

**UCGE Reports  
Number 20153**

Department of Geomatics Engineering

**A Java Implementation for Open  
GIS Simple Feature Specification**

(URL: <http://www.geomatics.ucalgary.ca/links/GradTheses.html>)

by

**Chuanyun Fei**

**October 2001**



THE UNIVERSITY OF CALGARY

**A JAVA IMPLEMENTATION FOR OPEN GIS SIMPLE FEATURE SPECIFICATION**

by

CHUANYUN FEI

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES  
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE  
DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF GEOMATICS ENGINEERING

CALGARY, ALBERTA

October, 2001

© Chuanyun Fei 2001

## ABSTRACT

Distributed GIS is the trend in the current GIS community. It has been recognized that interoperability is one of major issues in the distributed geocomputing environment. To respond to non-interoperability problem, OGC creates a series of specifications to form an open framework as GIS standards. These standards are increasingly accepted by GIS software vendors, geodata and geoprocessing providers, and users. This research focuses on an implementation of OpenGIS Simple Features Specification in the Java computing platform, which is an important family member of OGC's specifications.

A Java version Implementation Specification for OpenGIS Simple Features is designed in this research based on the review and analysis of the OGC Abstract Specification, OGC implementation specifications for SQL, OLE/COM and CORBA, and other related works done by other organizations. The Geometry Data Model, the spatial component of OpenGIS Simple Features, was designed and implemented following the new Implementation Specification. The Template Union Model for buffer operation was introduced, and some new algorithms were developed. The reasonable geometry object classification logic made the designed model more extendable and implementable. The UML technology and Java standards applied in the design and implementation procedures made the model more maintainable and distributable. An easy-to-use Conformance Testing Suite was also developed to check whether or not each implementation is strictly compatible with the requirements of the new Simple Features Implementation Specification. The application example demonstrated that the design and implementation of the designed specification in this thesis are successful.

## ACKNOWLEDGEMENTS

I wish to express my deep gratitude to my supervisor, Dr. C. Vincent Tao, for his advice, guidance, encouragement, and support throughout my graduate studies. I am also grateful to Dr. Marina Gavrilova, Department of Computer Science, U of C, for her valuable suggestions about the algorithms of computational geometry during my thesis research. My appreciation also goes to Mr. Shuxin Yuan for his contribution in GeoEye<sup>TM</sup> 1.0, and postdoctoral fellows, Mr. Chaowei Yang and Mr. Quanke Wang, for their cooperation, discussions and suggestions in the development and improvement of the thesis related software, and Mr. Andrew Hunter, Mr. David Alton and Mr. Suen Lee for the proofreading of the draft of this thesis. I am also grateful to the secretaries, technicians and computer specialists and graduate students in the Remote Sensing and GIS Lab of the Department of Geomatics Engineering, who, in one way or other, helped me during my graduate studies. Finally my appreciation and thanks go to my family, my dearest wife, Xinxin Zhang for their endless love, understanding, devotion and support, which make this thesis possible.

## TABLE OF CONTENTS

<b>APPROVAL PAGE .....</b>	<b>ii</b>
<b>ABSTRACT.....</b>	<b>iii</b>
<b>ACKNOWLEDGEMENTS .....</b>	<b>iv</b>
<b>TABLE OF CONTENTS .....</b>	<b>iv</b>
<b>LIST OF FIGURES .....</b>	<b>viii</b>
<b>LIST OF TABLES .....</b>	<b>x</b>
<b>CHAPTER 1 INTRODUCTION .....</b>	<b>1</b>
1.1 Research Background.....	1
1.2 Objectives and Limitations.....	8
1.3 Outline .....	10
<b>CHAPTER 2 INTEROPERABILITY.....</b>	<b>12</b>
2.1 Background.....	12
2.1.1 GIS Trend.....	12
2.1.2 Overview of Interoperability.....	15
2.2 Interoperability Levels.....	19
2.3 Interoperability Approaches .....	23
2.3.1 Overview.....	23
2.3.2 OGC's Approach .....	25
2.4 Summary.....	31
<b>CHAPTER 3 RESEARCH METHODOLOGY .....</b>	<b>32</b>
3.1 Existing Work Review.....	32
3.2 Design And Implementation.....	33
3.3 Testing Suite .....	235
3.4 Case Study .....	35
<b>CHAPTER 4 EXISTING WORK REVIEW.....</b>	<b>37</b>
4.1 OGC's Implementation Specifications .....	37
4.2 Other Existing References.....	39
<b>CHAPTER 5 JAVA IMPLEMENTATION .....</b>	<b>46</b>

5.1	General Logical Model Design.....	46
5.1.1	Design Criteria .....	46
5.1.2	Geometry Entity Design.....	47
5.1.3	Geometry Entity Classification Criteria.....	52
5.2	Geometry Data Model Design.....	53
5.2.1	Abstract Geometry Data Model Design.....	54
5.2.2	Geometry Data Model.....	55
5.2.3	Factory Design .....	57
5.3	Geometry Implementation.....	57
5.3.1	Functions.....	58
5.3.2	Geodata Storage .....	79
5.4	Characteristics Analysis .....	83
<b>CHAPTER 6 TESTING SUITE .....</b>		<b>85</b>
6.1	Objectives.....	85
6.2	Testing Suite Design.....	86
6.2.1	Testing Suite Generation.....	86
6.2.2	Testing Procedure .....	86
6.2.3	Design Considerations .....	86
6.2.4	Components of Testing Suite.....	87
6.3	Implementation.....	88
6.4	Issues .....	91
6.4.1	Dataset.....	91
6.4.2	Distribution.....	94
<b>CHAPTER 7 CASE STUDY.....</b>		<b>97</b>
7.1	Background.....	97
7.2	System Design .....	97
7.2.1	Architecture.....	98
7.2.2	Data Model.....	100
7.3	Implementation.....	103
7.3.1	Database Implementation.....	103
7.3.2	GIS Prototype Implementation .....	104

7.4 Case Conclusions.....	109
<b>CHAPTER 8 CONCLUSIONS AND RECOMMENDATIONS .....</b>	<b>110</b>
8.1 Conclusions .....	110
8.2 Recommendations .....	114
<b>REFERENCES.....</b>	<b>116</b>
<b>APPENDIX DATA MODEL NOTATION.....</b>	<b>125</b>

## LIST OF FIGURES

Figure 1.1	The Relationship between Geodata, Geoprocessing and Geocomputing.....	1
Figure 1.2	Abstract Specification Topic Dependencies (OGC, 1999) .....	5
Figure 1.3	The Structure of OGC Feature (Gardels, 1996) .....	6
Figure 2.1	The Evolution of GIS Architecture .....	12
Figure 2.2	Interoperability Levels.....	20
Figure 2.3	Interoperability Approach Categories .....	24
Figure 2.4	The Relationships in Open Geoprocessing .....	25
Figure 4.1	The Difference in the Geometry Model of SQL, OLE/COM and CORBA Implementation Specifications .....	38
Figure 4.2	Geometry Hierarchy in OpenGIS Implementation Specification of OLE/COM and SQL.....	38
Figure 4.3	Geometry Hierarchy in OpenGIS Implementation Specification of CORBA	39
Figure 4.4	Comparison of the Classification under Different Rules.....	42
Figure 4.5	Geometry Interface Hierarchy in the Proposed Data Model from Oracle and MapInfo.....	43
Figure 4.6	Geometry Interfaces and Objects Hierarchy in Cadcorp’s Data Model ... ..	44
Figure 4.7	Geometry Interfaces and Methods Hierarchy in Bonn’s Data Model .....	45
Figure 5.1	Inheritance Relationship of Interface and Class in Java (D not permitted)...	49
Figure 5.2	Possible Roles of Geometry in Conceptual Model.....	54
Figure 5.3	Abstract Geometry Data Model in Interfaces .....	54
Figure 5.4	Geometry Data Model.....	56
Figure 5.5	Factory Hierarchy in Data Model .....	57
Figure 5.6	Example Instance of DE-9IM .....	62
Figure 5.7	Examples of the Touches Relationship.....	63
Figure 5.8	Examples of the Crosses Relationship.....	64
Figure 5.9	Examples of the Within Relationship .....	65
Figure 5.10	Examples of the Overlaps Relationship .....	65
Figure 5.11	Relationships among the Eight Relational Operations .....	67

Figure 5.12	Two Intersected Polygons .....	69
Figure 5.13	Intersections of the Two Polygons .....	70
Figure 5.14	Connectivity List of Point <i>x</i> .....	70
Figure 5.15	Labeled Polygons.....	73
Figure 5.16	Union Results.....	76
Figure 5.17	Line Object's Buffer .....	76
Figure 5.18	Original Buffer Templates .....	79
Figure 5.19	Example of UTM .....	79
Figure 5.20	Polygon Object Structure in Pure Object Method .....	81
Figure 5.21	Polygon Object Structure in Data Object Method .....	81
Figure 5.22	Data Objects Used in the Design.....	82
Figure 6.1	Interface after Loaded Data in the Prototype System .....	90
Figure 6.2	Buffer Testing in the Prototype System .....	91
Figure 6.3	Test Data Concept (OGC, 1999c) .....	92
Figure 6.4	Points in the Blue Lake Dataset (OGC, 1999c) .....	92
Figure 6.5	Unzipped File Structure of the Testing Suite .....	94
Figure 7.1	Architecture of the Geotechnical Data and Analysis Sharing System .....	99
Figure 7.2	Borehole Profile Diagram ... ..	100
Figure 7.3	A Conceptual Geotechnical Data Model in Oracle .....	101
Figure 7.4	Geospatial Data Model .....	103
Figure 7.5	Object Communication between Client and Server .....	106
Figure 7.6	New GeoEye™ User Interface .....	107
Figure 7.7	Interface of Profile Analysis.....	107
Figure 7.8	Data Registration Interfaces in GeoServNet Server .....	108

## LIST OF TABLES

Table 2.1	An Overview of Three Generation of Interoperability in R&D (Sheth, 1999).....	17
Table 2.2	OGC Released Implementation Specifications.....	29
Table 2.3	OGC Conforming Products .....	30
Table 5.1	Java's Naming Conversion (Sun, 1999) .....	50
Table 5.2	Basic Functions in Geometry Data Model.....	60
Table 5.3	Examples of Template .....	78
Table 5.4	Memory Expense of the Three Methods.....	83
Table 6.1	Implementation Status of the Functionality in the Testing Suite.....	89
Table 6.2	Comparison of the Implementable Objects in the Data Model and Dataset.....	94
Table 7.1	View Schemes in Geotechnical Database.....	104
Table 7.2	GeoEye™ 1.0 Functions (Yuan, 2000).....	106

# CHAPTER 1 INTRODUCTION

## 1.1 Research Background

Essentially, the purpose of a GIS application is in applying geoprocessing to the geospatial dataset from six aspects (Figure 1.1) for explaining events, predicting outcomes, planning strategies and making decisions.

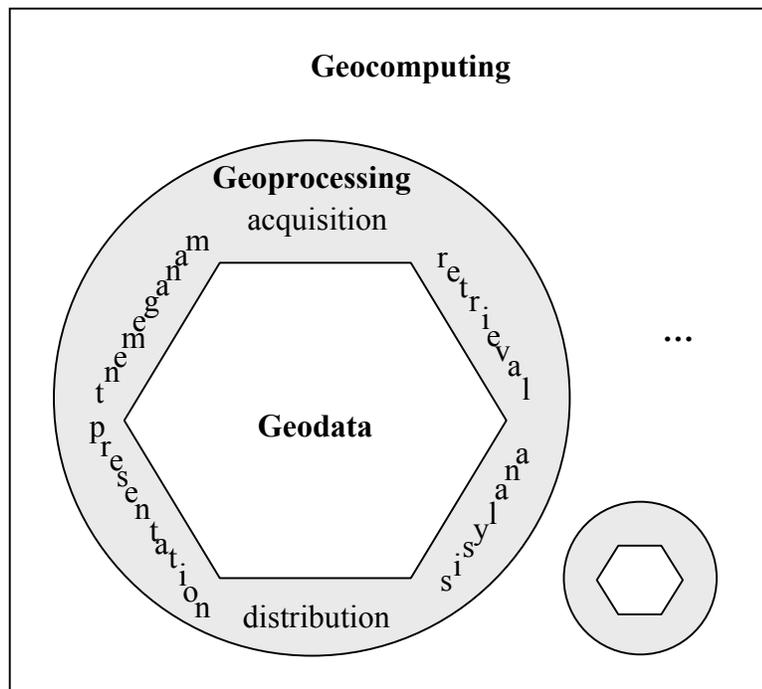


Figure 1.1 The Relationship between Geodata, Geoprocessing and Geocomputing

Geodata describes phenomena directly or indirectly associated with a location (and time, and orientation) relative to the surface of the Earth. The overall rate of geodata collection has increased rapidly and the data formats tend to be complex (OGC 1996). Sharing data is a

cumbersome, daunting, frustrating, error-prone, sometimes totally impractical task. The obstacles of geodata sharing are broadly referred as “non-interoperability.”

Geoprocessing can be any kind of digital computing that uses geodata. This means the use of a series of special methods to process the geodata in a software and hardware environment. During the passed 30 years, many different methods for acquiring, storing, processing, analyzing, and viewing geodata have been developed, mostly independently from one another (OGC 1996). The software that uses and produces geodata is itself varied and complex. That’s why the non-interoperability problem exists in geoprocessing and the question “How can we enable software from different vendors, which use very different data structures, to communicate, process and share data with each other” has often been asked in the GIS community.

Non-interoperability is a big issue of data and functionality sharing in GIS. Interoperability has been the common concern of GIS software vendors, geographic information providers and GIS users. Although there is no common definition of interoperability, the term “interoperability” suggests an ideal world in which the geodata, geoprocessing and geocomputing can be easily divided and reassembled to achieve reuse in data, application and system levels.

The solution for achieving interoperability is creating common standards. Like automobile manufacturing, the standardized parts provided by different manufacturers can be assembled to produce a car, GIS geoprocessing should be based on standards. The GIS community has invested much effort to establish standards, such as the geodata format standards, but those efforts are not done with the whole of the GIS industry in mind. The lack of industry-wide well-defined standards in GIS makes it impossible to achieve interoperability.

In response to the problem of non-interoperability and its many negative ramifications for industry, government, and academia, the Open GIS Consortium, Inc. (OGC) was formed in 1994. The OGC is a not-for-profit trade association dedicated to promoting new technical and commercial approaches to interoperable geoprocessing. The members of OGC share a positive vision of a national and global information infrastructure in which geodata and geoprocessing resources move freely, fully integrated with the latest distributed computing technologies, accessible to everyone, “geo-enabling” a wide variety of activities that are currently outside the domain of geoprocessing, opening new markets and giving rise to new kinds of businesses and offering new benefits to the public. Geoprocessing software vendors, database software vendors, visualization software vendors, system integrators, computer vendors, telecommunications companies, universities, information providers, and federal agencies have joined the Consortium to participate in creating a software specification and new business strategies that will help solve these problems and fulfill these lofty goals.

Defining OpenGIS Specifications for the GIS community is the approach for OGC to fulfill its goal. The specifications, widely accepted and understood in GIS, define (OGC, 1996):

- The Open Geodata Model: A general and common set of basic geographic information types that can be used to model the geodata needs of more specific application domains, using object-based and/or conventional programming methods.
- OpenGIS Services: The set of services needed to 1) access and process the geographic types defined in the Open Geodata Model and 2) provide capabilities to share geodata within communities of users who use a common set of geographic feature definitions and translate

between different communities of users that use different sets of geographic feature definitions.

- An Information Communities Model that employs the Open Geodata Model and OpenGIS Services in a scheme that establishes:
  1. A way for a community of geodata producers and users who already share a common set of geographic feature definitions to efficiently and effectively maintain these definitions and to catalog and share datasets conforming to these definitions.
  2. An efficient and optimally accurate way for different communities of geodata users and producers to share geodata despite their dissimilar sets of geographic feature definitions. For example, civil engineers, geologists, and agronomists may seek to share soils data despite the fact that they characterize soil types differently according to their different professional objectives. The Information Communities Model defines a scheme for automated translation between different geographic feature lexicons.

OGC has two kinds of specifications: Abstract Specification and Implementation Specification. The Abstract Specification is organized into separate topic volumes in order to manage the complexity of the subject matter and to assist parallel development of work items by different Working Groups of the OGC membership. The 16 topics (see Figure 1.2) are organized into two Central Themes: Sharing Information and Providing Services. Topics 12, 13, 15 and 16 are concerned with providing geospatial services. The remainders are centered on sharing geospatial information. Topic 5, 6 and 7 are fundamentally concerned with the handling and exposing of geospatial information from three different perspectives: Features with Geometry, Coverages and Imagery. It is clear that Topic 6 is an extension of Topic 5, and Topic 7 is an extension of Topic

6. Topic 1, Feature Geometry, provides the geometry structure for Topic 5. The other topics in Sharing Information are supporting topics for Topic 5, 6 and 7. From an architecture's perspective, Topic 5 is the key topic in the Sharing Information theme.

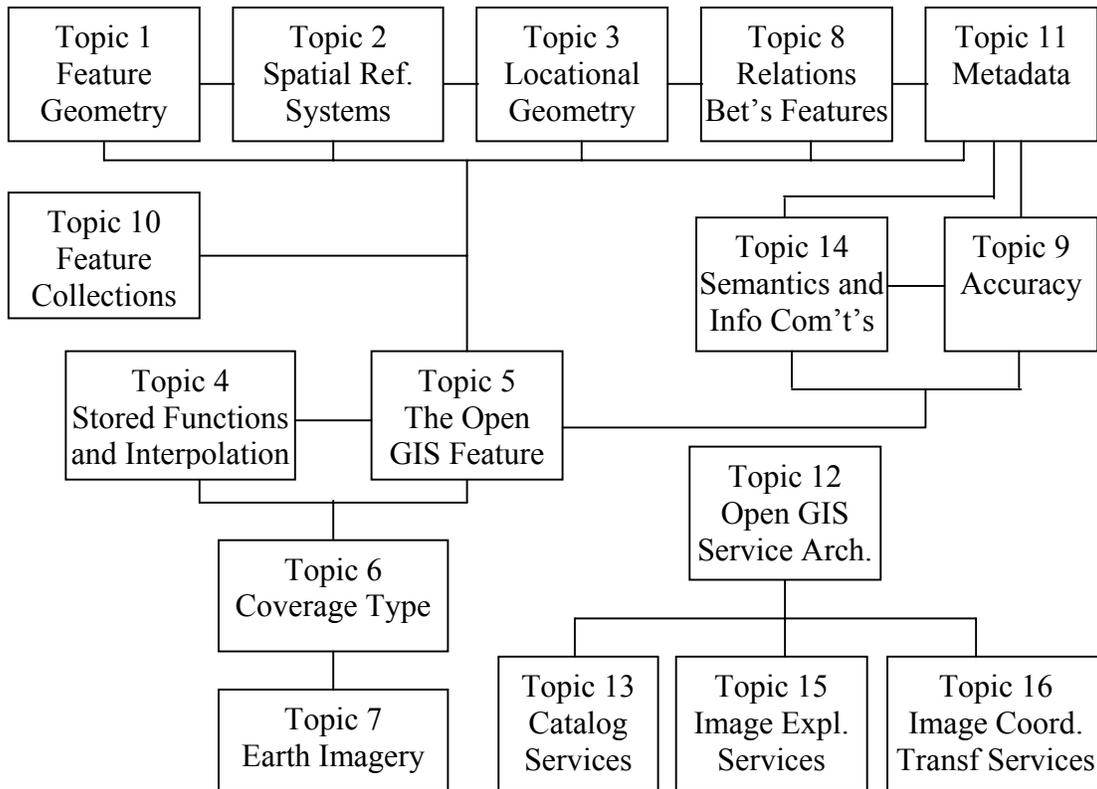


Figure 1.2 Abstract Specification Topic Dependencies (OGC, 1999)

The Feature Geometry in Topic 5 has three components (see Figure 1.3) in OGC specifications. For the **metadata component**, the standardization work has been done by other organizations, such as ISO, FGDC. It is not necessary for OGC to define other metadata standards. The **semantic component** is too complex and is a huge challenge for OGC to standardize it at this moment. To get the focus, OGC currently puts its efforts on the **spatial component** in the *Feature Object*. In OGC's released specifications: OpenGIS Simple Features Implementation Specification for CORBA, for OLE/COM, and for SQL, only the Simple Features are defined.

Here the “simple” is something of a misnomer and merely recognizes that the specification does not include access to all aspects of a feature (Cuthbert, 1999). Only the Geometry in spatial components of the *Feature Object* is addressed in this thesis. That is why the thesis title is an implementation for Simple Features.

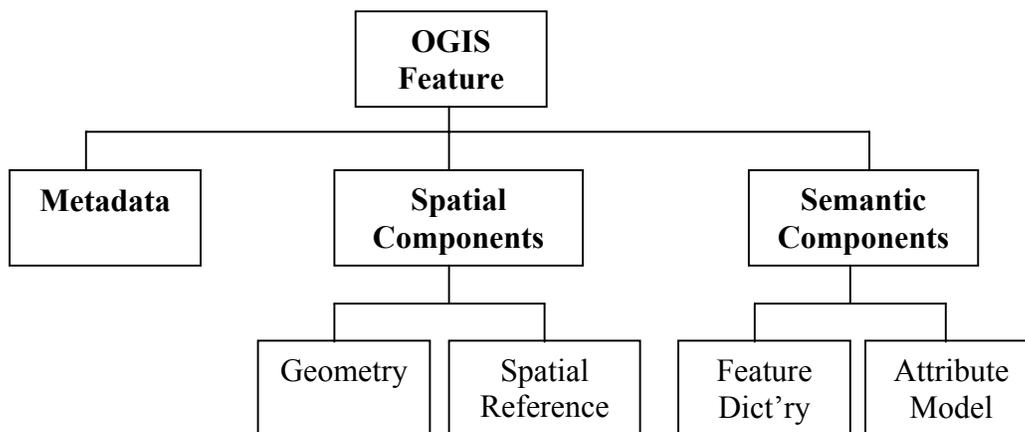


Figure 1.3 The Structure of OGC Feature (Gardels, 1996)

OGC’s Implementation Specifications for Abstract Implementation focus on different Distributed Computing Platforms (DCPs). OGC began with an assumption that there were a small number of DCPs that could be exploited. There are, therefore, three Implementation Specifications of OpenGIS Simple Features in SQL, CORBA and OLE/COM for the Abstract Specification. Changes to the technology are ongoing. Java is not only a popular programming language, but is also becoming a major DCP in current network environment. It is necessary to review OGC’s fundamental assumption and develop an Implementation Specification for the Java DCP. The following characteristics of Java show that it is an ideal tool to implement a new version of OpenGIS Simple Features in Java computing platform.

**Object-oriented.** Object-oriented design is the mechanism for defining how modules “plug and play”. The object-oriented facilities of Java are essentially those of C++, with extensions from Objective C for more dynamic method resolution. Most members of the OGC Technical Committee believe that an object approach is the ideal way to realize the goals and objectives (OGC, 1998).

**Architecture Neutral.** Java was designed to support applications on the network. The compiler generates an architecture neutral object file format - the compiled code is executable on many processors anywhere on the network, given the presence of the Java runtime system. There are no platform-dependent aspects of the Java language including primitive data types, libraries. It gives Java the capability to cross different hardware and software platforms.

**Ubiquity.** Java is embedded in many Web browsers. The Application Programming Interfaces, the Advanced Windowing Toolkit, the Java Foundation Classes (JFC), and JDBC, are leading toward even more deployment of Java. Java provides ways to call already compiled (legacy) code (native methods) to integrate the legacy software, which presents the idea: “Code once, run anywhere.”

**Flexibility.** The mechanism of dynamic class loading allows the virtual machine to load and define classes at runtime. The class-loading mechanism is extensible and enables classes to be loaded via the network. Combined with other features of Java, such as generating bytecode and architecture neutrality, the Java application can be dynamically changed, updated and controlled at runtime via network with little effort (Cornell and Horstmann, 1997).

**Introspection.** Java code can discover information about the variables, methods, and constructors of loaded classes, and can use reflected variables, methods, and constructors to operate on their underlying counterparts in objects, all within the security restrictions. The ability of introspection is to discover and dynamically load new class definitions and, as a consequence, improve the ability of an object to serialize and distribute itself (Cornell and Horstmann, 1997).

Java's Interface-Class mechanism maps OGC's Abstract Specification-Implementation Specification strategy. The Java's Interface technology can take on the role to draw out OGC's framework in a Java environment. Class technology can efficiently code the framework into Java by using Java's rules. As OGC's implementation specifications evolved, Java implementation can conform to OGC's updates with few coding.

## **1.2 Objectives and Limitations**

The major objective of this research is to design and propose a specification of OpenGIS Simple Features for Java. The focus of the research is on the simple features' design, implementation and testing. The objectives of this thesis research:

- Design a Java version Specification of OpenGIS Simple Features. This design is made with reference to the specifications, SQL, OLE/COM and CORBA, and the characteristics of the Java computing platform.

- Develop a Conformance Testing Package to test the conformance of the claimed products with the OpenGIS Simple Feature Specification for Java. The testing software and documents are developed.
- Implement a prototype Geometry Data Model based on the designed OpenGIS Simple Features Specification for Java. All the mandatory requirements and most of the optional requirements are implemented.
- Develop an application example to demonstrate and test the Geometry Data Model.

The design of a specification is a technologically complex task. Adding to the challenge is that many issues have not reached a common consensus in GIS at present. However, due to the time limitations, many of the following listed issues are not included in this research.

- This research focuses on the technological issues. The scientific and societal issues involved in GIS systems are ignored.
- This research does not intend to implement all the designs although a simple feature specification has been proposed. The Spatial Reference System has not been implemented in the research prototype system.
- This research focuses on the simple features. The other elements of the feature, such as metadata and semantics are not addressed in this research.
- This research focuses on the Java computing platform. The issues of interoperability between the Java version specification and other platform's specifications are not discussed in the thesis.

### 1.3 Outline

Eight chapters, including this introductory chapter, are organized as follows.

Chapter Two addresses interoperability, one of major issues in the GIS community. Interoperability attracts more and more researcher's attention. The background of this problem, the levels of interoperability and the approaches to address this problem are detailed. The OGC's approach, the effective means, is analyzed clearly from several aspects in this chapter.

Chapter Three introduces the research methodology applied in this research. The research methods and developing tools used in this thesis are presented in this chapter.

Chapter Four reviews some works related to this research. The Geometry Data Model in OGC's Implementation Specifications for SQL, OLE/COM and CORBA is presented. The weakness of other works done by other organizations is analyzed respectively. This review provides the reference and a start point for this research.

Chapter Five designs the Implementation Specification of OpenGIS Simple Features for Java. The design considerations, criteria of the geometry object classification, and characteristics of this design are discussed clearly in this chapter, includes the details of the logical model design and geometry data model design. The implementation of the prototype Geometry Data Model is detailed here. A new buffer model, Template Union Model, is introduced.

Chapter Six proposes the conformance Testing Suite for the OpenGIS Simple Feature Implementation Specification for Java. The suite design consideration and implementation are detailed in this chapter. The testing dataset and suite distribution issues are discussed.

Chapter Seven examples a case study based on the designed OpenGIS Simple Feature Implementation Specification for Java. The prototype Geometry Data Model is embedded into the client and server sides in the client/server architecture to build up an application system. This prototype system tests and demonstrates the design in Chapter Five.

Chapter Eight concludes the research. Conclusions are drawn based on the research. Also recommendations for the future research are given.

## CHAPTER 2 INTEROPERABILITY

### 2.1 Background

#### 2.1.1 GIS Trend

Advances in computing and communications technology are constantly breaking new ground in processing speed, storage capacity and miniaturization, diversity and complexity of input and output devices, and transmission capabilities of networks. With the growth and popularity of the computer network, more and more computing is moving towards a networked computing environment. In this same way, GIS computing architecture has evolved (a) from mainframe GIS to desktop GIS, (b) from desktop GIS to network-based (Internet/Intranet) client/server GIS, and (c) from client/server GIS to ubiquitous distributed GIS (Tao 2000). Distributed GIS (DGIS) systems, including the client/server GIS and ubiquitous GIS, are becoming the mainstream of GIS and has attracted more and more researchers and industries to invest into this next generation of GIS.

