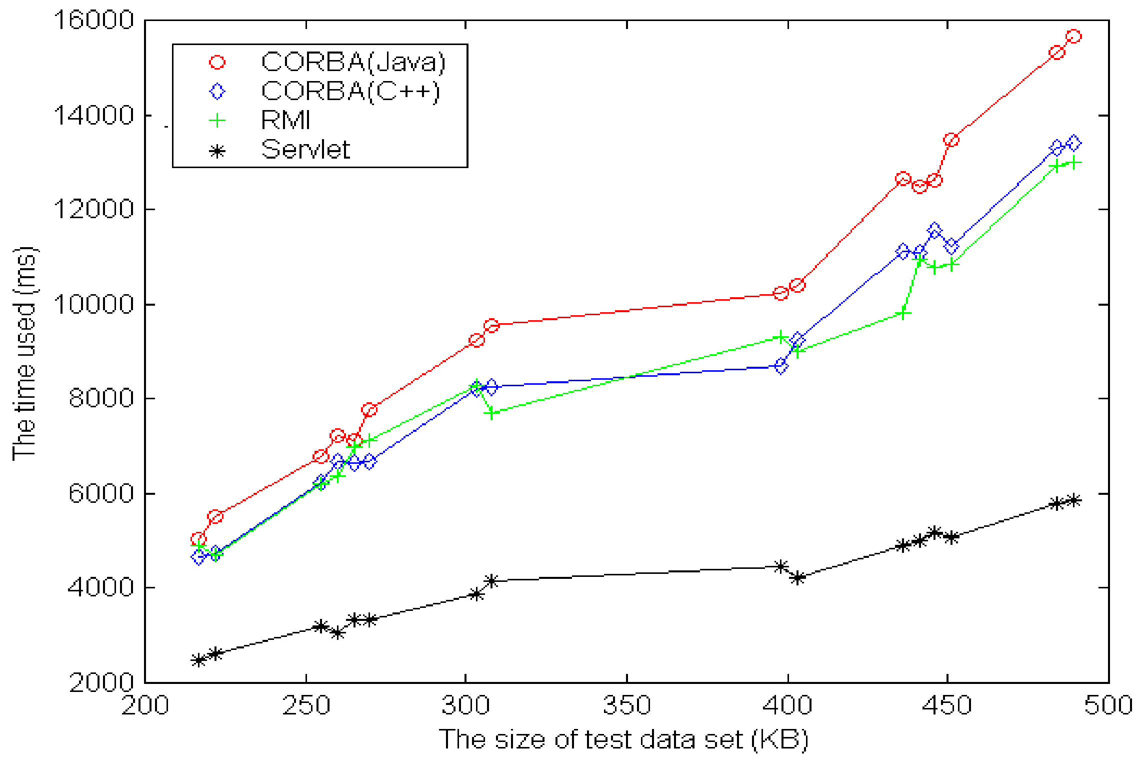


Figure 7.2: Performance comparison of the 4 types of contextual data servers

URL address the client can find the servlet program. In the others, a programmer doesn't have to worry how to connect to the server and where to find the server program; all this work is done by the *name service* of the ORB system. That is, the osagent Smart Agent of VisiBroker for the CORBA approaches and rmiregistry for the RMI approach were used to find a distributed object. The process of finding out where objects reside requires substantial time.

Based on this analysis, the three sub-systems (query, CLI data service, and CCRS data service) are all implemented using servlet techniques.

7.2.3 Test Results of the Query Server

The query server cooperates with the contextual data server to perform the func-

Table 7.9: The search time for keywords only.

Trial No.	Keywords	# of Items Found	Search Time (ms)	Time (ms) per item
1	earth science	6,503	4,927	0.76
2	oceans	1,516	90	0.06
3	canada land inventory	1,232	60	0.05
4	land use	339	10	0.03

(Test time: 11:00–11:30 AM, March 26, 2001)

Table 7.10: The search time for bounding boxes only. (Match factor $k = 0.01$.)

