A PROTYPE of FEATURE-ORIENTED GIS

by

JINMU CHOI

(Under the Direction of E. Lynn Usery)

ABSTRACT

With an emergence of geographic information systems (GIS), spatial data have been represented by a traditional layer-based data model using geometry and thematic attributes. In the real world, geographic phenomena also possess relationships among them and temporal attributes. Since late 1980, a feature concept has been developed that uses three dimensions for representation: space, theme, and time. A feature approach in developing GIS has the potential for more accurate representation of geographic reality because the features correspond to the basic level of human cognitive observation. The aim of this study is to derive a feature data model and to develop a feature-oriented GIS (FOGIS). The objectives are 1) to apply the object-oriented model and feature concepts to GIS and construct a feature data model, 2) to develop a spatial data input procedure using an interactive rule-based expert system, 3) to apply the input procedure for spatial data generation, 4) to design and implement a FOGIS architecture, and 5) to demonstrate the utility of FOGIS by entering, storing, analyzing, and displaying spatial data for multiple types of geographic features. A feature data model was derived from the feature concept. Classes in the feature data model were derived using abstraction methods in Spatial Data Transfer Standard (SDTS) and key concepts of object-orientation. A prototype system was then developed using the feature data model. This system consists of an input module, an

import/export module, a feature database, and a display module. Specifically, the input module includes the expert system to provide a guideline to users for producing land use/cover data through an interactive question-and-answer sequence. The display module can be used to effectively visualize the hierarchical structure of objects in the feature data model. The developed system was applied for several geographic applications. The input module with an expert system was applied for thematic mapping of land use/cover in Athens, Georgia. The FOGIS was applied for the visualization of multi-temporal representations of buildings and coastlines in Camp Lejeune Marine Corps Base, North Carolina. Also, thematic relationships were used for spatial query on the roads in Camp Lejeune area.

INDEX WORDS: feature data model, expert system, feature database, multi-temporal representation, thematic relationship

A PROTOTYPE OF FEATURE-ORIENTED GIS

by

JINMU CHOI

B.A., Seoul National University, Korea, 1993

M.A., Seoul National University, Korea, 1998

A Dissertation Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

ATHENS, GEORGIA

2004

A PROTOTYPE OF FEATURE-ORIENTED SYSTEM

by

JINMU CHOI

Major Professor:	E. Lynn Usery
Committee:	Chor-pang Lo Thomas W. Hodler Marguerite Madden Xiaobai Yao

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
December 2004

Dedicated to My Parents

ACKNOWLEDGEMENTS

I am grateful to many people for their help and support. First of all, I would like to express my sincere appreciation to my advisor, Dr. E. Lynn Usery, for his unconditional support throughout my graduate study. This study would have been impossible without his constant support and guidance. His insightful comments made a big improvement of my dissertation research. He was always available to answer my questions, and so generous and considerate to understand all circumstances. I will always look upon his insights and integrity in my whole academic carrier. It was truly an honor to work with him for these years.

I wish to extend my sincere appreciation to the committee members, Drs Chor-pang Lo, Thomas W. Hodler, Marguerite Madden, Xiaobai Yao for their comments and suggestions on my study. I am particularly indebted to Dr. Lo for his help during the first year of my graduate study. He encouraged and supported me to start my new life in the U.S. without any problem. It was of great help for me to work with him for NASA grant. The financial support from the Center for Remote Sensing and Mapping Science (CRMS) and Department of Geography is gratefully acknowledged. I was very fortunate to be involved in several research projects in CRMS during my graduate study and it has been a grate pleasure to work with them.

There are so many people I would like to thank for their help. Thanks go to Drs Roy A. Welch, Tomas R. Jordan, Kavita K. Pandit for their generosity and encouragement during my graduate study at UGA. I also appreciate endless support and encouragement by friendly senior scholars, Drs Jeong-Chang Seong, Soohong Park, Cha Yong Ku, Chul-sue Hwang.

I would also like to express my gratitude to my fellow graduate students, to Byong-Woon Jun, Hwahwan Kim, Minho Kim, Hyeon Gang Jeong, Bryan Nicholson, Molly Azami, Yangrong Ling, Yanfen Le, and Bo Xu, who are always good friends. Thanks go to Ms. Audrey W. Hawkins, Ms. Emily Duggar, Ms. Loretta Scott, Ms. Jodie Guy, and Ms. Donna M. Johnson in the geography office for their help and kindness.

Finally, I would like to thank to my parents and families in both Phohang and Daejeon Cities in Korea for their understanding and patience over the past several years. Bora Kim, my wife, is such a wonderful woman and gives me a great family and support. Jimin, my little son, is the present and future to me. Without their love and patience, it would have not been able to accomplish my dissertation research. I deeply appreciate all their love and support, and will keep it forever.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS v

LIST OF TABLES x

LIST OF FIGURES xi

CHAPTER

1 RESEARCH OBJECTIVES AND BACKGROUND 1

1.1 Introduction 1

1.2 Research Objectives 2

1.3 Background 6

1.4 Research Methodology 10

1.5 Summary 19

1.6 Chapter Organization 20

2 OBJECT-ORIENTATION WITH VISUAL BASIC FOR FEATURE-ORIENTATION 23

2.1 Introduction 23

2.2 History of Object-Orientation 24

2.3 Key Elements of Object-Orientation 27

2.4 Objects and Classes 32

2.5 Summary 39

3 FEATURE DATA MODEL FOR FEATURE REPRESENTATION.....42

3.1 Early Data Model: Hierarchical and Network Data Model.....42

3.2 Relational Data Model.....45

3.3 Object-Oriented Model.....49

3.4 Feature Data Model with Object-Orientation for GIS.....51

3.5 Discussion: Feature Data Model vs. Existing Data Model in GIS.....63

3.6 Summary.....65

4 SYSTEM DEVELOPMENT WITH FEATURE DATA MODEL.....67

4.1 Introduction.....67

4.2 Class Design.....69

4.3 System Design and Implementation.....72

4.4 Feature Database.....76

4.5 Multi-Temporal Representation of a Coastal Area.....80

4.6 Summary.....85

5 SPATIAL DATA GENERATION WITH RULE-BASED EXPERT SYSTEM.....87

5.1 Introduction.....87

5.2 Integration of GIS and a Rule-Based Expert System: System Architecture.....90

5.3 System Development.....92

5.4 Application: Interpretation of Aerial Photographs.....96

5.5 Summary.....105

6 SYSTEM INTEGRATION FOR FEATURE-ORIENTED GIS (FOGIS).....107

6.1 System Architecture for FOGIS.....107

6.2 System Integration for FOGIS.....107

6.3 Application I: Feature Management using Time....117
6.4 Application II: Retrieval of Features using Thematic Relationship....122
6.5 Summary....126
7 CONCLUSION AND DISCUSSION....128
7.1 Conclusion....128
7.2 Contributions and Limitatins....132
7.3 Further Study....133
REFERENCES....135
APPENDICES....142
A CLASS MODULES AND COLLECTION CLASSES ON FEATURE IN FOGIS....142

LIST OF TABLES

Page

Table 1.1: Terminology for Feature and Object 4

Table 3.1: Standard Query Language (SQL) Commands (Kochhar and Lad, 1998) 47

Table 3.2: Terms for Spatial Data Abstraction in SDTS (USGS, 1997) 55

Table 3.3: Correspondence between Entities and Objects (USGS, 1997).....56

Table 4.1: Groups of Classes in Feature Data Model 69

Table 4.2: Data and Methods for Basic Geometry Classes 71

Table 5.1: Accuracy Assessment of Classified Land Use/Land Cover Data.....104

Table 7.1: Relation between Research Theme and Solving Problems 129

LIST OF FIGURES

Page

Figure 1.1: Prototype User Interface of Feature-Oriented GIS....2

Figure 1.2: Prototype User Interface of Input Procedure with Expert System in FOGIS 3

Figure 1.3: Research Flow for Feature-Oriented GIS and Geographic Applications....11

Figure 1.4: Feature with Time Attribute in a Feature Data Model....14

Figure 2.1: Common Data Type vs. Abstract Data Type 28

Figure 2.2: Qualifications of Public Property and Method....30

Figure 2.3: Object Creation in Visual Basic 34

Figure 2.4: Creating Classes in Visual Basic....36

Figure 3.1: File Structure with a Repeating Group (Hierarchical Structure in a File)....43

Figure 3.2: Decomposition without Repeating Group (Hierarchical Structure in Two Files)....43

Figure 3.3: Data Structure and Network Pointer Structure (Graham, 2001) 44

Figure 3.4: Relational Calculus and Algebra....47

Figure 3.5: Conceptual Framework of Geographic Feature (Usery, 1996b) 53

Figure 3.6: Feature Data Model using Object-Oriented Paradigm for Vector Data....59

Figure 3.7: Hierarchical Structure of Classes in a Feature Data Model 60

Figure 3.8: Feature Type and Feature Instances with their Own Relationships 61

Figure 3.9: Feature Representation with Two Temporal Attributes....62

Figure 4.1: Attribute Classes Inherited from Geometry and Type Group 70

Figure 4.2: System Design from Feature Data Model 73

Figure 4.3: Main GUI of the System for Feature Data Model....74
Figure 4.4: GUI for the Import/Export Module....74
Figure 4.5: GUI for the Display Module....75
Figure 4.6: Table Structure for Classes in Feature Database....79
Figure 4.7: Study Site....80
Figure 4.8: Floyd Hurricane Paths on the Study Area on September 16 in 1999....81
Figure 4.9: Source Data for Building Features with Different Time....82
Figure 4.10: Populating Features on Three Different Sources....83
Figure 4.11: Feature Query based on the Time....84
Figure 4.12: Temporal Differences of Buildings between 1998 and 1999....85
Figure 5.1: Architecture of Integrated System....90
Figure 5.2: Main Window of Hybrid System....93
Figure 5.3: Table Structure and Relations in the Database....95
Figure 5.4: Decision Tree for Rule Construction....97
Figure 5.5: Symbolization of Rules Decision Tree....98
Figure 5.6: Rules using If-then Structure....100
Figure 5.7: Processing Steps for Land Use using Expert System....101
Figure 5.8: Study Area for Expert Mapping....102
Figure 5.9: Land Use/Land Cover Mapping from Aerial Photographs....103
Figure 6.1: System Architecture for FOGIS....107
Figure 6.2: User Interfaces of FOGIS....108
Figure 6.3: Feature Generation Procedures in the Input Module....109
Figure 6.4: Import/Export Procedure in FOGIS....110

Figure 6.5: User Interface for Projection Selection 111

Figure 6.6: Table Structure in Feature Database from Feature Data Model.....113

Figure 6.7: Tree Viewer for Feature Data.....114

Figure 6.8: Tree Viewer and Display Module for a Feature Database 115

Figure 6.9: Feature Query using Feature Relationship 116

Figure 6.10: Camp Lejeune Study Area 117

Figure 6.11: Coastline Changes in Different Image Source in Different Time 118

Figure 6.12: Input Procedure for Populating Coastline 119

Figure 6.13: Coastlines Extracted from Three Different Time Image Sources 120

Figure 6.14: Multi-temporal Situation of Coastline Feature.....121

Figure 6.15: Roads in Camp Lejeune Study Area 123

Figure 6.16: Importing Procedures for Populating Road Features 124

Figure 6.17: Road Query using Feature Relationship (Thematic Relationship).....125

CHAPTER 1

RESEARCH OBJECTIVES AND BACKGROUND

1.1 Introduction

Since late 1980, a feature concept has been developed to represent geographic phenomena in geographic information systems (GIS) (Guptill *et al*., 1990; USGS, 1997). A feature representing a geographic phenomenon is an entity in a real world and an object in computer technology, and may include combinations of basic geometric elements. Features are the units of aggregation and analysis instead of a theme or layer (Usery, 1993).

In computer science, an object-oriented paradigm has been built and applied for both programming and data modeling (Booch, 1994; Graham, 2001). In a conventional programming paradigm, programming is separated from data models. However, objects in an object-oriented model have both data (status or attributes) and behavior (functions or methods). Objects are abstracted entities in the real world so that the object-orientation may be a better support for complex spatial data in GIS.

There are still some limitations in GIS even with its rapid advances. Some thematic relations such as 'part of' are not contained in a conventional layered model in GIS. A feature in different time periods cannot be saved as one object. However, it is possible to take advantage of the advances in a feature concept and an object-oriented paradigm in order to overcome some of the current limitations in GIS

1.2 Research Objectives

The primary goal of this research is to develop a prototype GIS that fully implements feature-oriented concepts based on object-oriented analysis, design, and implementation for spatial data input, storage, analysis, and output (Figure 1.1). The input procedure will generate spatial data for both conventional GIS and the developed system (feature-oriented GIS) in order to increase usability. The input procedure also will include the ability to generate spatial data based on an interactive rule-based expert system for neophytes in GIS (Figure 1.2).

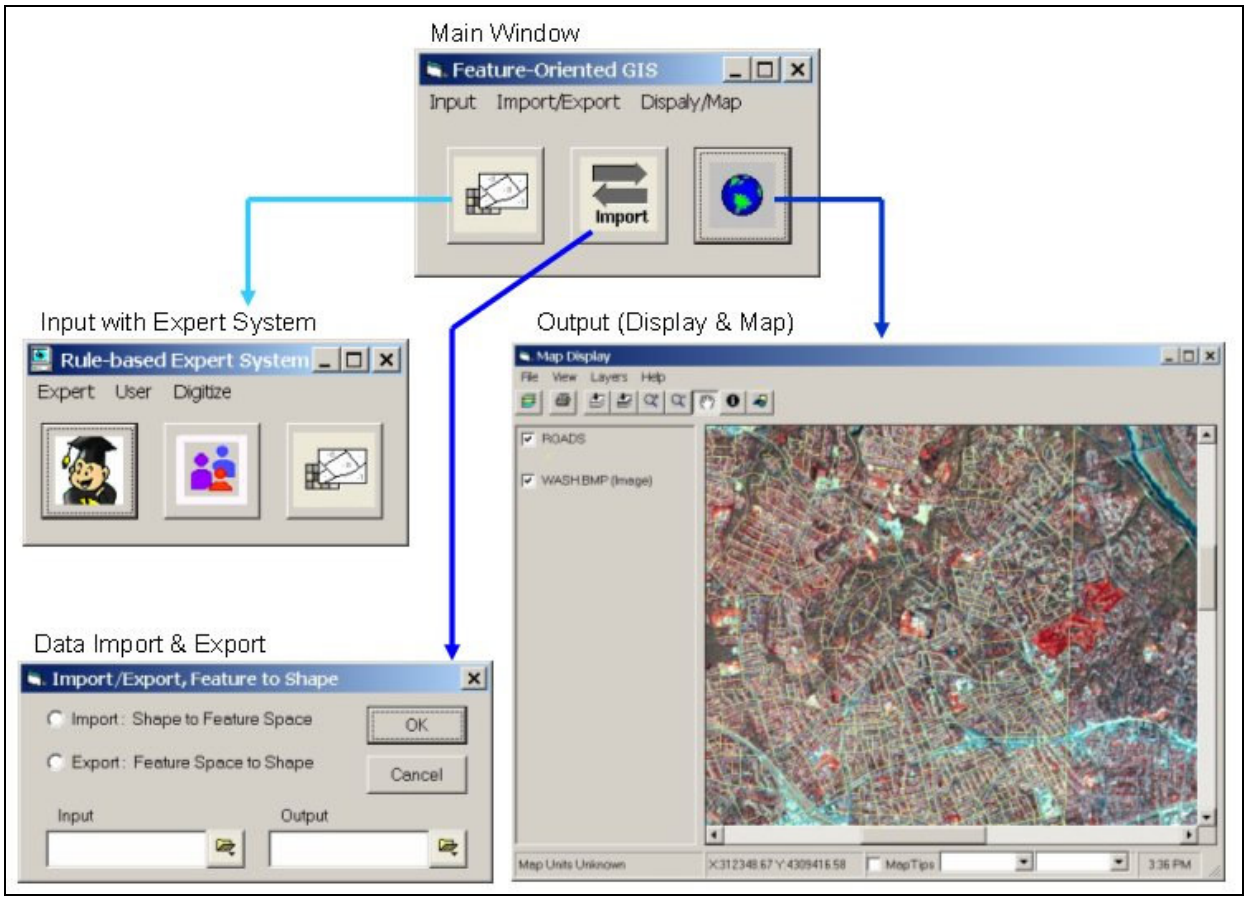


Figure 1.1 Prototype User Interface of Feature-Oriented GIS

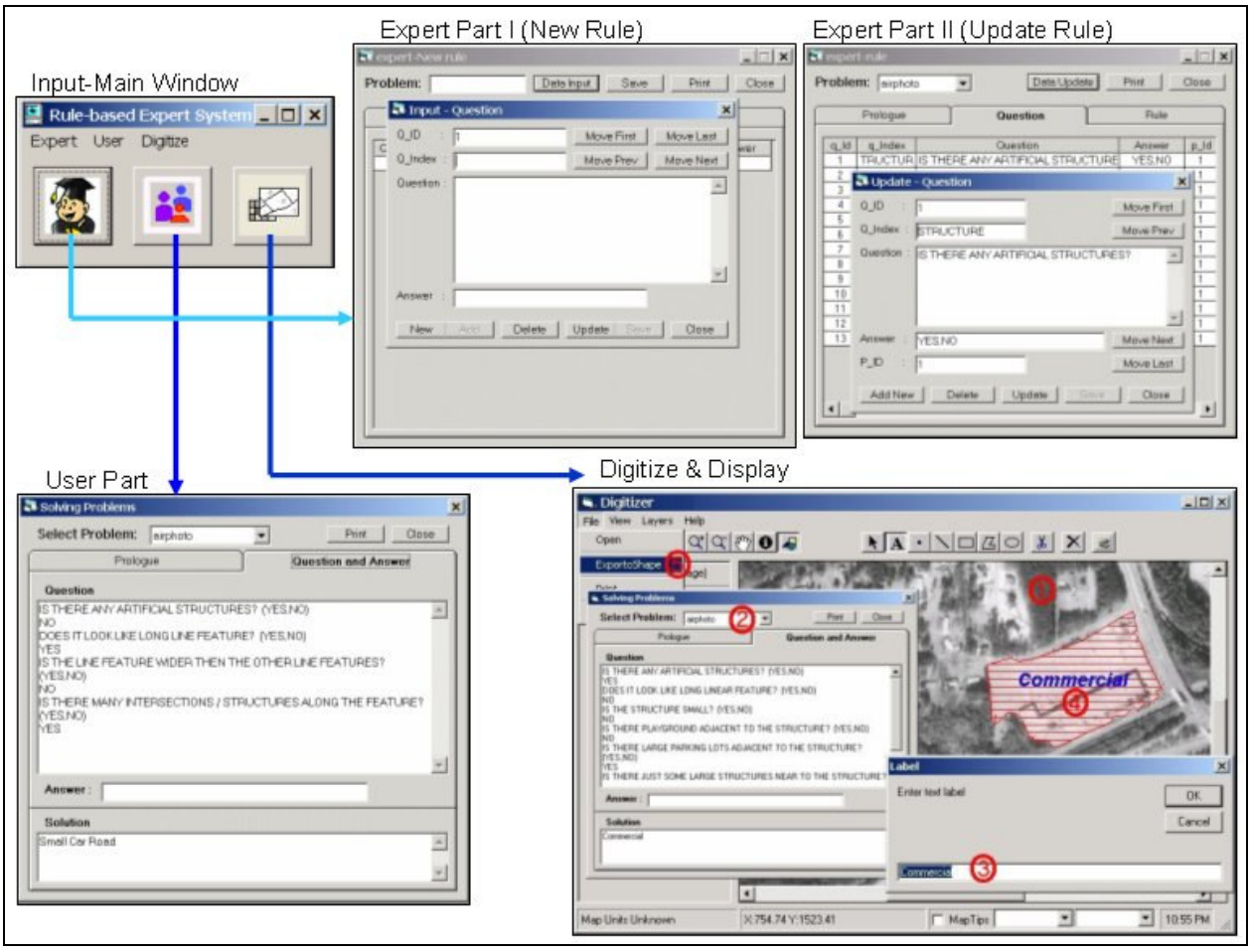


Figure 1.2 Prototype User Interface of Input Procedure with Expert System in FOGIS

The term, Feature-Oriented Geographic Information System (FOGIS), is defined as a geographic information system in which a feature can be entered, stored, analyzed, and displayed as an object wholly in the digital environment of a computer. Therefore, in FOGIS, the data model and programming methods are based on the object-oriented model. Also, several other terms used in this research are identified in Table 1.1.