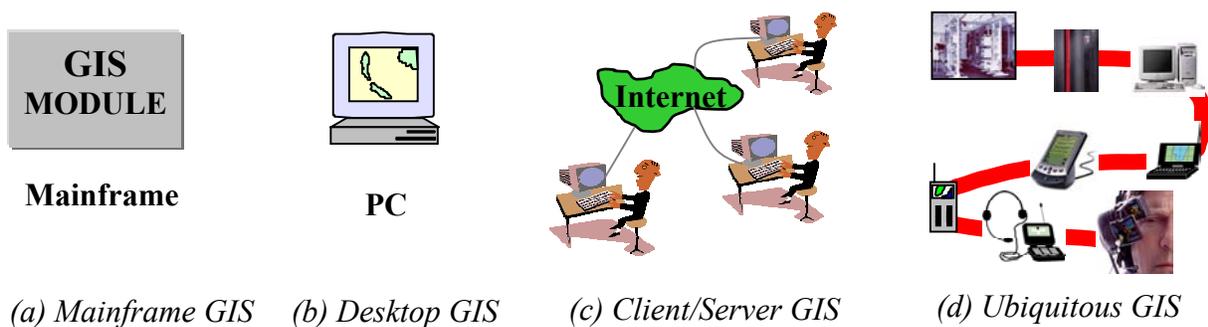


Figure 2.1 The Evolution of GIS Architecture

Generally, an ideal DGIS system distributes its components: hardware, software, data, user and management into the networked computing environment: LAN, Enterprise, WAN, Intranet and Internet. The components can be dynamically integrated together to finish specific tasks. The hardware, operation system and user are inherently dispersed in the environment. Most of researches, therefore, are done on the GIS software, geodata and DGIS system management and maintenance at current stage. From a software engineering viewpoint, these research topics can be divided into following categories:

### **Geodata Sharing**

The Internet provides a convenient way to share the geospatial datasets dispersed across different agencies (consultants, government agencies, universities, colleges, etc.). Much research was diverted to this topic at the beginning stages of DGIS. The data providers create metadata database, catalog and clearinghouse, and put them onto the Internet for interested users to find their required dataset. Desired data can be delivered by simply downloading from the data provider's site or with a more traditional mail system in the original or required data format. Metadata is the key issue among the obstacles of geodata sharing systems in this typical geodata sharing scenario. The research on the metadata standard, metadata semantics, metadata auto-generation and management, and metadata simplification is very hot.

### **GIS Service Accessing**

Current commercial Web GIS, such as ArcIMS<sup>®</sup> and MapGuide<sup>®</sup>, results from the evolution of the traditional stand-alone GIS by separating the graphical user interface (GUI) from the system and moving it onto the Internet. The GIS Service Accessing addresses is how the distributed

clients access the centrally controlled GIS services hosted on the server machine. There are two approaches to transfer geodata from server side to client's browser: image/picture and vector data (Yun, 1999). At present, in order to improve the speed of system response, some server side functions are moved to the client side, such as display, zoom, and pan. GIS service involves many related technologies: server and client technology, data transfer, protocol and communication. The system developer pursues how to improve the system's performance. How to design the client side to use the open API according to standard specification such as OpenGIS and W3C recommendations for interoperability in distributed computing environment is attracting much attention.

### **Distributed Geoprocessing**

Distributed geoprocessing is at a high implementation level of DGIS. At the GIS service stage, only the client side is distributed on the Internet, and the server is often centralized. However, the client and server are distributed on the networked environment at the distributed geoprocessing stage. It allows the dynamic configuration of the system during the running time. It is a new research direction for DGIS.

Distributed computing offers advantages in its potential for improving availability and reliability through replication; performance through parallelism; sharing and interoperability through interconnection, and flexibility, incremental expansion and scalability through modularity and componentization (Kramer, 1994). DGIS, a branch of distributed computing, supports "geocomputing at anytime and anywhere in any devices", no matter where the distributed geodata and geoprocessing are, at a client site or at a server site. Many important issues are

involved in the development and deployment of DGIS, such as security, integrability, system dynamic configuration, and so forth. However, it has been recognized that the top challenge is interoperability of both geodata and geoprocessing (Buehler and McKee, 1998).

### **2.1.2 Overview of Interoperability**

Interoperability is not entirely new. It has been discussed within the broader information system communities and certain aspects have been implemented outside of GIS for years. Interoperability in GIS has received a significant research attention in recent years.

In the 1970s and 1980s, GIS projects typically collected their datasets from analog sources (maps, field measurements and surveys), making the data collection and maintenance the main cost factor in a project. This has been replaced in many cases by exchanging, selling, and purchasing existing datasets, leading to the demands for open and interoperable systems.

Systems are designed according to what needs to be done, not what can be done. Those system does not live in splendid isolation but are typically embedded into large information technology infrastructures with a growing need to exchange information and services. The goal of interoperating GISs is to achieve an automated process that will allow us to use data and software services across the boundaries that their collectors and designers envisioned. An *ad hoc* integration may work in a few specialized cases, but it is often difficult or impossible to generalize such an approach. The deficiencies of *ad hoc* interoperability become apparent when interoperating subparts, which are supposed to be extended by integrating new components or adding functionality or replacing subparts with new software pieces. The difficulties are

primarily in the semantics of the diverse implementations. Compatible semantics of geospatial information are a key characteristic of interoperating GISs and powerful methods to capture and describe geospatial semantics are critical.

There are three periods of interoperability evolution (Sheth, 1999). During the second half of the 1970s, we saw the ability to deal with hardware, operating systems, and communications heterogeneity; although with evolution in each of these, new issues have to be continuously addressed.

During the 1980s, we saw significant progress in managing heterogeneity and support interoperability or integration in environments with structured databases and traditional database management systems (DBMSs). There is a large body of work during the first generation in dealing with heterogeneity associated with data models or schematic issues, DBMSs including query languages, concurrency control, commit and recovery, etc.

During the 1990s, the emersion of distributed computing, middleware technology, and standards has allowed us to increase the focus on the heterogeneity. This has particularly supported syntactic and structural interoperability, and allowed us to address issues at the information level. As the future information system addresses the information and knowledge level issues, it will increasingly require semantic interoperability. Semantic interoperability requires that the information system understand the semantics of the user's information request and those of information sources, and uses mediation or brokering to satisfy the information request as well as it can. Table 2.1 provides an overview of the three generations of systems in interoperability.

Table 2.1. An Overview of Three Generation of Interoperability in R&D (Sheth, 1999)

	<b>Generation I</b>	<b>Generation II</b>	<b>Generation III</b>
<b>Interoperability concerns</b>	<u>System</u> , data	System, <u>data</u> , information	system, data, <u>information</u> , <u>knowledge</u>
<b>Interoperability scope</b>	Handful of interconnected computers and databases	Tens of systems on a LAN, databases and text repositories	Enterprise-wide and global scope
<b>System architecture</b>	Terminal access, point-to-point; remoter access, mainframes & minicomputers, client-server(two tier)	Client-server (three tier)	Network, distributed, and mobile
<b>Communication protocol</b>	Proprietary (IBM domain), TCP/IP	TCP/IP, HTTP, CORBA	Internet/Web/Java, distributed object, multi-agent mobile, component
<b>System model</b>	Relational and E-R	Object-oriented	Component-based, multi-modal
<b>Interoperability dominant</b>	Multi-databases or <u>federated</u> databases	Federated information systems, <u>mediator</u>	Mediator, information <u>brokering</u>
<b>Interoperability types</b>	System (computer and communication); limited aspects of syntax	Syntax (data types and formats), structure (schematic,	Semantic (increasingly domain-specific)

	and structure (data model); transparency of location, distribution, replications	query languages and interfaces)	
<b>Interoperability approaches</b>	Structural and data representation	Understanding of a variety of metadata, comprehensive understanding of schematic heterogeneity	Comprehensive use of metadata, increasing emphasis on semantics and ontology supported approaches
<b>Types of data</b>	Structured databases and files	Structured databases, text repositories, semi-structured and structured and data in generic and domain specific formats	All forms of digital media with increasing support for visual/spatiotemporal/scientific/engineering data
<b>A few representative applications</b>	Integration of business databases or public databases	Digital library, integrated access to heterogeneous data for a software team	Digital earth, environmental phenomena, multi-step and multi-modal intelligence analysis

In short, the characteristics of GIS interoperability can be summarized as follows:

- **Openness and Transparency.** For the GIS software designers and developers, they should provide open models, open systems to support direct communications among GISs; for system builders and integrators, the details behind the common interfaces are hidden.
- **Exchange and Extensibility.** For the GIS service providers, the data and functionality can be easily exchanged among GISs. The open data model can easily integrate or be integrated with other domain data models.
- **Simplicity and Similarity.** For GIS users, they can easily migrate from one system to another for the common user interfaces. Their experience and knowledge are still useful in the new system environment.

## **2.2 Interoperability Levels**

Table 2.1 presents interoperability in several aspects, but it is difficult to identify a number of different ways in which interoperability might be realized. In general, interoperability can be classified into five levels between two or more geographically distributed independent GISs illustrated in Figure 2.2.

Hardware and Network interoperability lies at the lowest level. A computer network consists of hardware, network protocols and other software. The hardware includes network interface cards and cables those link them together. The network protocols are rules and procedures used to communicate among the connected system. TELNET is a good example of this kind of interoperability - the user logs into a remote system without any knowledge of the network protocol it supports. International Standard Organization (ISO) has issued some standards to

achieve this level's interoperability, such as the well defined seven layers in its reference model, and the successful TCP/IP protocol. At this level, the hardware and network only provides links in the whole system, users usually communicate without any direct service from the remote GIS.

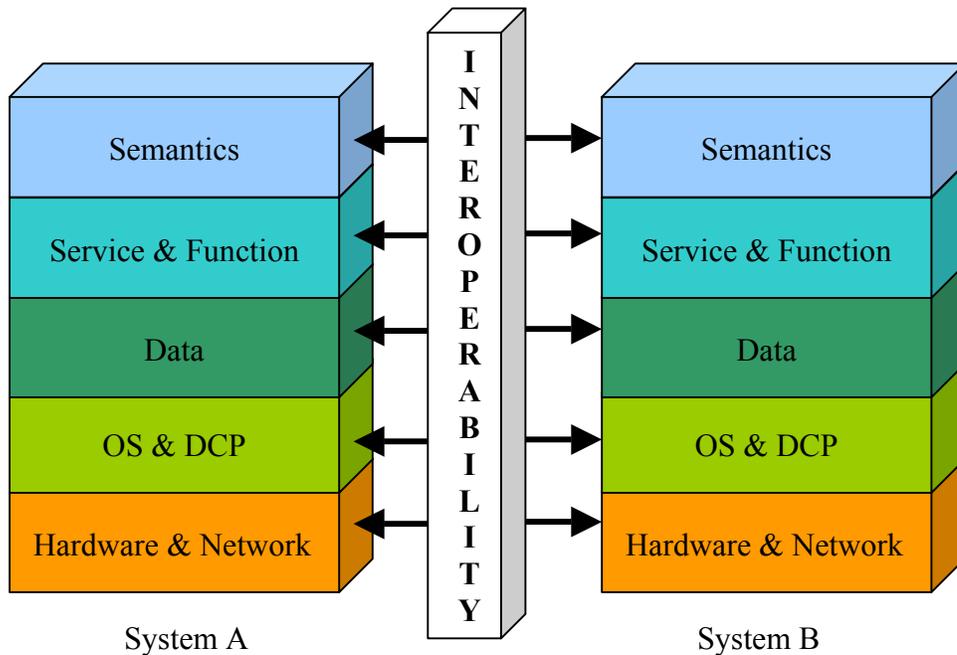


Figure 2.2 Interoperability Levels

Interoperability located at the Operation System (OS) and Distributed Computing Platform (DCP) level ensures that users can connect to a host and interact seamlessly, regardless of the operating system, and merely transfer data file between systems. A good example of this level's interoperability is FTP. The major problem of this method is that users have to possess knowledge of the operation system that runs at the remote machine in order to interact with it.

The third level of interoperability is the Data, which is composed of spatial data files and spatial databases. Users can download data file in a standard format and the system automatically identifies, and consequently converts the format to that of the user. Or the spatial database users

query and access the datasets by standard language - SQL. The response is formatted and the spatial datasets can be distributed. At the lower interoperability – Hardware & Network or OS & DCP, the peers can communicate to each other, but it doesn't mean that the applications or data files hosted on one peer can be run and opened at the other side. Many metadata standards (ISO/TC 211, FGDC), spatial data formats (such as FIPS and SDTS) and data syntax and structures are developed in GIS to achieve data sharing and exchanging. But this level's interoperability is not just a matter of file transfer or access; it includes files and directories naming, access control, access methods, and file management (Bishr, *et al.* 1999). For the spatial DBMS users, they can query the remote systems using their own, local query language, and it is also possible to analyze and display the remote datasets. However, The main weakness of this level's interoperability is that, users have to have prior knowledge of what data exist at the source, and have to search its contents using its user interface and query language. The prior knowledge includes the data's semantics and underlying data model in DBMS. Interoperability only achieves at the data level, and no interoperability for the information and knowledge (business rules) hiding in the data.

Interoperability of Functions and Services is at the fourth level. It includes the interoperability between the software components, modules or methods. Users are provided with a single open data model, that is the basis of all the upper spatial analysis functions and GIS services. For the multi-database's integration practice, the system provides a "virtual" global data model for the modules of geoprocessing. This virtual data model is only reflecting the data's relationships and structures stored in the spatial databases. The mission of this kind of data model is supporting the upper level's application logic and business logic. OGC's efforts try to address this level's

interoperability by providing a framework consisting of Open Geodata Model, Services Model and Information Community Model (Buehler & McKee, 1998). OGC's model serves the easy system integration and interoperability from the software side. Although users can access and use those functions and services seamlessly at this level, a proper knowledge on their semantics is still required. Two GISs with two different thematic views can have the same data model, but they vary in their assumption and semantics. OGC tries to provide the formal and rigorous definition of the Geometry objects and features, and one single well-defined wide-accepted framework to remove the ambiguity of the definitions of the data objects and data model, and to achieve a common semantics in different geospatial information communities. OGC's framework can make the interoperable GIS users transparently access and share distributed data and services regardless of the underlying GIS platform.

An interoperable GIS can be built on the lower four levels of interoperability mentioned above. The intention of interoperability at the fifth level is to provide seamless communication in data/information/rules/knowledge between GISs without having prior knowledge of their underlying semantics and spatial theories. The different applications may have different semantics, which stems from the fact that each application may have its own worldview and representation of the real world. An important consideration for research is that interoperating GISs need to go beyond providing interoperability for different geometries. Geometry is an integral part of GIS interoperability; however, the formation of particular geometries through shapes, orientations, and connectivities is driven in the first instance by different ways to conceptualize the world. If one considers the differences in spatial datasets that users want to integrate, one quickly realizes that they stem from different conceptualizations more often than

from the choice of different geometric data models. These conceptual issues are a significant part of the semantics of spatial information. The differentiation of semantics is a key issue in this level's interoperability (McKee, 2000). This kind of interoperability is the toughest, and there are only a few touches at current research progress in this field.

The efforts from IT (Information Technology) are involved in five interoperability levels addressed above, but they focus more on the bottom two levels. GIS researchers provide their special contributions to the upper three levels, which are application domain related. Generally, interoperability discussed in GIS context is referred to the upper three interoperability levels, we call it GIS interoperability. Interoperability mentioned in this thesis stands for the GIS interoperability.

## **2.3 Interoperability Approaches**

### **2.3.1 Overview**

All the approaches to achieve the GIS interoperability need to address a methodological and a thematic question (Vekovski, 1999):

- How is the mapping formulated? That is, using graphical symbols, mathematical formalisms, computer languages, natural language, and drawings.
- What is the content of the mapping? That is, the *story* written in the selected *language*.

Table 2.1 gave out some general interoperability approaches in the three generation systems.

From the technology and engineering viewpoint, the approaches to GIS interoperability can be

divided into three categories (Figure 2.3): open geodata, open geoprocessing and open geocomputing. For example, to increase the geodata interoperability, the XML and GML technology has been used for encoding geodata, and SVG technology has been used to display geodata; for geoprocessing and geocomputing interoperability, the software component technology is used to implement interoperability.

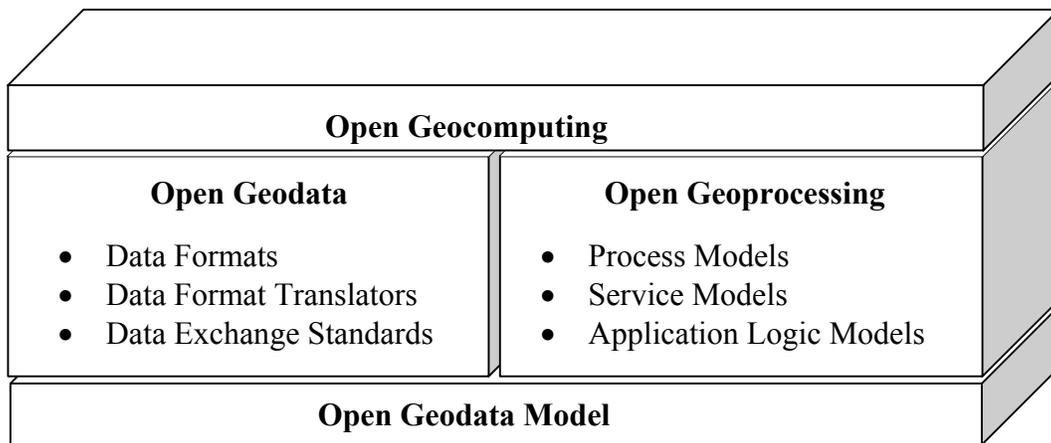


Figure 2.3 Interoperability Approach Categories

“Open Geodata Model” belongs to those three categories. In the Open Geodata category, it opens the data model in spatial databases to support the upper systems; on the other hand, in the Open Geoprocessing category, it opens the inside data model of the GIS software to the upper applications. The Open Geocomputing covers the Open Geodata and Open Geoprocessing to provide a neutral, open geocomputing environment.

In the Open Geoprocessing category, the opened Process Models allow users to access the functions in the internals of the application and organize them in desired ways. The opened Service Models make any application become a service provider, and allow other participating applications to invoke desired functions within it. The opened Application Logic Models make it

possible that one application can become a network service provider that can be invoked by users or applications distributed on the network, without taking care of the underlying data flows and process flows. Their relationships can be simplified into Figure 2.4.

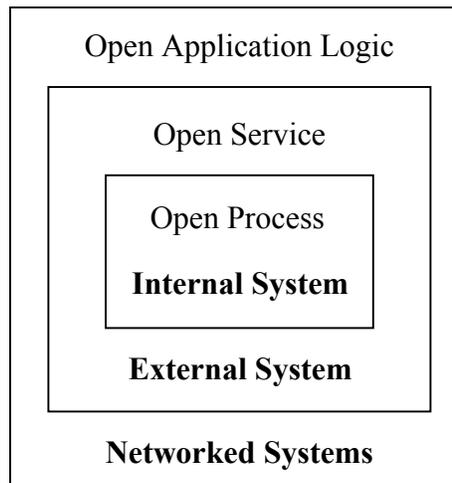


Figure 2.4 The Relationships in Open Geoprocessing

### 2.3.2 OGC's Approach

#### Strategy

#### *"Interoperability by means of specification"*

OGC's approach to interoperability is defining specifications for the mapping from real world things to abstract entities and providing implementation specifications giving concrete implementations on a specific computing infrastructure. A series of OGC's specifications form a framework to provide standards for overcoming the barriers of interoperability. The framework includes:

- A common means for digitally representing the Earth and Earth phenomena, mathematically and conceptually; and

- A common model for implementing services for access, management, manipulation, representation, and sharing of geodata between Information Communities.

The development and deployment of successful interoperability strategies requires standardization that covers many application areas, many different views and conceptualizations of the world (OGC, 1996). Standardization processes are often driven by market forces and vendors trying to position their particular technology and product as a common concept. OGC's strategy to achieve interoperability is pushing its specifications as the standards to present a common concept for the GIS community via the practices of its industrial, academic and governmental members.

## **Method**

There are many perspectives on space, which can be identified in the GIS literature, but the entity-field view and scientific-cognitive view are the two common perspectives (Leung *et al.*, 1999). The entity view sees the space in the real world as a collection of discrete entities (vector view). Entities are identifiable constructs or units that have relatively well defined boundaries and unambiguous descriptions, such as wells, rivers, or city areas. On the other hand, the field view sees the space as a continuous world (raster view) and their description is meaningful only at a particular point, for example topography, air quality or soil types. The entity-field view is the mapping mechanism between the real world and GIS world adopted by OGC to present two geographic types:

- Features – represent atomic, uni-valued geographic entities, whose location is a function of their definition, i.e. “where is what” representation; and

- Coverages – describe complex geographic phenomena in terms of maps, images, and fields, and are a function of the spatial/temporal domain, i.e. “what is where”.

In OGC specifications, each geodata object of the two geographic types is designed of a complexity, which includes three major components (Figure 1.3):

- Spatial components composed of geometries, such as points, lines, polygons, grids, and spatiotemporal referencing, defined by projections, coordinate systems, and allowable transformations;
- Semantic components defining the meaning of an object’s elements in terms of a “real-world” model, using a catalog or data dictionary; and
- Metadata components describing any additional information required interpreting an object correctly in the context of an information community.

Interoperability, in the context of the OpenGIS Specification, is software components operating reciprocally (working with each other) to overcome tedious batch conversion tasks, import/export obstacles, and distributed resource access barriers imposed by heterogeneous processing environments and heterogeneous data. OGC’s mapping mechanism is presented into the following hierarchical model to transparently support the use of multiple data and processing resources within a single workflow and session:

- The Essential (Abstract) Model – a model of the "facts" consisting of "real" objects (entities, attributes and relations) and instantaneous events. This is the codable structure of the real world as it is perceived by the specification writers.

- The Specification Model – a generic model of the software, what states it can be in and the way it responds to stimuli (events or messages) by changing state and generating responses (also events). That is, the model consists of "ideal" software objects and "ideal" events.
- The Implementation Models – Models of the software objects in specific executing software environments, and how the software objects communicate in those software environments. The models are of actual software objects, which have types, states, and properties, and communicate by sending messages. Each implementation model is hosted on a specific Distributed Computing Platform (DCP). The major identified DCPs include the Component Object Model (COM) from Microsoft, Common Object Request Broker Architecture (CORBA) from the Object Management Group, Distributed Computing Environment (DCE) from the Open Software Foundation, and Java from Sun.

### **Promotion**

The Testing Program (TP) and Interoperability Program (IP) are launched by OGC to keep all the activities following OGC's direction. The purpose of the TP is to keep the candidate products following the OpenGIS Implementation Specifications. The program provides a process for testing conformance of products to OpenGIS Implementation Specifications, and, eventually, for testing interoperability between conformant products. Once conformance testing occurs and is successfully completed, OGC will license vendors of such systems to use OGC's marks (trademarks or certification marks) that will identify to users the capability of products with respect to OpenGIS Implementation Specifications. The conforming products are listed in Table 2.3. This program composed of two phases:

- **Conformance Testing** determination that a product implementation of a particular Implementation Specification fulfills all mandatory elements as specified and that these elements are operable;
- **Interoperability Testing** determination that a product implementation of an Implementation Specification interoperates with other product implementations of the same Implementation Specification, different but related Implementation Specification(s), or within a particular computing environment.

The IP is responsible for testing the specifications' feasibility, reliability and other technical issues to improve its framework's interoperability. IP began with the Web Mapping Testbed (WMT1), in which key standards were developed that enable much easier overlay and use of maps on the World Wide Web. WMT1 set a precedent for testbeds in which competing vendors work together in rapid-prototyping projects to achieve interoperability goals set by testbed sponsors. The IP enables users and providers of geospatial technologies to cause and direct rapid and revolutionary changes in "spatial services," the broad set of Web-based information services that involve geospatial information.

### **Current Status**

OGC has released its Abstract Specifications and Implementation Specifications in Table 2.2.

Table 2.2 OGC Released Implementation Specifications

<b>Specification Name</b>	<b>Version</b>
Simple Features Implementation Specification for OLE/COM	1.1

Simple Features Implementation Specification for CORBA	1.0
Simple Features Implementation Specification for SQL	1.1
Catalog Interface Implementation Specification	1.0
Grid Coverages Implementation Specification	1.0
Coordinate Transformation Services Implementation Specification	1.0
Web Map Server Interfaces Implementation Specification	1.0
Geography Markup Language (GML) Implementation Specification	2.0

Table 2.3 OGC Conforming Products

Vendor	Product Name	Version	Level*
ESRI	Spatial Database Engine for Informix	3.0.2	SFS TF
ESRI	Spatial Database Engine for DB2 Data joiner	3.0.2	SFS TF
ESRI	Spatial Database Engine for Oracle	3.0.2	SFS NG
Oracle Corporation	Oracle8 Spatial Cartridge	8.0.5	SFS NG
Oracle Corporation	Oracle8i Spatial	8.1.5	SFS NG
Oracle Corporation	Oracle8i Spatial	8.1.6	SFS NG
Oracle Corporation	Oracle8i Spatial	8.1.7	SFS NG
Cadcorp	Cadcorp Simple Features for COM component, used in Cadcorp SIS	5.1	SFO
Cadcorp	Cadcorp Simple Features for COM component, used in Cadcorp PowerMap	1.0	SFO
Hitachi Software Engineering Co., Ltd.	HiRDB Spatial Search Plug-in	01-01	SFS
BG Hitachi Software Engineering Co., Ltd.	GeoMation GeoAdapter/J for HiRDB	01-00	SFC
Cadcorp	COM Object Library Used in Cadcorp SIS V6.0and Cadcorp PowerMap V1.0	SIS V6.0	CT OLE/ COM

\*: **SFS**: Open GIS Simple Features Specification for SQL, **SFO**: OpenGIS Simple Features Specification for OLE/COM, **SFC**: OpenGIS Simple Features Specification for CORBA, **SFS NG**: Normalized Geometry, **SFS BG**: Binary Geometry, **SFS TF**: Types and Functions

Source: OGC web site at <http://www.opengis.org/>

## 2.4 Summary

Initiated in 1960's (Coppock and Rhind, 1995), GIS software experienced several development phases and gradually evolved into the mainstream of Information Technology. The world of computing is moving towards network-centric computing. DGIS is the trend in GIS. It has been recognized that interoperability among the involved components of a DGIS system makes it possible to achieve this progress (Yuan, 2000). Interoperability has been the common concern of GIS software vendors, geodata providers and users.

The background and research progress of interoperability is overviewed in this chapter. Interoperability can be achieved in five levels. The general approaches to solve interoperability issues are summarized. OGC proposed its own solutions for interoperability issues. OGC's approach, including the strategy, method and promotion program, is introduced.

A series of OGC's specifications build up a framework for interoperability. OGC's framework is becoming standard, and is recognized and understood in GIS. The OpenGIS Simple Features define a common Geometry Data Model for OGC's framework. The results of this thesis research, implementation of OpenGIS Simple Features in Java computing platform, provide the Open Geodata Model discussed in OGC's models (see page 3) and in the Figure 2.3 (see page 24) to support the upper models or services, such as Open Geoprocessing, OpenGIS Services Model. Before reaching the design and implementation of OpenGIS Simple Feature's Java version, the research methodology in this thesis will be addressed in the following chapter firstly.

## CHAPTER 3 RESEARCH METHODOLOGY

The implementation of OpenGIS Simple Features consists of two components: design of the Implementation Specification of OpenGIS Simple Features and a real implementation based on the designed Implementation Specification. A Testing Suite, accompanied with this Implementation Specification, is developed to ensure that the real implementation follows the specification and meets the requirements defined in the specification. A Case Study, which contained the implementation, is used to demonstrate the reasonableness of the design and the performance of the implementation. Four steps are performed to reach the objectives of this thesis research.

### 3.1 Existing Work Review

Some related works done by OGC and other bodies are reviewed firstly, which provides a background and start point for this research. The analysis of existing works points out the direction for this research.