Trial No.	Bounding Boxes	# of Items Found	Search Time (ms)	Time (ms) per item
1	[0, 360; -90, 90]	3,914	2,473	0.63
2	[0, 180; -90, 90]	2,340	270	0.12
3	[180, 360; -90, 90]	3,508	952	0.27
4	[225, 300; 40, 60]	4,128	1,342	0.33

(Test time: 12:00–12:30 PM, March 26, 2001)

tion of catalog server. The query server was installed in the *butter* machine and it was tested by using the Internet Explorer 5.0 web browser running on the same machine. The search time is the time that it takes the client to get query results after the request is sent out. Search times for querying for keywords only, querying for bounding boxes only, and combined querying for keywords and bounding boxes are listed in Table 7.9 to Table 7.11. Table 7.12 shows the effect of the match factor k on the search results and the effect is depicted in Figure 7.3. The results in these tables are the average of 10 separate runs for each trial.

Table 7.11: The search time for combining keywords and bounding boxes. (Match factor $k = 0.01$.)

Bounding Boxes	Keywords	# of Items Found	Search Time (ms)	Time (ms) per item
$[0, 360; -90, 90]$	earth science	3,794	7,751	2.04
$[0, 360; -90, 90]$	oceans	1,196	2,754	2.30
$[0, 360; -90, 90]$	land use	80	2,694	33.68
$[0, 180; -90, 90]$	earth science	2,265	5,278	2.33
$[0, 180; -90, 90]$	oceans	842	411	0.49
$[0, 180; -90, 90]$	land use	46	324	7.04
$[225, 300, 40, 60]$	earth science	2,878	6,499	2.26
$[225, 300, 40, 60]$	oceans	753	1,452	1.93
$[225, 300, 40, 60]$	land use	246	1,352	5.50

(Test time: 12:30–1:30 PM, March 26, 2001)

Table 7.9 and Table 7.10 show that key word searches can take up to 4.9 seconds compared to bounding box searches times of less than 2.5 seconds. The per item search times of bounding box searches range from 0.12 to 0.63, whereas that of keyword searches vary from 0.03 to 0.76. Some keyword searches take much more time than others, e.g. "earth science" via "land use", since there are so many data sets contain the keywords "earth science". The search time for all bounding boxes searches does not vary significantly. Table 7.11 indicates that the combined search time is basically the sum of the two separate search times.

Table 7.12 shows that there is a significant effect of the match factor k on the search result. For a fixed bounding box $[200, 350; 20, 84]$, the variation of match factor from 0.90 to 0.50 results in a small change of the number of found items from 0 to

Table 7.12: The effect of the match factor k on the search results. (Bounding Box = [200, 350; 20, 84])

k	**# of Items**	**Time (ms)**	**Time (ms) per item**	k	**# of Items**	**Time (ms)**	**Time (ms) per item**
0.90	0	0	0	0.20	781	140	0.179
0.80	12	10	0.833	0.10	2476	640	0.258
0.70	22	10	0.455	0.01	3040	980	0.322
0.60	56	10	0.179	0.001	3537	1422	0.402
0.50	203	40	0.197	0.0001	5102	2774	0.544
0.40	291	50	0.172	0.00001	5111	2845	0.557
0.30	485	80	0.165	10^{-8}	5673	3135	0.553