Table 1.1 Terminology for Feature and Object

Term	Definition
Feature	• A feature represents geographic phenomena. • A feature is an entity in a real world and an object in the digital environment.
Object	• An object is an entity in the digital environment that combines the properties of methods and data.
Object-oriented approach	• The object-oriented approach is a method for developing systems and databases. • Object-orientation includes object-oriented analysis, object-oriented design, and object-oriented implementation using object-oriented programming.
Feature-oriented approach	• The feature-oriented approach is a method for developing a GIS using an object-oriented approach. • The feature-oriented approach uses object-orientation for both conceptual design (using object-oriented analysis and design) and programming for implementation.
Feature-based approach	• The feature-based approach is a method for developing GIS using both object-oriented approach and a relational model. • The feature-based approach uses object-orientation for conceptual design (using object-oriented analysis and design) and uses a functional programming for implementation.

Specifically, the objectives of this research are as follows:

1) To apply the object-oriented model and feature concepts to GIS and construct a feature data model;
2) To develop a spatial data input procedure using an interactive rule-based expert system in order to generate spatial data for both conventional GIS and FOGIS;
3) To demonstrate the usability of the input procedure with an interactive rule-based expert system for spatial data generation by land use/cover mapping of urban features from aerial photographs;
4) To design and implement a FOGIS architecture with the input procedure and a feature database for data storage, which includes both spatial data analysis procedures for time and thematic relationships and a spatial data output procedure for feature display;
5) To demonstrate the utility of FOGIS by entering, storing, analyzing, and displaying spatial data for multiple types of geographic features.

For the analysis ability of FOGIS, the multi-temporal occurrences of features will be retrieved by simple query operation for searching features in a feature database and the features can be queried by thematic relationships.

1.3 Background

With an emergence of a feature concept, generic lists of features useful in GIS operations have been compiled (Guptill *et al*., 1990; USGS, 1997). The Topologically Integrated Geographic Encoding and Referencing (TIGER) system of the U.S. Census Bureau (Trainor, 1990), Digital Line Graph-Enhanced (DLG-E) of the U.S. Geological Survey (USGS) (Guptill *et al*., 1990), feature lists for military requirements by the U.S. Defense Mapping Agency (Defense Mapping Agency, 1987), and the Feature and Attribute Coding Catalogue (FACC) of the National Geospatial-Intelligence Agency (NGA) (Digital Geographic Information Working Group, 2000) are examples of feature lists that are useful for each organization. However, these feature lists tend to vary according to their applications. Usery (1993) already tried to build a conceptual framework of features, which should be effective for various applications and resolutions using region theory, cognitive category theory, cartographic principles of abstraction, and set theory.

Region theory in geography provides a basis for the geographic feature concept (James and Jones, 1954). A region is an area which is not only homogenous in terms of criteria, but also possesses a unique character in terms of a particular association of phenomena. Therefore, a region in region theory is an entity for the purpose of thought based on an intellectual concept to group associated phenomena. A feature is also an intellectual concept with attributes and relationships for a geographic phenomenon. Cognitive category theory provides a method to aggregate features based on similarity (Rosch, 1978; Lakoff, 1987). Human cognition can build an identity of a feature based on perception and judgment. Categorical boundaries of features are also constructed from the human cognition based on their similarity. Cartographic principles provide abstraction and generalization methods for geographic phenomena, which are well-

established for various map scales (Robinson *et al*., 1984; McMaster, 1989). Set theory has a clear and well-established collection of rules so that it provides a method to build thematic groupings of geographic phenomena with no ambiguity (Gatrell, 1983 and 1991). For example, physical materials provide a common set of attributes in buildings and multispectral classification of remotely sensed data into land use/cover categories is based on a statistical set of spectral distribution (Usery, 1996a).

With an emergence of the object-oriented paradigm, several researchers have attempted to apply object-oriented concepts to GIS since 1990. Some researchers have used object representations directly for geographic phenomena without the feature concept. Worboys *et al*. (1990), for example, tried to build object-oriented data models for spatial databases.

For a database design, the means of representation is provided by the data model. The relational model is limited with respect to semantic content (expressive power) (Codd, 1970; Chen, 1976). Using only relations, a common difficulty in computer-aided design (CAD) and GIS is the gap between the richness of the knowledge structures in the application domain and the relative simplicity of the data model in which these knowledge structures are expressed and manipulated (Worboys *et al*., 1990). Many real world problems in GIS are not naturally expressible in terms of relations such as one-to-one, one-to-many, and many-to-many.

Real world phenomena can be captured by an abstraction that arises from recognition of similarities between certain objects, situations, or processes in the real world (Booch, 1994). Common abstraction of similar objects can be grouped and implemented as a class of a feature in an object-oriented model. Therefore, object-oriented models have the capability to express more readily the knowledge structure of real world phenomena than relational models.

Wachowicz (1999) chose object-oriented analysis and design methods to build a versioning module in GIS for deriving temporal differences of an object. Versioning is the tracking of the evolution of an object's state through time. The need for object versions to have separate identities is deemed a fundamental temporal issue, and the identity approach supported by object-oriented databases would be better suited for incorporating time into GIS than the value-based approach of the relational database (Barrea *et al*., 1991; Loomis, 1992). The time concept can be modeled and implemented effectively using object-oriented methods. A single object and identity with versions can be used to capture temporal differences. Each version of an object is associated with the attributes or relationships of the object.

For a spatial database, Milne *et al*. (1993) built a spatial data model based on object-orientation and tested it using both an object-oriented database (ONTOS) and a relational database (ORACLE). For contour retrieval, ONTOS was about 10 times faster than ORACLE (ONTOS, 1991). For the implementation of GIS, Vckovski (1998) used the object-oriented model for high interoperability based on the complexity reduction, which is accomplished by encapsulation (information hiding) of properties (data) and operations (functions or methods). An object's complexity of properties and methods is hidden for the object user. Objects are accessed only by sending and receiving a set of messages.

Other researchers have used feature-based representations of geographic phenomena in applying object-oriented concept to GIS. For example, Usery (1993 and 1994b) found a feature-based approach in building GIS may have the potential to better support geographical models and analytical procedures with accurate representation of geographic reality because the features correspond to the basic level of human cognitive observation. When people are questioned about what is viewed in a geographic scene, the answers are not likely to include geometric elements

such as points, lines, areas, and grid cells, but rather geographic entities such as rivers, lakes, buildings, and roads.

With a conceptual framework of features, a prototype feature-based GIS (FBGIS) has been developed for both vector and raster data representation (Usery, 1994a and 1994b; Usery *et al*, 2002). Features can be characterized by a three-dimensional framework: space, theme, and time. Each dimension includes both attributes and relationships to build a FBGIS. Although a FBGIS has been implemented using relational database techniques, the logical implementation strategy for the feature-based conceptual model is the object-oriented approach (Usery, 1996b).

Tang *et al*. (1996) used object-oriented concepts such as encapsulation, inheritance, and polymorphism for building an object-oriented feature-based data model of a FBGIS. As satate previously, encapsulation is a method of hiding information in a class definition and only data declared as 'public' in an object definition can be accessed by other objects. Inheritance is a method of data transfer in objects hierarchy. Objects in sub class inherit information from objects in super class. Polymorphism is a method to invoke different methods with the same message. The same name can be used for methods in different classes, which make different results.

These studies have shown that the object-oriented paradigm is better for GIS than conventional data models and programming methods. The reason is that the object-oriented paradigm is closer to real world situations, features and their relations, which are very complex. GIS deal with very complex spatial data that are captured from the real world.

1.4 Research Methodology

Five specific problems in conventional GIS are identified, which may be addressed by a feature approach based on object-orientation.

- Disjunctive thematic relations such as 'part of' are not contained in the conventional layered model in GIS.
- The same feature from different time periods cannot be saved as a single object.
- For multiple scale application, the geographic data model should allow multiple geometry types for features.
- Feature retrieval by name will consume significant resources in the conventional GIS.
- An expert system that can help neophytes generate and analyze spatial data is not available.

However, first four limitations in conventional GIS can be solved by a feature approach based on object-orientation. For final limitation, the integration of an expert system and GIS has been tried.

This research mainly consists of three parts: feature-oriented model development, system implementation, and application using geographic features. The feature-oriented model will start with a review of object-orientation and apply it to GIS modeling as a feature-oriented GIS model. It is the step to accomplish the first objective in this research. A prototype GIS will be developed based on feature-orientation. The implementation step will accomplish the second and fourth objectives. Finally, a developed system will be applied for generation, storage, analysis, and display of geographic data (Figure 1.3), which accomplish the third and fifth objectives. As a spatial analysis, time query will be used for differentiating the multi-temporal situations of

features. With time query, feature query will be also performed with the feature thematic relationships.

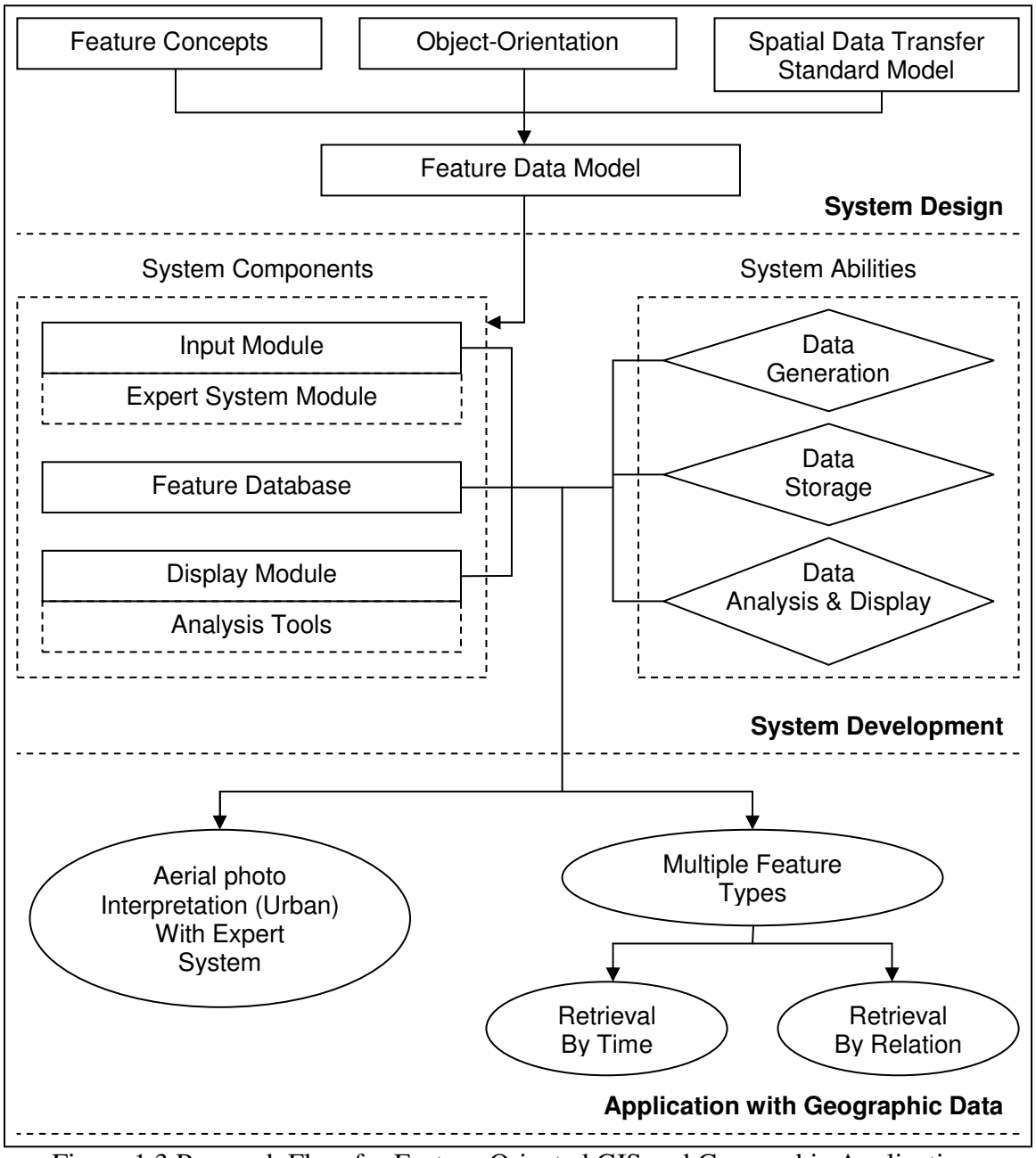


Figure 1.3 Research Flow for Feature-Oriented GIS and Geographic Applications

1.4.1 Solving Problems with Feature-Orientation

This research focuses on building a feature-oriented GIS using object-oriented methods. Therefore, the feature-oriented model should start with a review of key concepts and elements in object-orientation that include programming methods and databases.

With the feature concept for geographic phenomena, current spatial data transfer standard (SDTS) specifications, and key concepts in object-orientation, a feature data model will be constructed and implemented as a prototype of a feature-oriented GIS including input, analysis, and output procedures (USGS, 1997). By constructing a feature data model and the system architecture based on the feature data model, the first three problems that are identified in this research can be solved.

1.4.1.1 Disjunctive Thematic Relation

Almost all current commercial GIS have been developed based on conventional programming methods and a layered or an entity-relational data model for database management and query (Peuquet, 1988; Rhind *et al*, 1991). In a layer-based model, spatial attributes (location) and relationships (topology) can be structured using basic geometric objects (points, lines, areas, and pixels) and planar topology. Thematic attributes are directly attached to the basic geometric objects, but certain thematic relations are not contained in a layered model.

For example, "Kennesaw Mountain *is a* national park" is a thematic relation that can be represented in a layered model by including the Kennesaw Mountain in a layer called 'National Parks'. However, "Kennesaw Mountain is *a part of* Atlanta" is a thematic relation that is difficult to be represented in a layered model because of the thematic disjunction of the park layer and the city limit layer. Thematic disjunction relations need a hierarchical model for effective

representation that may not be represented in a layer-based model. Therefore, conventional GIS have limitations to contain spatial data fully.

The object-oriented model basically uses hierarchical representation for inheritance of attributes and methods of an object. Therefore, thematic disjunction relations may be implemented directly in a feature data model using the object-oriented paradigm.

1.4.1.2 Temporal Representation of Features

In a conventional GIS based on the relational model, the temporal representation of spatial data has been problematic because it is difficult and consumes significant resources to implement. For example, if a lake's boundary has changed through time and a user wants to save the changes, then the snapshot of the lake boundary for each time could be saved with its time as an attribute in two different lake layers including all other lakes which might not have changed through the time. The difference of the lake boundary can be calculated easily by the overlay of the two lake layers. However, the lake cannot be retrieved as the same one by its own name from the two different lake layers. It is impossible for two objects to have the same name with different attributes in the relational model, which means that an object in a different time should be saved as two different objects and the two objects could not be modeled as the same one by the relational model.

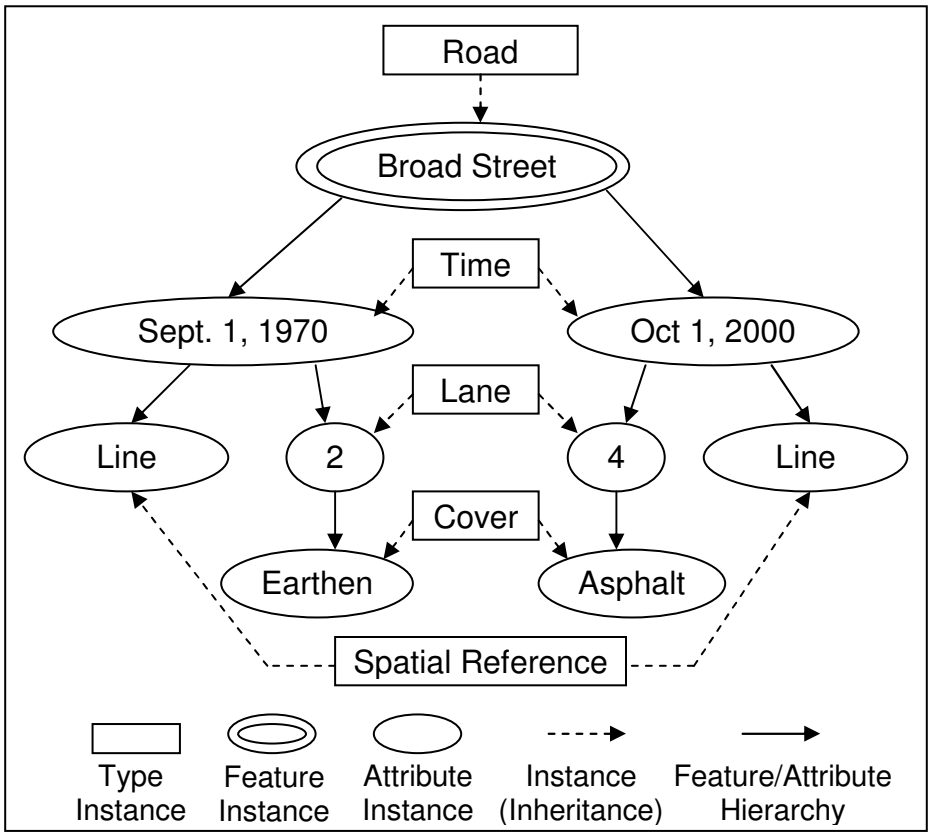


Figure 1.4 Feature with Time Attribute in a Feature Data Model

In the object-oriented paradigm, an object in a different time can be saved as the same object with different attributes or situations because an object can be created as two instances that inherit basic attributes and then save different situations through time. For example, if there is a road named 'Broad Street' and that road has been changed from September 1, 1970 to October 1, 2000 by constructing a new shortcut between two locations on the road which was originally a bypass, in an object-oriented design, a feature instance called 'Broad Street' has two sub-trees of attributes, one for attributes bound to September 1, 1970 and the other to October 1, 2000 (Figure 1.4).

A feature data model will be based on the object-orientation, which can save time as one of a feature's attributes. Therefore, FOGIS based on a feature data model can save a feature with different situations through time.

1.4.1.3 Multiple Geometry Types for Features

The conventional layer-based data model has a limitation on the geometry type for storing features in a layer. If multiple geometry types for a feature type were needed for an application, the same number of layers should be created for each to contain one geometry type. For example, if buildings have to be mapped at a certain map scale, then all buildings will be mapped as polygons or points in a layer-based model. However, if some buildings have to be mapped as polygons and the others as points because of the size of buildings related to the application scale and symbolization, it cannot be contained in a layer. It needs two layers based on the different geometry type even though both layers are containing the same feature type (Buildings).

However, the feature data model can contain multiple geometry types for a feature. For mapping buildings, the large buildings can be stored as polygons and the small buildings can be stored as points. During the digitizing process, a building can be mapped either as point or polygon according to the proper scale of an application. Therefore, the representation of features, both points and polygons, can be used for representing building features.

1.4.2 Solving Problems with System Implementation

This prototype system will consist of four main modules: (1) database, (2) data generation and input module with expert system, (3) data display module, and (4) analysis module for temporal representation and feature search using thematic relationships (see Figure 1.3). Each module will be designed for specific purposes from data input to output.