This research is only a part of OGC's framework. It must be considered in the context of OGC's specifications. OGC's Abstract Specification is the root of all Implementation Specifications, which was referenced by this research. The OpenGIS Simple Features Implementation Specifications for SQL, OLE/COM and CORBA are reviewed, which is the start point of this research. The Conformance Testing Suites for SQL and OLE/COM released by OGC are the templates for the developing of Java version's Conformance Testing Suite.

There is an Internet forum discussing Java based Simple Feature Implementation Specification issues (<http://groups.yahoo.com/group/sf4java/>, last check on May 25, 2001). Some ideas from this discussing forum are referenced in the design.

Some bodies from industry and academy have done some efforts on the implementation of OpenGIS Simple Features in Java computing platform. The works from the alliance of Oracle and MapInfo, Computer Aided Development Corporation Ltd. (Cadcorp), and the University of Bonn are analyzed in this research. Some weakness in their design has been pointed out and overcome in our design.

### **3.2 Design And Implementation**

This section is divided into two parts: the design of the Implementation Specification for Java and the implementation of the design. The Implementation Specification is the guideline for GIS software developing activities. The quality of the Implementation Specification is very important for the implementation. To keep the design at high quality, two kinds of criteria are created as follows:

- Design Criteria. All the design procedures follow the requirements of the design criteria.
- Classification Criteria. The classification method of geometry objects is the key point in the design of the implementation specification. A more reasonable classification criteria are created to organize different kinds of geometry objects into the new geometry data model.

Unified Modeling Language (UML) is the communication standard of visual modeling. The UML provides a smooth transition between the business domain and the computer domain. The UML makes it possible for all members of a design team to work with a common vocabulary, minimizing miscommunication and increasing efficiency. Visualizing the components and relationships of a complex system make it far easier to understand than describing it in words.

The system design tool Rational Rose 2000, built on UML technology, is used in all the design procedures to improve the productivity and standardize the efforts. It uses static and dynamic views of a logical model and a physical model to capture the in-process products of object-oriented analysis and design. The embedded notation creates and refines these views, and represents each kind of model element and relationship in graphical icons. It visualizes and manipulates the model's elements and properties by diagrams and specifications. Two tasks were done in Rational Rose environment:

- The Implementation Specification of OpenGIS Simple Features for Java is designed in diagrams. The Implementation Specification's input/output interfaces, and the definitions, relationships and properties of elements, such as use cases, objects, classes and components are translated into a code framework for implementation by this tool.
- Some existing works, such as Oracle/MapInfo, Cadcorp and Bonn, are converted into diagrammatic models by the Reverse Engineering tool. The diagrams facilitate the comparison and analysis of the referred models.

Borland's Jbuilder 4 is a comprehensive group of highly productive tools for creating scalable, high-performance, platform-independent applications using the Java programming language.

Jbuilder 4 is a 100% Pure Java integrated development environment (IDE), which supports the JDK 1.3 technologies. Any program written in Java can be run, debugged, and worked within Jbuilder 4 environment. The prototype Geometry Data Model, Testing Suite and Case Study in this thesis are developed and tested in this software developing tool.

### **3.3 Testing Suite**

The Conformance Testing Suite ensures that each implementation follows the requirements of the Implementation Specification it based on. OGC has released two Conformance Testing Suites for SQL and OLE/COM. This Java version Conformance Testing Suite refers to the two released suites. The mandatory and optional methods defined in the Geometry Data Model are organized into two categories and tested separately in this suite. The OGC's standard testing dataset is embedded into this suite to keep the consistent in testing data with OGC's released suites.

This testing suite is designed in Rational Rose 2000 and is implemented in Jbuilder 4. This testing suite is implemented in a Java Applet to facilitate distribution. This testing suite can be easily initiated in general Java enabled browser, and the testing results are inherently distributed with the testing suite.

### **3.4 Case Study**

A GIS-based Geotechnical Data Sharing and Analysis system is developed in the Case Study section. This system's client viewer is changed from the Java GIS software GeoEye™ 1.0 (Yuan, 2000). The old data model in GeoEye™ 1.0 is replaced by the prototype Geometry Data Model implemented from the Java version Implementation Specification of OpenGIS Simple Features. Some new functions, such as the Profile Analysis, are developed in the system. This system's server, GeoServNet, adopts the same Geometry Data Model as that of the client. GeoServNet is designed in Rational Rose 2000 and implemented in Jbuilder 4. The design and implementation in this thesis is tested in this system.

## CHAPTER 4 EXISTING WORK REVIEW

### 4.1 OGC's Implementation Specifications

The OGC has released three Implementation Specifications on different DCPs titled: The OpenGIS Simple Features Specification for

- OLE /COM
- CORBA
- SQL

The first version for the three implementation specifications was released in 1996. Many companies implemented the SQL or OLE/COM specification within their products (see Table 2.3). The OLE/COM and SQL specifications were revised in 1999 to respond to improvements in technology, and their conformance testing suites were redeveloped and made freely available. Due to the fact that the implementation of CORBA is complicated, compared to the SQL or OLE/COM versions, there are fewer companies trying to implement this specification. The released CORBA specification is still the first version so far, and the testing suite for CORBA specification continues to undergo developing.

OGC's Implementation Specifications are valuable references for this thesis's work. There are a few differences in the underlying Geometry Data Model between the three released versions illustrated in Figures 4.1, 4.2 and 4.3. The entities *Ring*, *LinearPolygon*, *MultiLinearPolygon*, *Line*, and the relationship between the *LinearRing*, *LineString* and *Ring* are not consistent in the models.

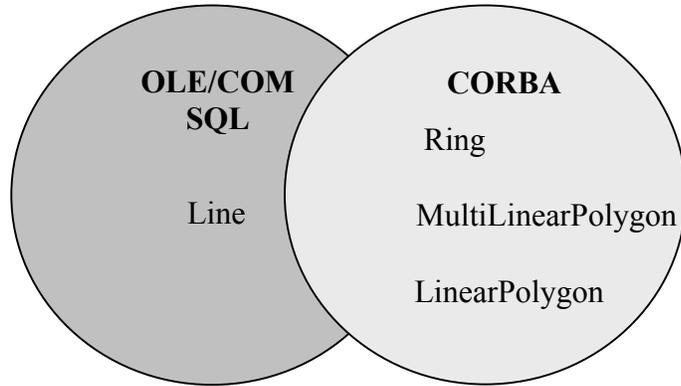


Figure 4.1 The Difference in the Geometry Model of SQL, OLE/COM and CORBA Implementation Specifications

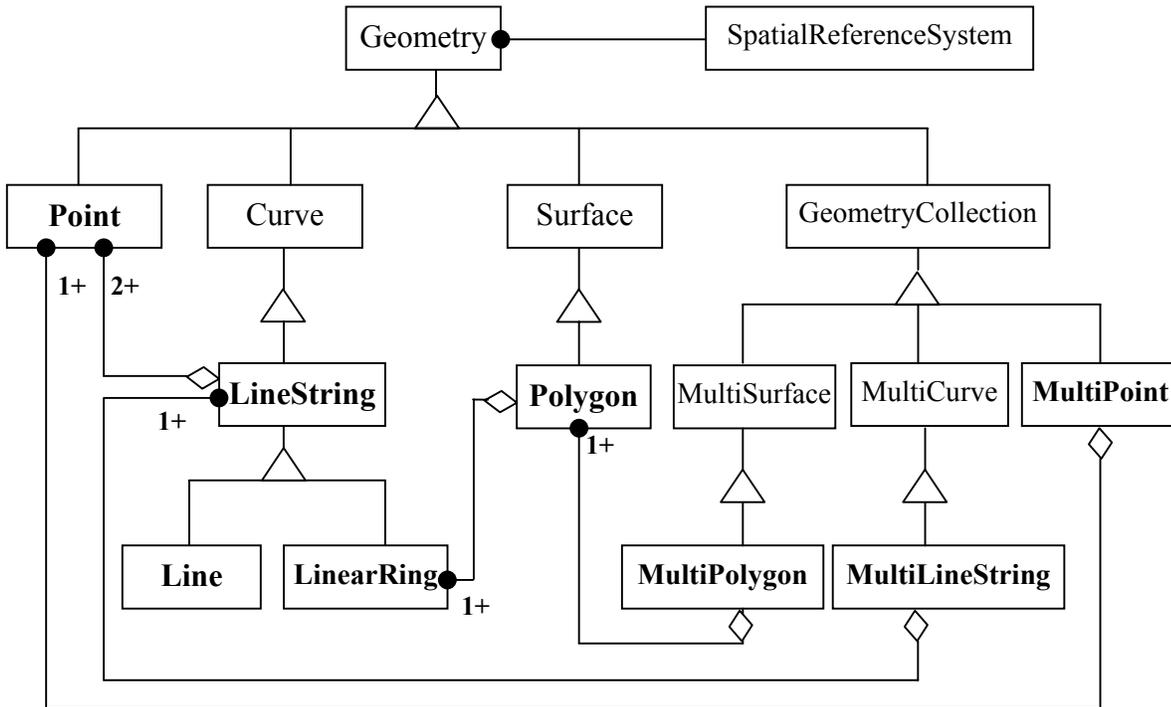


Figure 4.2 Geometry Hierarchy in OpenGIS Implementation Specification of OLE/COM and SQL

(See Appendix for Data Model Notation)

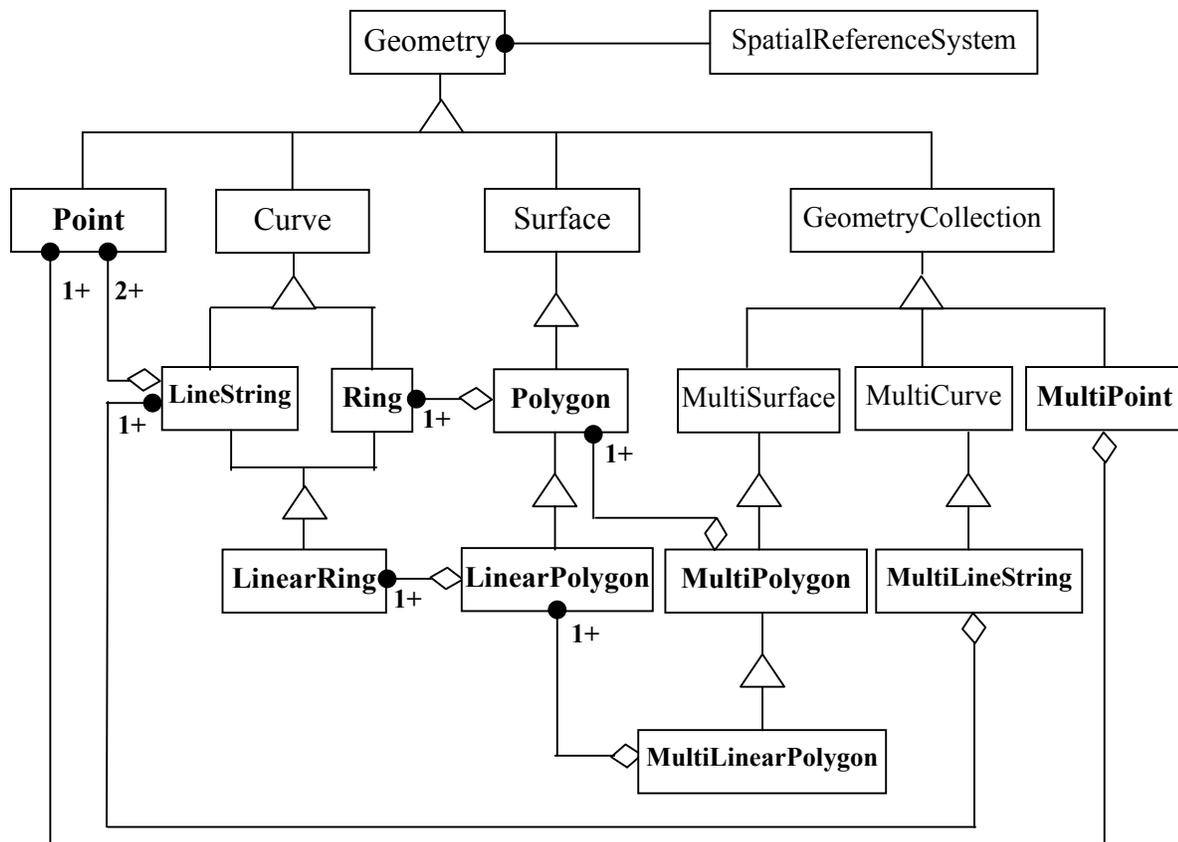


Figure 4.3 Geometry Hierarchy in OpenGIS Implementation Specification of CORBA

*(See Appendix for Data Model Notation)*

## 4.2 Other Existing References

The Oracle/MapInfo alliance, Computer Aided Development Corporation Ltd. (Cadorp), and the University of Bonn have done some efforts on the Java implementation of OpenGIS Simple Features from academic and industrial perspectives.

### Oracle/MapInfo

Oracle made many efforts to extend its database's capability to manage spatial datasets in RDBMS and ORDBMS by providing a spatial extension (Spatial Cartridge<sup>®</sup> in Oracle 8<sup>®</sup>, Spatial<sup>®</sup> in Oracle 8i<sup>®</sup> and Oracle 9i<sup>®</sup>). Oracle has integrated the implementation of OpenGIS

Simple Features Specification for SQL at Normalized Geometry level (SFS NG, see Table 2.3) into the products Oracle 8, 8i and 9i. MapInfo and Oracle entered into a relationship to develop a Java based version of OpenGIS SF Implementation Specification (see Figure 4.5 for Geometry Data Model). Unfortunately, the relationship has since dissolved.

The Geometry Data Model contained in Oracle and MapInfo's efforts followed the specification for SQL. The entities *CurveString*, *CurvePolygon*, *MultiCurveString* and *MultiCurvePolygon* were added into OGC's SQL model to represent the non-linearly interpolated entities in the real world. Currently, most of the commercial GISs only have the ability to process linearly interpolated entities. In fact, many entities are non-linear or compounds of linear and non-linear entities in the real world. It is reasonable that the linear entities are simplified and specialized from non-linear entities in the hierarchy of the proposed model.

Unfortunately, the presentation, algorithms and processing for linear and non-linear entities are quite different in software development. The presentation of objects *Arc* and *Line*, as an instance, is different: *Line* objects only record the (x, y) value of the start point and end point, but for the *Arc* object, the parameter, curvature, needs to be recorded. In addition, the complexity is quite different: the non-linear entities are expressed by high order equations, and the linear entities are expressed by first order equations. From a software engineering viewpoint, the relationships between the *CurveXXX* entities and their sub-entities are not suitable for software development.

## **Cadcorp**

Cadcorp, one of the principal members of OGC, has undertaken many efforts in the implementation of OpenGIS Simple Features Implementation Specification for OLE/COM (see Table 2.3). Cadcorp utilizes the component technology to implement SF, which is a different approach when compared with other organizations whose products have passed OGC's Conformance Testing.

In Cadcorp's Java based SF implementation (see Figure 4.6), the *MultiGeometry* is unnecessary in the hierarchy of the geometry data model. The existing entity *GeometryCollection* can contain all the other kinds of geometry types. Its sub-entities are restricted to specific kinds of entities, such as *MultiPoint*, which only contain *Point* entities.

Some names of entities in this model are not consistent with OGC's released Implementation Specifications. For example, *MultiCurve* and *MultiSurface* are replaced by *CurveCollection* and *SurfaceCollection*, respectively. The entity *LinearCurve* is added in this model to narrow the underlying entities to linear curve objects. It makes sense.

### **University of Bonn**

The GIS & Remote Sensing Research Unit at the Department of Geography in the University of Bonn has implemented a Java implementation of OpenGIS Simple Features. They developed a slightly modified Java implementation of the OGC Simple Features for CORBA Specification (see Figure 4.7). Information and source code of the software can be found at <http://katla.giub.uni-bonn.de/sfjava/>.

There are two changes in Bonn's model compared with CORBA Implementation Specification (OGC, 1998): (1) The CORBA Implementation Specification defines the client and server side implementation for each entity, but Bonn's model only defines the entity itself; (2) The *LinearRing* entity is a child of *LineString* and *Ring*, but it can only inherit from *LineString* during the model implementation because Java language has the single inheritance restricting at Class level. That means, *LinearRing* can not inherit any characteristics from object *Ring*.

In Bonn's model, the naming system of the methods followed the CORBA Implementation Specification. For a Java version, paralleled with the other three versions, it should follow Java's conventions during naming the methods. The big issue in this model is the logical problem among *LineString*, *Ring* and *LinearRing*. If the classification rule applied on *Curve*'s sub-entities, **linear** and **non-linear**, the *LinearRing* can only be a sub-entity of *Ring*; If the classification rule applied on *Curve*'s sub-entities **closed** and **non-closed**, the *LinearRing* can only be a sub-entity of *LineString* (see Figure 4.4).

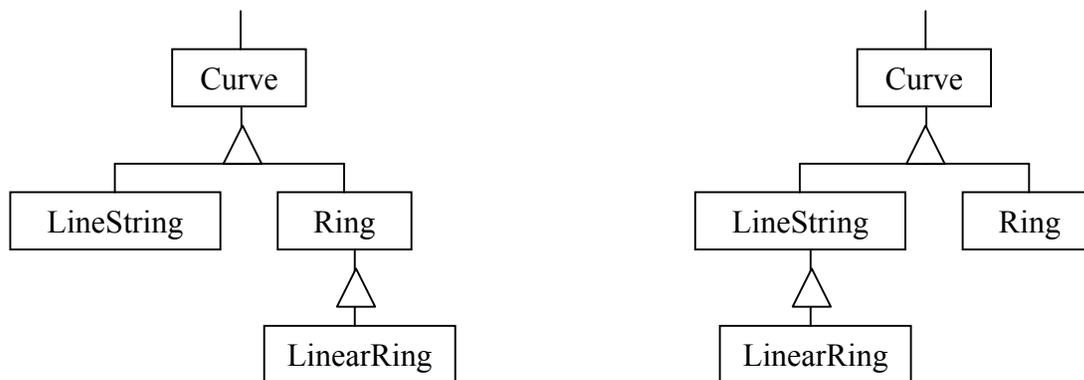


Figure 4.4 Comparison of the Classification under Different Rules

(left: linear and non-linear classification rule; right: closed and non-closed classification rule)

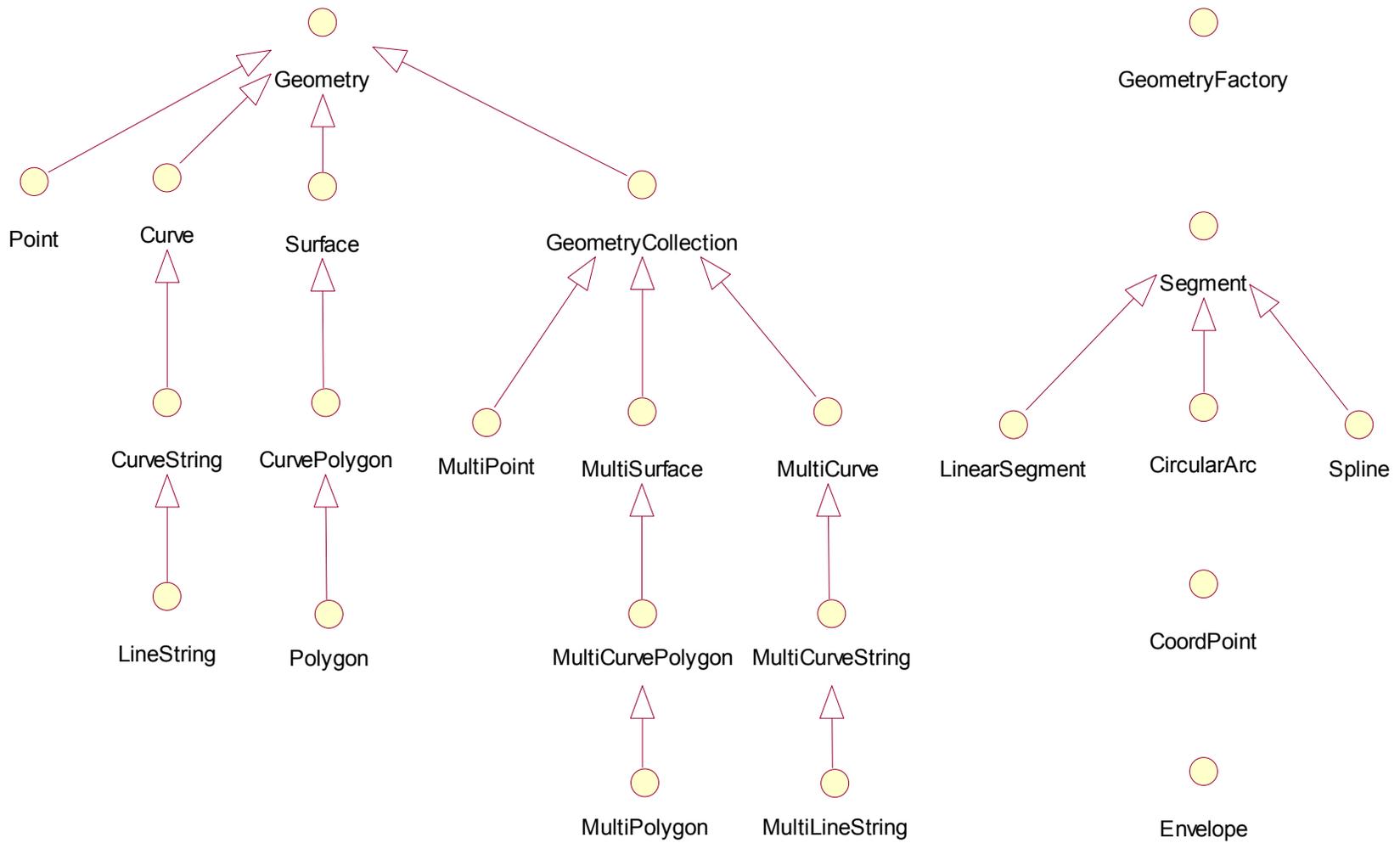


Figure 4.5 Geometry Interface Hierarchy in the Proposed Data Model from Oracle and MapInfo  
*(See Appendix for Data Model Notation)*

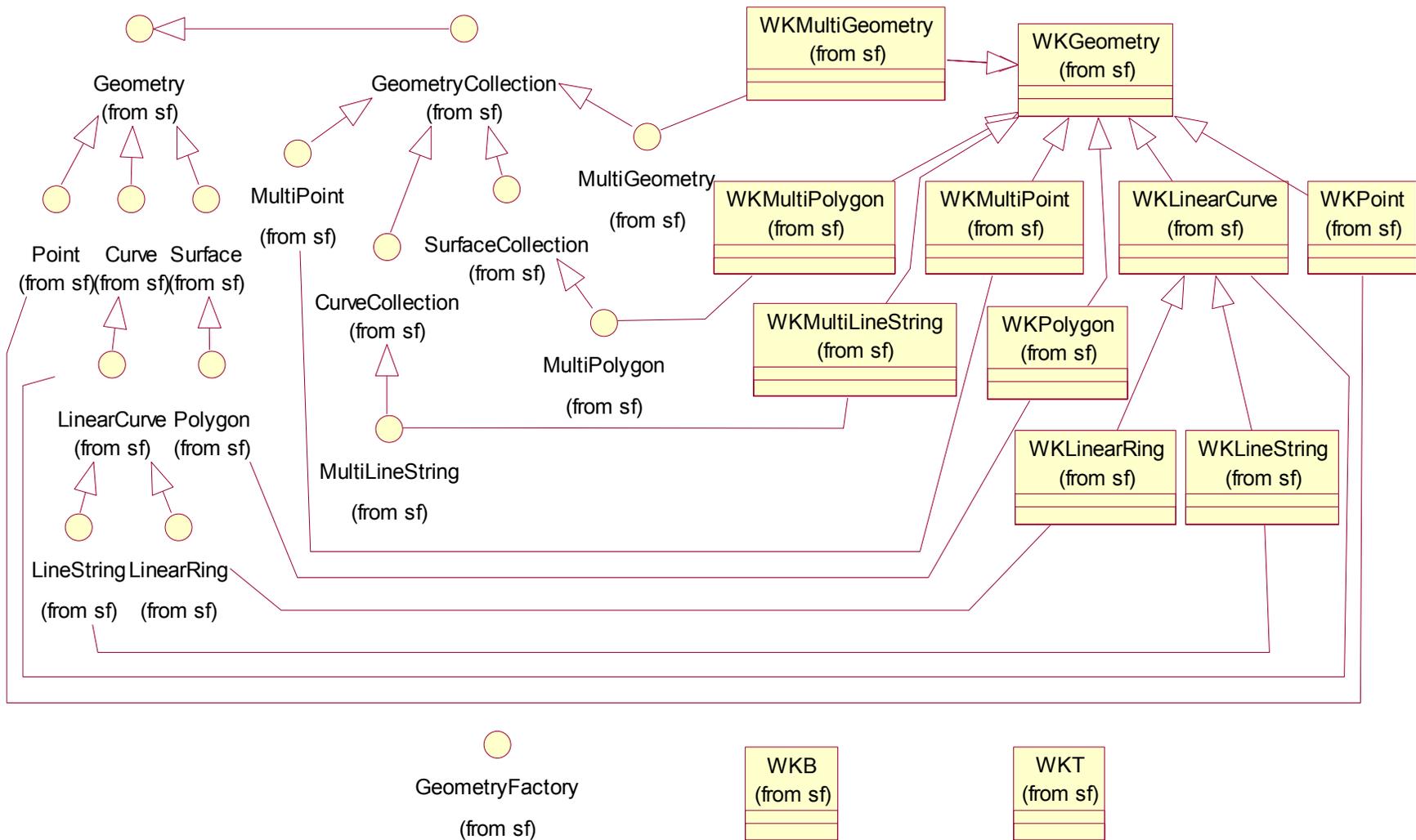


Figure 4.6 Geometry Interfaces and Objects Hierarchy in Cadcorp's Data Model

(See Appendix for Data Model Notation)

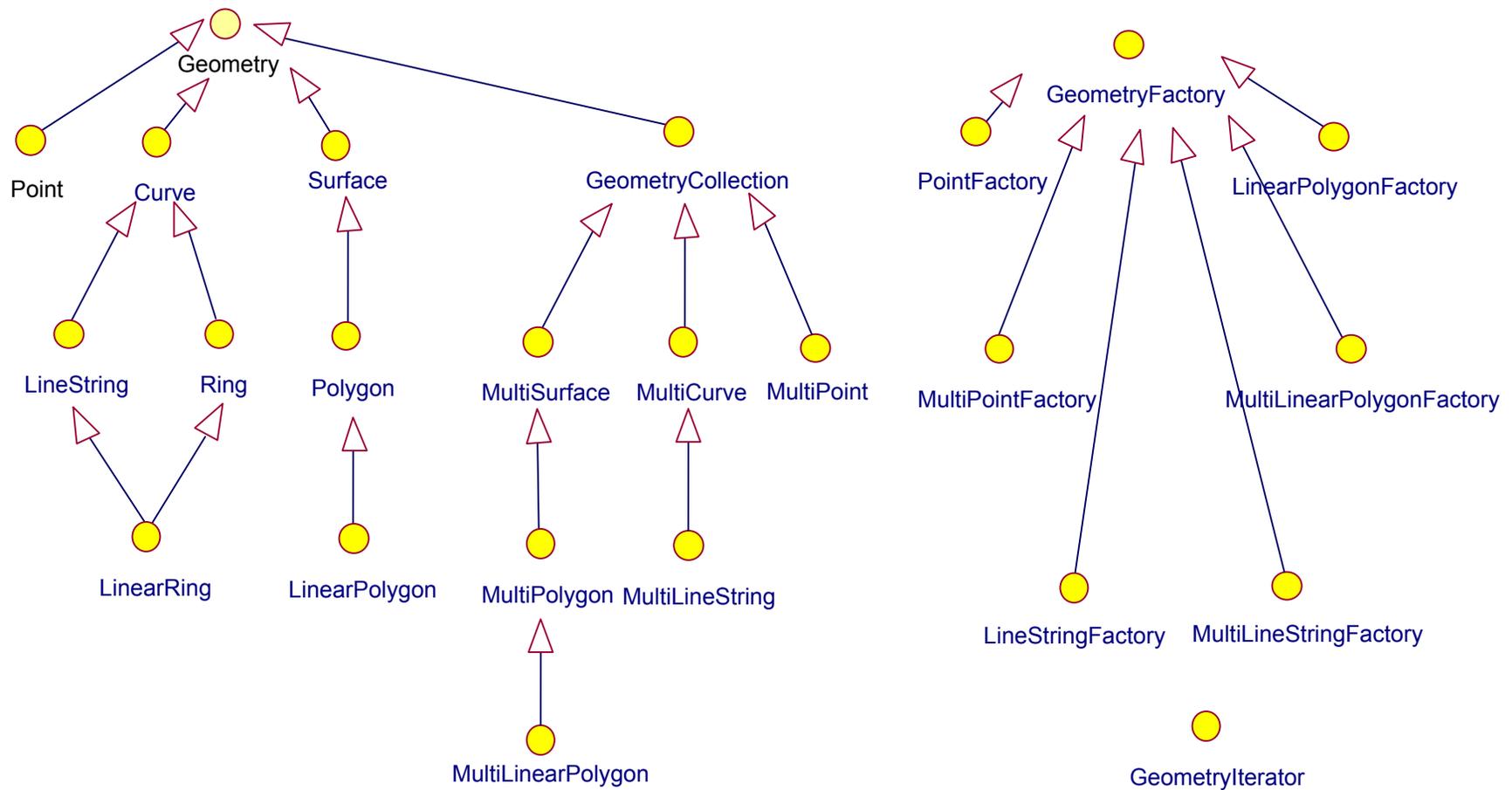


Figure 4.7 Geometry Interfaces and Methods Hierarchy in Bonn's Data Model

*(See Appendix for Data Model Notation)*

## CHAPTER 5 JAVA IMPLEMENTATION

Some weakness and limitations of the existing works analyzed above. The purpose of the new design of the Geometry Data Model is overcome those weakness. The details of the design and implementation of the new model will be addressed in this chapter.