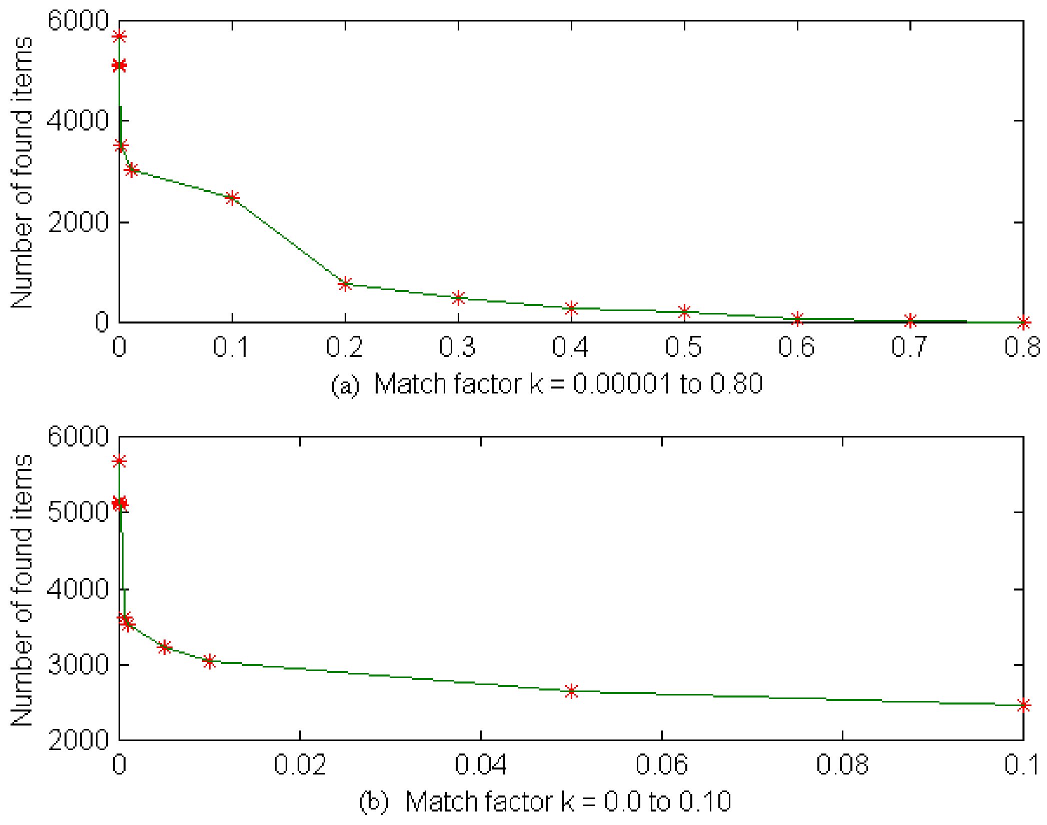


Figure 7.3: The effect of match factor k on the number of returned items.

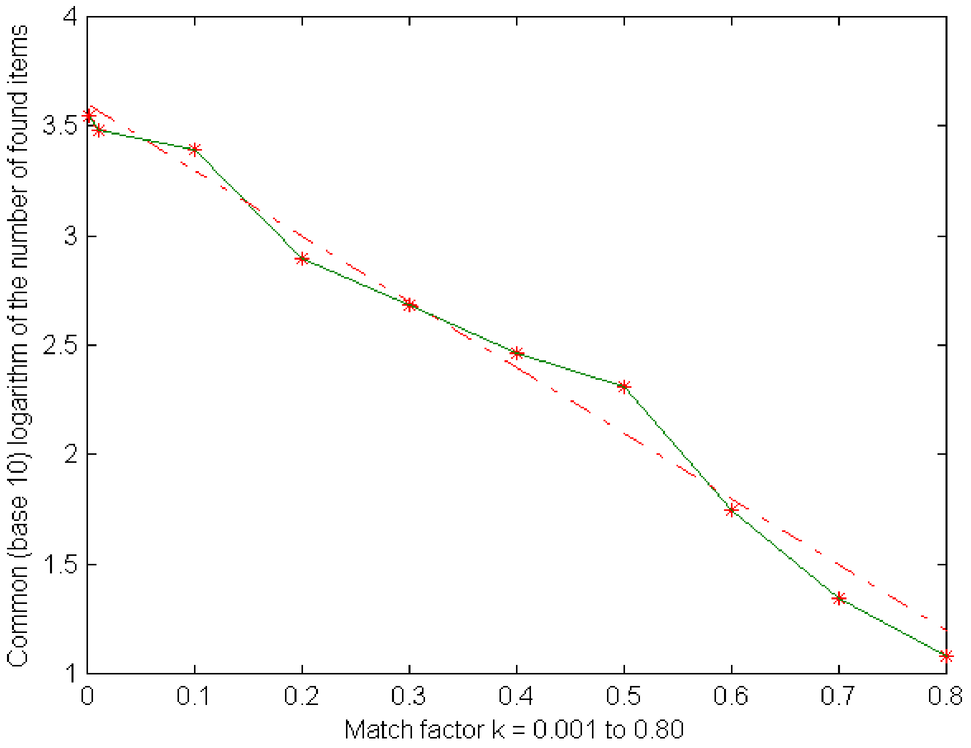


Figure 7.4: The effect of match factor k on the logarithm (base 10) of the number of returned items.