For the database, a feature database will be designed and used for physical storage of features from the feature data model using a Microsoft Access database. The input and display

modules can be developed with MapObjects which is a set of mapping software components that can be customized for mapping applications (ESRI, 1999a). Visual Basic 6.0 will be used to implement FOGIS including the input procedure with the rule-based expert system, the output procedure, and the analysis modules (Harris, 1997; Ryu, 1999; Stephens, 2000).

Specifically, the input module with an expert system will be developed using a rule-based method in an interactive mode in which a user can determine a solution from the result of an expert system for generating vector data (points, lines, and polygons) from image data such as aerial photographs or satellite images. The input module should be compatible with a conventional GIS by producing spatial data as shape file format.

The analysis module for temporal representation will be developed to retrieve different temporal situations of a feature from a database. The versioning concept and algorithm for feature retrieval will be used (Barrea *et al*., 1991; Loomis, 1992; Wachowicz, 1999). Versioning means maintaining an extra copy of old data existed on another time period. While versioning needs copies of layers in a layer-based model, it needs copies of only some features that have been changed through time periods in a feature data model. Also, a feature searching tool will be developed to retrieve the other related features using thematic relationships.

The analysis module will demonstrate the temporal representation in the third problem which is solved in the previous step. The final two problems that are identified in this research can be solved in the implementation step.

1.4.2.1 Feature Retrieval by Name

Spatial data are originally captured from real world features. Features should be retrieved by name, but the retrieval may cause problems in conventional GIS. A feature retrieval using

names in conventional GIS using a relation data model requires accessing the same attribute over many instances of the geometric or topologic data, because a feature's name is not an identifier but a thematic attribute of geometric elements such as points, lines, and areas in a commercial GIS (Guptil *et al*., 1990). A feature's name can be an identifier of a feature in the feature-based approach so that all spatial and non-spatial information including geometrical and topological relationships are directly linked to the identifier. Therefore, a feature's name can be used to retrieve all geometric or topologic information of a feature at once.

In a conventional layer-based model, a layer is a basic unit for data input and output so that it consumes resources to retrieve all information of a feature from a layer by name. If attributes of a feature are saved in different locations such as files and tables, the retrieval of all attributes of the feature will consume significant system resources to link the files or tables. This problem may also be related to the system performance. However, in a feature-based approach, a feature is a basic unit for data handling so it is fast and easy to use the name as a primary identifier for retrieving all related information of a feature. In the system implementation with a feature data model, a feature name will be used for an identifier for the feature to retrieve its attributes and relationships.

1.4.2.2 Expert System for Neophyte, Novice User

Almost all current commercial GIS software packages lack utilities or tools that can help neophytes in GIS to generate and analyze spatial data with proper explanation in a step-by-step fashion. An expert system could be incorporated with GIS to help neophytes to generate and analyze spatial data.

An expert system is a knowledge-based artificial intelligence technique because this system solves problems with the knowledge of experts (Carrico *et al*., 1989). The basic unit of knowledge in these expert systems is the rule which is a conditional test-action pair, for example, if condition is true, then action.

In remote sensing and GIS, previous studies using rule-based expert systems have focused on the automatic interpretation of aerial photographs or satellite images or on the automatic mapping of spatial data for experts (Tatsuyama, 1987; Usery *et al*., 1988; Moller-Jensen, 1990; Foschi and Smith, 1997; Wilson, 1997). However, the rule-based expert system can be applied to the interactive image interpretation for neophytes.

The input procedure will be developed using an interactive rule-based expert system, which will show how an expert system can be integrated with GIS. In FOGIS, an interactive rule-based expert system will be used for the interpretation of aerial photographs in order to generate spatial data from the interpretation.

1.4.3 Geographic Application

The developed system will be tested using geographic applications. The first application is to map urban features from aerial photographs for testing the data input procedure with the interactive rule-based expert system. The data input modules with the expert system will be applied for urban area mapping since a rule for urban mapping will be simple and can be generated as a general rule over locations. A general rule for urban mapping can be constructed by the key elements of the image interpretation: tone, texture, pattern, size, shape, shadow, and situation (Avery and Berlin, 1992).

The second application is to save, analyze, and display urban vector data for testing the database, analysis, and display modules because urban areas include various complex features. The FOGIS will be applied for multiple feature type data such as coastlines, buildings, roads, and administrative boundaries of Camp Lejeune area (the U.S. Marine Corps Base) in North Carolina to test the performance. Two analysis operations will be executed: representation of multi-temporal instances of a feature and a feature search using thematic relationships.

Both coastlines and buildings will be represented geometrically as area and have simple attributes and relations that are easy to use for the temporal analysis. Changes in buildings are evident since there was a hurricane in the study area between two dates of image sources. Multi-temporal instances of building features can be used to clearly visualize the temporal differences of buildings. The road data are linear and have comparatively complex attributes and relations that are useful to test the feature retrieval by name and searching features based on the thematic relationships. The road data are provided in shape file format so that they can also be used for testing the import/export module.

1.5 Summary

Five problems in conventional GIS will be solved by a feature-oriented approach using the object-oriented model. The feature-orientation step that accomplishes the first objective, which is to construct a feature data model with a feature concept and object-orientation, will solve the first three problems. First, disjunctive thematic relations such as ‘part of’ are not included in the conventional layer-based model, but are incorporated in a feature-based approach by a hierarchical representation. Second, a feature that exists at different time periods cannot be saved as one object in a conventional GIS based on a relational database, but a feature can have

versioned attributes of one object through time in a feature data model. Third, a layer in a layer-based model cannot contain multiple geometry types for representing different size of features, but features (buildings) in a feature data model can be any type of geometry instances (either point building or polygon building).

The implementation of FOGIS that accomplishes the second and fourth objectives, which are developing input procedure based on an expert system and implementing FOGIS based on a feature data model, will solve the final two problems. A feature retrieval by name in conventional GIS needs to access all related instances of geometric and topologic data so that it consumes significant system resources, but the name is the primary identifier for retrieving features including their attributes and relationships in the FOGIS. An expert system will be incorporated in the input procedure of FOGIS for helping neophytes to generate and analyze spatial data.

The geographic applications will accomplish the third and fifth objectives, which are demonstrations for the utility of a rule-based expert system and FOGIS. Specifically, the analysis module will include two operations: multi-temporal representation and a feature search using thematic relationships.

1.6 Chapter Organization

The construction of a feature data model is mainly based on the object-oriented design and analysis methods. The development of a feature-oriented system is also based on the object-oriented programming. Therefore, in Chapter 2, the basic concepts of the object-oriented paradigm will be reviewed. This chapter includes the historical background of the object-oriented paradigm, the key elements of object-oriented programming for the system development

(such as abstraction, encapsulation, hierarchy, and so on), and the concepts and design methods of object and class.

In Chapter 3, the previous data model for the data container will be reviewed. The hierarchical and network model is a physical data model and the relational data model is logical data model. More recently, the object-oriented model has emerged and includes both data and methods. Based on the feature concepts, the SDTS data model, and object-orientation, the feature data model will be designed.

In Chapter 4, the feature data model will be implemented to construct FOGIS. The implementation will include an import/export module, a feature database for physical data storage, and a display module. Also, a time query tool will be added. To test the implemented system, time query for the multi-temporal situations will be executed on building features.

In Chapter 5, for spatial data generation, the input procedure will be designed and implemented. Specifically, the expert system will be designed and implemented as a part of the input module. To test the input procedure incorporated with the expert system, the application will be focused on land use/cover mapping from the image sources. The land use/cover categories will be decided by the expert system. Accuracy assessment will be conducted on the resulting land use/cover map in order to test the usability of the rule in the expert system.

In Chapter 6, system modules developed in both Chapter 4 and 5 will be integrated into one complete system as FOGIS. A feature population tool for the input module, projection tool for import/export module, and tree viewer for the display module will be developed and added to FOGIS during an integration process. Also, a feature searching tool using thematic relationships will be added in the display module as one of the spatial analysis tools.

Finally, in Chapter 7, all five problems observed in Chapter 1 will be discussed and summarized as conclusions. The discussion will include the level of accomplishment for each problem addressed in Chapter 1 through FOGIS development. Limitations and further research directions also will be discussed.

CHAPTER 2

OBJECT-ORIENTATION WITH VISUAL BASIC FOR FEATURE-ORIENTATION

2.1 Introduction

While both programming and a data model are separated in the conventional programming paradigm, the object-oriented model includes both. The reason is the object in an object-oriented model has both data (status) and behaviors (functions or methods). In the conventional programming paradigm, data have their own structure based on the data model and are independent from the functional action that is implemented by procedural programming.

Many GIS have been developed based on the conventional programming method with an entity-relational data model. The entity-relational data model has some limitations to fully contain spatial data because spatial data are originally captured from the real world features. Features have complex relationships with other features. However, the relational data model cannot capture these complex relationships because the relationships between entities (tables) are modeled by the linkage of tables in the entity-relational models. There is no room for the relationship itself. Therefore, conventional GIS only focus on the spatial relationship of topological information as feature relationships. Other relationships of features such as thematic and temporal relationships are not currently handled in the layer-based GIS model. The object-oriented model provides a solution for those complex relationships so that it has been suggested for building a GIS (Graham, 2001). Version-management is another important demand for GIS applications and can be embodied more effectively by object-oriented programming (Wachowicz, 1999).

In this chapter, important concepts of the object-oriented model will be reviewed as the main methods for designing the feature data model and building FOGIS. The next section provides a historical review of object-orientation. Key concepts and characteristics of object-orientation will be reviewed and discussed with the meaning in Visual Basic, which is the main programming language for FOGIS. Also, the concepts of objects and classes are discussed, which are the containers of the data and methods.

2.2 History of Object-Orientation

The earliest work in computing was concerned with programming. Later, concepts of design and analysis arose. Similarly, in object-orientation, programming first attracted attention and object-oriented design and analysis later become major areas of endeavor (Graham, 2001). Therefore, the history of object-orientation should start with object-oriented programming (OOP), and move on to object-oriented design (OOD) and object-oriented analysis (OOA).

The history of object-oriented programming starts with the development of Simula language in 1965, based on the ALGOL-60 language, which was specifically oriented towards discrete event simulation (Dahl and Nygaard, 1966). In 1967, Simula-67 was developed to solve problems of conventional language in simulation modeling, which introduced the basic concept of object and class first (Dahl el al., 1968). The inheritance mechanism, through which a class could inherit the data from super classes, was also introduced in Simula-67.

In the mid-1970s, the term “object-oriented” finally came into the language with the programming language Smalltalk. Smalltalk was developed at the Xerox Palo Alto Research Center (PARC) based on not only Simula but also the doctoral work of Alna Kay (Rentsch, 1982). Smalltalk was influenced by both the notion of classes and inheritance of Simula and the

functional abstractions of the LISt Processing (LISP) language (McCarthy, 1960). In Smalltalk, everything is perceived as an object and objects can communicate with each other by passing messages.

There have been five releases of Smalltalk between 1972 and 1980 from Smalltalk-72 to Smalltalk-80. Generally, Smalltalk has a complete programming environment, including editors and browsers. Smalltalk has influenced not only the design of every subsequent object-oriented programming language, but also the look and feel of graphic user interfaces such as the Macintosh user interface and Motif (Booch, 1994). One of the reasons for the success of object-oriented programming was the complexity of these user interfaces and the high cost of building them. Without the inherent reusability of programming code in the object-oriented concept, these interfaces could not have been built on such a wide scale (Graham, 2001).

After the advent of object-orientation, there were two demands of users: the development of a user interface and reusable software components (modularity). User interface development possessed no significant data management problems, but there were performance problems associated with early object-oriented languages. This led to the development of new languages, such as Eiffel, and to extensions of existing efficient conventional languages such as C and Pascal. The modularity has been addressed by Ada language developed by U.S. Department of Defense. Any changes in a module have no effect on the other modules. Object-oriented programming promises to enable system developers to assemble systems from reusable components, thus addressing modularity. Several researchers (Biggerstaff and Richter, 1989; Prieto-Diaz and Freeman, 1987) have argued that the higher the levels of reuse the greater the benefit for the object-oriented programming in a general software engineering context.

Almost all object-oriented methods have in common basic characteristics such as methods associated with objects, inheritance of attributes and methods from super classes, and the ability to define the type of objects, their attribute types and relationships. However, these object-oriented databases still suffer from the lack of standards in query language. The lack of a standard for object-oriented query languages has caused differences in query language syntax, completeness, standard query language (SQL) compatibility and treatment of encapsulation (Cattell, 1994). Recently, two approaches in object-oriented databases have been developed: pure object-oriented databases and hybrid object-relational databases.

As object-oriented programming began to mature, interest shifted to object-oriented design and analysis or specification, which developed as an approach to improving our understanding of the concepts, activities, rules, and assertions of the object-orientation paradigm. Computer-aided software engineering (CASE) has become increasingly important as a graphical tool for supporting object-oriented analysis and design (Graham, 2001). The increasing products of CASE tools are based on the composition of graphical symbols and notations depicting the semantics and features. The most important benefits of using CASE tools are automatic code generation and enhancement of productivity. However, the rules and methods provided by CASE tools are inappropriate or even non-existent in some innovative applications (Wachowicz, 1999).

Now, the object-oriented method is a part of the general toolkit of the software developer such as Java, Visual C++, Visual Basic, and so on. It means that object-oriented programming can be regarded as a mature discipline worthy of regular use by commercial organizations. In the next section, the key concepts and characteristics of object-orientation will be reviewed and discussed based on the programming methods of Visual Basic, which is the major tool for building FOGIS.

2.3 Key Elements of Object-Orientation

Five key elements of the object-oriented model include abstraction, encapsulation, modularity, hierarchy, and polymorphism. The methods to accomplish each element in the object-oriented programming language, especially with the Visual Basic, will be discussed (Roman, 1997).

2.3.1 Abstraction

Abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provides crisply defined conceptual boundaries, relative to the perspective of the viewer. Objects with common abstraction based on their similarity can be grouped as a class.

There are several sets of abstraction: entity abstraction, action abstraction, virtual machine abstraction, and coincidental abstraction (Graham, 2001). An object represents a useful model of problem-domain entity using entity abstraction, provides a general set of functions or operations through action abstraction, ties together operations using virtual machine abstraction, and packages a set of operations that have no relation to each other by coincidental abstraction. Deciding upon the right set of abstractions for a given domain is the central problem in object-oriented design. Generally, programmers use entity abstractions because they are directly parallel to the vocabulary of a given problem domain. Abstractions give us an inside view of an object.

The outside view of an object or abstraction can be explained by the contract model (Meyer, 1988). In the contract model, a client is any object that uses the resources of another object as a server. The behavior of an object can be specified by considering the services or operations that it provides to other objects. The entire set of services or operations of an object is

called its protocol, which is the way an object may act and react. The protocol constitutes the entire outside view of the abstraction.

In Visual Basic, the abstraction data type is an example of abstraction of a basic data type. The abstraction data type is a user defined data type or class, which is different from common data type variables such as integer, double, or string data types.

The data type is used as a variable and initialized by assigning a value to the variable. To use the abstract data type, the object of 'CFeature' has to be initialized by the set method. Then the instance ('instFeature') will be created. In Figure2.1, an instance ('instFeature') of 'CFeature' is created by 'New' parameter from the 'CFeature' that is an abstract data type or a class and then instantiated as 'instFeature' by 'Set' methods. 'CFeature' is an abstraction for all geographic features that may have common data and methods.

Data Type	Abstract Data Type
Dim X as Integer X = 200	Dim instFeature as CFeature Set instFeature = New CFeature

Figure 2.1 Common Data Type vs. Abstract Data Type

2.3.2 Encapsulation

Encapsulation is the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior (Booch, 1994). Encapsulation serves to separate the contractual interface of an abstraction and its implementation. Encapsulation is achieved through information hiding. The structure of an object is hidden, as well as the implementation of its methods. Each implementation encapsulates details about which no client may make assumptions. Implementation of a class comprises the representation of the abstraction as well as the mechanisms that achieve the desired behavior.

For example, in C++, members may be placed in the public, private, or protected parts of a class. Members declared in the public part are visible to all clients. Members declared in the private part are fully encapsulated, and members declared in the protected part are visible only to the class itself and its sub class. Specifically, the notion of friends that can be used to permit access each other's private parts is supported in C++ (Zaratian, 1998). In Visual Basic, public and private declarations are allowed.

Encapsulation also allows the inclusion of validation code to help catch errors in the use of the exposed interface. If any variable is defined as private and referred from an outside interface, it will automatically generate errors. If the public interface is used for a variable that could be any data or method, then it can be inherited and used on the other classes.

2.3.3 Modularity

In programming, the act of partitioning a program into individual components can reduce its complexity to some degree. Although partitioning a program is helpful for this reason, a more powerful justification for partitioning a program is that it creates a number of well-defined, documented boundaries within the program. Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules by grouping logically related abstractions and minimizing the dependencies among modules (Booch, 1994). Classes and objects form the logical structure of a system. These abstractions are placed in modules in order to produce the system's physical architecture. Modules serve as the physical containers in which the classes and objects of logical design will be declared.

The greatest advantage of modularity is the cost in that the decomposition into modules will reduce the software cost by allowing modules to be designed and revised independently. The

identification (abstraction) of classes and objects is part of the logical design of the system, but the identification of modules is part of the system's physical design. Therefore, modularity makes reuse of classes and objects convenient.

In Visual Basic, the module has two types: form modules and class modules. The main similarity is both modules can have both data (property) and methods. The difference is the form module has a visible interface but a class module does not. Also, the way to access public properties and methods is different. With a class module, one class can only access another class's public properties and methods by instancing of the super class. With a form module, the public properties and methods of a form can be directly accessed by the other modules (Figure 2.2). However, both modules make reusability possible.

Class Module	Form Module

class CStudent {
  public CStudent_Propt1
  public CStudent_Meth1
}
	form Display {
	  public Display_Propt1
	  public Display_Method1
	}

Dim Jimmy as CStudent Set Jimmy = New CStudent	-- No need to create Instance of form Display
Jimmy.CStudent_Propt1 Jimmy.CStudent_Meth1	Display.Display_Propt1 Display.Display_Method1

Figure 2.2 Qualifications of Public Property and Method

2.3.4 Hierarchy

The fourth element of object-orientation is hierarchy that is a ranking or ordering of abstractions. Two most important hierarchies in system design are class structure using the "is a" hierarchy for generalization or specialization and object structure using the "part of" hierarchy for aggregation. Inheritance is the most important method of the "is a" hierarchy. It defines a

relationship among classes and allows groups to be easily reused. Aggregation is the most important method of the "part of" hierarchy. It permits the physical grouping of logically related structures.

In Visual Basic, the explicit class inheritance technique is not provided, but the references for the variables of sub class to super class make the hierarchy possible. Also, the interface inheritance is allowed, by which a class can inherit properties and methods defined in the interface class (Harmon and Sawyer, 1999).

The principles of abstraction, encapsulation, and modularity are synergistic. An object provides a crisp boundary around a single abstraction, and both encapsulation and modularity provide barriers around this abstraction. Also, in almost all applications, there are too many abstractions to comprehend at one time. Encapsulation helps manage this complexity by hiding the inside view of these abstractions. Modularity provides a way to cluster logically related abstractions. Hierarchy in system design helps simplify the understanding of the problem.

2.3.5 Polymorphism

Polymorphism represents a concept in type theory in which a single name may denote objects of many different classes that are related by some common super class. Therefore, any object denoted by this name can respond to some common set of operations. Polymorphism is the ability to process objects differently depending on their data type or class by redefining methods for derived classes. For example, given a base class 'shape', it is possible to define different 'area' methods for any number of derived classes, such as polygons, circles, and rectangles. According to the shape of an object, the 'area' method to the object will return the correct results.

In Visual Basic, the clear example of polymorphism is any method that uses the polymorphic variable 'Variant'. The 'Variant' variable will be a different form for integer, double, or string in run time so that the same name method with 'Variant' variable will work differently based on the input data for the variable. Through the interface inheritance, polymorphism is accomplished.