### 5.1 General Logical Model Design

The logical model design defines a Java standard for OpenGIS Simple Features Implementation Specification. This design is based on, but not limited to the CORBA Implementation Specification, it also refers to OLE/COM and SQL specifications and other existing Java practices, especially Bonn's efforts. The following section will discuss the design from two aspects: Design Criteria and Geometry Entity Design.

#### 5.1.1 Design Criteria

Three rules have been applied in this logical model design: Extensibility, Effectivity and Implementability, and Java Standard.

*Extensibility* means this data model can be easily expanded. That is, it supports adding of new objects, which are not yet recognized, mentioned or implemented. Extensibility makes the data model active as time elapse.

*Implementability* ensures that the designed data model is an efficient software development guideline. Implementability is a pre-required criterion in the model design. *Effectivity* is a higher criterion than implementability. The data model can improve the algorithms' executable efficiency, and speed the code's transfer in the network environment from one site to another. The data storage strategy, for example, needs to be considered during the design stage. Generally, the volume of geodata is very large. If the data and their operation methods are strictly encapsulated together in one class, the size of higher level classes, which are composed of the lower classes, could be very large, because everything contained in the lower classes, including the data and methods, are encapsulated into the higher level classes.

**Java Standard** refers to Java as a widely accepted programming language and DCP in the IT community. Java has its own convention and rules, and therefore forms its own style. As a Java implementation, the new data model must follow existing Java standards. The name of a method, for example, is composed of a verb plus a noun. The Interface and Class structure can improve the system's extensibility.

The other criteria, such as expressivity and simplicity, and unambiguity and completeness, have been applied at the definition level of the data model in OpenGIS' Implementation Specification.

### **5.1.2 Geometry Entity Design**

Geometry Entity Design is the essential part in this logical model. The geometry model is built up from the ground: geometry entities. The major geometry entities in GIS have been well defined and discussed in OGC's documents and specifications. Our design focuses on how to organize geometry entities and their relationships.

### *Structure Elements*

The concepts of Package, Interface and Class are applied to organize and structure geometry entities into the design and implementation in this thesis. Interface and Class are used to represent the abstract and solid geometry entities. The Package is used to organize the interfaces and classes in this model.

Interfaces map the OpenGIS SF specification, and Classes implement the Interfaces (specification). One interface abstractly defines one geometry entity and its capability. The structure of interfaces represents the Geometry Data Model in a neutral manner. Each class refers to its interface, and implementation of the geometry entity's capability. The relationship between interface and class is one to one or many, that means, one interface can have several different implementations (classes), and different developers can implement the same interface into different classes. This approach is an effective solution for OGC's goal: providing a standard and unifying view of the real world.

In a Java environment, one interface can inherit many other interfaces to overcome the limitation that one Java class is only permitted to inherit one other class (see Figure 5.1).

However, the interface's multi-inheritance only transfers the variables from several super interfaces into their sub-interface, that means, some features of the geometry entity are transferred and the activities of the geometry entity are left. Java's multiple inheritance is different with other object-oriented languages, such as C++. Java's multiple inheritance can not keep the consistency between the super-objects and the sub-objects in activities. The multiple inheritance rule has lost its meaning in this geometry data model and can be discarded during the geometry entity design.

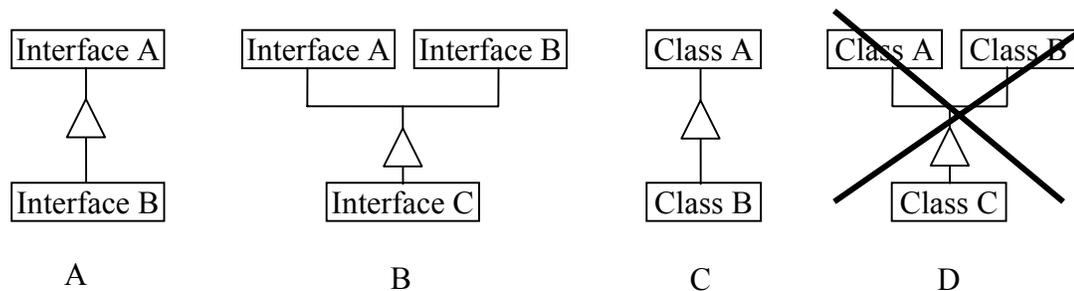


Figure 5.1 Inheritance relationship of Interface and Class in Java (D not permitted)

*(See Appendix for Data Model Notation)*

The closed interfaces or classes in relationships can be grouped together to form a package, which is like the class library in other OO languages. Packages make the structure clear and easily manageable. Following Java's convention, there are two packages *org.opengis.sf.geometry* and *com.uc.geoservnet.geometry* used in this design. Package *org.opengis.sf.geometry* contains all the geometry interfaces defined by OGC. The implementation (classes) of the geometry interfaces are organized within *com.uc.geoservnet.geometry*, which stands for one implementation coming from the GeoServNet system (geoservnet) at the University of Calgary (uc).

### *Naming System*

Java is a free-form programming language, but the code conventions are important to programmers for a number of reasons (Sun, 1999):

- 80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the original author.
- Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
- If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.

Naming systems are composed of a series of naming rules, which applied on the package name, interface name, class name, variable name and method name. These rules are borrowed from the Java environment to facilitate coding for programmers, especially for the programmer in a team environment. The following style (see Table 5.1) is used by Sun itself in virtually all of the code, and seems to be supported by most Java development environments. The Java conversion naming system is used in the design and implementation in this thesis.

Table 5.1 Java's Naming Conventions (Sun, 1999)

<b>Type</b>	<b>Rules for Naming</b>	<b>Examples</b>
Packages	The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names, currently	com.sun.eng com.apple.quicktime.v2 cdu.cmu.cs.bovik.chees

	com, edu, gov, mil, net, org, or one of the English two-letter codes identifying countries as specified in ISO Standard 3166, 1981. Subsequent components of the package name vary according to an organization's own internal naming conventions.	
Classes	Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).	class Raster; class ImageSprite;
Interfaces	Interface names should be capitalized like class names.	interface RasterDelega; interface Storing;
Methods	Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.	run(); runFast(); getBackground();
Variables	Variable names should not start with underscore _ or dollar sign \$ characters, even though both are allowed. Variable names should be short yet meaningful. The choice of a variable name should be mnemonic - that is, designed to indicate to the casual observer the intent of its use. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.	int i; char c; float myWidth;
Constants	The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by under-scores ("_"). (ANSI constants should be avoided, for ease of debugging.)	static final int MIN_WIDTH = 4; static final int MAX_WIDTH = 999;

To help a programmer easily distinguish if the Java file in this design is an interface or a class, all the name of interfaces is prefixed with “I” as in Sun’s Interface naming convention. The remainder files without the prefix “I” are classes. For example, *IPoint* stands for an interface, which defines the features and activities of a point object. *Point* is a class, which defines and implements the features and activities of a point object. If *Point* class implements the interface *IPoint*, it inherits the features and empty activities from interface *IPoint*, and the activities are implemented.

In fact, there exists mixed geometry entities. The *ArcString*, for example, is composed of linear strings and non-linear arcs. The name of a mixed geometry entity is composed of each kind of element’s name regardless of how many kinds of elements it contains. The name of the mixed geometry entity is a sequence of names of all contained elements started from the main part.

Paul Daisey noticed that there are some differences in naming the methods in OGC released Implementation Specifications, and posted a list of the difference to the discussing forum (<http://groups.yahoo.com/group/sf4java/message/23>) in Apr 12, 1999. The Java’s code conventions for naming methods were adopted in this design.

### **5.1.3 Geometry Entity Classification Criteria**

The purpose of the Geometry Data Model design is to assemble the geometry entities defined in OpenGIS’ specifications. It is very difficult to enumerate all the geometry

entities in GIS and other relative information bodies, like CAD, and there exists semantic differences in some geometry entities. In this case, OGC only provides a framework of the 2D open geodata model and leave the model's extensibility to users and future improvements. In this thesis, the defined geometry entities match OGC's framework and the primary geometry entities are listed. The following criteria are used to map data types into the framework by order:

Rule 1: Classifying by spatial extension dimension: zero dimension, one dimension, two dimension and collection entity;

Rule 2: Classifying by linear, non-linear and complex entity; and

Rule 3: Classifying by close and open property.

## **5.2 Geometry Data Model Design**

The Geometry Data Model is the kernel part or basic part for GIS software or geospatial related information systems. The real-world entities and phenomena are mapped into the computer system by this Geometry Data Model. The quality of a Geometry Data Model defines the fate of its upper software. At the conceptual level, the Geometry Data Model plays the roles described in Figure 5.2.

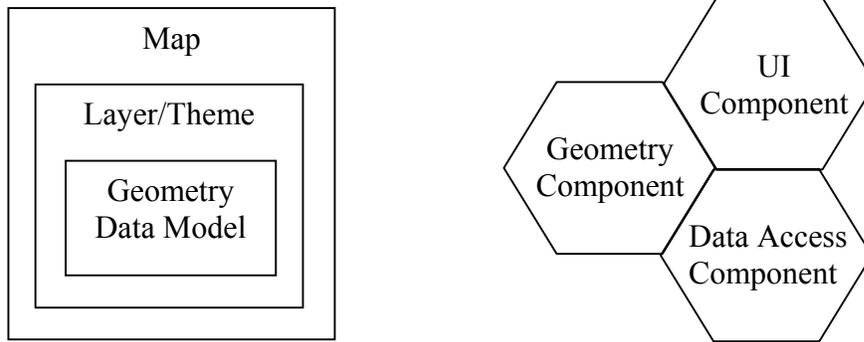


Figure 5.2 The Possible Roles of Geometry in Conceptual Model

(left: GIS software; right: as “Plug-and-Play” component in Information System)

### 5.2.1 Abstract Geometry Data Model Design

Based on the design criteria and referred to the existing works discussed above, the designed Abstract Geometry Data Model is shown in Figure 5.3.

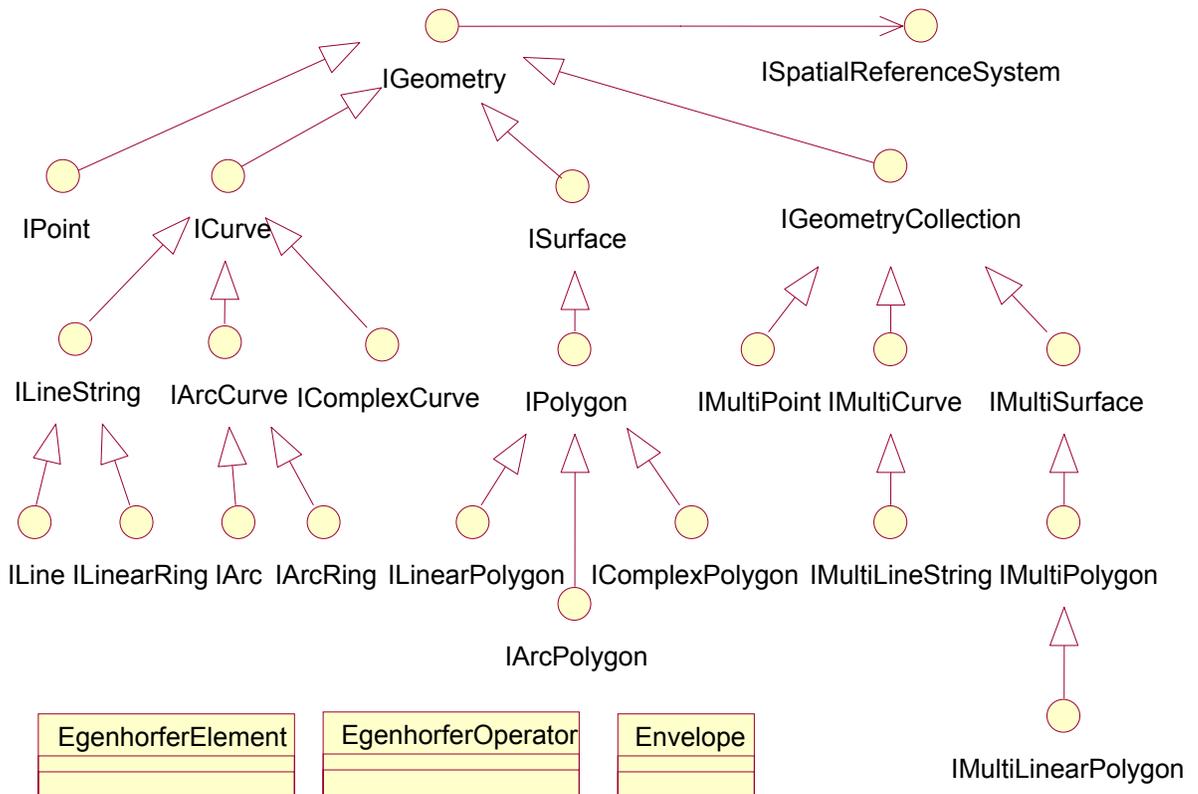


Figure 5.3 Abstract Geometry Data Model in Interfaces

*IGeometry* is the parent interface in this model. The *IPoint*, *ICurve*, *ISurface* and *IGeometryCollection* follow Rule 1 of the Geometry Entity Classification and located at the second level in the framework, which represent the zero dimension, one dimension, two dimension and compound entity, respectively. The underlying geometry entities inherited from them belong to those four categories classified by spatial extension.

Classification Rule 2 is applied on the geometry entities at the third level. The entities are classified into linear, non-linear and complex categories. The *Curve*'s sub-types, for example, are ordered into the *LineString* (linear entity), *ArcCurve* (non-linear entity) and *ComplexCurve* (complex entity). The entity *ArcString* composed of *Arc* and *LineString* elements belongs to the *ComplexCurve* category.

Classification Rule 3 is applied to the sub-types of the geometry entities applied Rule 2. The *ArcCurve*'s sub-types, for example, are classified into close entity *ArcRing* and open entity *Arc*. The *Line* and *LinearRing* are used the same rule.

### **5.2.2 Geometry Data Model**

Based on the Abstract Geometry Data Model, the designed Geometry Data Model is illustrated in Figure 5.4. The variables for each geometry entity are listed out. The methods in this model will be analyzed in the section: Geometry Implementation.



### 5.2.3 Factory Design

There exist many formatted geospatial datasets in GIS. Those widely used formats are called Well-Known Structure (WKS). The constructor methods in geometry object classes can create instances for each geometry object defined in this model from the WKS directly. But this approach is not convenient for WKS datasets in many cases. To facilitate the users to create instances of each geometry object from the WKS datasets, this data model provides a series of factories, which can directly create objects from the datasets by hiding the details of creating objects and setting their features. The factories provided in this data model are listed in Figure 5.5.

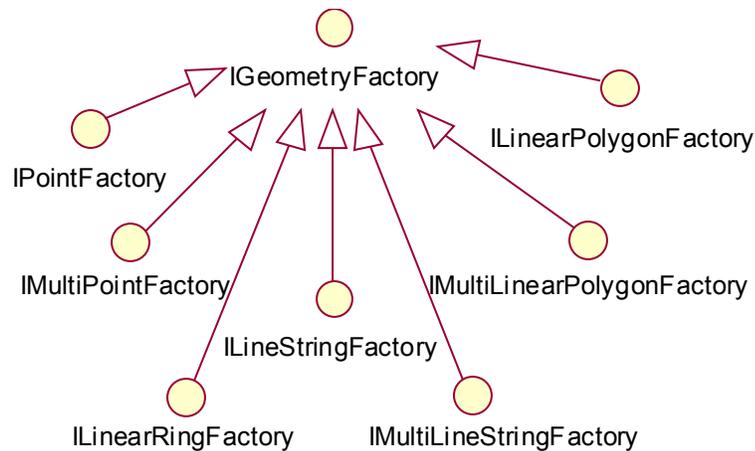


Figure 5.5 Factory Hierarchy in Data Model

### 5.3 Geometry Implementation

The implementation of geometry is translating the design patterns into real Java code. The implementation is composed of two parts: implementation of the interfaces, and implementation of the objects, which completes the methods of the inherited interfaces. In fact, the

implementation of the interfaces is very simple with the help of the design tool, Rational Rose. This tool can automatically generate the code frame for the interfaces and the classes. The methods are empty in the generated classes. Most of this thesis' work is designing the algorithms and completing those methods. The following section will discuss two aspects of the implementation: functions and data storage.

### **5.3.1 Functions**

The functions addressed here are the system's spatial operations. Generally, operators over more than two operands are typically broken down into a sequence of binary operations. There is no consensus on a canonical set of spatial operators (Egenhofer *et al* 1999). Different applications use different operators, although some operators (such as intersection) are more common than others. Spatial operators can be classified in several ways, reflecting fundamentally different perspectives and objectives. A common distinction is based on different geometric properties of spatial relations, leading to groups of topological, directional, and metric relations. Topological relations are invariant under topological transformations, such as translation, rotation, and scaling (Egenhofer & Franzosa, 1991). Direction relations refer to the location of two spatial objects with respect to a reference frame (Peuquet & Zhan 1987, Retz-Schmidt 1988, Frank 1991), yielding quantitative values (e.g. 32°12') or qualitative values (e.g. south and north-west, or left and right). Metric relations (Peuquet 1992, Hernandez *et at* 1995) capture distances, either quantitatively (e.g. 71.4 km) or qualitatively (e.g. near or far).

In the three existing Implementation Specifications (SQL, OLE/COM and CORBA), OGC has defined the spatial operations based on the state-of-the-art research results of this topic. The rigorous definitions for each function, including input and output arguments, can be found in the documents at OGC's web site <http://www.opengis.org/>. Those functions defined in this design refer to the existing specifications and are divided into six categories shown in Table 5.2. The six kinds of functions are located at the root in the Geometry Data Model (*IGeometry/Geometry*), that means, all the geometry objects in the model inherit those functions. From another perspective, those functions can be divided into two categories, mandatory functions and optional functions, which shown in OGC's conformance testing documents.

Most of the operations in Table 5.2 are the research interests in the discipline of Computational Geometry. There are many effective algorithms for those operations in the computational geometry literature (Berg *et al* 1997, Goodman and O'Rourke 1997). Listing all the implementation details of the spatial operations in this thesis will confuse you and make you lost your focus. We, therefore, only discuss the main implementations for relational operators, set operators and constructive operators described in Table 5.2 (the XXX stands for all the implementable geometry objects: *Point*, *LineString*, *Polygon*, *GeometryCollection*, *MultiPoint*, *MultiLineString* and *MultiPolygon*).

Table 5.2 Basic Functions in Geometry Data Model

<b>Kind</b>	<b>Function</b>	<b>Category</b>
Geometry creation	XXX createFromXXX(xxx); XXX createFromWKSXXX(wksxxx, srs); XXX createFromWKBXXX(srs, byte[]);	Mandatory
Geometric characteristics	Boolean isEmpty(); boolean isSimple(); boolean isClosed();	Mandatory
WKS operators	WKSGeometry export();	Mandatory
Constructive operators	Geometry copy(); Geometry boundary(); Geometry buffer (double distance); Geometry convexHull();	Optional (except Copy())
Metric operators	Double distance (Geometry other);	Optional
Set operators	Geometry intersection (Geometry other); Geometry union (Geometry other); Geometry difference (Geometry other); Geometry symmetricDifference (Geometry other);	Optional
Relational operators	Boolean equals (Geometry other); boolean touches (Geometry other); boolean contains (Geometry other); boolean within (Geometry other); boolean disjoint (Geometry other); boolean crosses (Geometry other); boolean overlaps (Geometry other); boolean intersects (Geometry other); boolean relate (Geometry other, EgenhoferOperator operator);	Optional

## Relational Operations

There are nine kinds of relational operations belonging to the optional functions in the Table 4.3. The definitions of those operations are formulized by OGC in mathematical format. Those definitions are based on the research results of Clementini, Di Felice, Egenhofer, *et al.* (Clementini, 1993, 1994, 1995, 1996 and Egenhofer, 1990, 1991, 1993), who created a widely accepted and rigorous theoretical system for spatial operations.

The relational operations deal with the topological relationships that are preserved under such transformations as rotation, scaling, and rubber sheeting. The model for binary topological relations used in this thesis is based on the usual concepts of point-set topology with open and closed sets. The binary topological relation between two objects,  $a$  and  $b$ , in Euclidean space  $E^d$  is based upon the intersection of  $a$ 's interior  $I(a)$ , boundary  $B(a)$ , and exterior  $E(a)$  with  $b$ 's interior  $I(b)$ , boundary  $B(b)$ , and exterior  $E(b)$ . The intersection of any two of interior, boundary and exterior relations can result in a set of geometries,  $x$ , of mixed dimension. For example, the intersection of the boundaries of two polygons may consist of a point and a line. Let  $dim(x)$  return the maximum dimension (-1, 0, 1, or 2) of the geometries in  $x$ , with a numeric value of -1 corresponding to  $dim(\emptyset)$ . The nine intersections between the six object parts describe a topological relation and can be concisely represented by a 3x3-matrix  $\mathfrak{I}_9$ , called the 9-intersection matrix (DE-9IM).

$$\mathfrak{I}_9(a, b) = \begin{pmatrix} I(a) \cap I(b) & I(a) \cap B(b) & I(a) \cap E(b) \\ B(a) \cap I(b) & B(a) \cap B(b) & B(a) \cap E(b) \\ E(a) \cap I(b) & E(a) \cap B(b) & E(a) \cap E(b) \end{pmatrix} \quad (1)$$

Figure 5.6 shows an example DE-9IM for the case where  $a$  and  $b$  are two polygons that overlap.

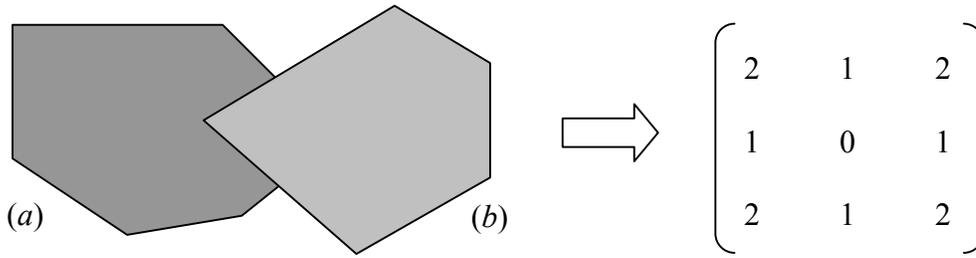


Figure 5.6 Example Instance of DE-9IM

A pattern matrix, stands for the DE-9IM, consists of a set of 9 pattern-values, one for each cell in the matrix. The possible pattern-values  $p$  are  $\{T, F, *, 0, 1, 2\}$  and their meanings for any cell where  $x$  is the intersection set for the cell are as follows:

$$p = T \Rightarrow \dim(x) \in \{0, 1, 2\}, \text{ i.e. } x \neq \emptyset$$

$$p = F \Rightarrow \dim(x) = -1, \text{ i.e. } x = \emptyset$$

$$p = * \Rightarrow \dim(x) \in \{-1, 0, 1, 2\}, \text{ i.e. Don't Care}$$

$$p = 0 \Rightarrow \dim(x) = 0$$

$$p = 1 \Rightarrow \dim(x) = 1$$

$$p = 2 \Rightarrow \dim(x) = 2$$

The nine relational operations can be defined as following. In the definitions, the term P is used to refer to 0 dimensional geometries (*Points* and *MultiPoints*), L is used to refer to one-dimensional geometries (*LineStrings* and *MultiLineStrings*) and A is used to refer to two-dimensional geometries (*Polygons* and *MultiPolygons*).

## Disjoint

Given two (topologically closed) geometries  $a$  and  $b$ ,

$$a.Disjoint(b) \Leftrightarrow a \cap b = \emptyset$$

Expressed in terms of the DE-9IM:

$$a.Disjoint(b) \Leftrightarrow (I(a) \cap I(b) = \emptyset) \wedge (I(a) \cap B(b) = \emptyset) \wedge (B(a) \cap I(b) = \emptyset) \wedge (B(a) \cap B(b) = \emptyset)$$

$$\Leftrightarrow a.Relate(b, "FF*FF****")$$

## Touches

The Touches relation between two geometries  $a$  and  $b$  applies to the A/A, L/L, L/A, P/A and P/L groups of relationships but not to the P/P group. It is defined as:

$$a.Touches(b) \Leftrightarrow (I(a) \cap I(b) = \emptyset) \wedge (a \cap b) \neq \emptyset$$

Expressed in terms of the DE-9IM:

$$a.Touches(b) \Leftrightarrow (I(a) \cap I(b) = \emptyset) \wedge ((B(a) \cap I(b) \neq \emptyset) \vee (I(a) \cap B(b) \neq \emptyset) \vee (B(a) \cap B(b) \neq \emptyset)) \Leftrightarrow a.Relate(b, "FT*****") \vee a.Relate(b, "F**T*****") \vee a.Relate(b, "F***T*****")$$

Figure 5.7 shows some examples of the Touches relation.

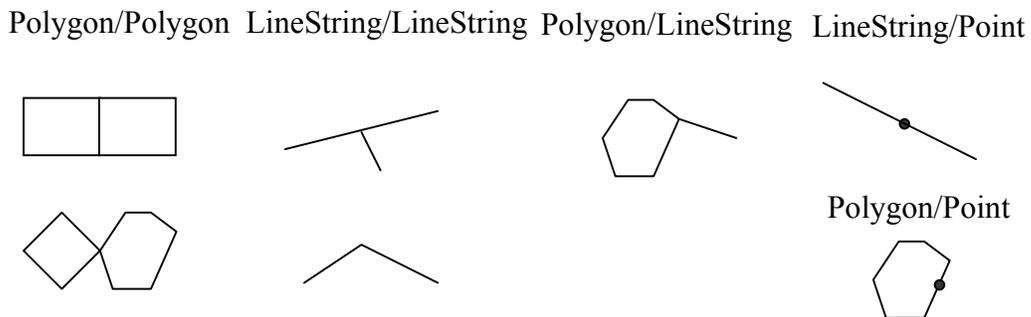


Figure 5.7 Examples of the Touches Relationship

## Crosses

“The Crosses relation applies to P/L, P/A, L/L and L/A situations. It is defined as:

$$a.Crosses(b) \Leftrightarrow (dim(I(a) \cap I(b)) < max(dim(I(a)), dim(I(b)))) \wedge (a \cap b \neq a) \wedge (a \cap b \neq b)$$

Expressed in terms of the DE-9IM:

Case  $a \in P, b \in L$  or Case  $a \in P, b \in A$  or Case  $a \in L, b \in A$ :

$$a.Crosses(b) \Leftrightarrow (I(a) \cap I(b) \neq \emptyset) \wedge (I(a) \cap E(b) \neq \emptyset) \Leftrightarrow a.Relate(b, 'T*T*****')$$

Case  $a \in L, b \in L$ :

$$a.Crosses(b) \Leftrightarrow dim(I(a) \cap I(b)) = 0 \Leftrightarrow a.Relate(b, '0*****')$$

Figure 5.8 shows some examples of the Crosses relation.