203, whereas the variation of match factor from 0.50 to 0.1 results in a large change of the number of found items from 203 to 2476. Furthermore, when the match factor changes from 0.1 to 0.0, the number of found items changes from 2476 to 5673. The per item search times (ms) falls in the range of $[0.15, 0.85]$. The effect of the match factor k on the number of returned items is depicted in Figure 7.3. Figure 7.4 shows the relation between the logarithm (base 10) of the number of returned items and match factor $k \in [0.001, 0.80]$. It seems from Figure 7.4 that there is a trend of $N = 10^{3.6}/10^{3k}$ for $k \in [0.001, 0.80]$, where N is the number of returned items.

Table 7.13: Test results of the CLI data service.

No.	**Data Size (KB)**	**Time (ms) for FC/TS**	**Time/KB for FC/TS**	**Time(ms) for TC/FS**	**Time/KB for TC/FS**
1	5,495	16,143	2.3	409,800	74.6
2	3,018	9,904	3.3	210,426	69.7
3	1,845	5,491	3.0	132,314	71.7
4	253	691	2.7	56,729	224.6
5	19	330	17.3	7,210	379.5

7.2.4 Test Results of the Geospatial Data Services

In this research, the ShowCLIMap applet cooperates with the CLIData Server to provide the CLI data service. We tested the performance of two architectural implementations of the CLI data service: Fat-Client via Thin Server (FC/TS) and Thin-Client via Fat Server (TC/FS) using three data sets which have different sizes. All the tests are conducted in the same machine *butter*. The test results are given in Table 7.13. The running time is the average time (10 time tests) measured from the client clicking a link to the CLI data service to the end of drawing a map on the screen (time in ms). The result shows that the running times for the Thin-Client via Fat-Server architecture is very large, whereas the performance of the Fat-Client via Thin-Server architecture is reasonable. For Fat-Client via Thin-Server architecture, the data transmission achieves the best performance of 349 bytes/ms. The best data transmission rate for Fat-Client via Thin-Server architecture is only about 15 bytes/ms.

Chapter 8

Conclusions and Future Work

In this research a distributed geospatial data integration system capable of performing geospatial queries and providing geospatial data services on multiple geospatial data servers has been designed, implemented and tested. In the following sections, the insights that have been gained through the work accomplished in this research and some possible further enhancements to the current implementation are discussed.

8.1 Conclusions

This research extended the previous work by Xiao [49] and Teng [43]. The Web-oriented Distributed Geospatial Data Warehouse (WDGSDW) system implemented in this research has added the capability to deliver distributed geospatial data in response to a query on the catalog server. In addition, a match factor k was added to the search engine to allow more flexible control of the query.

CORBA, RMI and Servlets techniques were tested to build the middle-tier component for the contextual data service. The Java Servlets technique was chosen to implement the whole system based on finding the CORBA and RMI techniques being 2 to 2.5 times slower, on average, compared to the Java servlet. Performance of 85 bytes/ms was observed, on average, for the servlet-based contextual data server.

The system is capable of querying multiple homogenous and heterogeneous geospatial databases in a geospatial query. The search engine is the kernel of the Web-oriented Distributed Geospatial Data Warehouse (WDGSDW) system. The search engine is implemented by combining the AVL-Tree for text search (keywords) and R-Tree for geographical search (bounding boxes) supporting a variable match factor restriction. The search engine is easy to build and update by providing appropriate ISO XML metadata files for any new geospatial data files. Experimental results from Chapter 7 indicate that keyword searches can take up to 4.9 seconds compared to bounding box searches times of less than 2.5 seconds on a catalogue containing 8188 entries. A combined keyword and bounding box search requires an average of 1.2 times more than the individual searches. The keyword search is dependent on the percentage of datasets containing the keywords. The search time (in ms per found item) ranges from 0.49 to 33.68, a 69 to 1 ratio.