2.4 Objects and Classes

2.4.1 Objects

Objects are entities that combine the properties of methods and data since they perform computations and save local state (Stefik and Bobrow, 1986). Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relations.

An object has a state and a unique identity. A state exhibits some well-defined behavior and a unique identity is the structure and behavior of similar objects that are defined in their common class. Attributes such as time, beauty, or color and emotions such as love and anger are not objects, but are all potential properties of an object. An object is defined by a crisp physical boundary, by a crisp conceptual boundary (i.e., intangible events or processes), or by a tangible fuzzy physical boundary (i.e., rivers, fog, or crowds of people).

The state of an object encompasses all of the static properties of the object and the current (or dynamic) values of each of these properties. A class is not an object because it does not represent a specific instance. Behavior is how an object acts and reacts in terms of its state changing by message passing. The state of an object represents the cumulative results of its

behavior. Operations such as modifier, selector, iterator, constructor, and destructor are services that a class offers to its clients. An object may be active or passive. An active object encompasses its own thread of control, whereas a passive object can only undergo a state change when explicitly acted upon.

An identity of an object is a unique property that distinguishes the object from all other objects. A structure is useful for a certain abstraction when the abstraction represents a simple record of other objects and has no behavior that applies to the object as a whole. However, a class is useful for an abstraction if the abstraction requires behavior more intense than just simple puts and gets of independent record items. An identity of an object may be lost when the object is copied by a reference (as a pointer). If a pointer of one object aliases another object then the original object designated by that pointer can no longer be named and the identity of the object is lost. It is problematic because it will cause memory leaks, memory-access violations, and even unexpected state changes. Therefore, it would be better to copy an object by value (or deepcopy) instead of by reference (shallowcopy) (Booch, 1994).

In Visual Basic, an object can be created either explicitly or implicitly (Figure 2.3). Explicit object creation needs two lines. The first line declares a variable named ‘Jimmy’ to be of type ‘CStudent’. The second line creates an object of type ‘CStudent’ by the ‘New’ keyword and assigns ‘Jimmy’ as a reference to that object by the ‘Set’ keyword. With these two lines, an instance ‘Jimmy’ of ‘CStudents’ exists. It is the explicit procedure to create an instance of the class ‘CStudent’. Implicit object creation needs only one line. The line declares ‘Jimmy’ as a variable of type ‘CStduent’ with the ‘New’ keyword. It will not create an instance (‘Jimmy’) immediately. When the object variable ‘Jimmy’ is used for the first time, the instance will be

created. Generally, the explicit object creation method is recommended for tracking the creation of instances.

Explicit Object Creation	Implicit Object Creation

class CStudent {
  public Last_name
  public CStudent_Meth1
}
 	
Dim Jimmy as CStudent
Set Jimmy = New CStudent
	class CStudent {
	  public Last_name
	  public CStudent_Meth1
	}
	
	Dim Jimmy as New CStudent
	…
	Jimmy.Last_name = “Choi”

Figure 2.3 Object Creation in Visual Basic

2.4.2 Classes

A class is a set of objects that share a common structure and a common behavior. A class represents only an abstraction that is the essence of an object, whereas an object is a concrete entity that exists in time and space. An object is simply an instance of a class. Classes can have an outside view known as an interface and an inside view known as an implementation or the secrets of its behavior. The interface can be separated as two or three types: public and private in Visual Basic, or public, protected, and private part in C++. Public parts are opened to all clients, protected parts to itself, sub class, and friend, and private parts to itself and friend.

The relation between classes might indicate some sort of sharing or some kind of semantic connection. There are three basic types of class relationships: generalization/specialization, whole/part, and association. The “is a” relation represents generalization/specialization of classes. The “part of” relation for object aggregation is also used for whole/part relation of classes. The association shows some semantic dependency between classes (i.e., ladybugs protect flowers).

The inheritance among classes uses the generalization/specialization of classes, and hence the hierarchy among abstractions is largely a matter of classification. The aggregation from the "part of" relation makes a class as a container of objects or instance of another class. There are two classes named A and B, and if an instance of class A can be created in the private part declaration of class B, the instance of class A does not exist independently of its enclosing class B instance. When an instance of class B is created or destroyed, the instance of class A will be created or destroyed. This kind of aggregation is direct containment by value (Booch, 1994).

A less direct kind of aggregation is containment by reference. When an instance of class B declared using Pointer, instances of each class A and B can be created independently. If and only if there is a whole/part relationship between two objects, there must be an aggregation relationship between their corresponding classes. If it is not sure that there is an "is a" relationship between two classes, then aggregation or some other relationship should be used instead of inheritance.

An instance of a class can be generated by value or by reference. Also, several different instances of a class can be created using different parameters. This kind of class is a parameterized class or a generic class that serves as a template for other classes. A template may be parameterized by other classes, objects, and/or operations. Instances of a class using different parameters are instances of distinctly different classes and not united by any common super class, even though they are derived from the same parameterized class. A parameterized class must be instantiated with its parameters before objects can be created.

A meta class is a class of classes. It means that instances of a meta class are themselves classes. The primary purpose of a meta class is to provide class variables that are shared by all

instances of the class and to provide operations for initializing class variables and for creating the meta class's single instance.

In Visual Basic, both class modules and collection classes are allowed to be used (Roman, 1997). A class module has two parts: properties and methods (Figure 2.4). Visual Basic does not allow the array type properties inside a class. Using a class module a new name of instance is needed, whenever an object is instantiated from the class. However, it may be impossible to get the new name of the instance in run time unless the name is known before the program is compiled. Therefore, with only a class module, it may be impossible to create multiple instances for which the name is currently unknown.

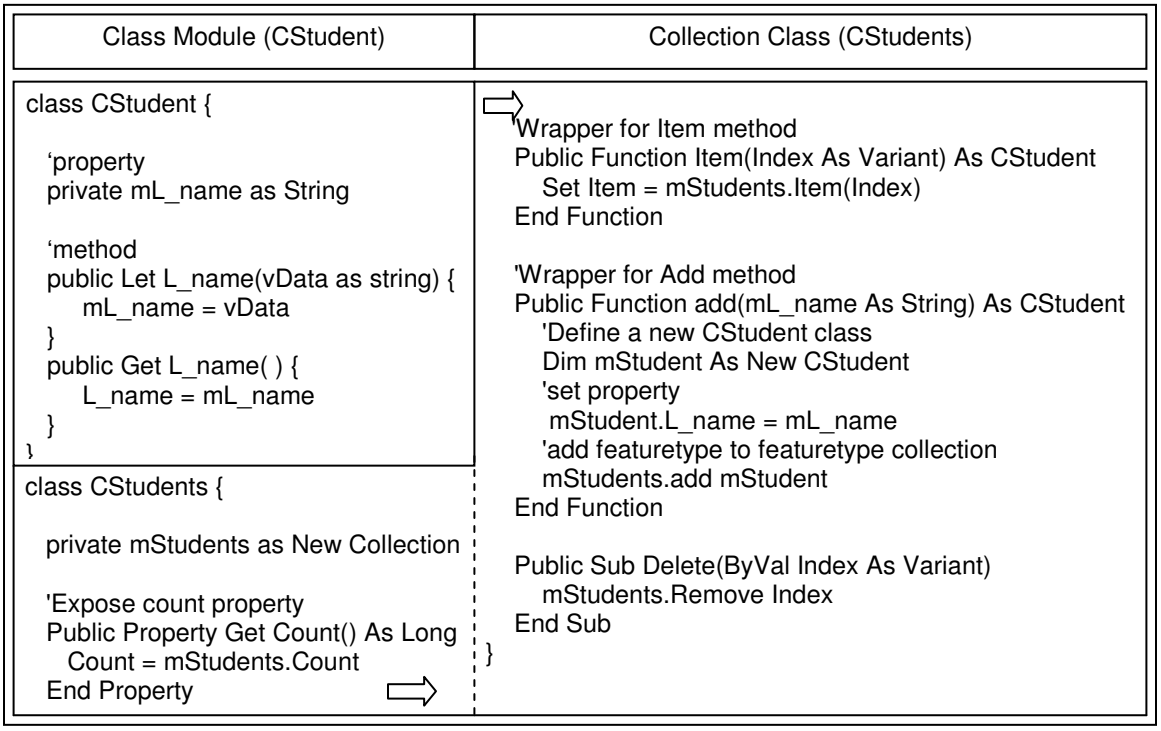


Figure 2.4 Creating Classes in Visual Basic

The 'CStudent' class in Figure 2.4 has private property that can be accessed only by the Let/Get methods inside the class. It hides the property from the outside of the class, which is an encapsulation. If multiple instances for a class need to be alive in run time, a collection class

should be used. The collection class allows making multiple instances using same name with a different index at any time the program is alive. The collection class also has both properties and methods, but the collection data type should be used for properties. For a collection class module, a single class for the collection should be defined and used inside the collection class for Add, Item, Count, and Delete methods (Figure 2.4). These methods of the collection method have been inherited from the Visual Basic Collection Interface, which is the interface inheritance.

2.4.3 Classification

The way to find classes and objects with our knowledge is a classification problem. Recognizing the similarity among entities shows the commonality within key abstractions and mechanisms, and eventually leads to smaller and simpler architecture. There are three ways for a classification: classical categorization, conceptual clustering, and prototype theory.

In classical categorization, all entities, that have a given property or collection of properties in common, form a category. Such properties are necessary and sufficient to define the category. However, natural categories tend to be messy. For example, most birds can fly but some cannot. In conceptual clustering, classes (or clusters of entities) are generated by first formulating conceptual descriptions of these classes and then classifying the entities according to the descriptions. It is closely related to fuzzy set theory, in which objects may belong to one or more groups, in varying degrees of fitness. There are some abstractions that have neither clearly bounded properties nor concepts. In prototype theory, a class of objects is represented by a prototypical object, and an object is considered to be a member of this class if and only if it resembles this prototype in significant ways (Booch, 1994).

There are several practical approaches to classify classes and objects: classical approach, behavior analysis, domain analysis, use-case analysis, informal English description, and structured analysis. These approaches are mainly used for object-oriented analysis, which is to find a way to model the world by discovering the classes and objects that form the vocabulary of the problem domain.

Several classical approaches use various sources of classes and objects derived from the requirements of the problem domain (Shlaer and Mellor, 1988; Ross, 1987; Coad and Yourdon, 1990). These are called classical because they derive primarily from the principles of classical categorization. Whereas these classical approaches focus upon tangible things in the problem domain, behavior analysis approach focuses on dynamic behavior as the primary sources of classes and objects. Classes are formed based on groups of objects that exhibit similar behavior (Shlaer and Mellor, 1992). These behaviors and services of objects are responsibilities; objects can be grouped by common responsibilities (Wirfs-Brock et el., 1990; Rubin and Goldberg, 1992).

Whereas the classical approach and behavior analysis are applied typically to a single and specific application, domain analysis seeks to identify the classes and objects that are common to all applications within a given domain. Experts can perceive important objects, operations, and relationships about the domain and consult to create a generic model of the domain (Arango, 1989; Moore and Bailin, 1988). However, it is difficult to find a domain expert.

All classical analysis, behavior analysis, and domain analysis depend on a large measure of personal experience. These approaches are unacceptable for the majority of applications, because such a process is neither deterministic nor predictably successful (Booch, 1994). However, Jacobson (1992) has formed use-case analysis. In use-case analysis, a case is a

particular form or example of usage or scenario. First, users make several scenarios as cases and then analysis proceeds by a study of each case to identify objects and classes with operations.

An informal English description method uses nouns and verbs in an English description of the problem (Abbott, 1983). The nouns represent candidate objects, and the verbs represent candidate operations or responsibilities of those objects. This approach is very useful because it is simple and makes the developer work in the vocabulary of the problem domain. However, sometimes it is difficult to use this approach because natural language may not express a problem fully (Booch, 1994).

A structured analysis uses data flow diagrams to provide a formal model of the problem. From this model, classes and objects can be identified, and data transformations between objects are assigned as operations or behaviors (Ward and Mellor, 1985). However, structured design with structured analysis is entirely orthogonal to the principal of object-oriented design. This approach uses data flow diagrams as a design rather than an essential model of the problem domain (Booch, 1994).

Cases or scenarios are very powerful tools to form a problem domain explicitly so that they can be used to drive the process of classical analysis, behavior analysis, and domain analysis (Booch, 1994). Therefore, among above approaches, a mixed approach of use-case analysis and one approach among classical analysis, behavior analysis, and domain analysis may be the best for identifying objects and classes.

2.5 Summary

In this chapter, with the brief historical review on object-orientation, the key elements and concepts are reviewed. Specifically, five key elements (abstraction, encapsulation, hierarchy,

polymorphism, and modularity), objects, and classes have been examined in Visual Basic, which is the main programming tool for implementation of FOGIS in this study.

Abstraction denotes the essential characteristics of an object that provide crisply defined conceptual boundaries. Objects with common abstraction based on the similarity can be grouped as a class. In Visual Basic, the abstraction data type is a user defied data type or class that is different from a common data type such as integer, double, or string.

Encapsulation is the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior. Therefore, encapsulation serves to separate the contractual interface of an abstraction and its implementation so it hides information of an object from the outside. In Visual Basic, public and private declarations are allowed. The public means it is open to the outside and the private means it is closed to the outside.

In programming, modularity is the act of partitioning a program into individual components, which can reduce its complexity to some degree. In Visual Basic, classes are form modules or class modules. The main similarity is both modules can have both data (property) and methods. With a form module, the public properties and methods of a form can be accessed directly by the other modules, which is not true in a class module. However, both modules make reusability possible.

The fourth element of object-orientation is hierarchy, which is a ranking or ordering of abstractions. In Visual Basic, the explicit class inheritance technique is not provided but the interface inheritance is allowed, by which a class can inherit properties and methods defined in the interface class. Also, the hierarchy of classes can be mapped by the references for the variables of sub class to super class.

Polymorphism means that any object denoted by name can respond to some common set of operations. In Visual Basic, the 'Variant' variable is called the polymorphic variable. The 'Variant' variable will take different forms as integer, double, or string in run time so that the same name method with the 'Variant' variable will work differently with different input data for the variable.

Objects are entities that combine the properties of methods and data. An object has a state and a unique identity. A state exhibits some well-defined behavior and a unique identity is the structure and behavior of similar objects that are defined in their common class. The state of an object encompasses all of the static properties of the object and the current (or dynamic) values of each of these properties. In Visual Basic, object can be created either explicitly or implicitly. Usually, explicit object creation is preferred because it creates an instance immediately.

A class is a set of objects that share a common structure and a common behavior. A class represents only an abstraction that is the essence of an object, whereas an object is a concrete entity that exists in time and space. An object is simply an instance of a class. In Visual Basic, both class module and collection classes can be used. Because Visual Basic does not allow the array type properties inside class, the collection class should be used to create multiple instances that use the same name with a different index at any time the program is alive.

Finally, classification is the way to identify classes and objects with our knowledge. Recognizing the similarity among things shows the commonality within key abstractions and mechanisms, and eventually leads to smaller and simpler architecture. A mixed approach of use-case analysis and one approach among classical analysis, behavior analysis, and domain analysis may be the best method for identifying objects and classes.

CHAPTER 3

FEATURE DATA MODEL FOR FEATURE REPRESENTATION

3.1 Early Data Models: Hierarchical and Network Data Model

Data storage began with the sequential file and the device for data storage was a tape. The data of the file was matched and read, and then the tape for the file was rewound for the next access. As more sophisticated storage devices, such as drums and disks emerged, the direct access method called random was possible. After this, hashing or storing index files could improve file access speeds. Index files made it possible to conceptualize a structural relationship and hence capture some of the structure of the real world or of the application.

Early databases used hierarchical or network data models with the structural relationships among index files. The hierarchical data model could only express certain kinds of structural relationships. The network data model allowed more general graphs to be constructed, but still it was very hard to change these relationships if a system had been already designed. If an attribute needed to be added or removed after a completed design, the network system might need to be redesigned. The early hierarchical and network data models suffered from inflexibility of their implemented systems.

The simplest data structure is a list, lists of lists, and the kind of tree-structured lists. These lists are written in a file. If a programmer works with a file, he will open a file or stream of data and read through it, record by record, until some condition is satisfied. For example, consider a university phone book that stores name, address, and telephone number for students. If one has the name and address, it is easy to find the phone number. If one, however, has the

phone number and address and wants the name, it is difficult to find the name because lists or files have logical structure in which the structure is sorted in a particular order. To find the name with phone number and address, a file should be read in a different order. The computerized university phone book can be viewed as a table (Figure 3.1).

Phone_No	Dept	No_of_Ext	Extension	Name
542-1234	CSCI	1	111	S. John
542-5452	GEOG	3	101	J. Choi
			102	N. Tom
			103	E. Man
542-6788	ELAN	1	141	P. Pod

Figure 3.1 File Structure with a Repeating Group (Hierarchical Structure in a File)

For the GEOG department, a repeating group that consists of extensions and names was included (Figure 3.1). The arrangement in the table has the logical structure of a tree or hierarchy. Given direct access using a logical pointer, a repeating group does not need to be kept in the same file. In Figure 3.1, the university phone book table can be separated into two files: a department file and an extensions file. The department file records contain logical pointers to the extensions file (Figure 3.2).

Department

Phone_No	Dept	Dept_Cod
542-1234	CSCI	10
542-5452	GEOG	13
542-6788	ELAN	15

Extensions

Dept_Cod	Extension	Name
10	111	S. John
13	101	J. Choi
13	102	N. Tom
13	103	E. Man
15	141	P. Pod

Figure 3.2 Decomposition without Repeating Group (Hierarchical Structure with Two Files)

There are two files with a common field to link them. It is logically a one-to-many relationship between departments and extensions in Figure 3.2. An extension cannot be in two departments in this scheme. This is a major limitation of the hierarchical approach to data structure. It is very difficult in the hierarchical model to deal with many-to-many relationships (Graham, 2001).

Many-to-many relationships can be captured by the network model. For example, let us suppose the stock information that is about particular stock items and their suppliers. The relationship represented in Figure 3.3 shows a many-to-many relationship between items and suppliers. For every item there may be many suppliers and for every supplier there may be many parts.

I_No	Item_Na	Description
i1	Marble	½" glass

*

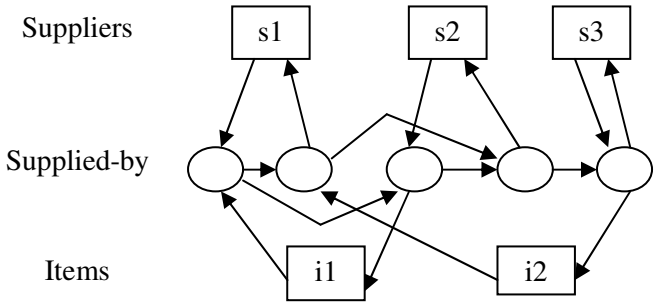
S_N	S_Name	Location	Price
s1	ABC	London	.30
s2	Aardvark	Paris	.30

I_No	Item_Na	Description
i2	Bat	Willow

*

S_N	S_Name	Location	Price
s1	ABC	London	20
s2	Aardvark	Paris	40
s3	Blue	New York	20

a) Data structure if items and suppliers by the supplied-by relationship



b) Network pointer structure for the supplied-by relationship

Figure 3.3 Data Structure and Network Pointer Structure (Graham, 2001)

In Figure 3.3, there are two ways to find item 2 (i2) supplied by s2. From the network pointer structure in the bottom part (b) of Figure 3.3, one way is to start with the supplier and look along the pointers for a connector linked to the item and the other way is to start with the item and look for the supplier.

In the network model, many-to-many relationship can be implemented efficiently by pointers. However, the maintenance associated with practical network databases may be expensive. A network diagram for a simple data structure such as that in Figure 3.3 is complicated. The relational data model overcomes some of the limitation of hierarchical and network data model in terms of flexibility.