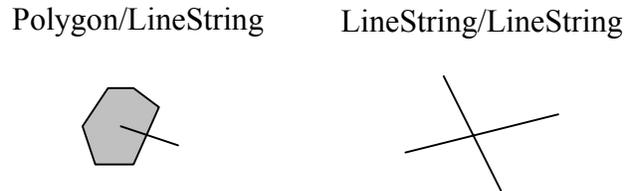


Figure 5.8 Examples of the Crosses Relationship

## Within

The Within relation is defined as:

$$a.Within(b) \Leftrightarrow (a \cap b = a) \wedge (I(a) \cap E(b) \neq \emptyset)$$

Expressed in terms of the DE-9IM:

$$a.Within(b) \Leftrightarrow (I(a) \cap I(b) \neq \emptyset) \wedge (I(a) \cap E(b) = \emptyset) \wedge (B(a) \cap E(b) = \emptyset) \wedge a.Relate(b, 'TF*F*****')$$

Figure 5.9 shows some examples of the Within relation.

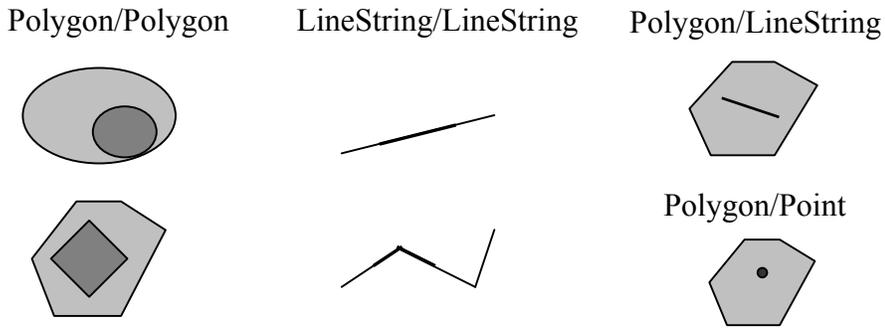


Figure 5.9 Examples of the Within Relationship

### Overlaps

The Overlaps relation is defined for A/A, L/L and P/P situations:

$$a.Overlaps(b) \Leftrightarrow (dim(I(a)) = dim(I(b)) = dim(I(a) \cap I(b))) \wedge (a \cap b \neq a) \wedge (a \cap b \neq b)$$

Expressed in terms of the DE-9IM:

Case  $a \in P, b \in P$  or Case  $a \in A, b \in A$ :

$$a.Overlaps(b) \Leftrightarrow (I(a) \cap I(b) \neq \emptyset) \wedge (I(a) \cap E(b) \neq \emptyset) \wedge (E(a) \cap I(b) \neq \emptyset) \wedge a.Relate(b, "T*T***T**")$$

Case  $a \in L, b \in L$ :

$$a.Overlaps(b) \Leftrightarrow (dim(I(a) \cap I(b)) = 1) \wedge (I(a) \cap E(b) \neq \emptyset) \wedge (E(a) \cap I(b) \neq \emptyset) \wedge a.Relate(b, "I*T***T**")$$

Figure 5.10 shows some examples of the Overlaps relation.

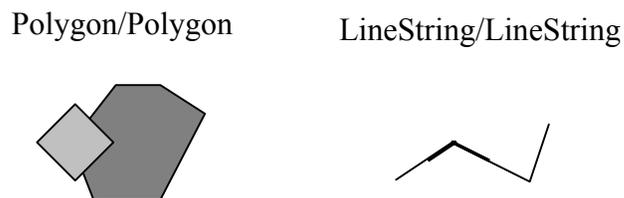


Figure 5.10 Examples of the Overlaps Relationship

## **Contains**

$a.Contains(b) \Leftrightarrow b.Within(a)$

## **Intersects**

$a.Intersects(b) \Leftrightarrow ! a.Disjoint(b)$

## **Equals**

$a.Equals(b) \Leftrightarrow b.Equals(a)$

The relationship between the geometry objects can be any one of the eight kinds of relational operations defined above. The relationship between the eight kinds of relational operations is illustrated in Figure 5.11 organized by the degree of closing of the two geometry objects. The *Relates* relationship generally describes the eight relationships on the DE-9IM.

Three processing steps are designed to calculate the relational operations:

- Minimum Bounding Rectangle (MBR) filter;
- Type filter; and
- Deep checking.

The MBR is a common spatial object access method. The MBR is the smallest X-Y-parallel rectangle, which contains entirely the spatial object it represents. Each geometry object(s) in this Geometry Data Model design has recorded its MBR as an attribute. MBR is a crude approximation of the geometry object, as they are usually accompanied by large arcs of “dead

spaces”, but it is a good filter for the relational operations. If the objects’ MBRs have the testing relationships, the relationships maybe exist between the objects; otherwise, the objects’ relationship couldn’t exist. The MBR filter is the fist step check because the relation operations applied on the MBRs are simpler than that of the real geometry objects.

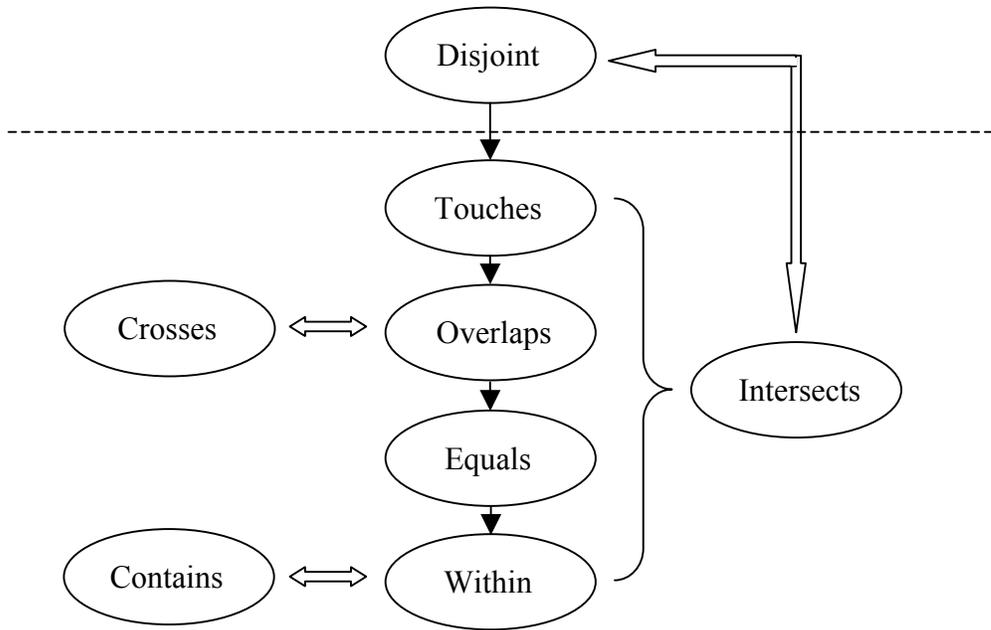


Figure 5.11 Relationships among the Eight Relational Operations  
(Double arrow: reciprocal relationship)

Each of the eight relational operations has its usage restrictions. The Overlaps operation, for example, can only be applied on the same two geometry objects, such as Polygon to Polygon, LineString to LineString and Point to Point. If the two geometry objects passed the MBR filter, the geometry Type filter will further eliminate some situations. Each of the geometries carries its geometry type as an attribute and geometry type retrieved method.

When the objects pass the first two filters, the precise calculation followed by the operation definitions will be done to make sure the relationship exists. At this step, the reciprocal operations, such as Disjoint/Intersects and Within/Contains, only calculate one relationship. For the Intersects operation, if one of the six relationships (at the left side of Intersects in Figure 5.11) exist, this relationship holds.

The three optional functions, `relate()`, `difference()` and `symmetricDifference()`, defined in the Implementation Specification, are not implemented in this thesis due to the limited time. The other functions in Table 5.2 are fully implemented and have passed the testing. Some implementation details can be found in next chapter, which discusses conformance testing.

## **Set Operations**

The Set Operations are the fundamental problems of computational geometry. Most researches focus on the intersection and union problems for different kinds of geometry. Many algorithms have been introduced in computational geometry literature.

The Intersection operation, from line-segment intersection to polygons' intersection, has been addressed by many researchers (Prparata and Shamos 1988, Bourke1989, Min *et al* 1991, Chazelle and Edelsbrummer 1992, O'Rourke 1993, Andrews 1994, Chan 1994, Balaban 1995, Berg *et al* 1997, Cigale and Zalik 1999, Zalik 2000). The algorithm for the intersection of two line segments in this thesis is the method introduced by Paul Bourke (Bourke 1989). The Set-based Intersection Algorithm introduced by Borut Zalik (Zalik 2000) is used in this thesis to check the intersection of two polygons.

The Union operation for polygons is complex, especially when polygons contain holes. The polygon defined in the OGC's document is simple polygon (no self-intersection, convex, can contain hole(s)). The intersection algorithm for polygons plays the key role in the polygon's union operation. The algorithm of polygon union is adapted from the closed set method introduced by Leonov, M. V. and Nikitin, A. G (personal discussion). This algorithm consists of four steps:

- Processing of the edge intersections;
- Edge and contour labeling;
- Collecting the resulting contours; and
- Generating result contours.

An example, Figure 5. 12, is used to illustrate the algorithm. Region A is a polygon consisting of one outer and one inner contour, and region B is a polygon consisting of one outer self-touching contour.

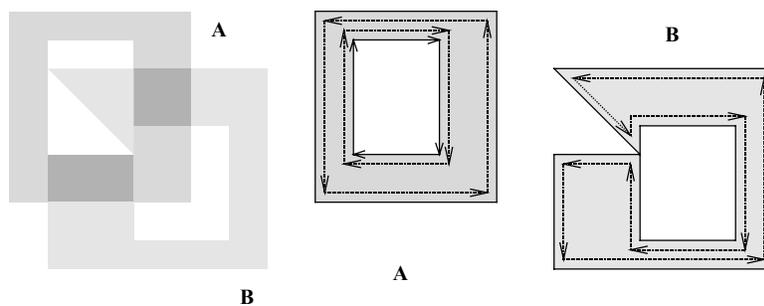


Figure 5.12 Two Intersected Polygons

## Edge Intersection Processing

If the edges share a common line segment, the endpoints of the common line segment are treated as the intersection points. All new intersection points are added as vertices to the input polygons  $A$  and  $B$ . Vertices corresponding to intersection points are called *cross-vertices* (see Figure 5.13). One geometric intersection point corresponds to at least two cross-vertices.

A cross-vertex connectivity list is introduced to record the cross-vertex. Figure 5.14 shows the cross-vertex connectivity list of point  $x$  in Figure 5.13. The corresponding cross-vertices are  $a_5$ ,  $B_2$  and  $B_9$ .

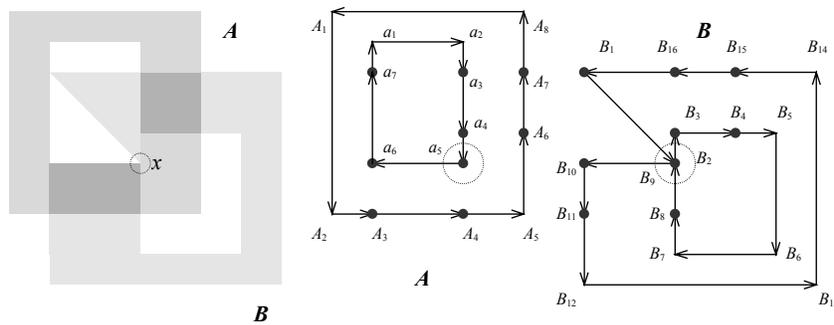


Figure 5.13 Intersections of the Two Polygons

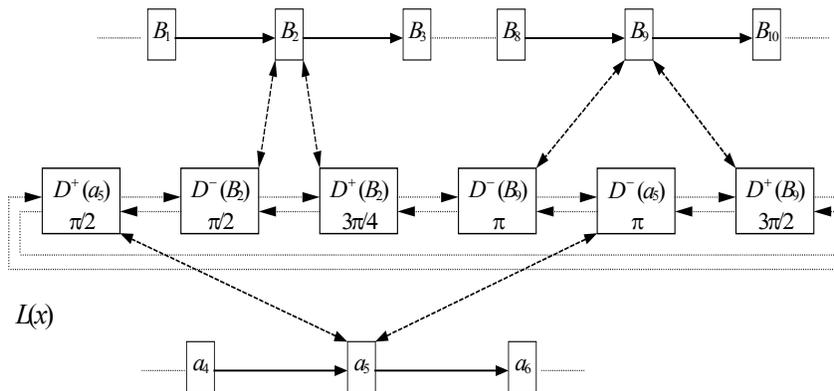


Figure 5.14 Connectivity List of Point  $x$

## Edge and Contour Labeling

The labeling method (Schutte 1995) is used to label the first step's results. Let  $C$  be a bounding contour of a polygon  $A$  or  $B$ . Let  $M$  be a polygon, which  $C$  does not belong to. Let  $E$  be an edge of a contour  $C$ . Then its *label* has one of the following values:

- *INSIDE* — when  $E$  lies inside  $M$ ;
- *OUTSIDE* — when  $E$  lies outside  $M$ ;
- *SHARED1* — when  $E$  coincides with an edge from polygon  $M$  with the same direction; and
- *SHARED2* — when  $E$  coincides with an edge from polygon  $M$  with the opposite direction.

The *label* of contour  $C$  has one of the following values:

- *INTERSECTED* — when  $C$  contains at least one cross-vertex;
- *INSIDE* — when  $C$  lies inside  $M$ ; and
- *OUTSIDE* — when  $C$  lies outside  $M$ .

Here is an algorithm for labeling  $C$  and its edges.

1 *C does not contain any cross-vertices*

If  $C$  lies inside  $M$ , it is labeled as *INSIDE*, otherwise — as *OUTSIDE*. The edges of  $C$  are not labeled at all.

2 *C contains at least one cross-vertex*

$C$  is labeled as *INTERSECTED* and all its edges are sequentially labeled. Let  $E_i(a, b)$  ( $i \in \{0 \dots n-1\}$ , where  $n$  is the number of edges in  $C$ ) be the edge to label.

### 2.1 $E_i$ does not contain cross-vertices as its endpoints

- 1)  $i \neq 0$ . The edge label value is copied from the edge  $E_{i-1}$ ;
- 2)  $i = 0$ . If  $a$  lies inside  $M$ , then  $E_i$  is labeled as *INSIDE*, otherwise  $E_i$  is labeled as *OUTSIDE*.

### 2.2 $E_i$ contains one or two cross-vertices as its endpoints

- 1)  $\exists$  edge  $F(c, d): F \in M \wedge a = c \wedge b = d$ .  $E_i$  is labeled as *SHARED1*.
- 2)  $\exists$  edge  $F(c, d): F \in M \wedge a = d \wedge b = c$ .  $E_i$  is labeled as *SHARED2*.
- 3) *Cross-vertices connectivity list(s) does not contain any vertices from  $M$*

Such situation is possible if  $C$  is not intersected by  $M$  and touches itself. The edge is labeled similarly to step 2.1.

- 4) *Cross-vertices connectivity list(s) contain vertices from  $M$ .*

For each vertex  $w_k$  the check is performed if  $E_i$  lies inside  $\angle(w_{k-1} w_k w_{k+1})$ . If  $E_i$  is inside one of such “labeling” angles, then it lies inside  $M$  and therefore is labeled as *INSIDE*, otherwise,  $E_i$  is labeled as *OUTSIDE*.

The labeled result is shown in Figure 5.15.

## **Result Contour Collection**

In this step we sequentially consider each bounding contour of  $A$  and  $B$ . Let  $C$  be the contour currently being considered. The label of contour or edge  $X$  is furthermore denoted as  $X.Flgs$ . Also *FORWARD* and *BACKWARD* will denote the original and the inverse directions of edges. Here is the algorithm for collecting the resulting contours.

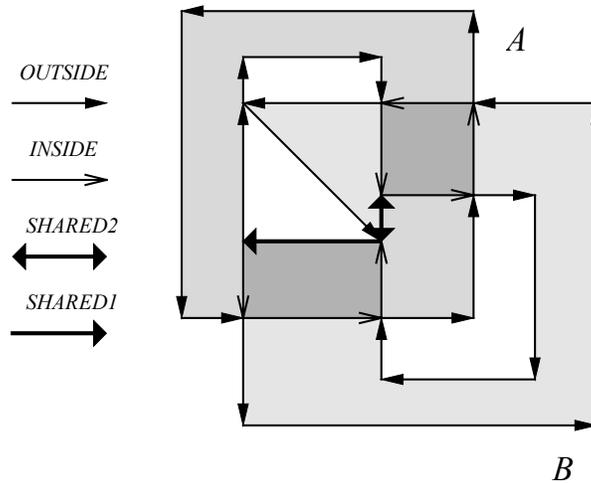


Figure 5.15 Labeled Polygons

- 1) If  $C.Flags \neq INTERSECTED$  and  $C.Flags = OUTSIDE$ ,  $C$  is added to  $R$  (result contour) depending on its label, the direction is set as *FORWARD*.
- 2) If  $C.Flags = INTERSECTED$ , we need to find which resulting contours can start from in  $C$ . Here is the algorithm for processing  $C$  ( $n$  is the number of vertices in  $C$ ). Each edge  $E$  has a bit flag  $E.Mark$  indicating if the edge has already included into one of the resulting contours, initially the flag is set to false for all edges.

```

For (  $i = 0, i < n, i++$  )
{
    if (  $EdgeRule(E_i, dir)$  and not  $E_i.Mark$  )
    {
        contour  $r$ ;
        if (  $dir = FORWARD$  )  $r = Collect(v_i, dir)$ ;
        else  $r = Collect(v_{i+1}, dir)$ ;
    }
}

```

*Include r into a set of resulting contours;*

}

}

where the functions are:

**boolean EdgeRule(edge E; dir (FORWARD, BACKWARD))**

*//the edge inclusion rule for union operation.*

{

**If ( $E.Flags = INSIDE$ )  $\vee$  ( $E.Flags = SHARED$ ) dir = FORWARD;**

}

**contour Collect(vertex v; dir (FORWARD, BACKWARD))**

{

*Create an empty contour r;*

**repeat**

*Add v to r;*

**if (dir = FORWARD) E = edge next to v;**

**else E = edge before v;**

**E.Mark = true;**

**if ( $(E.Flags = SHARED1)$  or  $(E.Flags = SHARED2)$ )**

*for edge, shared with E, set its Mark to true;*

**v = vertex next to v in direction dir;**

**if (v is a cross-vertex) Jump(v, dir);**

**until (current edge is marked);**

**return r;**

}

**Jump(vertex v; dir (FORWARD, BACKWARD))**

{

**if (dir = FORWARD) d = prev(D(v));**

```

else d = prev(D(v));
{ prev(D) denotes descriptor nearest to D in clockwise direction }
found = FALSE;
repeat
    e = edge corresponding to d;
    if (not (e.Mark) and EdgeRule(e, newdir))
    {
        v = vertex corresponding to d;
        if ((e is next to v) and (newdir = FORWARD) or
            (e is previous to v) and (newdir = BACKWARD))
        {
            dir = newdir;
            found = TRUE;
        }
    }
    d = prev(d);
until (found);
}

```

### Resulting Contour Generating

The result polygon for the Union operation is composed of one outer and two interior contours. It's very simple to arrange the bounding contours generated by previous steps into R because they already have proper orientation. The result polygon for the example is shown in Figure 5.16.

The Difference and symmetricDifference operations have not been implemented in this thesis due to time limitations. However, they further the Union operation by changing the resulting contours collection rules.

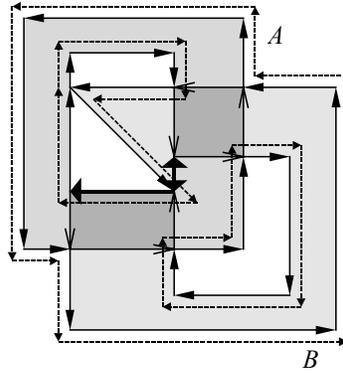


Figure 5.16 Union Results

### Constructive Operations

There are some effective algorithms for the Boundary and convexHull operations in computational geometry literature (Berg *et al* 1997, Goodman and O'Rourke 1997). The Buffer operation is complex compared with the Boundary and convexHull operations. The details of Buffer operation are illustrated in this thesis.

A Template Union Model (TUM) evolved from the Unit-combined Model (Cheng & Yuan, 2001) is introduced in this thesis to calculate the buffer for any kind of linear geometry defined in the Geometry Data Model. The following buffer example, illustrated in Figure 5.17, is an introduction to the TUM model. We create a Circle with a particular radius for *Line* object *AB*'s Buffer operation. The Circle rolls from the start point *A* to the end point *B* (left), and generates the buffer area of *Line AB* (right). In this example, the Circle is the Template in TUM model, the spatial Union operation for all the Circles generates the *Line AB*'s buffer. In TUM model, the Union operator in Set Operators is the key operation.

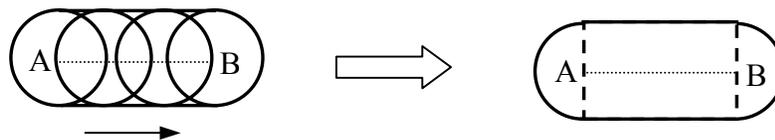


Figure 5.17 Line Object's Buffer

## **Template**

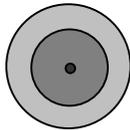
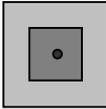
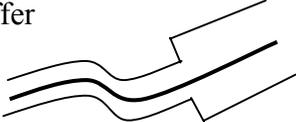
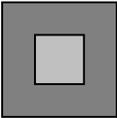
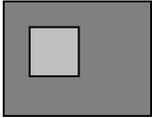
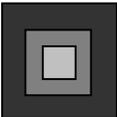
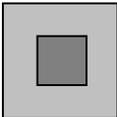
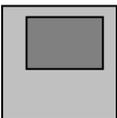
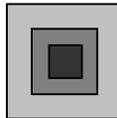
The Buffer operation is an important function in GIS analysis. The study of propagation of pollution or flooding with natural or artificial barriers requires the construction of a geometric buffer. The buffer methods are different in different application domains. The common buffer methods are generalized in Table 5.3 as the templates used in TUM model. It's a big challenge to list all the templates, but the software system can provide the extension capability from the viewpoint of software engineering.

Analyzing the templates in Table 5.3, we have found that the simple LineString Template (only has two points) is the Original Template. The Point, LineString and Polygon templates can be generated from the Original Template. If the Original Template has only one point, it becomes the Point Template. The LineString Template is the spatial union of all the segments' Original Templates. The Polygon Template is the spatial union of Original Templates of polygon's edges.

## **Algorithm**

As analyzed above, the Original Template is the root for all templates listed in Table 5.3. In the TUM model, only the shape of the Original Template is really calculated. In fact, there are four kinds of Original Templates, one is illustrated at the right of Figure 5.17, and the other three are illustrated at the right side of Figure 5.18. The Original Templates are divided into two categories: two-side buffer and one-side buffer. The Variable Buffer Templates are generated from the union of the shape for each edge in the Polygon or segment in the LineString applied different one-side Original Templates. The Multiple Buffer Templates are combined of multiple templates with different radius.

Table 5.3 Examples of Template

Point	Circle Buffer		Circle Multiple Buffer	
	Square Buffer		Square Multiple Buffer	
LineString	Constant Buffer		Variable Buffer	
	One-side Buffer		Multiple Buffer	
Polygon	Exterior Constant Buffer		Exterior Variable Buffer	
	Exterior Multiple Buffer		Interior Constant Buffer	
	Interior Variable Buffer		Interior Multiple Buffer	

Polygon's buffer is a little complex. If the polygon object hasn't a hole, the two-side exterior buffer will be calculated along the boundary of the polygon first, then the generated exterior buffer will be combined by union with the original polygon to form the buffer object (see Figure 5.19). If the polygon has hole(s), the two-side exterior buffer and one-side interior buffer(s) are

calculated respectively, then the generated exterior buffer is combined by union with the original polygon and interior buffer zone are removed.

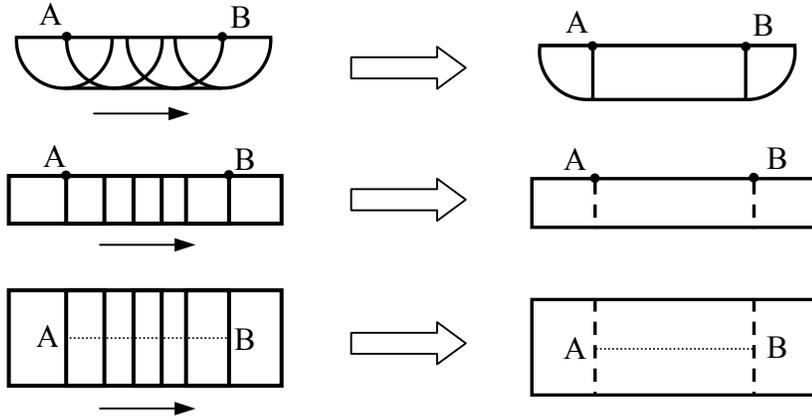


Figure 5.18 Original Buffer Templates

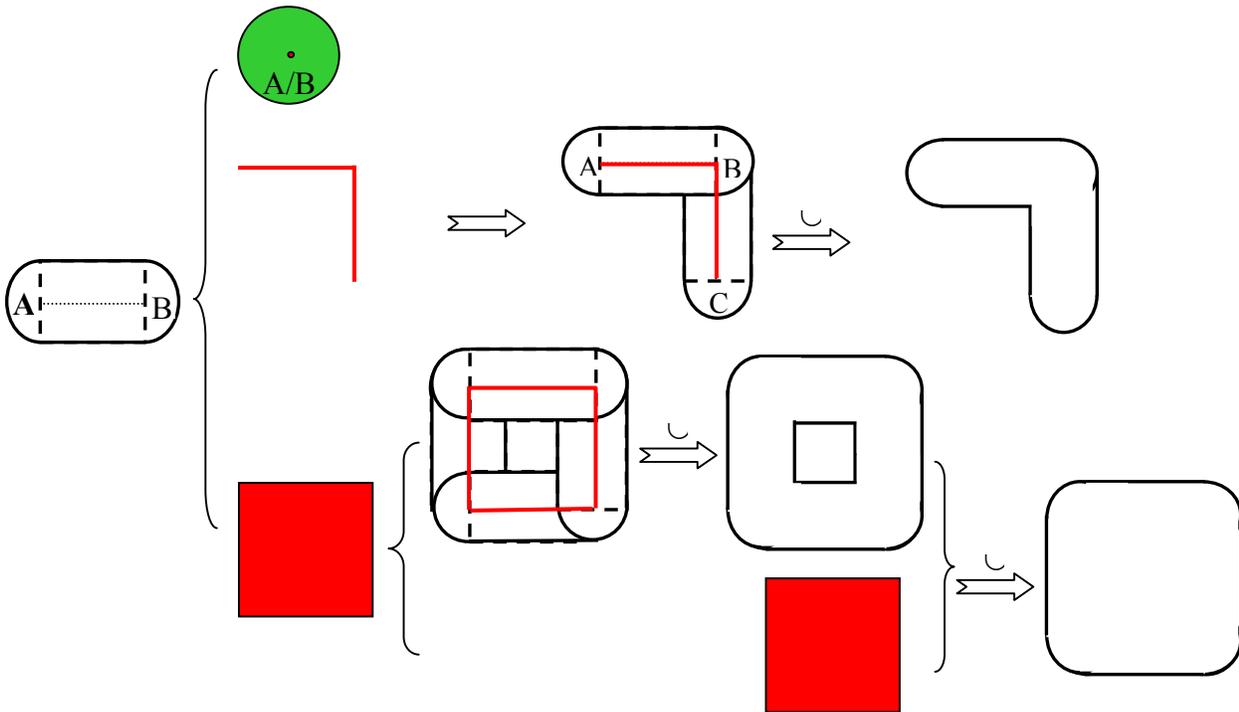


Figure 5.19 Example of TUM

### 5.3.2 Geodata Storage

How to effectively organize large datasets in a system memory will make a difference to the system performance. The goal of geodata storage design is to reduce system memory use while keeping performance at a reasonable level.