It is also observed that there is a significant effect of the match factor k on the search result. For a fixed bounding box $[200, 350; 20, 84]$, when the match factor changes from 0.90 to 10^{-8}, the number of found items changes from 4 to 5673 and the per item search times (ms) falls in the range of $[0.15, 0.85]$. There is also a trend of $N = 10^{3.6}/10^{3k}$ for $k \in [0.001, 0.80]$, where N is the number of returned items.

Two actual geospatial data services, CLI data and CCRS data, were implemented in this research. The implementation provides a three-tier architecture to support a flexible, expandable, and most importantly, server-side transparent realization. The Fat-client via Thin-server architecture for CLI data service achieves the best performance of 349 bytes/ms, about 23 times as fast as the Thin-client via Fat-server architecture.

8.2 Future Work

The search engine is the key component of the implemented Web-oriented Distributed Geospatial Data Warehouse (WDGSDW) system and the query performance is determined mainly by the keyword searching which requires an exact string match. We expect that the search performance can be improved if we develop an appropriate data structure capable of supporting efficient geospatial queries (combined text and geographical queries). *Tries* [11] is one of the promising candidate index techniques which can used to build the search engine, since it can efficiently index spatial data as well as text data supporting approximate string match. Therefore, further work of this research may start by investigating if the trie data structure can compete with the combined AVL-Tree and R-Tree technique.

Another enhancement to the current system is to implement a sophisticated tool meeting the OpenGIS standard which can than provide geospatial data services for a wide variety of geospatial data formats.

The usability of the system can further be improved by developing a "geospatial web crawler" which runs around the entire web collecting information about geospatial data servers over the Internet and creating corresponding ISO XML metadata files. These metadata files then can be stored in the catalog server and used to up-date the search engine in a continuous, unattended fashion.

References

[1] Adelson-Velskii, G. M. and Landis, E. M. (1962), "An Algorithm for the Organization of Infromation", *Soviet Math. Doklady* 3 (1962), 1259–1263.

[2] Art&Computer Hackbarth (2001), Homepage: http://www.tiffy.de/, 2001.

[3] BEA Inc. (2001a), "Introduction to BEA WebLogic Server", March 2001, available from http://e-docs.bea.com/wls/docs60/intro/index.html.

[4] BEA Inc. (2001b), "Developing WebLogic Server Applications", March 2001, available from http://e-docs.bea.com/wls/docs60/programming/index.html.

[5] Box, J. et al (2000), "Simple Object Access Protocol (SOAP) 1.1", W3C Note 08 May 2000, available from http://www.w3.org/TR/SOAP/.

[6] The Canada Centre for Remote Sensing (2001), Images and Data Services, March 2001, available from http://www.ccrs.nrcan.gc.ca/ccrs/imgserv/seeimge.html.

[7] Chaudhuri, S and Dayal, U. (1997), "An overview of data warehousing and OLAP technology", ACM SIGMOD Record, 26: 65-74, 1997.

[8] Edwards, J. (1999), 3-Tier Server/Client at Work (Rev.Ed), John Wiley & Sons, New York, 1999.

[9] Farley, J. (1998), *Java Distributed Computing*, O'Reilly, CA, 1998.

[10] FGDC (1998), Federal Geographic Data Committee, "Content Standard for Digital Geospatial Metadata (CSDGM Version 2)", Document FGDC-STD-001-1998, available from http://www.fgdc.gov/metadata/contstan.html

[11] Fredkin, E. H. (1960), "Trie memory", *Coummunications of the ACM*, 3(9): 490-499, 1960.

[12] GEOIDE (2001), "Designing the technological foundations of geospatial decision-making with the World Wide Web (DEC#2)", 2001, available from http://www.geoide.ulaval.ca/Public/an/ProgrammesRD/Projets/Project2.html

[13] GeoGratis (2001), GeoGratis homepage, http://geogratis.cgdi.gc.ca/frames.html, 2001.