3.2 Relational Data Model

The relational data model was the first formal data model as a theoretical basis of database, and based on first order predicate calculus (FOPC) (Codd, 1970). The relational model supported a relationally complete, non-procedural query language. Now, standard query language (SQL) is used to communicate with databases for data definition and data manipulation. Without the need to reconstruct a complex pointer system, relational databases can be used to refine complex organizational structures.

The basic idea of the relational model is that data are represented as a series of tables. Repeating groups or implicit hierarchies, and fixed structural links are not allowed. Logical relationships between the data are constructed at run time or are held in tables themselves. Tables can be used to represent both entities and relationships (Chen, 1976). Also cross-reference tables can be rebuilt without reorganizing the basic data. This is a great advantage in case many changes in data are needed.

The hierarchical or network model is a physical data model, but the relational model is a logical one. The logical relational model allows users to view one data structure in many different ways such as user views. This is one important benefit of relational databases leading to a higher degree of user acceptance.

The relational model consists of two intrinsic and two extrinsic parts. The intrinsic parts are a structural and a manipulative part. A structural part uses notions of domains, attributes, tuples, n-ary relations, and primary and foreign keys. A manipulative part includes tools: relational algebra and calculus (Graham, 2001).

In a structural part, given a list of sets A_1, …, A_n, each set of A_i is a called a domain. An attribute is a label for each domain. A tuple is a container that has only one element from each domain. In a table, domains are columns and tuples are rows. Attributes are column names. A relation is defined mathematically as any subset of a Cartesian product of sets so that a relation would be a table. An n-ary relation means that there are n attributes in a relation. A primary key set of attributes uniquely identifies each tuple at any given time. A foreign key set of attributes is used to link logically a relation to the other relations.

In a manipulative part, the most used language based on relational calculus and algebra is SQL (Figure 3.4). The SQL was partly derived from the motivation to produce a 'structured' language. This kind of relational query language operates on tables, or sets of tuples. A query always returns a table. The SQL consists of five functional languages: data query language (DQL), data manipulation language (DML), data definition language (DDL), transaction control language (TCL), and data control language (DCL) (Table 3.1) (Kochhar and Lad, 1998). There are several advantages in using SQL. First, it is efficient in that a lot of work can be done with only several lines of commands. Second, it is easy to learn and use because SQL uses English

like commands and a simple syntax. Last, it is functionally complete so that it can define, retrieve, and manipulate data in a table.

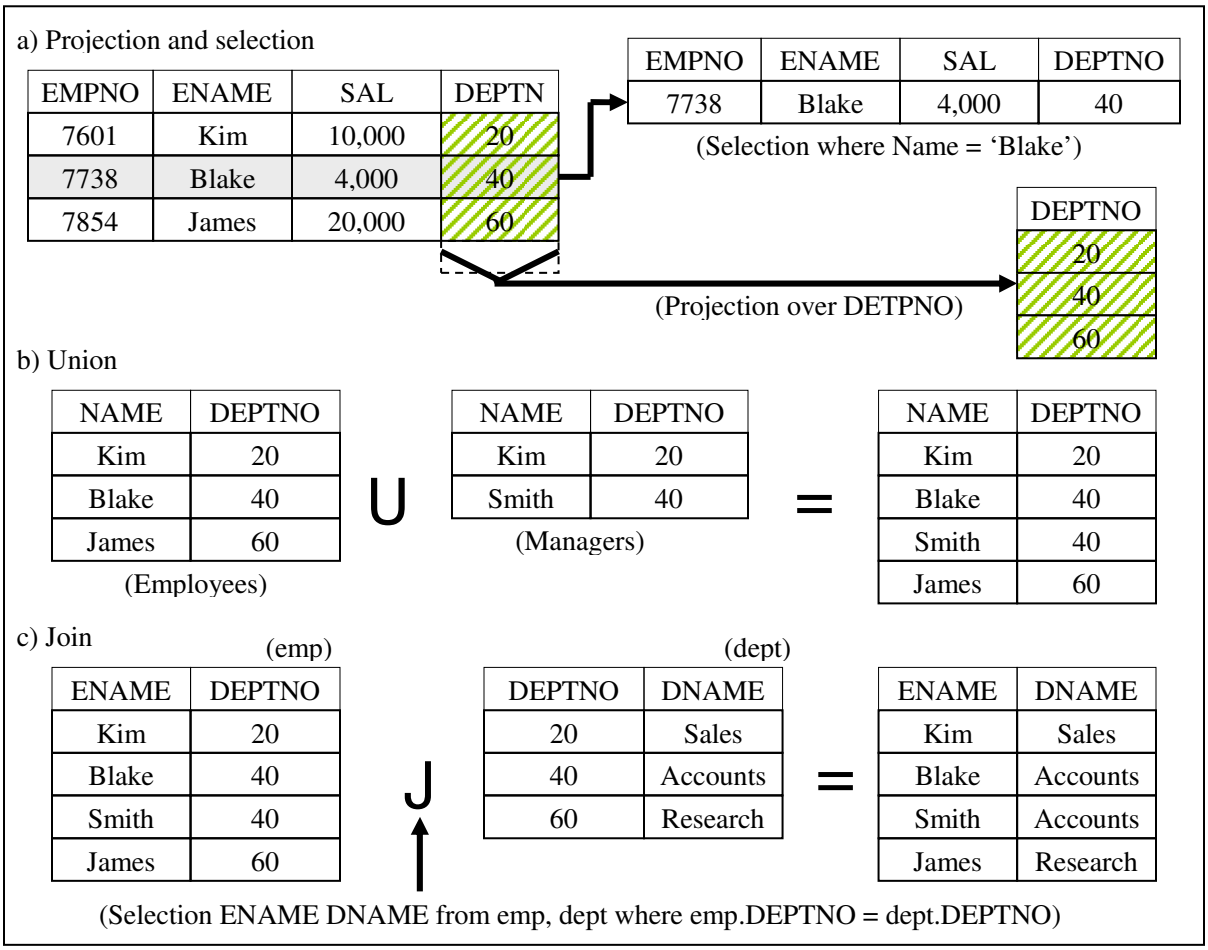


Figure 3.4 Relational Calculus and Algebra

Table 3.1 Standard Query Language (SQL) Commands (Kochhar and Lad, 1998)

Type	Command	Description
Data Query Language: DQL	SELECT	Data retrieval from table
Data Manipulation Language: DML	INSERT, UPDATE DELETE	Insert new rows or update existing rows in a table, or delete unwanted rows from a table
Data Definition Language: DDL	CREATE, ALTER DROP, RENAME TRUNCATE	Create, alter, or remove data structure from a table
Transaction Control Language: TCL	COMMIT, OLLBACK SAVEPOINT	Control or manage changes of a table executed by DML command
Data Control Language: DCL	GRANT, REVOKE	Control or manage access privileges for database and data structure

The extrinsic parts include an integrity part with respect to both entities and reference, and a design part consisting of the theory of normal forms. Five normal forms have been derived to restrict a relation and explain the integrity part and the design part of a relation model. Graham (2001) summarized the normal forms as follows. First normal form (1NF) insists that attribute value entries are atomic; that is, values must be numbers or strings. Second normal form (2NF) says that a primary key uniquely identifies tuples. Third normal form (3NF) says that, with 2NF, the other attributes are independent of one another and are all functionally dependent on the primary key. Fourth normal form (4NF) helps to avoid redundancy and fifth normal form (5NF) prevents a lossy joins, i.e. data is lost when two or more relations are joined and then decomposed to the original form.

In an integrity part, two types of integrity should be considered in a relational model: entity integrity and referential integrity. Entity integrity means that primary key cannot include an attribute that may take null as a value. Referential integrity means that, given relation R and S, if a relation R contains a foreign key whose values are taken from the primary key values of a relation S, then every occurrence of the foreign key in R must either match a primary key value in S or be null. The relation R and S may be the same. In five normal forms, 3NF also explains a functional dependency that is a special form of integrity constraint of a relational model. The theory of normal form can be considered as a hierarchy from 1NF (as a base) to 5NF (as a top). Therefore, the hierarchy of five normal forms shows an aspect of bottom-up design for a relational model.

A relational model is very strong because it is based on a logical formal theory that makes it possible to have a relationally complete non-procedural query language such as SQL. The

logic ensures that any job of this language can be proved mathematically. With a relational model, a database structure can also be easily changed without complete redesign of the database.

There are several weaknesses in the relational model (Grahm, 2001). First, the relational model handles relations, which are directly perceived in multidimensional form, unnaturally by refusal to recognize any object of higher type (abstract data types such as lists, vectors, graphics or drawings, etc) as atomic because of the theory of first normal form. Also, any method cannot be stored ready for use in the database. Second, normalized relations also rarely correspond to any object in the real world. This means a loss of information. The decomposition of an application is driven by considerations of computation or logic, rather than modeled on the structure of the application. Therefore, normalized relations cannot be reusable in isolation and be the assembly of systems from components. Finally, first normal form also disallows recursive queries that are queries about an entity's own relationships. An example of such a relationship is the parenthood relationship on the person entity.

With the emergence of object-oriented programming languages, the object-oriented model provides solutions to some of the limitations of a relational model. An object-oriented model allows abstract data types that correspond to objects in the real world. Also, the inheritance property of an object-oriented model can be applied for recursive queries (Graham, 2001).

3.3 Object-Oriented Model

Conceptually, an object in an object-oriented model stores both attributes and methods together. In an actual implementation, attributes and methods of complex objects may not be physically stored with the object identifier, but pointers to the physical location may be stored.

Generally, objects with attributes are stored in a database but methods of those objects are rarely stored in a database. Methods are implemented by the run-time system. It is important that any query language cope with this question of physical storage quite transparently. Therefore, most object-oriented databases are tightly coupled with a particular object-oriented programming language for both queries and implementation of methods.

In an object-oriented model, behavioral abstraction is emphasized and means the inheritance of encapsulated methods. In object-oriented database systems, it is very important to model the structural properties of data and make explicit inheritance and aggregation structure. For explicit inheritance and aggregation structure, several methods can be used to group objects: classification, association, composition, and aggregation.

Inheritance can be derived by classification and association. Classification is a method to group objects in classes according to their common structural and behavioral properties. For example, students and professors can be grouped as people. Association means that instances are grouped into a class according to some shared property. For example, the set of all things is red. Classification is used for essential objects and represents a subset relationship, and association is used for accidental objects and represents a membership relationship.

Objects can be grouped as components of some composite object in two ways: composition and aggregation. Composition is the way to group objects as parts of the structure of an object. For example, a car is composed of wheels, transmission, body, and so on. Aggregation is a method to group objects as a set of attributes of an object and includes both conceptual composition and physical composition. For example, a computer is a concept aggregated from concepts including name, operating system, several hardware and software, location, access right, and so on.

In an object-oriented model, inheritance and aggregation can interact. An object that may be an aggregate of several objects contains attributes and methods of those objects. An object that inherits attributes and methods from a parent object also inherits aggregation structure of the parent object.

Compared to the relational data model, an object-oriented model removes the need to perform expensive joins when objects are used in an application since objects are stored as coherent wholes. This makes them potentially much more efficient for applications involving complex objects such as web servers, multimedia databases, geographic information systems (GIS), and CAD/CAM systems.

3.4 Feature Data Model with Object-Orientation for GIS

With the emergence of GIS, the data model for capturing geographic phenomena has been focused on the spatial geometry and thematic attributes of the phenomena have been linked to the geometry. The basic geometries in the geometric model are point, line, polygon, and grid. Different geometric types need different types of data structure with their thematic attributes, so multiple geometry types cannot not be handled by one theme or layer.

The layer-based model emerged to effectively manage the geometric based data. As it allows only one type of geometry to be entered into a theme or layer, multiple geometric types of geographic phenomena must be represented by multiple layers. In the real world, geographic phenomena have not only spatial geometry and thematic attribute, but also relationships with other phenomena and temporal attribute. The layer-based data model, however, cannot contain the relationships among phenomena. Also, multi-temporal instances of a feature cannot be represented by a layer and more than two layers must be used for representing the changes of a

feature through time. The same phenomenon in different layers may not be the same any more because it may have different object IDs in different layers.

Since late 1980, there have been significant advances in both the data model of GIS and the programming paradigm of computer science. A feature concept has been developed to represent geographic phenomena in GIS (Guptill *et al*., 1990; USGS, 1997). A feature is a defined geographical entity and its object representation in the digital environment, so that it is the unit of aggregation and analysis instead of a theme or layer (Usery, 1993). A feature is also represented by multiple dimensions using space, theme, and time with their relationships among features. Therefore, the feature approach for a geographic data model may be the best solution for the multiple representations of real world geographic phenomena.

A feature data model can be constructed basically from the feature conceptual framework, the abstraction methods defined in SDTS, and key concepts of object-orientation. With the feature conceptual framework, features are represented by three dimensional aspects: space, theme, and time. Classes in the feature data model can be derived from the abstraction methods of spatial data in SDTS.

3.4.1 Conceptual Framework of a Feature

A conceptual framework of features has been built, which should be effective for various applications and resolutions (Usery, 1993). The framework includes spatial, thematic, and temporal dimensions with attributes and relationships for each dimension to build GIS using feature concept (Figure 3.5).

	Space	Theme	Time
Attributes	point, line, area, volume, pixel,…	color, size, shape, …	date, duration, period, …
Relationships	topology, direction, distance, …	is_a, kind_of, part_of, …	is_a, was_a, will_be, …

Figure 3.5 Conceptual Framework of Geographic Feature (Usery, 1996b)

As an example of building GIS using the conceptual framework for structuring features, a prototype feature-based GIS (FBGIS) has been developed for both vector (Tang *et al*., 1996; Usery *et al*, 2002) and raster data representation (Usery, 1994a and 1994b). Although FBGIS has been implemented using conventional programming techniques, the logical implementation strategy for the feature-based conceptual model was the object-oriented approach in which objects are constructed based on either sets or category theory concepts of prototypes.

3.4.2 Feature Abstraction Methods in SDTS Data Model

The establishment of a standard for the interchange of spatial data has been of significant interest to users and producers of spatial data, because of the potential for increased access to an sharing of spatial data, the deduction of information loss in data exchange, the elimination of the duplication of data acquisition, and the increase in the quality and integrity of spatial data. The spatial data transfer standard (SDTS) is a robust way of transferring spatial data between dissimilar GIS with the potential for no information loss.

As the application of computers in geography and cartography grew within the federal government, the U.S. Geological Survey (USGS) was designated the agency responsible for

developing spatial data standards in 1980. In 1992, after twelve years of development, the resulting standard, SDTS, was approved as Federal Information Processing Standard (FIPS) Publication 173. The FIPS version has been superceded by current version (ANSI NCITS 320-1998) and was ratified by the American National Standards Institute (ANSI) in 1998 (USGS, 1997).

Features are captured by an abstraction, which arises from recognition of similarities between certain objects, situations, or processes in the real world (Booch, 1994). The conceptual model of the SDTS is a good example of abstraction for geographic features (USGS, 1997). The SDTS conceptual model has three parts: a model of spatial phenomena, a model of the spatial objects used to represent phenomena, and a model of spatial features that explains how spatial phenomena and spatial objects are related. Four basic terms (classification, generalization, aggregation, and association) of an object-oriented model have been defined and explained in SDTS for spatial data so that each term can be directly used for a feature data model (Table 3.2). Classification can be used for defining a feature class. Generalization and aggregation can be used for grouping features into a feature type class. Association can be used for describing attributes and relationships of a feature.

Table 3.2 Terms for Spatial Data Abstraction in SDTS (USGS, 1997)

Term	Definition	Example
Classification	The assignment of similar phenomena to a common class.	Route 10, Lake Herric, and Chesterfield County are all classified phenomena.
Generalization	A process in which classes are assigned to other classes. The general class includes all the instances of the constituent classes.	Sewers are included in the more general class of utilities.
Aggregation	The operation of constructing more complex phenomena out of component phenomena.	A lock is an aggregation of walls, gates, and a reservoir.
Association	The assignment of phenomena to sets, using criteria different from those used for classification.	Concrete roads may be associated with concrete sewers, concrete locks, and other phenomena constructed of concrete.

A phenomenon is a fact, an occurrence or a circumstance. Route 10, Broad Street, Athens & Clarke County are all phenomena that belong to a class of phenomena. A characteristic of such a class is called an attribute, whose value is a specific quantity or quality of the attribute assigned to a phenomenon in that class. Whether a given phenomenon belongs to a class is determined by the definition of the class that includes characteristics that distinguish the class from the other classes. Those classes of phenomena are called entity types, and the individual phenomena are called entity instances. An entity instance is not further subdivided into phenomena of the same type. For example, an entity type is 'Bridge' with attribute 'Name' and 'Material'. An instance of this type, Bridge, might be the "Newton Bridge" composed of "Steel".

Entity instances may be aggregated into instances of different types of entity. A lock is partly composed of walls but it is not itself a wall. Entity types can be generalized into themes based on the definitional characteristics shared by more than one type. For example, 'Transportation' is defined by a function of both 'Railway' and 'Road'. Association of entity

instances is defined in terms of characteristics other than those used to define an entity type. A common association is the spatial domain, which groups all entity instances having coordinates within a specified range. A relationship is a special case of an association and exists between entity types.

Under the spatial object model, entity instances have a digital representation, an entity object, which may consist of one or more spatial objects. A spatial object may be an aggregation of other spatial objects, not all of which necessarily represent an entity instance. Entity objects also have generalizations and associations. In general, all characteristics of phenomena (entities) and their digital representations (objects) are the same (Table 3.3).

Table 3.3 Correspondence between Entities and Objects (USGS, 1997)

	Phenomena	Digital Representation
Class	Entity Type	Entity Object Class
Characteristic	Attribute	Attribute
Instance	Entity Instance	Entity Object
Characteristic	Attribute Value	Attribute Value
Generalization	Theme	Theme
Association	Spatial Domain	Spatial Domain

Entity objects have locational attributes, non-locational attributes, and relationships (topology, thematic and temporal relationships). Spatial objects will be used to compose the locational attribute of an entity object. A set of simple spatial objects are defined in SDTS, which are either primitive objects (not aggregated from any other spatial objects) or aggregated only from spatial objects from different classes (polygons are not aggregated from polygons, only

from rings, chains or strings). There is also the composite object that may be aggregated from simple objects or from other composites.

Under the feature data model, a feature instance is an instance of a defined entity and its object representation, and belongs to a feature type. If a class is viewed as a set whose members are the instances of the class, then a feature type is the intersection of the entity type and the entity object class. Therefore, all characteristics of entity and object related to the terminology such as classification, generalization, aggregation, and association will be the same for a feature.

3.4.3 Key Concepts in Object-Orientation for Feature Data Model

Even though object-oriented terms are ideal methods for a feature data model, the object-oriented model cannot replace the feature data model because an object can represent not only a feature but also a feature type, relationship, or any kind of geometric types. All things are objects in the object-oriented paradigm.

Three key concepts of the object-oriented paradigm used for building a feature data model are inheritance, encapsulation, and polymorphism. In an object-orientation with a feature, the encapsulation can hide information (data and method) in a class definition over hierarchical inheritance so that a class can be designed to represent characteristics of a feature that are relevant to a particular application. For example, schools and houses inherit attributes and methods of buildings that are basically area features. If schools have to be represented as area features and houses have to be represented as point features according to the resolution of an application, the houses need not inherit the area size attribute or the area size calculation method. A building class can encapsulate specific data and methods by constraints (Borgida, 1988).

The inheritance from super class to sub class makes abstraction of geographical features possible. For example, highway and cart track objects can inherit instance variables and methods from a road object. Sub class objects (highway and cart tracks) may have additional instance variables and methods unique to the sub class that the super class object (road) does not have. Polymorphism allows the same message with different instance references to invoke different methods for the appropriate classes according to the instance reference so that spatial operations on different but similar objects are made easier. For example, a method for distance measurement between two features can be applied to compute the distance between two buildings and between a building and a road, although the internal algorithm of computation is different.