Object-oriented (OO) technology is used to organize geodata. OO technology is adopted in this design in response to the evolution of software engineering. The strategy used to deal with large geodata sets should be migrated to Object methods via traditional Array technology, which organizes the spatial data in a simple linear sequence. There are two object-based methods to optimize the geodata storage strategy: pure object and data object. The Pure Object method follows rigorously OO technology to organize the spatial data and its operations. The Data Object method adopts the object concept, but each object's data only contains spatial data, no operations are applied on it (see Figures 5.20, 5.21 and 5.22 for details).

Object technology encapsulates the spatial data and its operations in an object. The complex object is composed of simple objects. For example, a *Polygon* is composed of *LinearRings*, each *LinearRing* contains many *Points*, each *Point* has a spatial data value (x, y) and some methods which process the data. The *Polygon*, therefore, has all the spatial data and methods coming from the underlying elements. Generally, the methods from the underlying elements are not all useful at a higher level. In this case, the pure object data storage method wastes considerable memory when the geodata set is very large. A simple test of this method has been undertaken and the results are listed in Table 5.4.

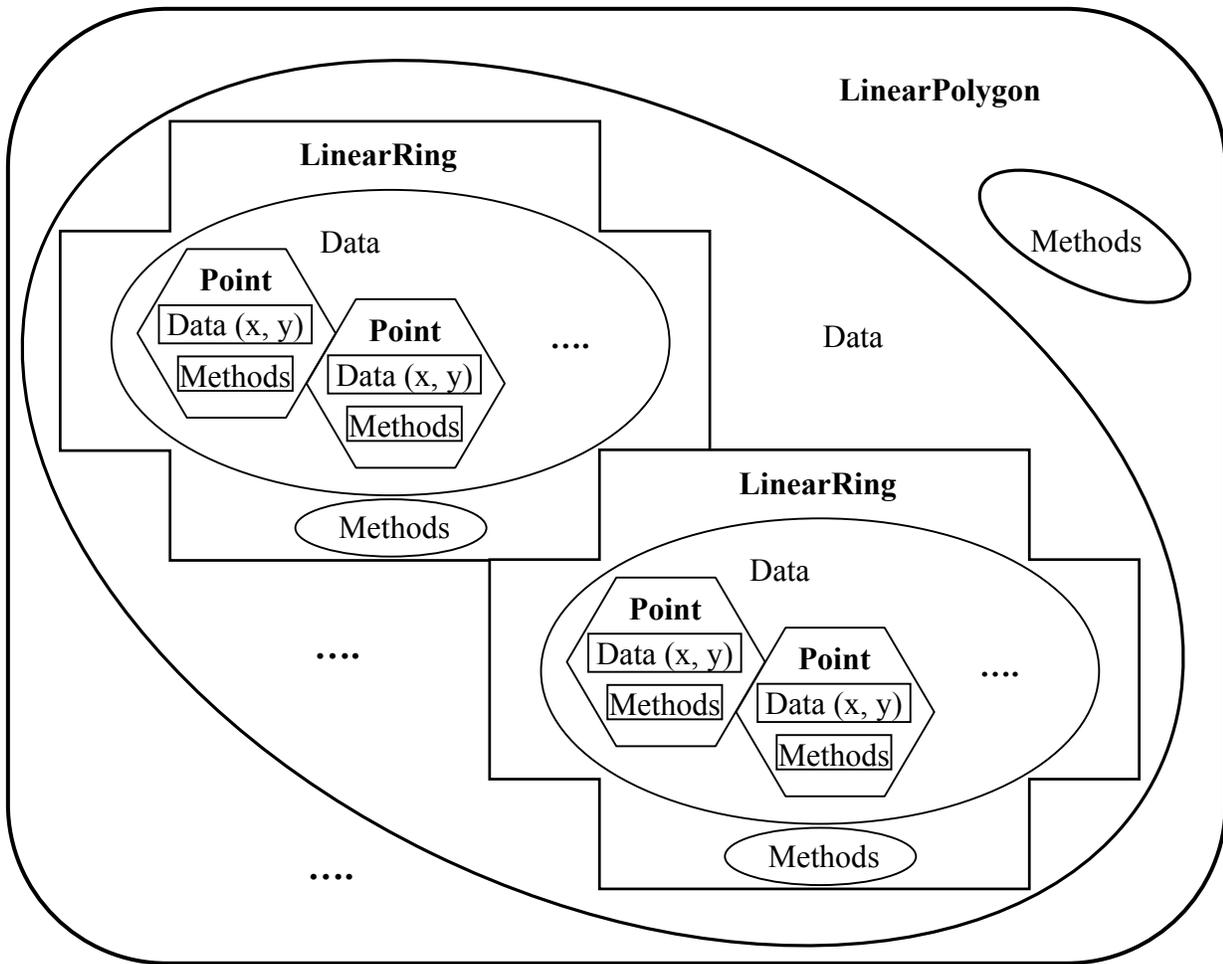


Figure 5.20 Polygon Object Structure in Pure Object Method

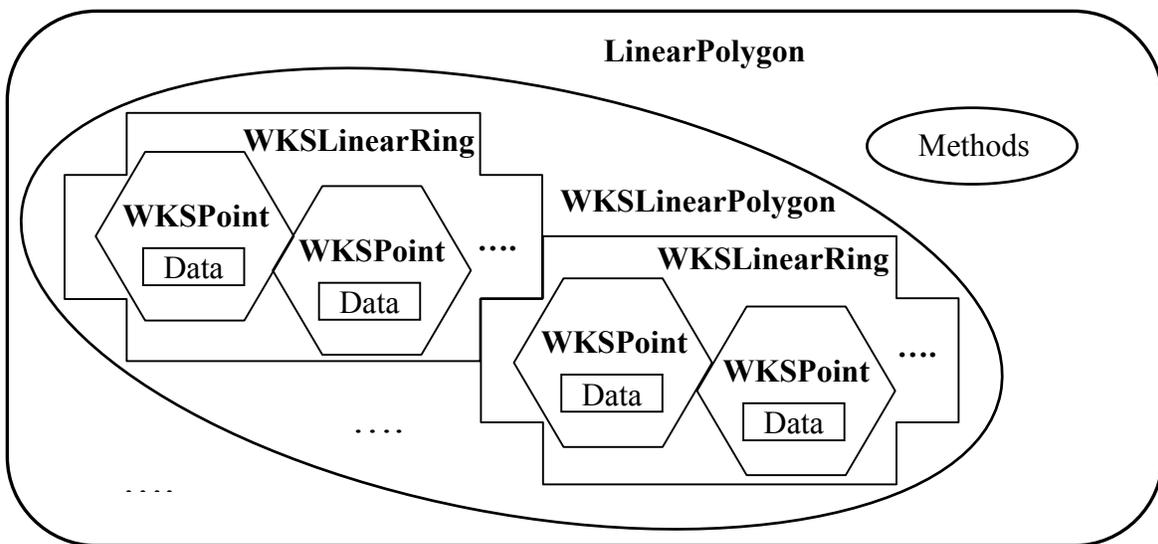


Figure 5.21 Polygon Object Structure in Data Object Method

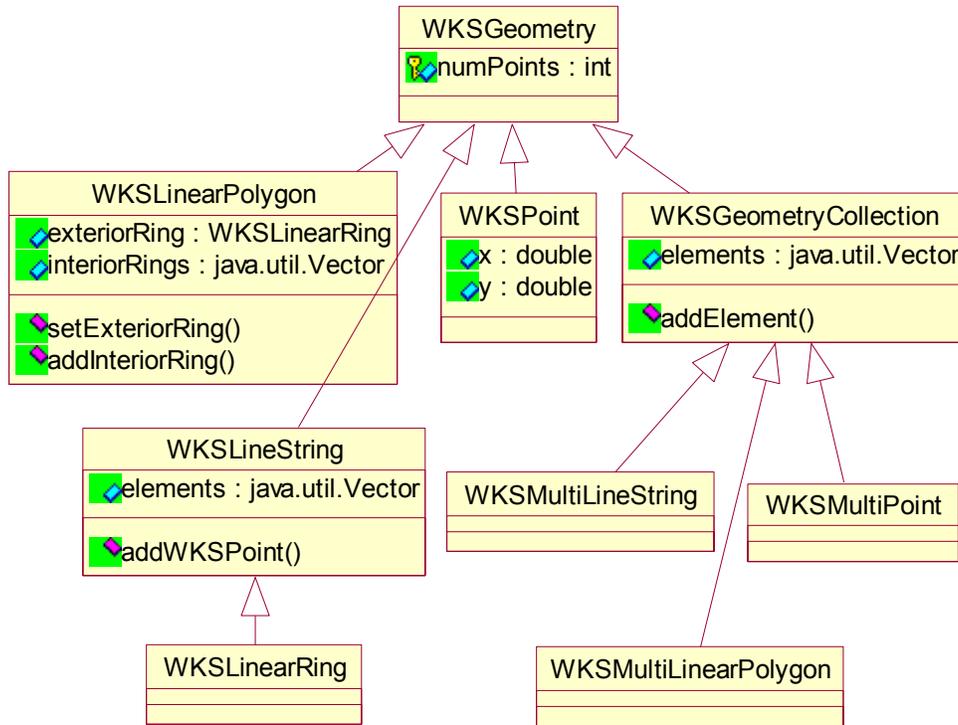


Figure 5.22 Data Objects Used in the Design

From this simple testing, we know that the Data Object method is a good solution for large geodata sets. In this design, the data objects illustrated in Figure 5.21 are used to store spatial data within the corresponding geodata objects. A *LinearPolygon*, for example, encapsulates a *WKSLinearPolygon* data object, which holds all the spatial data of this *LinearPolygon*, and the operations applied on the spatial data.

Retrieving the elements for one object is a problem in this storage strategy. In the pure object method, when we retrieve a Point from a LinearPolygon object, the function will return a Point object including its data and methods. But in the data object method, the primary elements are WKSPoint objects, not Point objects. To avoid the problem, the element retrieval function invokes proper Factories depicted in Figure 5.5 to create the objects on the fly.

Table 5.4 Memory Expense of the Three Methods

	Pure Object	Data Object	Array
Total Used Memory Expense(Mb)	38.32	21.14	64.34*

Environment: Pentium II, Windows NT 4.0, RAM: 128 Mb.

Data: a shape file, roadnet.shp, contains 18,289 polylines. For the Array method, the type is double, the capability is 500.

\*: this value is under loaded, 6,000 polylines; when loaded with 12,000 polylines, the system runs out of memory.

#### 5.4 Characteristics Analysis

The design and implementation details of the Geometry Data Model are discussed above.

Compared with the existing references, this design has its own characteristics as following:

- Disseminativity. The software system design standard, UML, is used during the whole design period to ensure this design is more readable and understandable. Design tool Rational Rose integrated all popular UML annotations together to standardize this design. On the other hand, Java's design and coding standards are used in this Geometry Data Model design. Sun and most of the Java software developers adopt Java's coding conversations and naming system, a real standard for Java software developing. These rules ensure design standards are maintained through all software architecture and coding levels.
- Extensibility. Extensibility endows the life to a standard design. At the current design stage, only linear 2D geometric objects are defined and implemented, but the geometry object's

logic relationship is clearly defined, and the inheritance hierarchy is embedded in the design architecture. It's very easy for other bodies or individuals to understand this model and extend the existing architecture to support the non-linear objects and complex objects. The object *RectangleRing*, for example, can be added into this model as a child of object *LinearRing*. On the other hand, the Java interpreter can execute Java byte codes directly on any machine to which the interpreter has been ported. Java language's interpreting strategy makes it possible to dynamically load the extended classes without interrupting the running system, and seamlessly integrate them with the legacy Java system. This characteristic make the Java software developing is more flexible, rapid and exploratory.

- **Functionality.** The Geometry Data Model in this design is more powerful than that of the traditional GISs. This geometry data model supports more geometric objects. Traditional GISs only support linear geometry objects, but this data model also supports non-linear objects and complex objects. Polygon objects in traditional GISs must be convex, but in this data model, concave polygons are supported. In addition, this geometry data model supports more operations at the data model level. There are nine kinds of relationship/spatial operations, such as intersects, cross, contains, union, buffer and so forth, defined in a data model level, and six of those operations are implemented in this work.
- **Manageability.** Java's Package technology is used to organize the model structure and code files. Similar functionality is organized into one package. One developing body can create a new package and contains their codes. The extended codes can easily be put into proper package by their functions or by organization.

## CHAPTER 6 TESTING SUITE

Software development is often the developer's personal creative. Given a common framework for software development, a resulting implementation may vary from developer to developer. How to test whether the software implementation is compatible to the Implementation Specification? OGC provides a Conformance Testing mechanism to ensure the compatibility of an implementation. This section will address the design and implementation of a testing suite for the Simple Features Implementation Specification of Java version.

### 6.1 Objectives

OGC's standards are public and can be downloaded from its web site. Maybe there are thousands of implementations for one specification, but only the OGC compatible implementations are valid. To validate implementations, OGC launched the Conformance Testing Program to make sure that development activities follow the OGC's direction. Conformance testing includes two phases: Conformance Testing; and Interoperability Testing. Current testing suites only have the conformance testing (no interoperability testing available), to check the compatibility of the candidate software to its claimed OGC's specification.

OGC has released the testing suites for Simple Features Implementation of OLE/COM and SQL, but there is no testing suite for Java version available. A testing suite for testing the conformance of Java implementations, therefore, is developed in this thesis, with respect to OpenGIS Simple Features Implementation Specification for Java. This testing suite only contains the conformance testing; the interoperability testing is not included.

## **6.2 Testing Suite Design**

### **6.2.1 Testing Suite Generation**

All the proposed testing suites, including source code and documents, must be submitted to OGC's Technical Committee for review. The source code will be checked by the Technical Committee. Only the passed candidate suites can be identified as OGC official testing suites. A completed testing suite contains the testing code, testing dataset and documentation.

### **6.2.2 Testing Procedure**

If a product is claimed to be OGC compatible, the product must pass the conformance testing. If the product passed the owner's own checking, the product and the testing result documents can be submitted to OGC for confirmation. Once the conformance testing is successfully completed again, OGC will license vendors of such systems to use OGC's marks (trademarks or certification marks) that will identify to users the capability of products with respect to OpenGIS Implementation Specifications.

### **6.2.3 Design Considerations**

This Java version testing suite is planned to be submitted to OGC's Technical Committee. To facilitate the testing process and the public usage, the following factors have been considered in this design.

## **Easy**

It should allow the public users to test their products using the testing suite with no or a little effort. The testing suite, therefore, should be **easy** enough for the Technical Committee and public users. The notes and comments must be embedded in the source code to ensure the readability. Java's code conventions must also be followed during the suite coding.

## **Public**

The testing suite is open source software that can be assessed via the Internet. The testing suite is developed as a Java Applet, which can run within a general web browser, like Internet Explorer or Netscape. This allows the testing over Internet and allows public involvement in the testing.

### **6.2.4 Components of Testing Suite**

According to the considerations discussed above, this testing suite is designed to be composed of three main parts: Graphic User Interface (GUI), data loading code and conformance testing code.

#### **GUI**

The GUI in testing suite is a very simple, it only provides a communication means between the testing suite and user. The GUI will guide the user on how to do the testing step by step. The testing dataset and the testing results will be displayed for user assessment. All testing indicators, success or failure, will be shown on screen.

#### **Data Loading**

This component of the testing suite loads the OGC's official testing dataset into the system. In OO technology, the data is encapsulated into its object. The data loading code will initiate the

Factories objects to create the instances of the geometry objects defined in the Simple Feature or SF Implementation Specification. The role of data loading is to check the candidate's capability of creating each of the geometry objects. Future conformance testing will be applied on the instanced objects created herein.

### **Conformance Testing**

Conformance testing will test the candidate product with respect to three criteria: object creation, basic methods and spatial relationship operations. The object creation is checked at the data loading stage. The other two checks will be done at this stage. Referring to the existing testing suites for OLE/COM and SQL, the testing of basic methods is mandatory and the testing of the spatial relationship operations is optional (see Table 5.2).

### **6.3 Implementation**

A Java applet is a powerful and flexible approach compared with other web solutions, such as CGI and ASP. This testing suite is implemented as a Java applet, and the testing dataset is embedded in the system by default. Slow downloading is a big issue for applications using Java applets. The browser needs to download the required classes from the server each time once the testing suite is initiated. There are two solutions to reduce the downloading time:

1. Simplifying packages and small classes by putting them into one Java compressed file;
2. Using Sun's standard classes.

In this testing suite, the class is implemented as simply as possible to meet the testing requirements and the interface's layout invokes only one Borland class. The standard Java

classes used in this testing suite are included in the Java Virtual Machine (JVM) embedded in the client's browser. This simple testing suite follows Java's coding conventions. The annotation in the source code can aid user navigation during configuration.

The testing suite has implemented the testing process for both mandatory and optional functions. Table 6.1 shows the status of the suite. The prototype system of this testing suite is illustrated in Figures 6.1 and 6.2. Using this testing suite, the candidate products can be tested by distributed evaluators. The testing results are displayed in the Results Box after each testing operation. For the candidate providers, they can easily adapt the testing suite to test their own products.

Table 6.1 Implementation Status of the Functionality in Testing Suite

Category	Function	Impl. Status
Mandatory	XXX createFromXXX(xxx);	
	XXX createFromWKSXXX(wksxxx, srs);	✓
	XXX createFromWKBXXX(srs, byte[]);	
	boolean isEmpty();	✓
	boolean isClosed();	✓
	boolean isSimple();	✓
	WKSGeometry export();	
	Geometry copy();	
Optional	Geometry boundary();	✓
	Geometry buffer (double distance);	✓
	Geometry convexHull();	✓
	double distance (Geometry other);	✓
	Geometry intersection (Geometry other);	✓
	Geometry union (Geometry other);	✓

Geometry difference (Geometry other);	
Geometry symmetricDifference (Geometry other);	
boolean equals (Geometry other);	✓
boolean touches (Geometry other);	✓
boolean contains (Geometry other);	✓
boolean within (Geometry other);	✓
boolean disjoint (Geometry other);	✓
boolean crosses (Geometry other);	✓
boolean overlaps (Geometry other);	✓
boolean intersects (Geometry other);	✓
boolean relate (Geometry other, EgenhoferOperator operator);	

Note: XXX stands for all the implementable geometry objects: Point, LineString, Polygon, GeometryCollection, MultiPoint, MultiLineString and MultiPolygon.

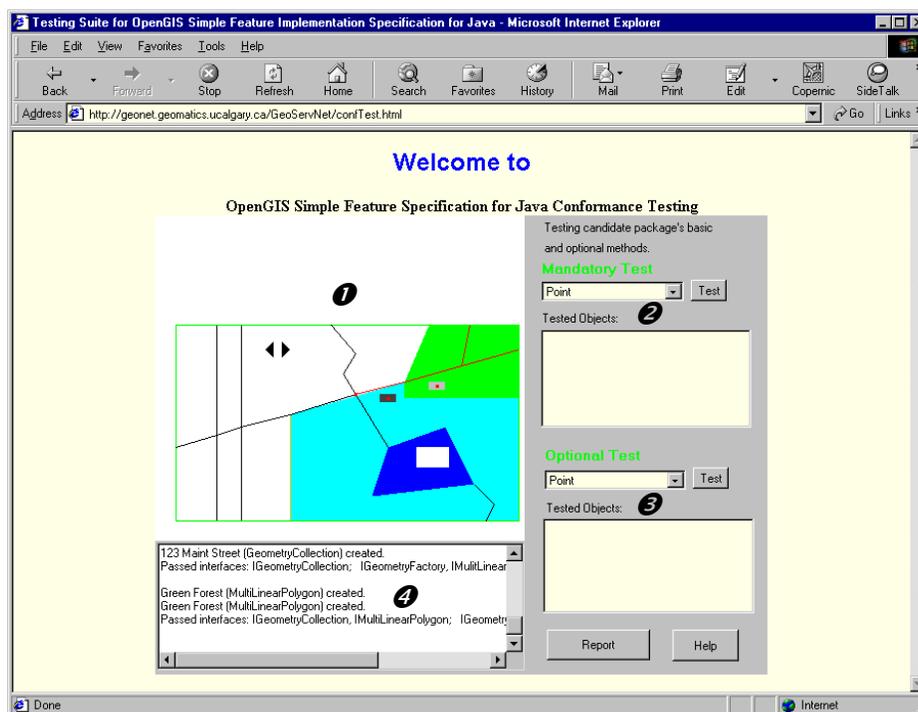


Figure 6.1 Interface after Loaded Data in the Prototype System

(1: Map Area 2: Mandatory Operations 3: Optional Operations 4: Results Box)

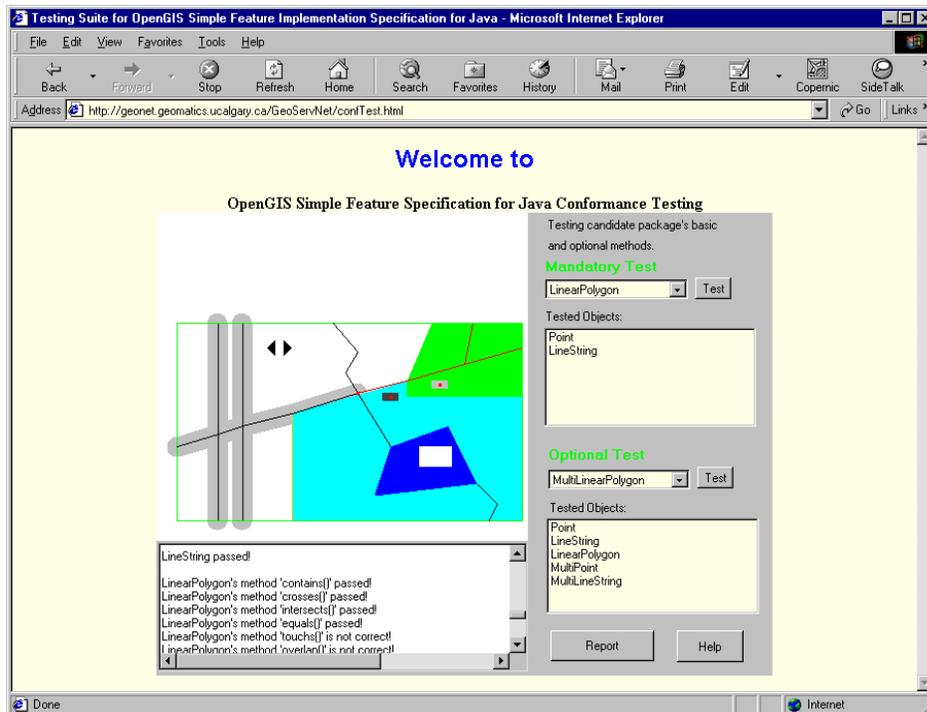


Figure 6.2 the Buffer Testing in the Prototype System

## 6.4 Issues

### 6.4.1 Dataset

OGC provides some testing datasets on its web site. The testing suites for OLE/COM and SQL use the same dataset to test functionality. In this Java version, the same dataset as that used in the OLE/COM and SQL versions (see Figure 6.3) is embedded into the testing suite.

The data is a synthetic dataset, developed by hand, to exercise the functionality of the specification. The semantics and points' coordinates of this dataset are defined in OGC released testing suites (OGC, 1999c). The following gives entire test data in Well Known Text (WKT) format (↵ indicates that the subsequent line of text is a continuation):

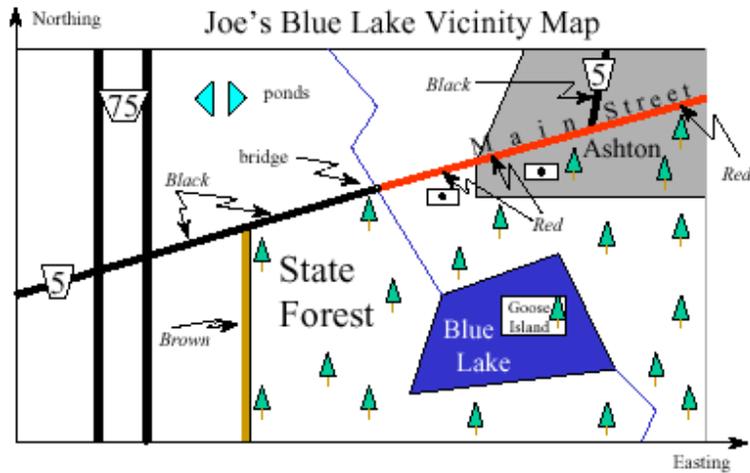


Figure 6.3 Test Data Concept (OGC, 1999c)

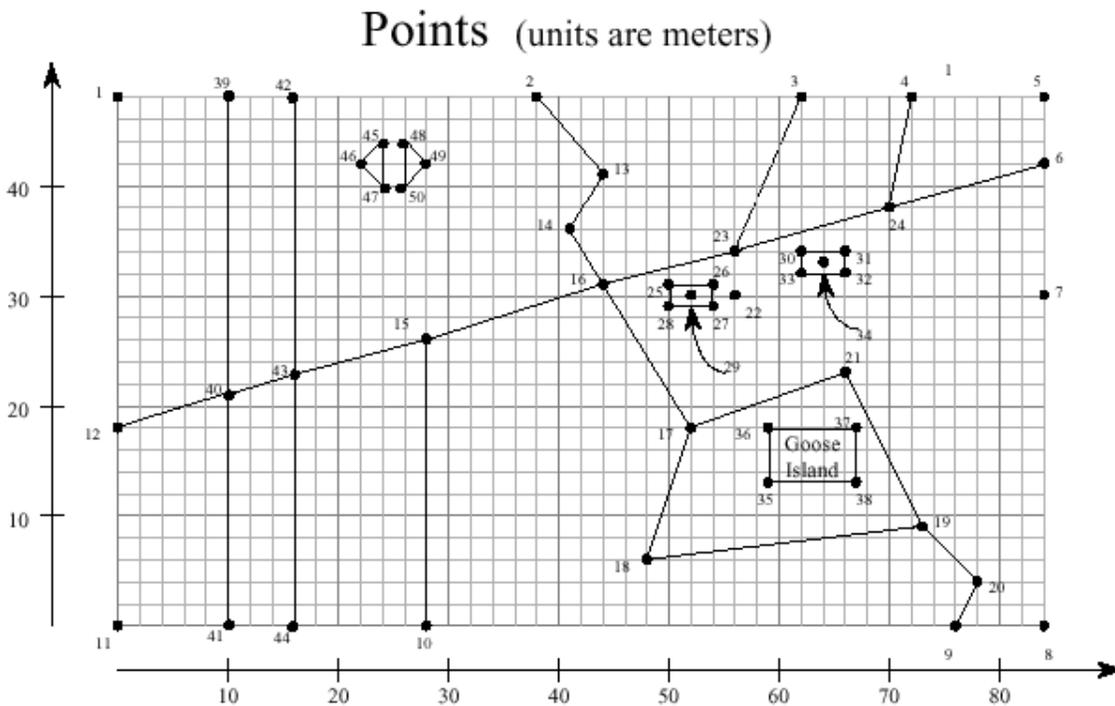


Figure 6.4 Points in the Blue Lake Dataset (OGC, 1999c)

```

PROJCS['UTM_ZONE_14N', GEOGCS['World Geodetic System 72', DATUM['WGS_72', SPHEROID
['NWL_10D', 6378135, 298.26]], PRIMEM['Greenwich', 0], UNIT['Meter', 1.0]],
PROJECTION['Transverse_Mercator'], PARAMETER['False_Easting', 500000.0],
PARAMETER['False_Northing', 0.0], PARAMETER['Central_Meridian', -99.0],

```

PARAMETER['Scale\_Factor',0.9996], PARAMETER['Latitude\_of\_origin',0.0],UNIT['Meter',1.0]]" ↵  
 101,"Blue Lake","POLYGON( (52 18, 66 23, 73 9, 48 6, 52 18), (59 18, 67 18, 67 13,59 13, 59 18) )" ↵  
 102,"Route 5","LINESTRING( 0 18, 10 21, 16 23, 28 26, 44 31 )" ↵  
 103,"Route 5","LINESTRING( 44 31, 56 34, 70 38 )" ↵  
 104,"Route 5","LINESTRING( 70 38, 72 48 )" ↵  
 105,"Main Street","LINESTRING( 70 38, 84 42 )" ↵  
 106,"Dirt Road by Green Forest","LINESTRING( 28 26, 28 0 )" ↵  
 109,"Green Forest","MULTIPOLYGON( ( (28 26, 28 0, 84 0, 84 42, 28 26), ↵  
 (52 18, 66 23,73 9, 48 6, 52 18) ), ( (59 18, 67 18, 67 13, 59 13, 59 18) ) )" ↵  
 110,"Cam Bridge","POINT( 44 31 )" ↵  
 111,"Cam Stream","LINESTRING( 38 48, 44 41, 41 36, 44 31, 52 18 )" ↵  
 112,"Cam Stream","LINESTRING( 76 0, 78 4, 73 9 )" ↵  
 113,"123 Main Street","GEOMETRYCOLLECTION( POINT( 52 30 ), ↵  
 POLYGON( ( 50 31, 54 31, 5429, 50 29, 50 31) ) )" ↵  
 114,"215 Main Street","GEOMETRYCOLLECTION( POINT( 64 33 ), ↵  
 POLYGON( ( 66 34, 62 34, 6232, 66 32, 66 34) ) )" ↵  
 115,"Neat Line","POLYGON( ( 0 0, 0 48, 84 48, 84 0, 0 0 ) )" ↵  
 117,"Ashton","POLYGON( ( 62 48, 84 48, 84 30, 56 30, 56 34, 62 48 ) )" ↵  
 118,"Goose Island","POLYGON( ( 67 13, 67 18, 59 18, 59 13, 67 13 ) )" ↵  
 119,"Route 75","MULTILINESTRING( (10 48, 10 21, 10 0), (16 0, 10 23, 16 48) )" ↵  
 120,"Stock Pond","MULTIPOLYGON( ( ( 24 44, 22 42, 24 40, 24 44) ), ↵  
 ( ( 26 44, 26 40,28 42, 26 44) ) )" ↵

The geometry objects in OGC standard testing dataset are listed above, but not all the implementable geometry objects defined in OpenGIS Simple Feature Implementation

Specification are represented. Table 6.2 shows the difference between the simple feature data model and the testing dataset in the implemented geometry objects.