[14] Guttman, A. (1984), "R-trees: A Dynamic Index Structure for Spatial Searching", *SIGMOD Record*, Vol. 14 No. 2, 1984, pp. 47-57.

[15] Han, J., Lakshmanan, L. V. S. and Ng, R. T. (1999), " Constraint-Based, Multidimensional Data Mining", IEEE Computer, vol. 32, no. 8, pp. 46-50, 1999.

[16] Harold, E. R. (1997), *Java Networking Programming*, O'Reilly, CA, 1997.

[17] Henning, M., and Vinoski, S. (1999), Advanced CORBA Programming with C++, Addison Wesley Longman, Inc., Reading, Massachusetts, USA, p. 16, 1999.

[18] Hunters, J. (1998), *Java Servlet Programming*, O'Reilly, CA, 1998.

[19] Inmon, W. H. (1992), *Building the Data Warehouse*. John Wiley & Sons, New York, 1992.

[20] Inprise Corporation (2000), VisiBroker for Java/C++: Programmer's Guide (version4.1), Scotts Valley, CA, 2000.

[21] Lake, R. and Cuthbert, A. (Eds.) (2000), "Geography Markup Language (GML)" v.1.0, OpenGIS Consortium Recommendation Paper, 2000, available from http://www.opengis.org/techno/specs/00-029/GML.html.

[22] ISO/TC 211 /WG3 Editing committee 19115 (2001a), "ISO/TC 211 Geographic information/Geomatics CD 19115.3, Geographic information - Metadata", March 2000, available from http://www.statkart.no/isotc211/dokreg10.htm [N930], pp. 16–85.

[23] ISO/TC 211/WG3 Editing committee 19115 (2001b), "ISO/TC 211 Geographic information/Geomatics Final text of CD 19115 Geographic information - Metadata", December 2000, available from http://www.statkart.no/isotc211/dokreg11.htm [N1024], pp.17–86.

[24] Mahmud, Q.H. (2000), Distributed Programming with JAVA, Greenwich, CT : Manning, 2000.

[25] McDonnell, R. and Kemp, K. (1995), *International GIS Dictionary*, Cambridge: GeoInformation International, 1995.

[26] Microsoft (2001), "MSXML Parser 3.0 Release", 2001, available from http://msdn.microsoft.com/downloads/default.asp.

[27] Morissette, D. (2000), "Arc/Info Export (E00) Format Analysis", April 2000, available from http://www.geocities.com/ vmushinskiy/fformats/files/e00.txt .

[28] Nickerson, B. G., Teng, Y. and Xiao, J. (2000), "Web-based query processing for geospatial data using XML and CORBA", Proceedings of GEOIDE 2000 Conference, Calgary, Alberta, May 24-25, 2000.

[29] City of Oakland Website (2001), http://www.oaklandnet.com/maproom/.

[30] OpenGIS Consortium Inc. (1999a), "The OpenGIS Abstract Specification, Topic 0: Abstract Specification Overview (Version 4)", 1999, available from http://www.opengis.org/techno/specs.htm.

[31] OpenGIS Consortium Inc. (1999b), "OpenGIS Simple Feature Specification for SQL", 1999, available from http://www.opengis.org/techno/specs.htm.

[32] OpenGIS Consortium Inc. (2000), " OpenGIS Web Map Server Interface Implementation Specification (Version. 1.0.0)", May 2000, available by contacting the OGC at revisions@opengis.org.

[33] OpenGIS Consortium Inc. (2001), Homepage, http://www.opengis.org/.

[34] Orfali, R., Harkey, D. and Edwards, J. (1996), The Essential Distributed Objects Survival Guide, John Wiley & Sons, New York, 1996.

[35] Ken Orr (2001), "Data Warehousing Technology ", 2001, available from http://www.kenorrinst.com/dwpaper.html.