3.4.4 Feature Data Model for Geographic Phenomena

A feature data model was derived from the feature conceptual framework with abstraction methods defined in SDTS and key concepts of object-orientation. With the feature conceptual framework, features are represented by three dimensional aspects: space, theme, and time (Figure 3.6). Classes in the feature data model were derived from the abstraction methods of spatial data in SDTS. A feature class was derived by classification. Feature, time, and attribute type classes were derived by aggregation and generalization. Finally, three attribute classes and various relationship classes were derived by association.

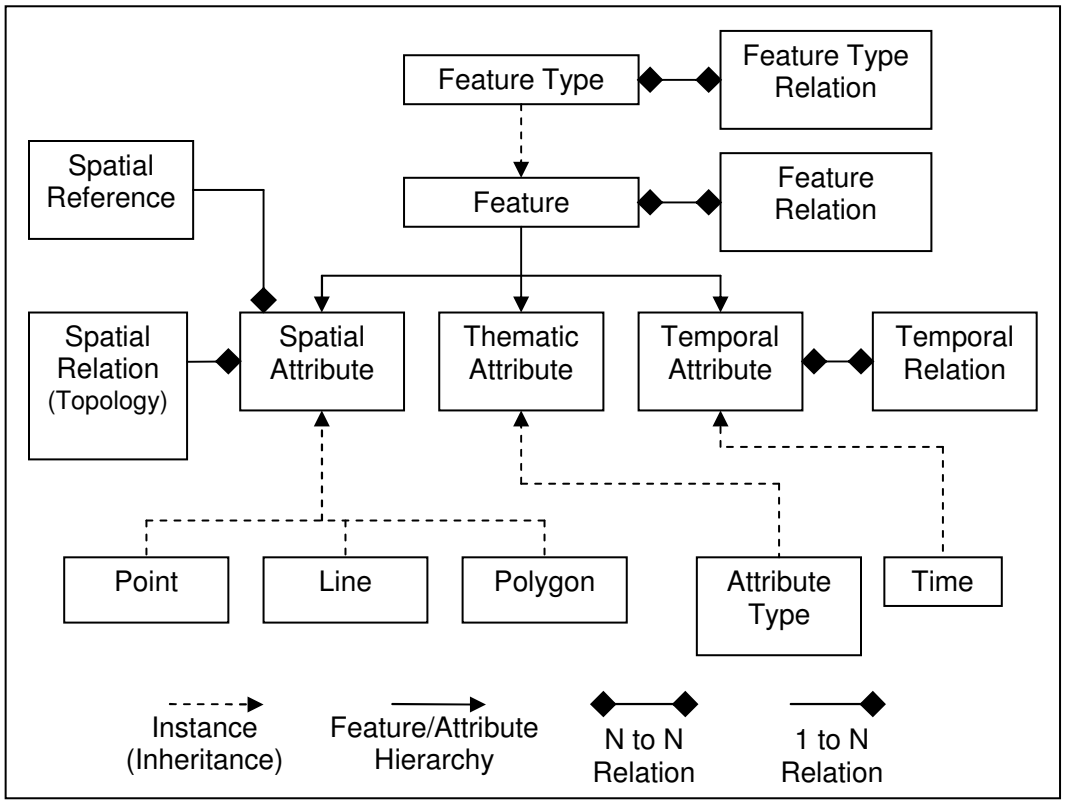


Figure 3.6 Feature Data Model using Object-Oriented Paradigm for Vector Data

Each dimensional attribute is connected to its own relationships. Spatial attributes are connected to the spatial relationship (or topology) with a many-to-one relation. Spatial attributes of each feature are parts of a spatial relationship. A spatial reference class can represent a geographic coordinate or a specific projection so that it has a one-to-many relation with spatial attributes. Temporal attributes are connected to temporal relationships with a many-to-many relation. Two temporal attributes are used for a temporal relationship and more than two relationships can reference the same temporal attribute. Specifically, the thematic relationships have been modeled by feature relationships because the thematic relationships in Figure 3.5 do not represent the relationships between thematic attribute themselves, but the relationships between features. Feature relationships also have a many-to-many relation with features. One feature relationship is composed of two features and more than two feature relationships can

refer to the same feature. The hierarchical structure between main classes in a feature data model such as feature type, feature, spatial attribute, thematic attribute, and temporal attribute is shown in Figure 3.7. Conceptually, a feature type class is the biggest class in a feature data model.

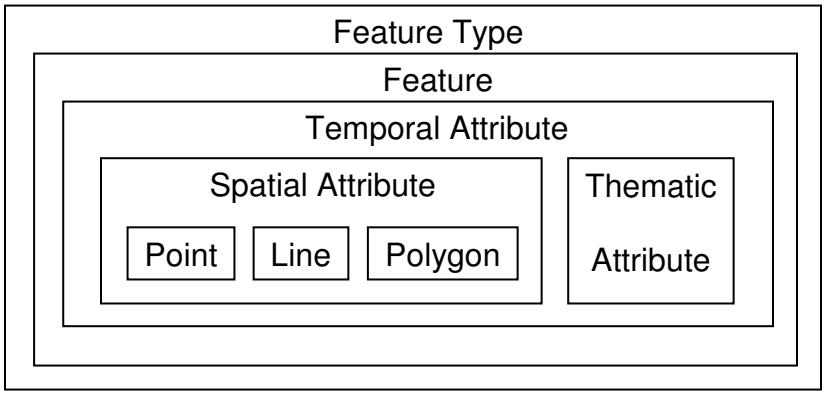


Figure 3.7 Hierarchical Structure of Classes in a Feature Data Model

A feature type class has been devised to represent a group of features based on their similarity. For all land parcels in general, a 'Land Parcel' as a feature type to which all land parcel features belong can be created. Both feature types and features are related to each other in the real world. These relationships can be implemented in a feature data model as a feature type relationship class and a feature relationship class that are used for binding two sets of feature types and features together, respectively. Relationship classes denote some form of direction such as 'flow into', 'belong to', 'part of', 'next to', and so on. Figure 3.8 represents feature type instances and their feature instances with their relationships. The 'Belong_To' relationship is an instance of feature type relationship. 'Next_To' and 'Brother_Of' relationships are the instances of feature relationship.

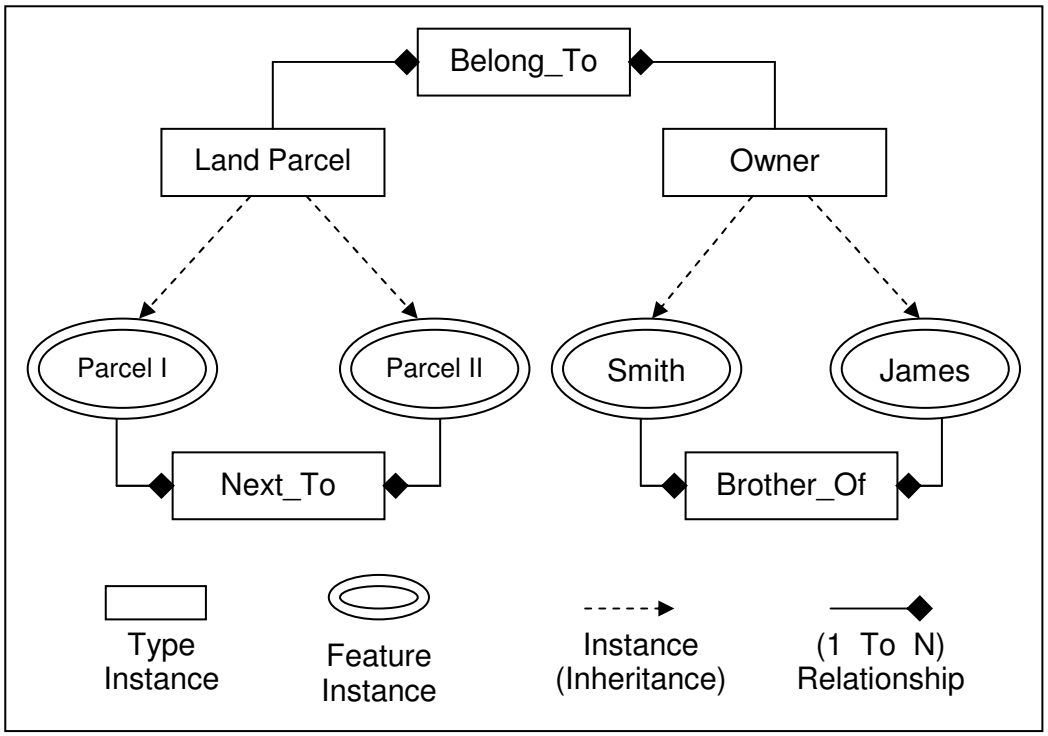


Figure 3.8 Feature Type and Feature Instances with their Own Relationships

With an object-orientation, an object can be instantiated as two instances that will inherit common attributes from super class. These two instances represent the same object in a different time with different attributes or changes through the time. If there is a building named 'Five & Ten' and that building has been changed from June 15, 1980 to July 10, 2002 by remodeling the roof type from wooden triangle to concrete flat and changing from grocery store to restaurant, in an object-oriented design, a feature instance called 'Five & Ten' has two sub-trees of attributes, one for attributes bound to June 15, 1980 and the other to July 10, 2002 (Figure 3.9). Each sub-tree contains a polygon describing a geometric attribute and a thematic attribute such as a roof type and material. Therefore, 'Five & Ten' can have two instances through time and be saved as one feature without duplicating the other buildings which have not changed through time.

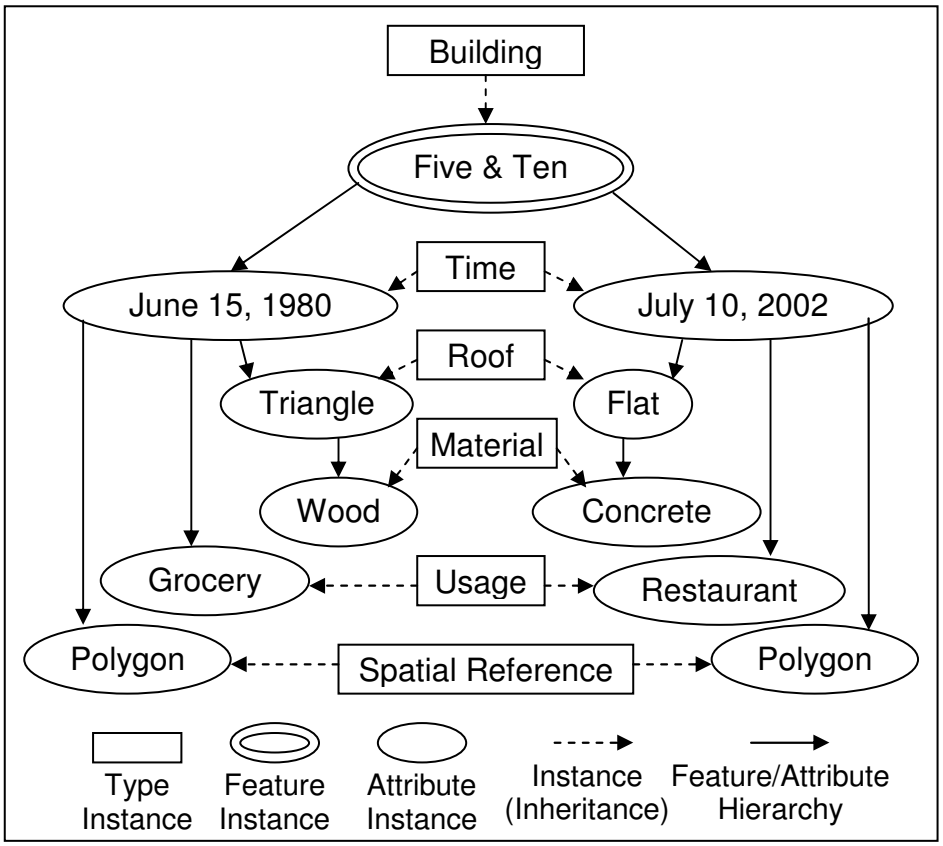


Figure 3.9 Feature Representation with Two Temporal Attributes

Attributes can be derived from Attribute Type objects to give a description in that '4' by itself has no meaning, but '4' derived from 'Lane' Type means 4 lanes (Figure 3.9). Therefore, Type in a feature data model is a convenient mechanism for making general descriptions for features and for imbuing attributes with more meaning.

Spatial attributes are used for spatially grounding these features to reality. In a feature data model, only three simple geometries among several geometric objects in the SDTS conceptual model are used to represent spatial attributes: point, line, and polygon (see Figure 3.6). A point is for a geometric point, which represents a set of coordinates (x, y, z). A line is merely a list of points. A polygon is just a line with the implication that it forms a loop, which is an essentially a list of points that describe a simple, closed polygon.

3.5 Discussion: Feature Data Model vs. Existing Data Model in GIS

The new feature data model derived in this chapter is modified from the existing feature-based data model and different from the existing layer-based geographic model in the following ways. First, the layer-based data model can only contain one geometry type such as point, line, or polygon in each layer. This mechanism has a limitation for the map representation. The feature representation on a map is always based on the scale. Therefore, the same feature types for different feature instances can be different geometric types. For building feature types, very small buildings will be mapped using point geometry rather than polygon geometry because of the difficulty for mapping it as polygon from the small scale source image data such as airphotos and satellite images. On the other hand, some large buildings should be mapped as polygon geometry rather than point geometry because of the size of the buildings. These two different sizes of buildings make it difficult to be contained in the same layer, even though they are in the same feature type. However, the feature data model has no limitation for the geometry type for each feature database. Also, for linear features, the highway can be saved as polygon and the central line for the highway can be saved as line under the same feature type in a feature data model.

Second, the layer-based data model can represent implicitly the thematic relationship among features in the same layer, but cannot describe the relationships among features in different layers. This makes disjunctive thematic relations. The disjunctive thematic relations can be solved by the hierarchical representation. A feature data model can represent any kind of thematic relationship among features because the feature data model uses basically the hierarchical representation from feature type to detail each feature's thematic attributes.

Third, in the layer-based data model, a feature with multi-temporal situations should be represented by several layers because one feature can only have one time attribute in one layer. Once a feature has been mapped into a different layer, the feature in each layer is not conceptually the same feature. However, in the feature data model, the feature can have multi-temporal situations. The temporal situations are from the same feature instance node so they reference exactly the same feature. Also, the feature can have different situation information according to the reference time because the spatial and thematic attributes and relationships of the feature reference the feature instance through the specific time node.

Someone may argue that a feature model is already used in a spatial database in a commercial GIS such as the geodatabase of Environmental Systems Research Institute (ESRI). However, the feature database based on the newly devised feature data model is different from ESRI's geodatabase for two reasons (ESRI, 1999b). First, the geodatabase still uses a layer-based model for each feature so that only one geometry type can be contained for one feature layer. A feature data model, however, can save different type of geometry for the same feature instance because the feature data model has no limitation on the geometry types for a feature instance.

Second, the geodatabase can only represent certain feature relationships. A feature in the real world has not only spatial relationships between features but also thematic and temporal relationships. To represent these relationships, the geodatabase can support the relationships between feature types. The example of the relationship between feature types would be the relationships between land parcel and land owner. However, a feature data model contains even more relationships: feature type relationship, temporal relationship, and thematic relationship. The feature type relationship can represent the same relationship as the relationship between land parcel and the owner in the geodatabase. The temporal relationship can capture the relationship

among different time spots for a feature. The temporal relationship will include “was a”, “will be” relations. The thematic relationship is the positional or thematic relation between features. The positional relationship include “is the right side of”, “is in the middle of”, “is the above of”, “is the below of”, and so on. The thematic relationship includes “is belong to”, “is part of”, “is connected to”, and so on.

3.6 Summary

In the first two sections of this chapter, existing data models were reviewed including hierarchical, network, and relational data models. The hierarchical data model may not deal with many-to-many relationships. A many-to-many relationship can be implemented efficiently in the network data model, but the network diagram is too complex even with simple data structure.

The hierarchical and network data models are physical models for managing data, but the relational data model is a logical model allowing multi-views for one data structure and flexibility for data management. However, the relational data model has limitations on abstraction of real world objects and recursive query of relationships.

The object-oriented model provides a solution to the abstraction of objects in the real world by allowing abstract data types and recursive queries using the inheritance property. Specifically, the object-oriented model removes the need of expensive joins in the relational data model so that it is potentially much more efficient for applications involving complex objects such as GIS and CAD/CAM systems.

With the advantage of the object-oriented model, a feature data model was derived from the feature concept, abstraction methods of SDTS, and the key concepts of an object-orientation. Geographic features can be effectively represented using three aspects: spatial, thematic, and

temporal dimensions. For each dimension, attributes and relationships are included in the feature data model.

Conclusively, the introduced feature data model has advantages over the traditional layer-based model in GIS. For example, multiple types of geometry can be used for the same feature class. Various types of feature relationships including spatial, thematic, and temporal dimension can be represented. Multi-temporal situations of a phenomenon can be saved in one feature. The feature data model also extends the capability of ESRI's geodatabase. Geodatabase can capture only the feature type relationships such as the relationship between land owner and land parcel, but the feature data model can represent even the feature relationships such as the relationship among land owners or land parcels (see Figure 3.8).

Even though a feature data model can effectively represent various relationships, there may be some disadvantages of the feature data model. First, it may be difficult to identify all relationships of a feature. Second, the identification of relationships of features requires manual effort because there is currently no automatic method for it. Finally, there is no standard term in the relationship representation of features so the same relationship may be represented by different terms. For example, 'Is A' relationship may be represented by 'Is A', 'Is-A', 'Is_A', and so on. This inconsistency makes the implementation of query procedures using the relationships difficult.

CHAPTER 4

SYSTEM DEVELOPMENT WITH FEATURE DATA MODEL

4.1 Introduction

A feature approach for system design and implementation can take advantage of the object-oriented paradigm because object-oriented analysis and design are conceptually compatible with the geographic feature model (Bian, 2000). Three basic abstraction levels in the object-oriented model are object-oriented analysis, object-oriented design, and object-oriented implementation. Object-oriented analysis used for the feature data model conceptualizes real world phenomena and identifies objects relevant to the phenomena. Object-oriented design is a formal model using attributes and methods of the objects that are identified at the object-oriented analysis level. Object-oriented implementation is a step to implement object abstractions (attributes and methods) in a computer environment (Abadi and Cardelli, 1996).

An object-oriented programming language cannot support persistent objects. The state of objects that are stored off-line on secondary storage devices continues to exist after a session terminates. Object-oriented languages have no ability to store and manage persistent objects off-line and no means for dealing with concurrency, security, and recovery in the manner of a database management system.

Object-orientation in databases has emerged in two forms: object-relational database and pure object-oriented database. There are two groups for each standard: the Committee for Advanced DBMS Function (CADF) and the Object Database Management Group (ODMG), respectively. The first group (CADF) laid the theoretical basis for object-relational databases and

the SQL3 query language that they support. This resulted in commercial object-relational databases such as Oracle 8 and Illustra. These object-relational databases support object-oriented features on top of a relational storage mechanism and use SQL3 as a query language. However, Taylor (1991) pointed out that an object identity (OID) is value-based, not reference-based, and is simply another candidate key because of the mathematical foundations of the relational model. Object-relational databases use a middle ground between object-oriented programming languages and relational databases by mapping language objects onto database tables invisibly and providing caching services to improve performance.

The second group (ODMG) laid the basis for standards for pure object-oriented databases and the object query language (OQL) for them (Atkinson *et al*., 1989). Specifically, OQL is part of the ODMG-93 standard (Cattell, 1996). The key discriminator between pure object-oriented databases and the object-relational databases is performance, though it depends on application type (Graham, 2001).

In this chapter, the designed feature data model in the previous chapter is implemented. Data model needs data structure for the implementation and the detail structure of classes in the feature data model was designed (Egenhofer and Herring, 1991). A system using the designed classes was designed and programmed. For the data storage of classes in the feature data model, the physical database structure was designed and implemented using a table structure in the relational database. Finally, multi-temporal instances of buildings were visualized using the developed system in order to test the built system and the database. In the study area, there was a hurricane between two dates of input image sources. Therefore, the application will be focused on the effective visualization of the different feature instances through the time.