Table 6.2 Comparison of the Implementable Objects in the Data Model and Dataset

Implementable Objects	Data Model	Testing Dataset
Point	X	X
LineString	X	X
Polygon	X	X
GeometryCollection	X	X
MultiPoint	X	
MultiLineString	X	X
MultiPolygon	X	X

### 6.4.2 Distribution

All of Java's .java files in the testing suite are zipped into one compressed file for easy distribution. When you unzipped the testing suite, the following file structure will be created on your system:

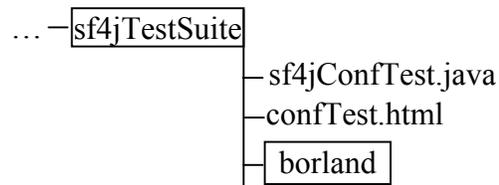


Figure 6.5 Unzipped File Structure in the Testing Suite  
(in Window environment; box indicated that it is a directory)

The files are divided into supporting files and key files. The key files show off this testing suite's features by support from the supporting files. Only the following two key files need to be adapted:

- sf4jConfTest.java
- confTest.html

The Java applet Sf4jConfTest.java is the essential part in this testing suite. The browser runs this testing suite by opening the HTML file confTest.html, which contains the compiled file of sf4jConfTest.java. There are three steps to adapt this testing suite:

1. import the candidate's implemented classes;
2. adjust the location of all the classes; and
3. compile sf4jConfTest.java.

The testing suite will invoke the implementation of the candidate product to the test conformance. First of all, we must tell sf4jConfTest.java which implementation will be tested. The first step is to import the candidate's implemented classes at the beginning of sf4jConfTest.java, such as:

```
import com.uc.geoservnet.geometry.*;
```

The annotation in the file will instruct the user how to do this.

To make sure that the sf4jConfTest.java can find the imported classes, you must adjust the location of the implemented classed in your system before compiling the main file. If the

compiler can not find the implemented classes, it will give an error message instructing you of the adjustments to be made.

The last step is compiling the amended `sf4jConfTest.java` file. It is assumed that the Java compiler has been installed on your system. If not, the compiler can be downloaded from Sun's web site. It's easy to compile this Java Applet following the instruction of the compiler. Make sure the name of the generated byte code file `sf4jConfTest.class` is the same in the `confTest.html` file before you run the testing suite.

## CHAPTER 7 CASE STUDY

The designed Geometry Data Model in this thesis has been applied in a research project *A GIS Based Geotechnical Data Sharing and Analysis System* (Tao *et al*, 2001). This case is used to test the design and demonstrate the performance of the model.

### 7.1 Background

The geotechnical technology is popularly used in Civil and Environmental Engineering projects and researches. Geotechnical data is often gathered from field instrumentation, site investigation, laboratory tests and model studies. The Geological Survey of Canada (GSC) compiled a computerized database of geological and geotechnical data for the Calgary urban area in the early of 1970's. Unfortunately, as time elapsed, the database appears to have been disused and neglected. It is realized that Calgary has about 100,000 wells on record represented by core and/or drill cuttings (Eyles, 1997) dispersed across many different agencies. The geotechnical data generated by and for the special projects was obtained at great expense. They are of huge potential value for other purposes. The University of Calgary in conjunction with the Calgary Geotechnical Society launched a project to develop a GIS based data sharing and analysis system to distribute the geotechnical data through the Internet.

### 7.2 System Design

### 7.2.1 Architecture

This geotechnical data sharing and analysis system is not only a geotechnical data distribution system, but also it should distribute the geotechnical analysis functionality into the Internet. There are some commercial Internet GIS software available in GIS industry, such as ESRI's ArcIMS<sup>®</sup>, Autodesk's MapGuide<sup>®</sup>, MapInfo's MapXtreme<sup>®</sup>, and so forth. Different Internet GIS software adopts different computer technology and spatial theories. Each software package has its own features and limitations. The existing commercial Internet GIS software has the capability to distribute the geospatial datasets. Most of them host the analysis functions at the server side for client invocation, a few of them can distribute the analysis functions to the client machine through the Internet computing environment (Limp, 2001). However, the limited extensibility is a big issue when building this data sharing and analysis system on these commercial Internet GIS software.

In this project, the geotechnical data of Calgary is dispersed among the different agencies, and is documented using different formats (hardcopy, GIS and CAD). It is also a fact that the databases and GISs used vary among the agencies. Considering the current status of the geotechnical data, the existing GIS and database software, and the state of the art technology of the Internet GIS and databases, the architecture of this geotechnical data sharing and analysis system has been developed, shown in Figure 7.1.

This system consists of three components: the spatial database, the Internet GIS server, and the front-end web interface. Oracle 8 is the central database, which manages the geospatial and

attribute geotechnical data. Oracle's SQL Plus, SQL\*Loader and other tools, such as FME, support the transforming of the geotechnical spatial data into and out of the data repository. The Java GIS server GeoServNet Server and Agent, and client GeoEye™ are developed in this project. The server and client viewer share the same Geometry Data Model designed and implemented in the previous chapters of this thesis. The spatial and attribute data retrieved from Oracle database to generate the simple features defined in OpenGIS Simple Features specification at the GeoServNet side on fly. The generated simple features contain the requested spatial and attribute information, which are translated to client side GeoEye™ to display, operation and query. The GeoServNet Server can be installed on more than one machine. The GeoServNet Agent middleware plays the role of balancing the visiting burden among the GeoServNet Servers.

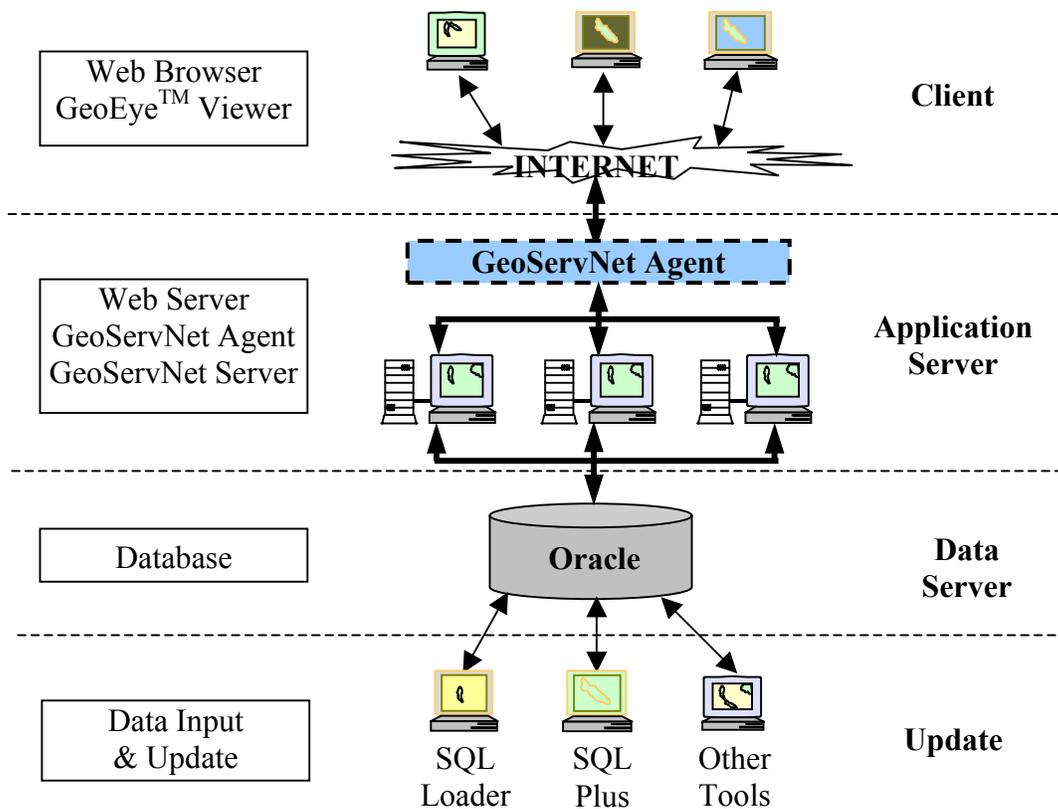


Figure 7.1 Architecture of the Geotechnical Data and Analysis Sharing System

## 7.2.2 Data Model

The data model in this system consists of two sub-data models: Geotechnical Data Model and Spatial Data Model. The Geotechnical Data Model is implemented in Oracle to management the attribute and spatial geotechnical datasets. The Spatial Data Model is implemented in the software: GeoServNet and GeoEye™.

### Geotechnical Data Model

The geotechnical data model in this system was designed based on the design guideline developed by the Geological Survey of Canada. There are some published data models (Lee et al. 1990, Oloufa et al. 1992, Adams et al. 1993, Giles 1994, Papacostas 1994, Eyles et al. 1997) in literature. The E-R pattern analysis technique was used to integrate the spatial and attribute data in a single data repository.

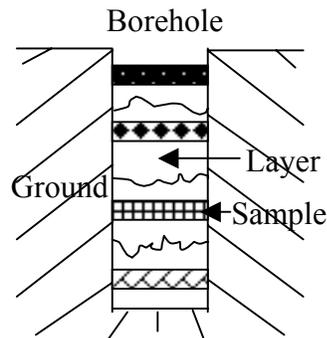


Figure 7.2 Borehole Profile Diagram

In the real world, the relationships between the entities of geotechnical field and laboratory test are illustrated at Figure 7.2 and 7.3. The BOREHOLE OBJECT entity extends the relational model to support the geotechnical spatial information in the database. The details of the geotechnical data model are discussed as following.

- PROJECT entity: The geotechnical data generated by the given projects, which are launched by different companies. In general, a project often drills many boring holes for the specific purposes. PROJECT entity is the root entity in the geotechnical data model.

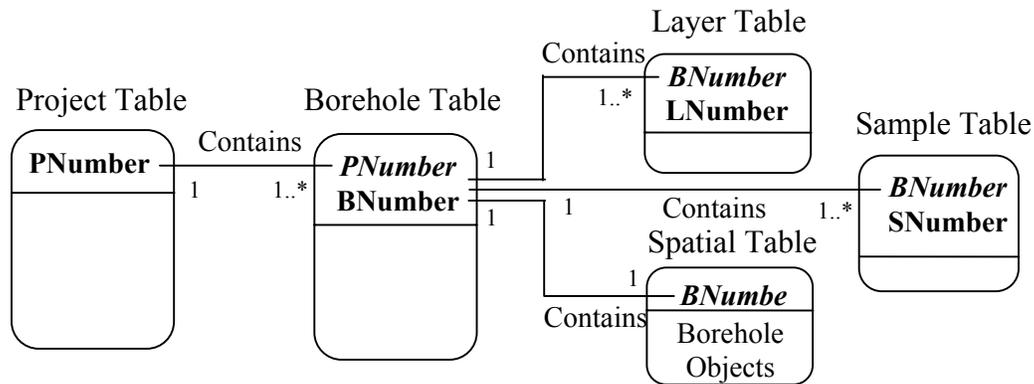


Figure 7.3 A Conceptual Geotechnical Data Model in Oracle

- **BOREHOLE** entity: Boreholes are investigation points drilled to retrieve information about the subsurface soil. The BOREHOLE entity provides information about the boreholes in the reference from which the subsoil information is taken. The borehole's information is presented in the boring log, which is contained in the project's reports.
- **LAYER** entity: A layer represents a soil stratum. The collection of a vertical ordering of layers forms a boring log. Each layer gives the information about the soil stratum.
- **SAMPLE** entity: A sample is an extraction of soil material from one layer of a borehole. Samples are used in tests to determine values of soil properties and parameters. Such values are reported in the SAMPLE entity.
- **BOREHOLE OBJECT** entity: A Borehole Object is a geospatial object represents a borehole's spatial feature in the real world. This special entity is used to organize the geospatial data.

Each borehole belongs to one project, and each project can have one or many boreholes reported in the corresponding project report. Therefore, the relationship between the PROJECT and

BOREHOLE entities is one-to-many. Similarly, each layer belongs to one borehole and each borehole contains one or many layers. The relationship between the BOREHOLE and the LAYER entities, and the relationship between the BOREHOLE entity and the SAMPLE entity is also one-to-many. Each borehole object corresponds with each borehole entity, so the relationship between the BOREHOLE OBJECT and BOREHOLE is one-to-one.

### **Spatial Data Model**

The spatial data model implemented in this system is shown in Figure 7.4. The root of this data model is the Geometry, which is the Geometry Data Model designed and implemented in previous chapters in this thesis. The Attribute Model stands for the Geotechnical Data Model implemented in Oracle. The combination of the spatial information from Geometry and the attribute information from Attribute generates the Simple Feature objects (the geospatial Metadata is not implemented in this system). The Feature Collection plays the role of Layer defined in GeoEye™ old version 1.0 to organize the similar Simple Features into one collection. Only the Map and GUIs in GeoEye™ 1.0 are kept in this spatial data model.

### **Connection of Models**

The Simple Feature object connects the two models in this system. The correspondences of Simple Feature's components Geometry and Attribute Model in the Spatial Data Model are the Borehole Object and other entities in the Geotechnical Data Model. If the system request data from Oracle, the data in the Borehole Object is transferred into the Geometry Data Model and the other attribute data is transferred into the Attribute Model, then they are combined together to generate the Simple Feature.

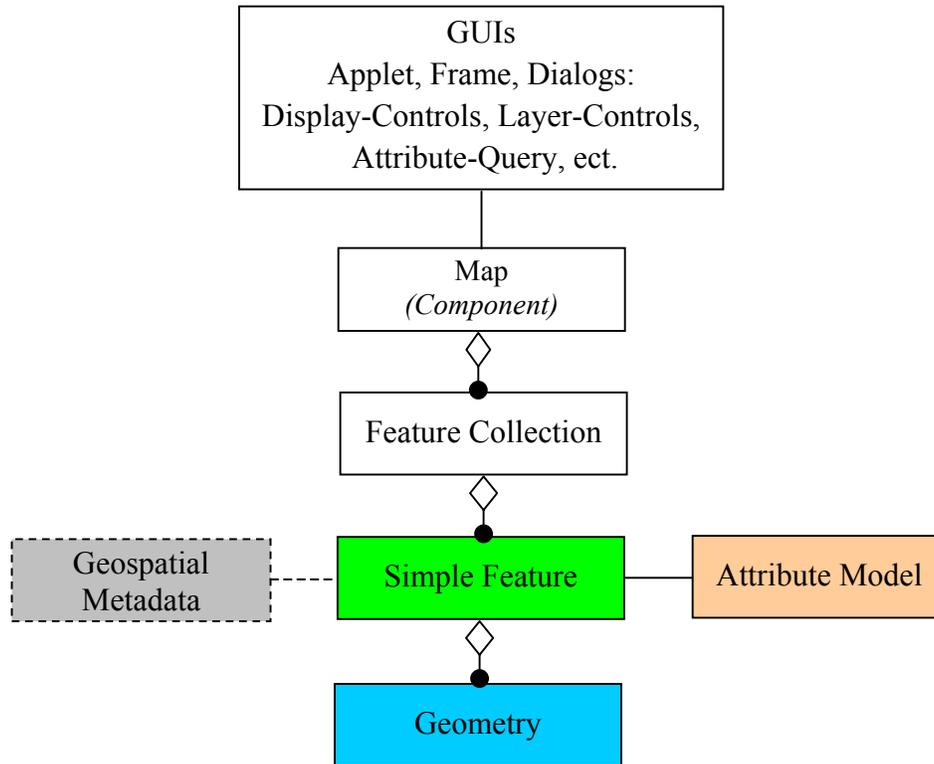


Figure 7.4 Geospatial Data Model  
*(See Appendix for data Model Notation)*

### 7.3 Implementation

#### 7.3.1 Database Implementation

The dispersed geotechnical databases have diversity in hardware, software, data model, and even the semantics. It's a big challenge to integrate these databases at a logical level in the distributed computing scenario. In this project, the data is transformed into one center-controlled database.

Data conversion is another issue in this case study. Many geotechnical and background datasets are in MicroStation's DGN or other CAD formats. However, many spatial datasets are in GIS

formats, such as ESRI Shapefile. There is a gap in the definitions of the objects in CAD and GIS. Some important information could get lost when transforming the CAD data to GIS data, such as transforming the DGN formatted data to ESRI's Shape data. It is more problematic that there was no standards regarding the geotechnical reports. Borehole sampling reports collected from various engineering companies are different in term of the diagram presentations, terminology used, items covered. It would cause huge efforts on populating these datasets into a database.

The various formatted geotechnical datasets are translated into ESRI's Shape File format and loaded into Oracle database. Based on the designed database data model detailed above, the geotechnical data generated from field tests and laboratory tests is loaded into Oracle and organized in several database view schemes in Table 7.1.

Table 7.1 View Schemes in Geotechnical Database

<b>View Name</b>	<b>Contained Item Number</b>
Field Test View	18
Shear Strength View	16
Consistency View	11
Consolidation View	11
Permeability View	6
Unit Weight View	9

### 7.3.2 GIS Prototype Implementation

The prototype system is composed of two coordinated components: client and server. The client viewer GeoEye<sup>TM</sup> is changed from the old version GeoEye<sup>TM</sup> 1.0 developed by Mr. Shuxin Yuan

(Yuan, 2000) by replacing the underlying data model to the Geometry Data Model implemented from the design of OpenGIS Simple Features Implementation Specification for Java. Some new functions are developed in this system.

### *New GeoEye™ Implementation*

GeoEye™ 1.0 is implemented as a Java Applet and is initiated in a general Web browser. Due to the underlying Spatial Data Model is changed and a new Geotechnical Data Model is introduced in the new system, GeoEye™ 1.0 has been done significant changes to keep these functions in Table 7.2. The framework of the spatial data model architecture and some algorithms in GUI and Map objects in the old version are used in the new GeoEye™.

It provides two data access scenarios: loading data from server side and client side. ESRI's Shape File formatted data file can be directly loaded into the system from local machine. If client request data from server side, the server retrieves the desired data from Oracle and generates it as geometry objects, then transfers them to the client. The Java's *ObjectSerialization* mechanism is adopted in the communication between the client and server (see Figure 7.5). The user interface of this system is illustrated in Figure 7.6.

The general GIS functions are not enough for this prototype system, many geotechnical domain functions, therefore, are necessary. Due to the limited time, a very simple Profile Analysis component is developed and added into the system. When the user draws a profile line on screen, this component will automatically select the objects crossed by the profile line, and a profile will be drawn in the pop up window (see Figure 7.7).

Table 7.2 GeoEye™ 1.0 Functions (Yuan, 2000)

Geodata Access	<ul style="list-style-type: none"> <li>- Local and Remote (Web Server) Shape File access</li> <li>- Access both spatial and non-spatial data</li> </ul>
Image layer support	<ul style="list-style-type: none"> <li>- Support GIF and JPEG images</li> <li>- Georeference image as map background</li> </ul>
Map display control	<ul style="list-style-type: none"> <li>- Repaint, Zoom (In/Out/Window/Extent), and Pan.</li> </ul>
Map overview window	<ul style="list-style-type: none"> <li>- Overview window On/Off, Resize, and Relocate</li> <li>- Layer On/Off in overview window</li> <li>- Display and Drag current map display position, etc.</li> </ul>
Layer control	<ul style="list-style-type: none"> <li>- Layer On/Off, Add/Remove Layers</li> <li>- Layer Reorder, Rename, Change Layer Color, etc.</li> </ul>
Attributes manipulation	<ul style="list-style-type: none"> <li>- Identify, Select by Point/Box, Attribute Table, etc.</li> </ul>
Drawings	<ul style="list-style-type: none"> <li>- Draw and Save Point, Polyline, and Polygon object.</li> </ul>
Project Management	<ul style="list-style-type: none"> <li>- Save and Reload project file.</li> </ul>

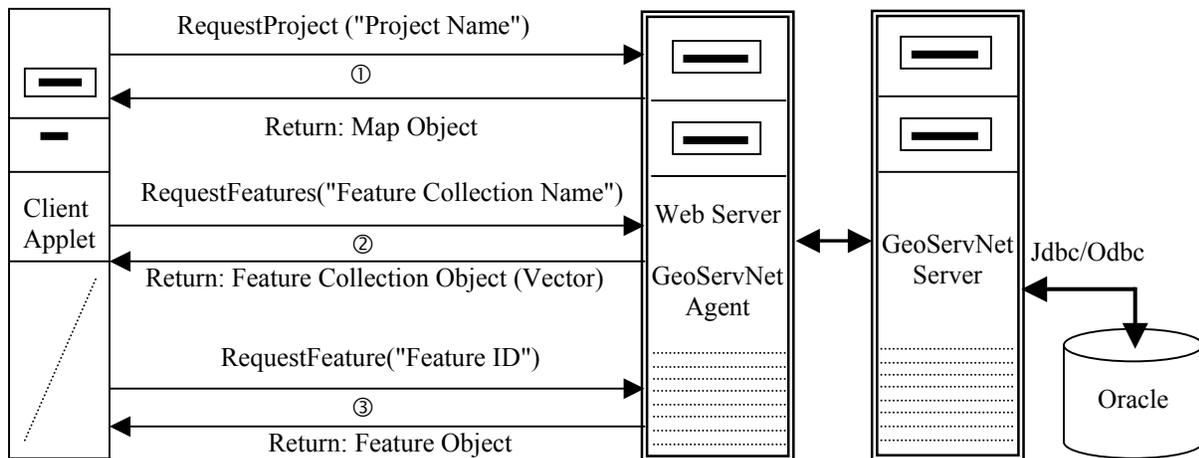


Figure 7.5 Object Communication between Client and Server

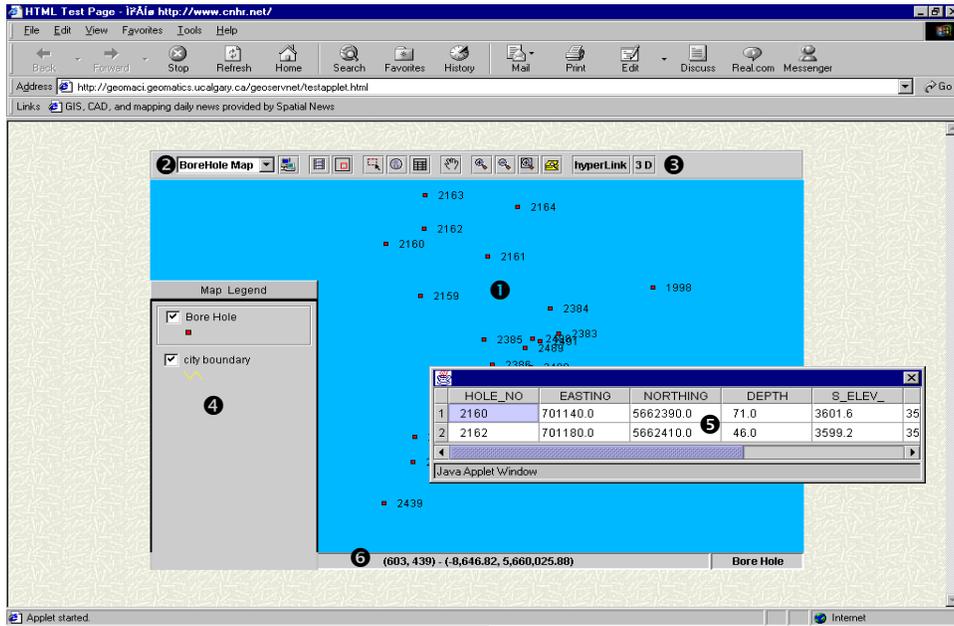


Figure 7.6 New GeoEye™ User Interface

(1): Map display area (2): Data Access (3): Tool Bar (4): Layer Control (5): Attribute display window for Identify (6): Status bar that displays mouse coordinates, scale and current layer)

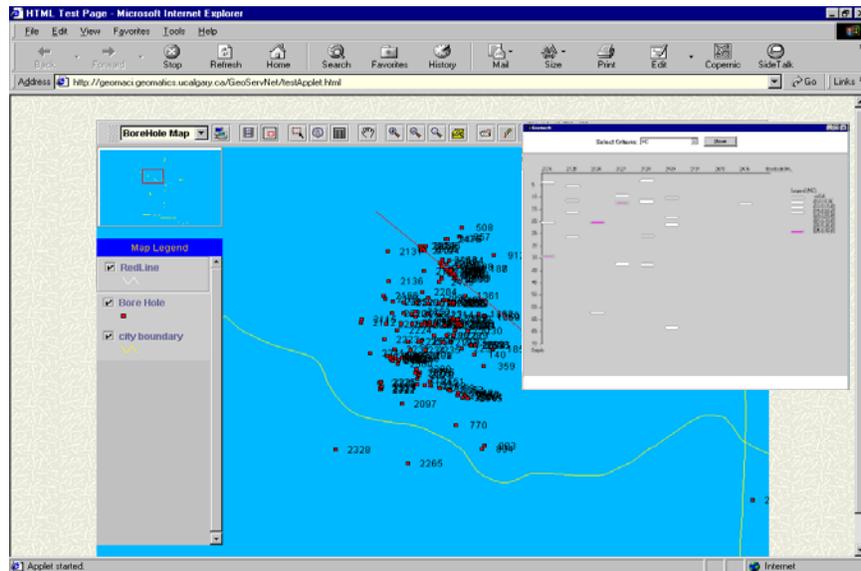


Figure 7.7 Interface of Profile Analysis

## GeoServNet Implementation

GeoServNet has two components: GeoServNet Server and GeoServNet Agent. The two server are developed as Java Servlets (Java's server side engine and APIs) and run inside the Java Servlet Engine and Web Server at the server side.