[36] Phyne, T. M. (1997), "Open Spatial Data Standards for the Information Highway", US EPA Scientific Visualization Centre, Research Triangle Park, North Carolina, 1997, available from http://www.gepg.psu.edu/ica/icavis/rhyne.html.

[37] The San Diego regional SanGIS (2001), Homepage, http://www.sangis.org/.

[38] Schussel, G. (1996), "Client/Server Past, Present, and Future", 1996, available from http://news.dci.com/geos/dbsejava.htm.

[39] Soy (2000), K. S., "Examination of Web-Based Geographic Information Systems Seminar in Information Policy: Digital Government – LIS 390.2 – Spring 2000", available from http://www.gslis.utexas.edu/ ssoy/federal/evalgis.htm.

[40] Sun Microsystems Inc. (2001a), “Using CORBA and Java IDL”, 2001, available from http://java.sun.com/products/jdk/1.2/docs/guide/idl/jidlUsingCORBA.html

[41] Sun Microsystems Inc. (2001b), “Java servlet API: The Power Behind the Server”, 20001, available from http://java.sun.com/products/servlet/index.html.

[42] Sun Microsystems Inc. (2001c), “Fundamentals of RMI”, 2001, available from http://developer.java.sun.com/developer/onlineTraining/rmi/RMI.html.

[43] Teng, Y. (2000), “Use of XML for query processing in Web-based geospatial data warehouses”, Faculty of Computer Science Technical Report TR00-135, University of New Brunswick, June 2000.

[44] Vinoski, S. (1997), “CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments”, *IEEE Communications Magazine*, Vol. 14, No. 2, 1997.

[45] Weiss, M. A. (1996), *Algorithms, Data Structures, and Problem Solving with C++*, Addison-Wesley, 1996.

[46] World Wide Web Consortium (2000), *Extensible Markup Language (XML) 1.0* (Second Edition), W3C Recommendation 6 October 2000, available from http://www.w3.org/TR/2000/REC-xml-20001006 (HTML file).

[47] Wessel, P. and W. H. F. Smith (1996), “A Global, Self-Consistent, Hierarchical, High-Resolution Shoreline Data Base”, J. Geophys. Res., 101, 8741-8743, 1996; data available from ftp://gmt.soest.hawaii.edu/pub/wessel/gshhs.

[48] Paul Wessel and Walter H. F. Smith (2000), ”The Generic Mapping Tools Version 3.3.6 Technical Reference and Cookbook”, University of Hawai’i at Manoa, School of Ocean and Earth Science and Technology, October 2000, available from http://imina.soest.hawaii.edu/gmt/gmt/doc/html/GMT_Docs/GMT_Docs.html

[49] Xiao, J. (2000), “WWW Access to Geospatial Data”, Faculty of Computer Science Technical Report TR00-133, University of New Brunswick, May 2000.

Vita

Candidate's full name: Lushu Li

Univeristy attended: PhD (Management Science - Operations Research), 1998
Southeast University
Nanjing, P. R. China

MSc (Applied Mathematicsand Control Theory), 1993
Nanjing University of Science and Technology
Nanjing, P. R. China

Publications:

1. Lushu Li and K. K. Lai (2000), "Fuzzy dynamic programming approach to hybrid multi-objective multi-stage decision-making problem", *Int. J. Fuzzy Sets and Systems* 117(1), 13-25.

2. Lushu Li and Z. H. Sheng (2000), "The structural property and weak convergence of fuzzy orthogonal measures, *Int. J. Fuzzy Sets and Systems* 112 (2), 271-276.

3. L.S. Li and K. K. Lai (1999), "A fuzzy approach to multiple objective transportation problem", *Int. J. Computers & Operations Research* 27 (1), 43–57.

4. K. K. Lai and Lushu Li (1999), "Fuzzy dynamic optimization approach to multi-objective resource allocation problem", *European J. Operational Research* 117(2), 293–309.

5. K. K. Lai, J. Xue and Lushu Li (1999), "Project Selection Modeling Using Fuzzy Multicriteria Evaluation and Fuzzy Boolean Programming", *Int J. Intelligent Control and Systems* 3(1), 19-38.