4.2 Class Design

In the feature data model represented in Figure 3.6, fifteen classes have been identified for representing geographic features: Feature Type, Feature Type Relation, Feature, Feature Relation, Spatial Attribute, Spatial Reference, Spatial Relation, Point, Line, Polygon, Thematic Attribute, Attribute Type, Temporal Attribute, Time, and Temporal Relation. With the exception of two classes (Feature and Spatial Reference), these classes can be grouped into four groups according to the similarity of the usage of the class: type, attribute, and relationship, and geometry classes (Table 4.1). The feature class is used for the storage of feature instances. The Spatial Reference class, derived for a visualization purpose as a total view of geographic features, has coordinate and projection information for the frame of geometric attributes of features and bounding box information for an effective visualization of features as a whole.

Table 4.1 Groups of Classes in Feature Data Model

Groups	Classes	Usage
Type	Feature Type, Attribute Type, Time	Storage of descriptive information for referring classes
Attribute	Spatial Attribute, Thematic Attribute, Temporal Attribute	Storage of detail value for a feature
Relationship	Feature Type Relation, Feature Relation, Spatial Relation, Temporal Relation	Storage of detail relationship for a feature
Geometry	Point, Line, Polygon	Primary form of spatial attributes
Others	Feature	Storage of descriptive information for a feature
	Spatial Reference	Storage of coordinate system and spatial bounding

Type classes include Feature Type, Attribute Type, and Time class. All these type classes are used to define a type so that they have the type information and its description as a data member. The time class may have only a time value as a data member. A feature only needs

feature name as a data member because detailed feature characteristics are represented by the attribute classes. A feature class is inherited from Feature Type class so that the feature class also has feature type information as a data member.

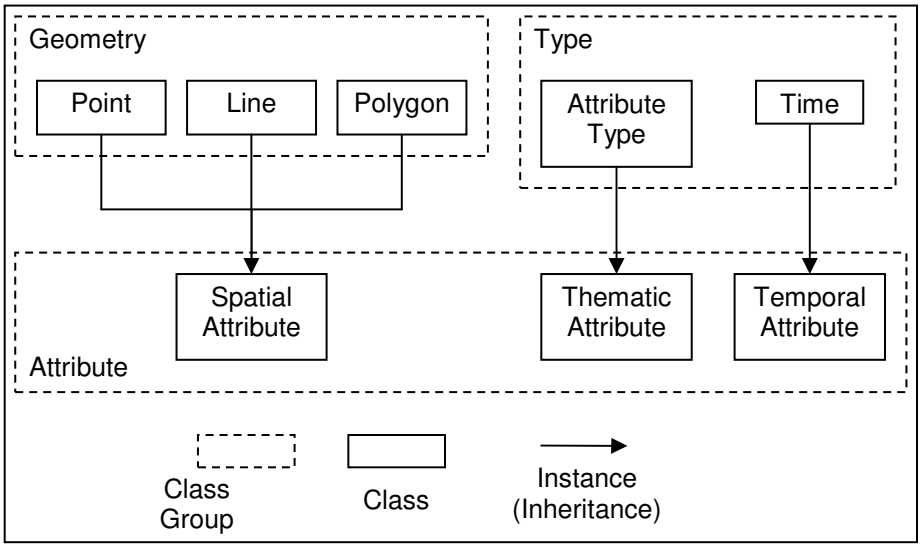


Figure 4.1 Attribute Classes Inherited from Geometry and Type Group

Attribute classes includes Spatial, Thematic, and Temporal Attribute. Attribute classes inherit information from other group classes: Spatial Attribute from geometry classes (Point, Line, and Polygon), Thematic Attribute from Attribute Type, and Temporal Attribute from Time class (Figure 4.1). Therefore, attribute classes can have the information and methods of related type or geometry classes by inheritance. Specifically, the attribute instances can only have meaning when they are instantiated by the related type classes. The number '4' has no meaning by itself but it means 4-lane when it is instantiated from the attribute type instance 'Lane'. The text 'October 2, 1999' has no meaning by itself but it can be considered as time when it is instantiated from time class.

Relationship classes include Feature Type Relation, Feature Relation, Temporal Relation, and Spatial Relation. The other relationship classes, except the Spatial Relation, have the relation

information and related objects as data members. The Spatial Relation class will have only one instance of geometric relationship of topology from all geometric attributes. Even though topology has more information than only spatial relationship, topologic structure with geometry is already known and is no different under the feature data model than it is under the layer-based model. The topology construction should be performed every time new feature instances are added in the database. The developed system focused on handling the feature thematic and temporal relationships, so the Spatial Relation class was not implemented in this prototype system.

Geometry classes are directly derived from MapObjects classes (ESRI, 1999a). These geometry classes contain not only geometry data but also methods (Table 4.2). All classes except geometry classes are designed simply for the data storage of objects in the prototype system, so they do not have methods except get and set methods for the encapsulation of their data members. If any action is needed for a class, it can be easily implemented later.

Table 4.2 Data and Methods for Basic Geometry Classes

Class	Data (Attribute)	Methods (Functions)
Point	X, Y, Z, Measure shapeType	Buffer, Difference, DistanceTo, DistanceToSegment, GetCrossings,Intersect, Union, Xor
Line	Extent, shapeType IsFullyMeasured, Length, Parts	Buffer, Difference, DistanceTo, etCrossings, Intersect, MultiplyMeasures, Offset, OffsetMeasures, ReturnLineEvent, ReturnMeasure, ReturnPointEvent, SetMeasures, SetMeasuresAsLength, UpdateMeasures, Union, Xor
Polygon	Area, Centroid, Extent, Perimeter, Parts, shapeType	Buffer, Difference, DistanceTo, GetCrossings, Intersect, IsPointIn, Offset, Union, Xor

4.3 System Design and Implementation

Usery (1996b) used an object-oriented approach to derive a feature as an abstraction of real world phenomena, but the FBGIS was implemented using conventional functional programming methods. Vckovski (1998) used the object-oriented approach for interoperable and distributed processing in GIS. High interoperability means that the complexity should be reduced while keeping high flexibility and expressivity. The complexity reduction is accomplished by encapsulation (information hiding) of properties (data) and operations (functions or methods), which is one of the basic concepts in object-orientation. Much of an object's complexity is hidden for the object user, and the object is typically accessed by sending and receiving a set of messages to and from the object. An object-oriented programming is therefore very useful to achieve interoperability.

Previous studies have shown that the object-oriented approach is a good method for the development of an information system. It leads to quality software which is extendible and reusable (Meyer, 1988). The reuse of software tends to improve the quality because reuse implies testing and debugging.

The conventional functional approach starts with the first question: "what does the system do?" This first question is adequate if the written procedure solves a fixed problem once and for all. Object-oriented design avoids the first question as long as possible, and starts with the decomposition of the categories of objects in the problem domain (Meyer, 1987). The object-oriented approach may also result in good user interfaces, including both visual and non-visual aspects because if the objects are well chosen, the user can easily form a mental model of the system (Oosterom and Bos, 1989).

The system developed in this chapter for a feature data model consists of file import/export module, database, and display module (Figure 4.2). The file import/export module can import a shape file into a feature database and export parts of a feature database into a shape file. The import module is mainly used for populating a feature database. For a database, a feature database has been designed in the next section. A feature database is used for the storage of the data members of the classes in the feature data model (see Figure 3.6). The display module is used for data visualization, not only for the geometry data, but also for the hierarchy of the feature data structure, thematic attributes, and background images or other file types of vector data as different layers.

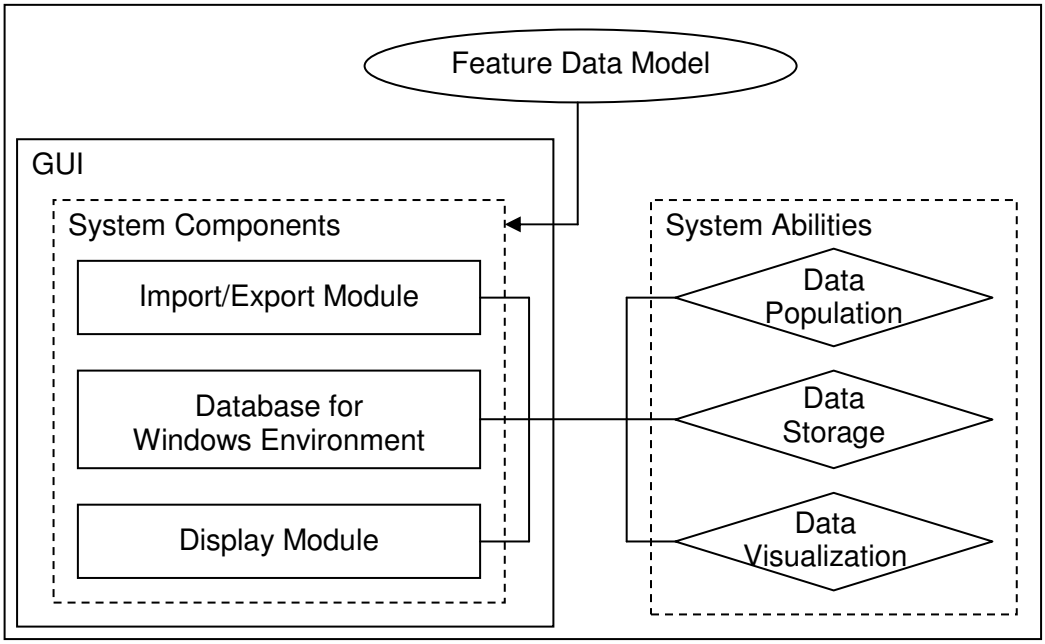


Figure 4.2 System Design from Feature Data Model

The system is implemented using Visual Basic 6.0 and some components in the display module have been customized from MapObjects2 (Ryu, 2000; Stephens, 2000). The main graphical user interface (GUI) has two buttons that are for the import/export and display modules (Figure 4.3). The import/export module has two components: main import/export interface and

sub-option menu interface (Figure 4.4). Once the import option is selected in the main import/export window, the sub-option interface will be displayed for the schema construction of the feature database.

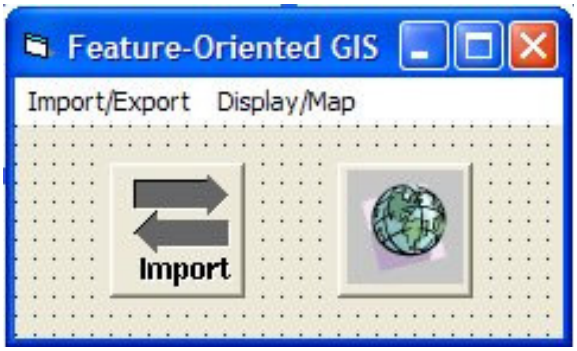


Figure 4.3 Main GUI of the System for Feature Data Model

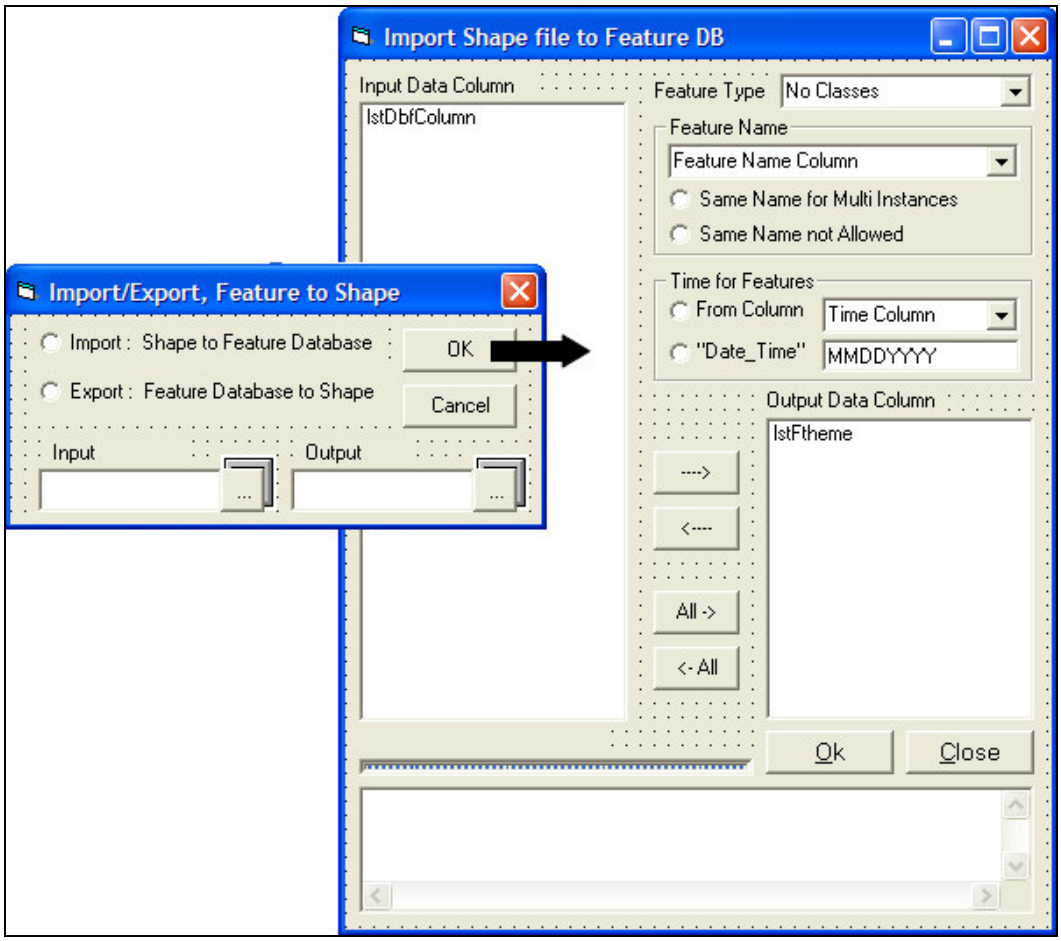


Figure 4.4 GUI for the Import/Export Module

For a schema construction of a feature database in the sub-option window, a feature type can be named from the input filename. The feature name, time, and thematic attribute in the feature data model can be populated from the selected column of the data file of the input shape file. For a feature name, the schema for a feature database allows the same name with different numbers to be used for multiple instances of a feature. For example, a feature 'Broad Street' can be instantiated as three different instances as 'Broad Street_1', 'Broad Street_2', and 'Broad Street_3'. With this naming scheme, three instances of 'Broad Street' can be handled as either one feature for searching features with a single name or three different features containing different situations such as a certain block for road construction or changing lane numbers through the same road.

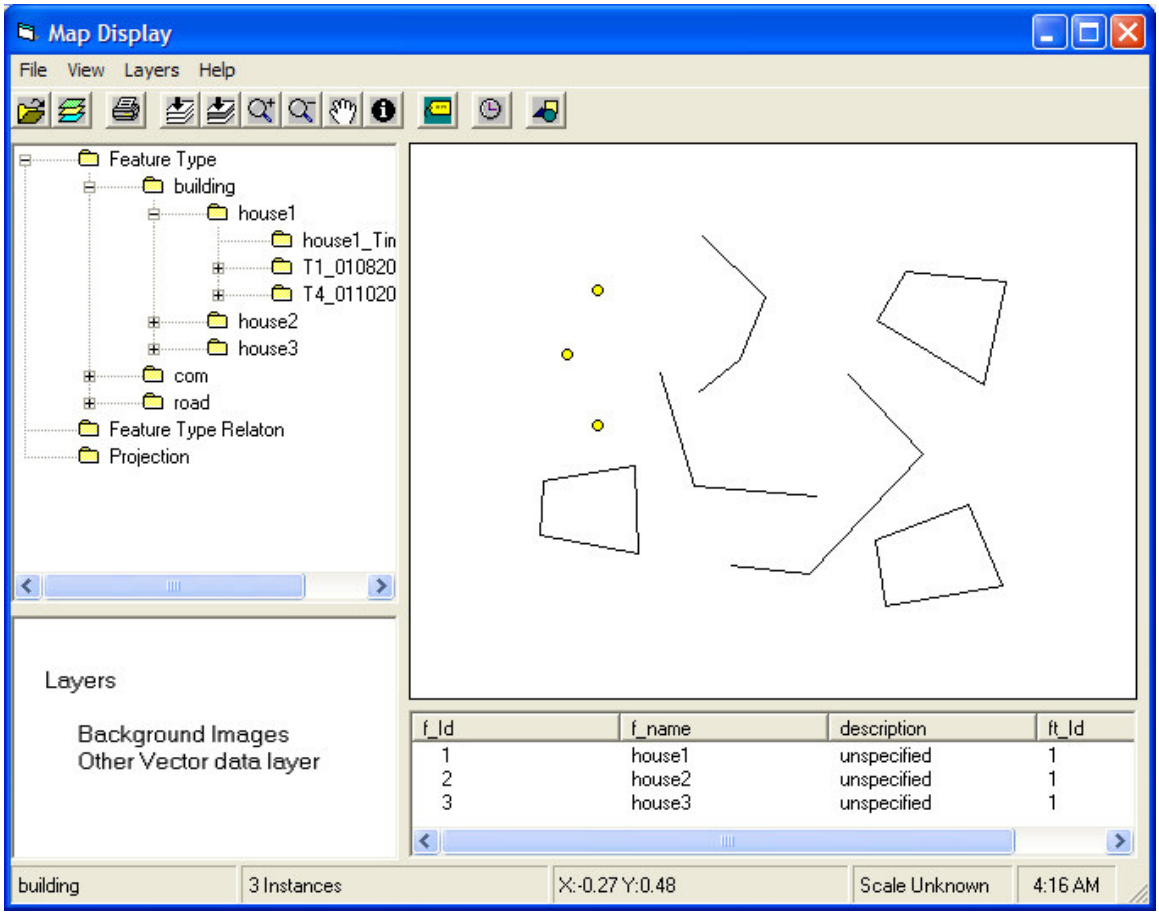


Figure 4.5 GUI for the Display Module

The display module is composed of four parts: a hierarchical viewer for a feature database, a map area for viewing geometry, an attribute viewer, and a layer manager (Figure 4.5). The hierarchical viewer uses a tree structure for the representation of the hierarchical structure of the feature data model. The map area is a canvas where the geometry is displayed. The map area is directly connected with the hierarchical viewer. If an instance of any class is chosen in the hierarchical viewer, then all objects under the chosen object in the hierarchy will be selected and displayed using different colors from those that are not selected.

The Attribute viewer is used to visualize the data members for each object. Like the map area, the attribute viewer is also directly connected with the hierarchical viewer. Once any object is selected, the data member of the object will be shown in the attribute viewer. Finally, the layer manager is used to load background images or vector data layers that are not in a feature database, but are used for the comparison or overlay analysis with the features in the feature database. For background images, various kinds of image formats can be supported such as ERDAS Imagine files (*.img), windows metafile (*.wmf), standard image data such as bitmap (*.bmp), gif (*.gif), jpeg (*.jpg), tiff (*.tif), and others.

4.4 Feature Database

For data storage, an object-oriented database may have better performance for applications in using an object-oriented programming language than a relational database (Graham, 2001). If a programming language is based on object-oriented programming environments, the data model is harmonious with the object representation in an object-oriented database. On the other hand, considerable data transformation, construction, and decomposition occur in the data transfer process into a relational database (Milne *et al*., 1993).

Milne *et al*. (1993) composed a spatial data model based on object-orientation from the draft of the SDTS, and tried to implement the spatial data model using a commercial object-oriented database, ONTOS (ONTOS, 1991). The performance test for spatial data between using ONTOS and a relational database, ORACLE, was executed, and the contour retrieval execution using ONTOS was about 10 times faster than using ORACLE.