GeoServNet Server is in charge of data publishing. It retrieves the attribute and spatial data from Oracle database to generate simple features based on the client's request. To improve the data loading performance, all the published datasets are registered in this server. Some general information of datasets, such as boundaries and spatial indices, is recorded by the server. The interface of the dataset registration is illustrated in Figure 7.8.

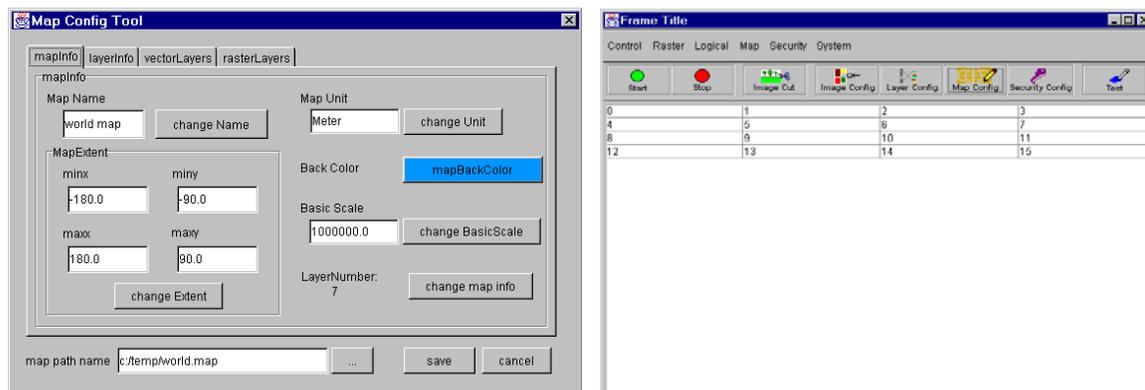


Figure 7.8 Data Registration Interfaces in GeoServNet Server

All client requests are accepted by GeoServNet Agent. The location information of the available datasets is recorded by this agent. It passes the data request to the proper GeoServNet Server to retrieve the desired datasets, then transfers the responses from GeoServNet Server to the client, who launched the data query.

## 7.4 Case Conclusions

A prototype system has been developed based on the design of the OpenGIS Simple Features Implementation Specification for Java. The new Geotechnical Data Model and Spatial Data Model are designed and implemented in this system. The simple feature is the primary object in the Spatial Data Model, which is simplified from the Simple Feature defined in OGC specifications by implementing the Geometry and Attribute Model. The Geometry is the designed and implemented Geometry Data Model.

This case study shows that the design is implementable. The Geometry Data Model implemented from the design of OpenGIS Simple Features Implementation Specification for Java is used in this case study to organize the spatial data. The methods implemented in the geometry objects are easily invoked by other objects outside of the Geometry Data Model. The integration of Geometry Data Model with other components in the system is straightforward. The new version of GeoEye<sup>TM</sup> works well in this case study.

This case study shows that the design is reasonable. The definitions of the geometry objects are complete with no ambiguity. The relationship between the geometry objects defined in the design is clear and logical. New geometry objects not included in this design can be easily integrated into this model. The organization of the design is simple to facilitate the developing of a system based on this design. The developer can easily locate the desired geometry objects.

## CHAPTER 8 CONCLUSIONS AND RECOMMENDATIONS

### 8.1 Conclusions

Geocomputing has migrated from the desktop to a networking environment. GIS software architecture has been evolving from stand-alone systems to client/server systems, and from client/server systems to ubiquitous computing systems. The Distributed GIS, including Web GIS, Internet GIS and Ubiquitous GIS, is becoming the mainstream in GIS industry. The boom of the Internet stimulates the huge demands for distributed GIS applications.

In a DGIS computing environment, interoperability is recognized as the major issue. In most cases, it's difficult to tell the difference between openness and interoperability in concept. Therefore, we often treat them as the same problem. Interoperability can be achieved at several levels. Many efforts have been made with respect to geodata interoperability; from metadata standards, such as the metadata standards from FGDC and ISO TC/211, to spatial data coding standards, FIPS (Federal Information Processing Standards) and SDTS (Spatial Data Transfer Standard). Research on interoperability at the application and semantics level has gained interests in recent years. In the application domain, existing DGIS applications are still bound by some specific GIS software. It's still difficult to build up interoperability between systems. The functions, application models and code in one system can not be invoked or migrated to another system directly.

It is recognized that standards are the key to address interoperability problem, and support component technology. It is clear that the use of software engineering standards is not enough to ensure development of interoperable DGIS applications, application domain standards are also important in determining their fate. The standards from OGC are a response to this problem. OGC has tried to create widely accepted domain standards for the GIS community. The open framework from OGC defines the blue print for GIS software, applications and services from the whole GIS community's viewpoint. The Geometry Data Model in OGC's nine worlds (see Figure 1.2) is the key component in OGC's standards system. The first released implementation specification of OGC is the simple feature implementation specification, which defines the Geometry Data Model and Spatial Reference System.

The existing released simple feature implementation specifications are for OLE/COM, SQL and CORBA. There is no OGC official specification for Java. In fact, Java is another important distributed computing platform and is becoming more and more popular as an open source language. The Java version Implementation Specification of OpenGIS Simple Features is an important family member in OGC's specifications.

This research, A Java Implementation for OpenGIS Simple Feature Specification, brings the following benefits to the GIS community:

- Defines a common data model based on OpenGIS and Java standards. A new OpenGIS Simple Feature Implementation Specification for Java is designed in this research. The Geometry Data Model is defined under the OGC's standards framework in Java's conventions. The common data model enriches the OGC's standard system to support the Java distributed computing platform.

- Push the development of DGIS and component based GIS in the GIS software industry. The DGIS and GIS components can be easily integrated and interoperable because they adopted the same underlying Geometry Data Model designed in this thesis. GIS component can be integrated with other non-GIS systems to provide spatial functionality. A component based GIS uses reusable GIS components to reduce GIS software's development cost; quicken development cycle and increase the software's flexibility and interoperability.
- Quicken the development of GIS systems and applications. The system integrator and application developer build up an application system is faster based on this common data model and other OGC's standards, such as GML (Geography Markup Language) and WFS (Web Feature Service).
- Provide a basis for interoperability. A common data model supports interoperability at the data model level, which is higher than geodata interoperability and lower than application and semantics interoperability. The designed data model in this research can push the realization of the geodata interoperability and help address application and semantics interoperability.

A number of conclusions can be drawn from the work presented herein:

- An OpenGIS Simple Feature Implementation Specification for Java is designed. Compared with related previous works reviewed in this thesis, the geometry object classification logic in the Geometry Data Model of this implementation specification is clearer. The Geometry Data Model has high extensibility to embrace the new geometry objects. This implementation specification can be easily maintained, improved and distributed with the

support from the UML technology. Rational Rose 2000 was used in all the design procedures to standardize the modeling of the implementation specification.

- A conformance Testing Suite was developed. The testing suite ensures that the implementation practices of the implementation specification strictly follow the definitions in the particular specification. This Java Applet testing suite that accompanies this design ensure the integrity of this work. Both of the mandatory and optional functions defined by the implementation specification were successfully tested in this suite.
- A prototype system was developed. The prototype Geometry Data Model was developed based on the Java version Implementation Specification of OpenGIS Simple Features. The implemented Geometry Data Model was embedded into the case study's system, which demonstrated that this design is reasonable and implementable. The performance of the system is acceptable.
- The design is standard. In this design, the GIS domain technology and standards are from OGC's specifications; Java's coding standards and conventions were used to code the implementation specification; the standard modeling technology, UML, was adopted in the design. About 20000 lines (about 500 Kb) 100% pure Java source code developed for the Implementation Specification, prototype Geometry Data Model, Testing Suite and Case Study system followed the Java standards and coding conventions.
- A new buffer algorithm was introduced. The Template Union Model is flexible and extendable to create buffer zones. Different kinds of templates can be combined freely to perform complicated buffer operations. New templates can be easily added into the model for specific application's requirements.

The documents for the implementation specification and conformance testing suite are compiled and will be revised and improved. It is our intention that the document can be submitted to OGC's Technique Committee for review and comments.

## **8.2 Recommendations**

The research on GIS standards is primary for GIS. This research addresses the more basic topic in the standard's research: Geometry Data Model. The value of the geometry data model for GIS is comparable to the cell in relation to a human's body. A lot of issues under this topic are worth of further research.

### (1) Topological Relationship

OGC moves away from an essential property of spatial data: *Intrinsic topological relationships*. Shared co-ordinates only have to be stored once and when correctly referenced reveal spatial relations by simple searching in stead of computer intensive relationship calculations (Cattenstart, 1998). Two neighboring polygons have a common boundary, for example, the boundary is stored separately for each polygon. The intrinsic topological relationship is removed.

### (2) Normalized Storage

OGC has classified geometries into simple and complex features. Rules for converting complex to simple geometries are absent. This leaves room for software developers to implement their own concepts in dealing with certain types of geometries. This situation calls for a further extension of geometry definitions and rules to convert from complex to simple entities in the

OpenGIS Data Model architecture. Normalized storage of geometries is absent as well as intrinsic topological relationship.

### (3) Measurement Features

The Simple Features defined in this implementation are still coordinate-based, which makes it impossible to apply traditional error analysis and estimate uncertainties in derived products (Goodchild, 2000). Measurement-based features should be imported into this model.

### (4) Professional Review

Interoperable GIS software must be open to peer review within the scientific community and open to continuous process improvement. The software should have embedded discipline-specific processes to provide credited functionality (Kottman 1994), and make it conform to “professional standards” or “best accepted professional practice” in software’s internals. It answers the question “why should we trust the output from the software”. Today’s commercial GIS primitives are seldom open or standard. The professional review mechanism should be imported into OGC’s standard framework.

### (5) Bridges between Different DCPs

There are three released Implementation Specifications for this topic. Here we propose the fourth for OGC. Those specifications are focusing on different Distributed Computing Platforms (DCPs). There are no “bridges” between the different DCPs (Cuthbert, 1999). Supporting multiple DCPs becomes increasingly untenable. The problem presents another interoperability issue to OGC.

## REFERENCES

- Abel, D. J. (1998). Towards integrated geographical information processing, *International Journal of Geographical Information Science*, 12(4), 353-371
- Adams, T. M., Tang, A. Y. S. and Wiegand, N. (1993). Spatial Data Models for Managing Subsurface Data, *Journal of Computing in Civil Engineering*, 7(3) 260-277
- Andrews, D. S., snoeyink, J. and Boritz, J. (1994). Further Comparison of Algorithms for Geometric Intersections Problems, *Proceedings 6<sup>th</sup> International Symposium on Spatial Data Handling*, 709-724
- Autodesk, (1997). Autodesk MapGuide: State-of-the-art network-centric GIS application architecture for publishing and accessing geodata, A White Paper Series of Autodesk Inc., Retrieved December 1999 from the World Wide Web: <http://www.autodesk.com/solution/gis/whtpaper/index.htm>
- Balaban, I. J. (1995). An Optimal Algorithm for Finding Segments Intersections, *Proceedings 11<sup>th</sup> Annual ACM Symposium on Computational Geometry*, 211-219
- Berg, M., van Kreveld, M., Overmars, M. and Schwarzkopf, O. (1997). Computational Geometry, Algorithms and Applications, Springer-Verlag, Berlin
- Bishr, Y. (1998). Overcoming the Semantic and Other Barriers to GIS Interoperability, *International Journal of Geographical Information Science*, 12(4), 299-314
- Bishr, Y.A., Pundt, H., Kuhn, W. & Rdwan, M. (1999). Probing the concept of information communities – A first step toward semantic interoperability, in Goodchild, M.,

- Egenhofer, M., Fegeas, R. & Kottman, C. (Ed.), *Interoperating Geographic Information Systems*, (pp. 55-69). Boston/Dordrecht/London: Kluwer Academic Publishers
- Bourke, P. (1989). Intersection Point of Two Lines, Retrieved February 9, 2001 from the World Wide Web: <http://astronomy.swin.edu.au/pbourke/geometry/lineline2d/>
- Buehler, K. (1998). OpenGIS Technology Development Overview, OpenGIS Consortium Presentations, Retrieved July 17, 2000 from the World Wide Web: <http://www.opengis.org/techno/presentations/over-view/index.htm>
- Buehler, K. & McKee, L. (Ed.). (1998). *The OpenGIS Guide: Introduction to Interoperable Geoprocessing and the OpenGIS Specification* (3rd ed.). Massachusetts: Open GIS Consortium.
- Cattenstart, G. C. (1998). Open Boundaries in GIS: OpenGIS and the SDO and SDE implementation. MSc dissertation at the Manchester Metropolitan University in conjunction with the Free University of Amsterdam.
- Chan, T. (1994). A simple Trapezoid Sweep Algorithm for Reporting Red/Blue Segment Intersections, *Proceedings 6<sup>th</sup> Canadian Conference on Computational Geometry*, 263-268
- Charles, B., et al. (1999). Adding an Interoperable Server Interface to a Spatial Database: Implementation Experiences with OpenMap, in A. Vckovski, K. E. Brassel & H. J. Schek (Ed.), *Interoperating Geographic Information Systems, Proceedings of Second International Conference: INTEROP'99* (pp. RT115-128), Zurich, Switzerland

- Chazelle, B. and Edelsbrunner, H. (1992). An Optimal Algorithm for Intersection Line Segments in the Plane, *Journal of ACM*, 39(1), 1-54
- Cheng, C. Q. and Yuan, W. (2001). A New Algorithm of Buffer Based on Unit-combined Model, *Journal of Image and Graphics (Chinese)* (accepted)
- Cigale, B. and Zalik, B. (1999). A Simple Polygon Splitting Algorithm, *Proceedings 15<sup>th</sup> Spring Conference on Computer Graphics*, Budmerice, Slovakia, 239-246
- Clementini, E. and DiFelice, P. (1994) A Comparison of Methods for Representing Topological Relationships, *Information Sciences*, 80, 1-34
- Clementini, E. and DiFelice, P. (1996). A Model for Representing Topological Relationships Between Complex Geometric Features in Spatial Databases, *Information Sciences* 90 (1-4), 121-136
- Clementini, E., DiFelice, P. and Califano, G. (1995). Composite Regions in Topological Queries, *Information Systems*, 20(6), 33-48
- Clementini, E., DiFelice, P. and Oostrom, P. (1993). A Small Set of Formal Topological Relationships Suitable for End-User Interaction, in D. Abel and B. C. Ooi (Ed.), *Advances in Spatial Databases — Third International Symposium, SSD'93*, LNCS 692, 277-295, Springer-Verlag, Singapore
- Coppock, J. and Rhind, D. (1995). The History of GIS in Geographical Information Systems - Principles and Applications, in D. Maguire, M. F. Goodchild and D. Rhind (Ed.), New York, 21-43
- Cornell, G. and Horstmann, C. S. (1997). Core Java (2<sup>nd</sup> Ed.), SunSoft Press

- Cuthbert, A. (1999). OpenGIS: Tales from a Small Market Town, in Vckovski, A. Brassel, K. E. and Schek, J. (Ed.), *Interoperating Geographic Information Systems*, Second International Conference, INTEROP'99, LNCS 1580, 17-28, Springer-Verlag, Singapore
- Egenhofer, M.J., Clementini, E. and Di Felice, P. (1994). Topological relations between regions with holes, *International Journal of Geographical Information Systems*, 8(2), 129-142
- Egenhofer, M. J. and Franzosa, R. (1991). Point-set topological spatial relations. *International Journal of Geographical Information Systems*, 5, 161-174
- Egenhofer, M.J., Glasgow, J., Gunther, O., Herring, J. R. and Peuquet, D. A. (1999). Progress in Computational Methods for Representing Geographical Concepts, *International Journal of Geographical Information Science*, 13(8), 775-796
- Egenhofer, M.J. and Herring, J. (1990). A Mathematical Framework for the Definition of Topological Relationships. *Proceedings of the Fourth International Symposium on Spatial Data Handling*, Zurich, Switzerland, 803-813.
- Egenhofer, M.J. and Sharma, J. (1993). Topological Relations between regions in  $\mathbb{R}^2$  and  $Z^2$ , in D. Abel and B. C. Ooi (Ed.), *Advances in Spatial Databases — Third International Symposium, SSD'93*, LNCS 692, 36-52, Springer Verlag, Singapore
- Elyes, N. and Doughty, M. (1997). Geoscientific Information Systems for Environmental Geology, *Environmental Geology of Urban Areas*, Geological Association of Canada, 495-506
- Frank, A. (1991). Aualitative Spatial Reasoning About Cardinal Directions, in D. Mark and D. White (Ed.), *A Proceedings of Autocarto 10* (pp. 148-167), Baltimore, MD

- Giles, D. (1994). A Geographical Information System for Geotechnical and Ground Investigation Data Management and Analysis, Retrieved on April 18, 2001 from the World Wide Web: <http://www.sgi.ursus.maine.edu/gisweb/spatdb/egis/eg94087.html>
- Goodchild, M. F. (1999). Measurement-based GIS. In W. Shi, M. Goodchild & P. Fisher (Ed.), *Proceedings of The International Symposium on Spatial Data Quality'99* (pp. RT1-9)
- Goodman, J. E. and O'Rourke, J. (1997). *Handbook of Discrete and Computational Geometry*, CRC Press LLC
- Hernandez, D., Clementini, E. and Di Felice, P. (1995). Qualitative Distances, in A. Frank and W. Kuhn (Ed.), *Spatial Information Theory – A Theoretical Basis for GIS, International Conference COSIT'95*, LNCS 988, 45-57, Springer Verlag, Berlin
- Karimi, H. A. (1997). Components of Interoperable Geographic Information Systems. Technical Report: *Interoperating GISs*, Ed. by M. F. Goodchild, M. J. Egenhofer & R. Fegeas.
- Kottman, C. A. (1994). Open GIS Software: An Industry View, *StandardView*, 2(3), 159-162
- Kramer, J. (1994). Distributed Software Engineering, 16th IEEE International Conference on Software Engineering (ICSE-16), 253-263, Sorrento
- Lee, F. H., Tan, T. S., Karunaratne, G. P. and Lee, S. L. (1990). Geotechnical Data Management System, *Journal of Computing in Civil Engineering*, 4(3), 239-254
- Leung, Y., Leung, K. S. and He, J. Z., 1999, A Generic Concept-based Object-oriented Geographical Information System, *International Journal of Geographical Information Science*, Vol. 13, No. 5, 1999, pp475-498

- Limp, W. F. (2001). User Needs Drive Web Mapping Product Selection. *GeoWorld*, 14(2), 8-18
- McKee, L. (2000). Geo-semantics and the "New Renaissance". *GEO AsiaPacific*, April, 2000.  
Retrieved July 11, 2000 from the World Wide Web:  
<http://www.opengis.org/info/gisworld/GeoAsiaPacific/GAP.htm>
- Min, W., Sheng, C. and Tang, Z. (1991). An Improved Plane-sweep Algorithm for Line Segment Intersections, in Staudhammer, J. and Peng, Q. (Ed.), *CAD/Graphics'91*, 192-197
- Oloufa, A. A. and Eltahan, A. A. (1992). Geotechnical Data Management: A GIS-based Approach, *Computing in Civil Engineering*, 590-597.
- Open GIS Consortium Inc. (1996). *The OpenGIS<sup>®</sup> Abstract Specification: An Object Model for Interoperable Geoprocessing*. (Document No. 96-015R1). Retrieved May 28, 2001 from the World Wide Web: <http://www.opengis.org/techno/>
- Open GIS Consortium Inc. (1998a). *The OpenGIS<sup>®</sup> Guide – Introduction to Interoperable Geoprocessing*. Retrieved April 21, 2001 from the World Wide Web:  
<http://www.opengis.org/techno/guide.htm>
- Open GIS Consortium Inc. (1998b). *The OpenGIS<sup>®</sup> Simple Features Specification For CORBA Revision 1.0*. Retrieved May 28, 2001 from the World Wide Web:  
<http://www.opengis.org/techno/>
- Open GIS Consortium Inc. (1999a). *The OpenGIS<sup>®</sup> Simple Features Specification For SQL Revision 1.1*. (Document No. 99-049). Retrieved May 28, 2001 from the World Wide Web: <http://www.opengis.org/techno/>

- Open GIS Consortium Inc. (1999b). *The OpenGIS<sup>®</sup> Simple Features Specification For OLE/COM Revision 1.1*. (Document No. 99-050). Retrieved May 28, 2001 from the World Wide Web: <http://www.opengis.org/techno/>
- Open GIS Consortium Inc. (1999c). *Conformance Test Guidelines for OpenGIS<sup>®</sup> Simple Features Specification for COM*. (Document No. 99-037r2). Retrieved June 12, 2001 from World Wide Web: <http://www.opengis.org/techno/conformance.htm>
- O'Rourke, J. (1993). *Computational Geometry in C*, Cambridge University Press, Cambridge
- Papacostas, C. S. (1994). Geotechnical Database, *Journal of Construction Engineering and Management*, 120(1), 211-221
- Peuquet, D.(1992). An Algorithm for Calculating Minimum Euclidean Distance Between Two Geographic Features, *Computers and Geosciences*, 18, 989-1001
- Peuquet, D. and Zhan, C. X. (1987). An Algorithm to Determine the Directional Relationship between Arbitrarily-shaped Polygons in the Plane, *Pattern Recognition*, 20, 65-74
- Preparata, F. P. and shamos, M. I. (1988). *Computational Geometry: an Introduction* (2<sup>nd</sup> Ed.), Springer-Verlag, New York
- Raj, G. S. (2001). A Detailed Comparison of CORBA, DCOM and Java/RMI, Retrieved January 12, 2001 from World Wide Web: <http://www.execpc.com/~gopalan/misc/compare.html>
- Retz-Schmidt, G. (1988). Various Views on Spatial Prepositions, *AI Magazine*, 9, 95-105
- Schutte, K. (1995). An Edge Labeling Approach to Concave Polygon Clipping, Preprint submitted 7 July 1995 to ACM Transactions on Graphics, Retrieved Feb. 2, 2001 from World Wide Web: <http://www.ph.tn.tudelft.nl/~klamer/clip.ps.gz>

Sf4java Newsgroup, <http://groups.yahoo.com/group/sf4java/>, last check on May 25, 2001

Sheth, P. A. (1999). Changing Focus on Interoperability in Information Systems: From System, Syntax, Structure to Semantics, in Goodchild, M., Egenhofer, M., Fegeas, R. & Kottman, C. (Ed.), *Interoperating Geographic Information Systems*, (pp. 5-29). Boston/Dordrecht/London: Kluwer Academic Publishers

Sun Microsystem. (1999). *Code Conventions for the Java™ Programming Language*. Retrieved June 22, 2001 from World Wide Web:  
<http://java.sun.com/docs/codeconv/index.html>

Tao, C.V. (2000). Development of Internet-based GIServices, *Proceedings of GIS2000 Conference* (CD-ROM), Toronto, Canada, March 14-16

Tao, C.V., Fei, C. and Wong, R. (2001). An Internet-GIS Based Geotechnical Data Sharing and Analysis System, *Proceedings of 2001 Earth Odyssey, 54<sup>th</sup> Canadian Geotechnical Conference & 2<sup>nd</sup> Joint IAT and CGS Groundwater Conference* (CD-ROM), Calgary, Canada, September 16-19

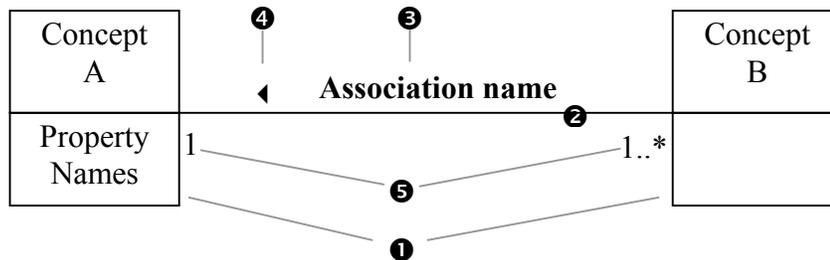
Tao, C.V., and Yuan, S. (2000a). GeoServNet: Renting Geotools over Internet, *Geo-Informatics*, 14(12), 12-15

Tao, C.V., and Yuan, S. (2000b). Developing of a GIS Service Model, *Proceedings of GITA 2000 Conference*, Denver, USA, March 26-29

- Vckovski, A. (1999). Interoperability and spatial information theory. in Goodchild, M., Egenhofer, M., Fegeas, R. & Kottman, C. (Ed.), *Interoperating Geographic Information Systems*, (pp. 31-37). Boston/Dordrecht/London: Kluwer Academic Publishers
- Yuan, S. (2000). Development of A Distributed Geoprocessing Service Model, M. Sc. Thesis, Department of Geomatics Engineering, University of Calgary
- Yun, S. (1999). The Internet GIS Infrastructure for Interoperability: MAP (Mapping Assistant Protocol), *GeoSolutions: Integrating Our World* (pp. RT41-45), Vancouver
- Zalik, B., Gombosi, M. and Podgorelec, D. (1998). A Quick Intersection Algorithm for Arbitrary Polygons, *Proceedings 14<sup>th</sup> Spring Conference on Computer Graphics*, Budmerice, Slovakia, 195-204
- Zalik, B. and Claphworthy, J. G. (1999). A Universal Trapezoidation Algorithm for Planar Polygons, *Computer & Graphics*, 23(3), 353-363
- Zalik, B. (2000). Two Efficient Algorithms for Determining Intersection Points Between Simple Polygons, *Computer & Geosciences*, 26(2000), 137-151

## APPENDIX DATA MODEL NOTATION

### 1. Conceptual Model Notation



In the above diagram:

❶: Concepts or types in a conceptual model. The upper part of the rectangle box gives the name of the concept and the lower part list the important properties of the concept. In many cases, the property name could be empty.

❷: The association between two concepts. In this thesis, dotted lines represent association between remote objects.

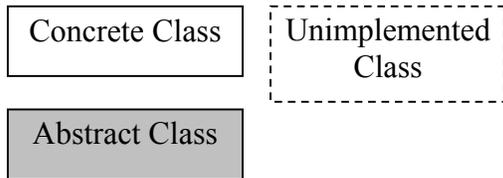
❸: Association name: The name that describe the association between two concepts.

❹: Direction of association: It gives the direction to read the association. The default direction of an association is from left to right or from top to bottom, in which the arrow can be ignored.

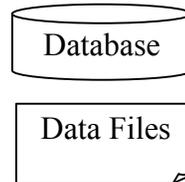
❺: Multiplicity of an association. For example: "1" means only one, "\*" means many, "1..\*" means 1 or more, and so forth.

## 2. Class Diagram Notation

Class Notation

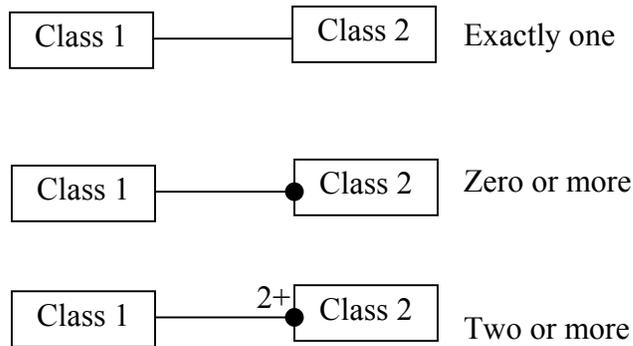


Dataset

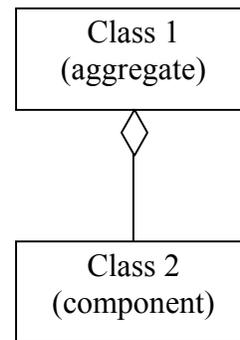


Association

Multiplicity



Aggregation



Inheritance