Persistence is the property of an object through which the object continues to exist after its creator ceases to exist, and/or the object's location moves from the address space in which it was created. The type of object persistence is one of the following (Atkinson *et al*., 1983): 1. Transient results in expression evaluation, 2. Local variables in procedure activation, 3. Own variables, global variables, and heap items whose extent is different from their scope, 4. Data that exists between executions of a program, 5. Data that exists between various versions of a program, and 6. Data that outlives the program. Introducing the concept of persistence to the object model gives rise to object-oriented databases. Persistence is archived through a modest number of commercially available object-oriented databases. Another reasonable approach to archive persistence is to provide an object-oriented skin over a relational database (Booch, 1994).

Though object-oriented databases have a number of advantages over relational databases, there are still unsolved problems concerning query optimization, locking, and non-procedural query languages. In an object-oriented database, it is difficult to implement query optimization fully, because optimization requires examining the implementation but object-orientation hides information by encapsulation. With distributed object-oriented databases, the guarantee of network-wide object identity is still unsolved. It is related to the locking and commit-strategy over a network. Therefore, users should consider object-oriented databases for performance

critical applications, and object-relational databases for record-oriented applications and extensions of existing systems.

There are also no explicit standard query methods such as SQL for object-oriented database. In object-orientation, objects themselves are the storage for the data members and the data members of an object may be different from one application to another. The objects scheme for an object-oriented database should be modified from one application to another in order to fully contain the data members of the objects. To avoid these problems, it is possible to implement an object-oriented paradigm into a relational database name as an object-relational database such as Oracle8.x (Oracle Corporation, 1998). In this way, the object-oriented paradigm can be used for system design and implementation and then the objects can be entered into tabular structures of the relational database as a physical storage of objects. Also, entered objects in tables can be easily retrieved by SQL in the relational database.

In this study, a Microsoft Access database has been used for the physical storage of the objects of the feature data model (Figure 4.6). The data members of each class in Table 4.1 will be entered into a table. The tables in a feature database are related to each other for populating the data members of the classes when the feature database is uploaded into the system based on feature data model. The entity-relation between tables in a feature database is only used for populating objects.

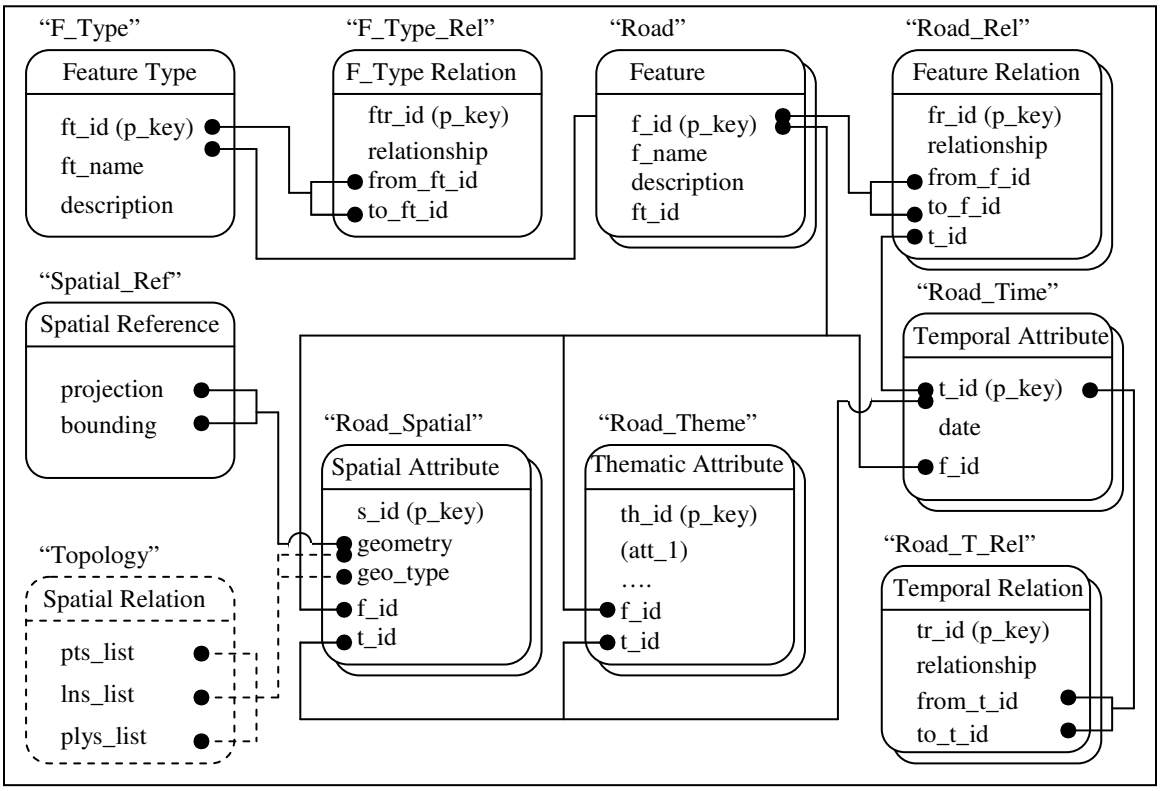


Figure 4.6 Table Structure for Classes in Feature Database

Once a shape file is imported into a new feature database, Feature Type, Feature Type Relation, and Spatial Reference classes will have an instance so that the corresponding table for each class will be created and populated. A feature table for Feature class will be named after the feature type instance such as 'Road' and 'Building'. All attribute and relationship classes, except Feature Type Relation class, will also be named after feature type name ('Road') with their own identifier ('Theme') such as 'Road_Theme' for thematic attribute of the 'Road' type features (Figure 4.6). The instances of Attribute Type class will be used automatically for populating the column names of the thematic attribute table. The instances of Time class will be used for populating instances of the 'date' column in the temporal attribute table.

If a shape file is imported into an existing feature database, then only the feature type instance related tables which are feature, feature relationship, spatial attribute, thematic attribute,

temporal attribute, and temporal relationship will be created and populated. The imported shape file should be under the same spatial reference information of the spatial reference table in the existing feature database. The feature type and feature type relationship tables are only populated with a new instance.

4.5 Multi-Temporal Representation of a Coastal Area

The study site is a part of the U.S. Marine Corps Base, Camp Lejeune, which is located along the North Carolina coast. The study site is very small (about 350m X 250m) to visualize individual buildings (Figure 4.7).

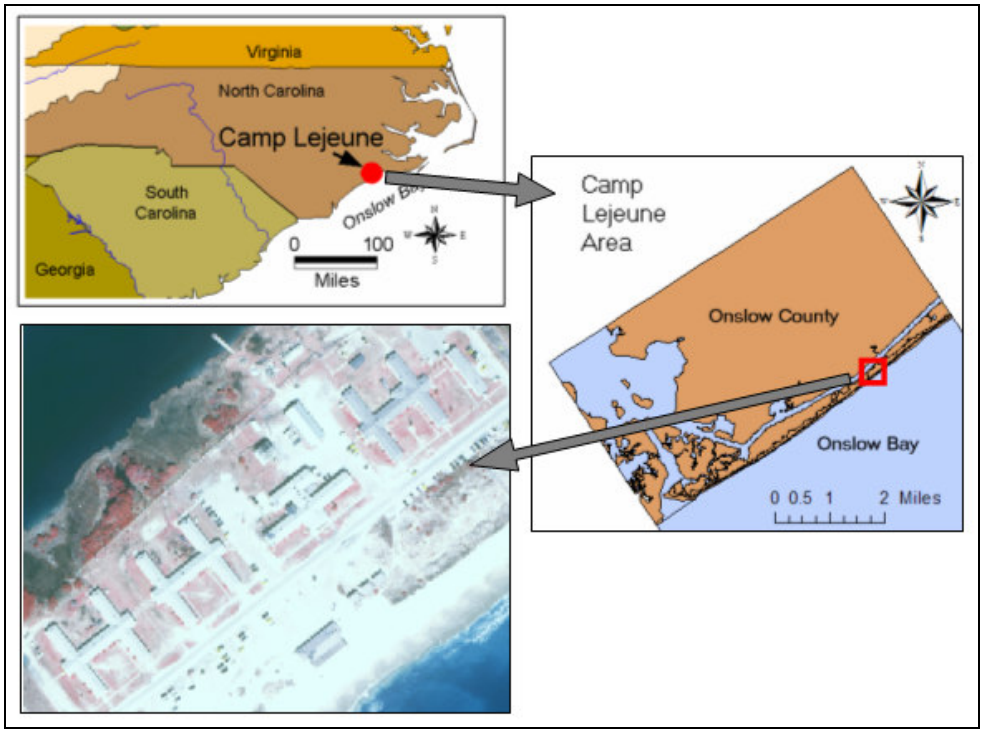


Figure 4.7 Study Site

On the study site, two different remotely sensed images were acquired less than one year apart: a color infrared airphoto (CIR) (09/1999) and Ikonos data (05/04/2004). In this application, buildings are mapped and visualized so very high resolution images were required. The CIR

airphoto image was scanned at 1.2 m ground resolution. The Ikonos image has 1 m ground resolution in the panchromatic band and 4 m in the multispectral bands. For enhancing visual interpretation ability, the Ikonos multispectral image has been pan-sharpened so that the resulting pan-sharpened image has a 1 m ground resolution. The pan-sharpening is an image enhancement method to merge the lower spatial resolution multispectal image with a higher resolution panchromatic image by replacing the intensity (or brightness) component of multispectal image with the panchromatic image.

Figure 4.8 Floyd Hurricane Paths on the Study Area on September 16 in 1999

Hurricane Floyd occurred over the study area on September 16, 1999 (Figure 4.8). When Floyd struck the study area, it was a level two hurricane with the wind speeds of 105 mph. A level two hurricane may cause some damage to roofing material, doors, and windows, and considerable damage to mobile homes and small craft in unprotected anchorages (NOAA, 1999).

In the study area, data extracted from the CIR image in 1999 and the Ikonos image in 2000 shows the removal of some features by Hurricane Floyd. Also, hypothetical data for 1995 has been created by duplicating 1999 data and removing one building from the duplicated data in order to visualize the addition of a feature. With these three input datasets, one building was built between 1995 and 1999 and eight buildings were destroyed between 1999 and 2000 (Figure 4.9).

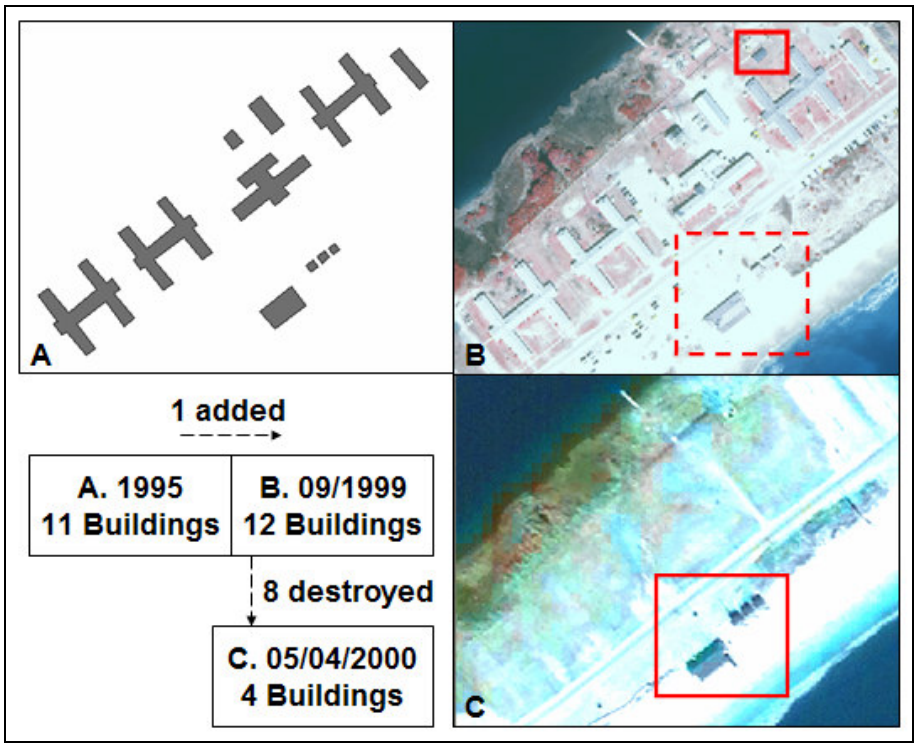


Figure 4.9 Source Data for Building Features with Different Time

During the population of features from different times, if a feature exists already in the feature database, the attributes and relationships of the feature refer to the existing feature based on the different times. A temporal relationship of the feature with different time can be explained by the 'Was_a' relationship: 'A feature in 1999 was a feature in 1995' If the feature does not exist previously, then the temporal relationship of the feature will have the 'built' relationship by itself without any reference to another feature: 'A feature has been built in 1995'. Figure 4.10

shows how building features are populated in the feature database from three different temporal datasets.

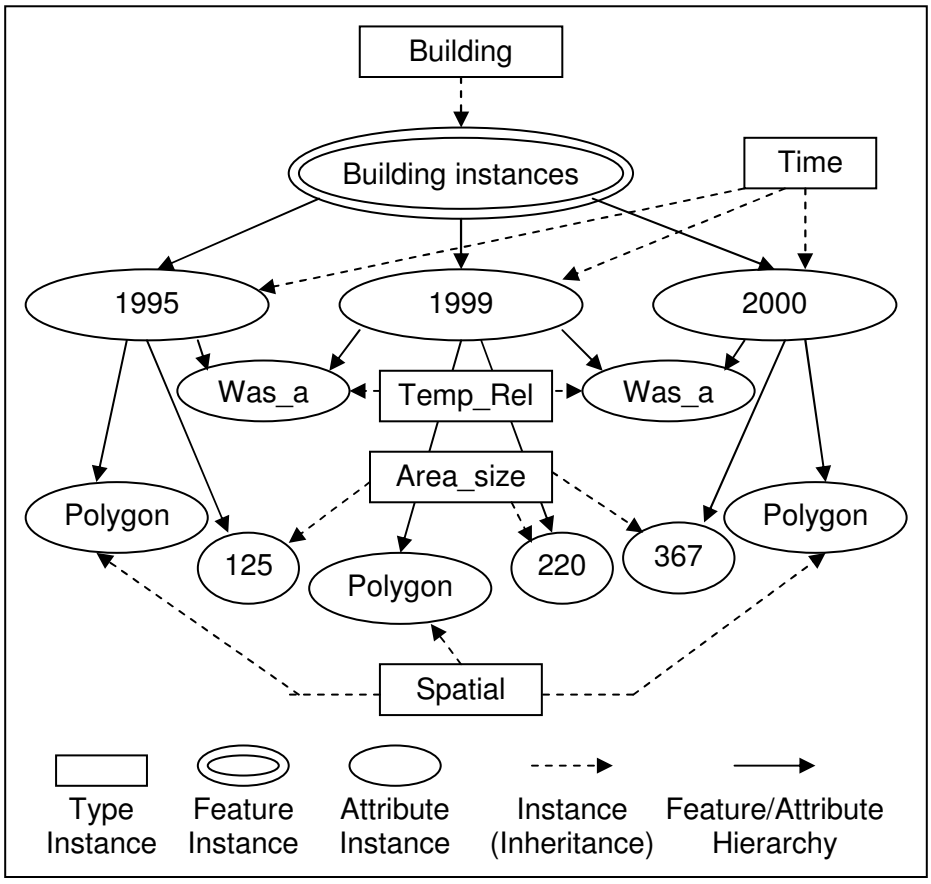


Figure 4.10 Populating Features on Three Different Sources

For the visualization of building features, the feature display module in the developed feature system has been used (Figure 4.11). The tree viewer in the display module shows the hierarchical structure of building features. Each feature has its time stamps and each time stamp has the feature relationship, spatial attributes, and thematic attributes. When a node in the tree structure is selected, the data members of the node will be shown in the attribute viewer (Right bottom section in Figure 4.11).

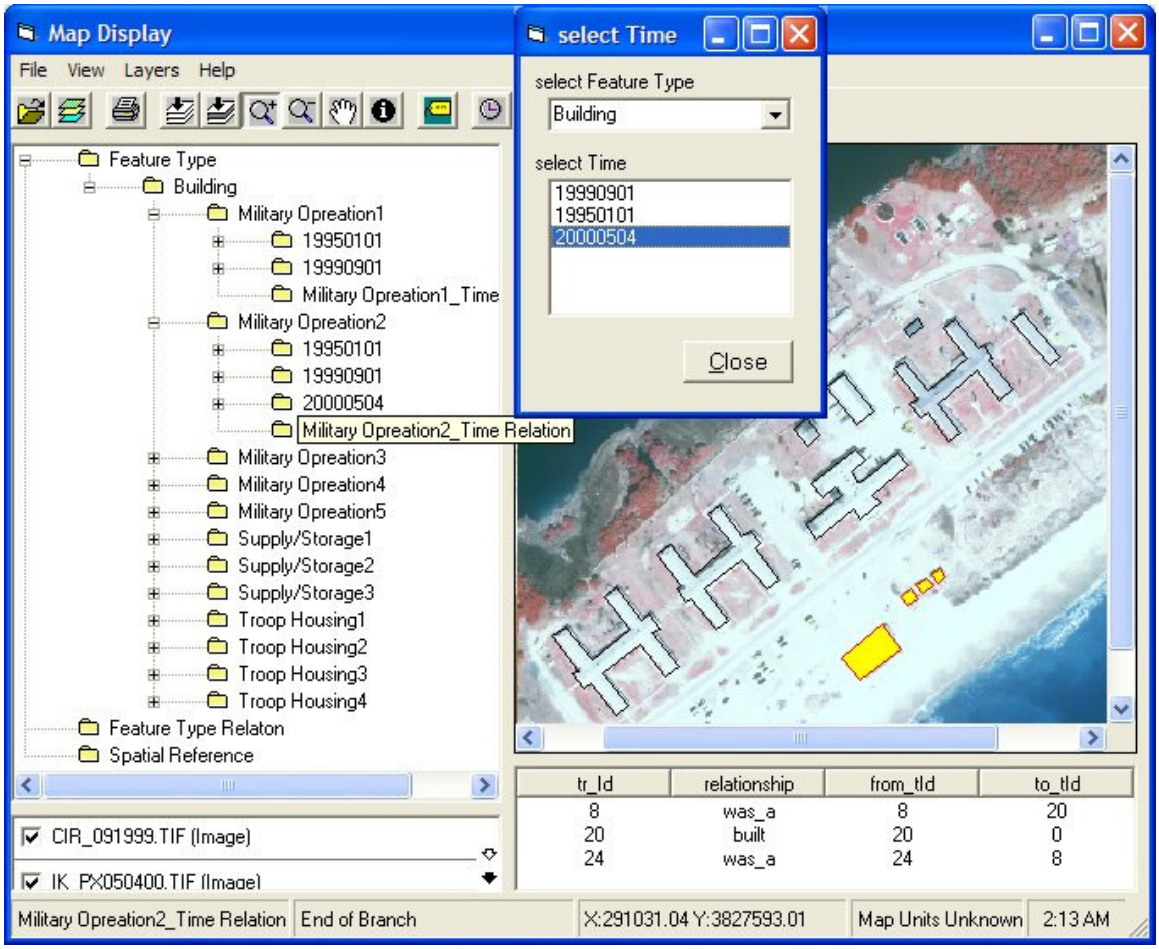


Figure 4.11 Feature Query based on the Time

The temporal differences of building features can be easily retrieved using the time selection tool ('select Time' window in Figure 4.11) that is loaded by selecting the clock icon on the menu bar. The geometry viewer in Figure 4.11 shows some building features selected when the specific time was chosen in the time selection tool. If another time is chosen in the time selection tool, then the features with the selected time will be shown. Therefore, features at the specific time spot can be easily and effectively queried and differentiated from the features at other time spots. Also, the time stamps of each feature can be shown easily under the feature instance node in the tree viewer. In this study, there were 12 building features in 1999, but only 4

buildings were left in 2000 after the hurricane that occurred during the time gap between the two datasets (Figure 4.12).

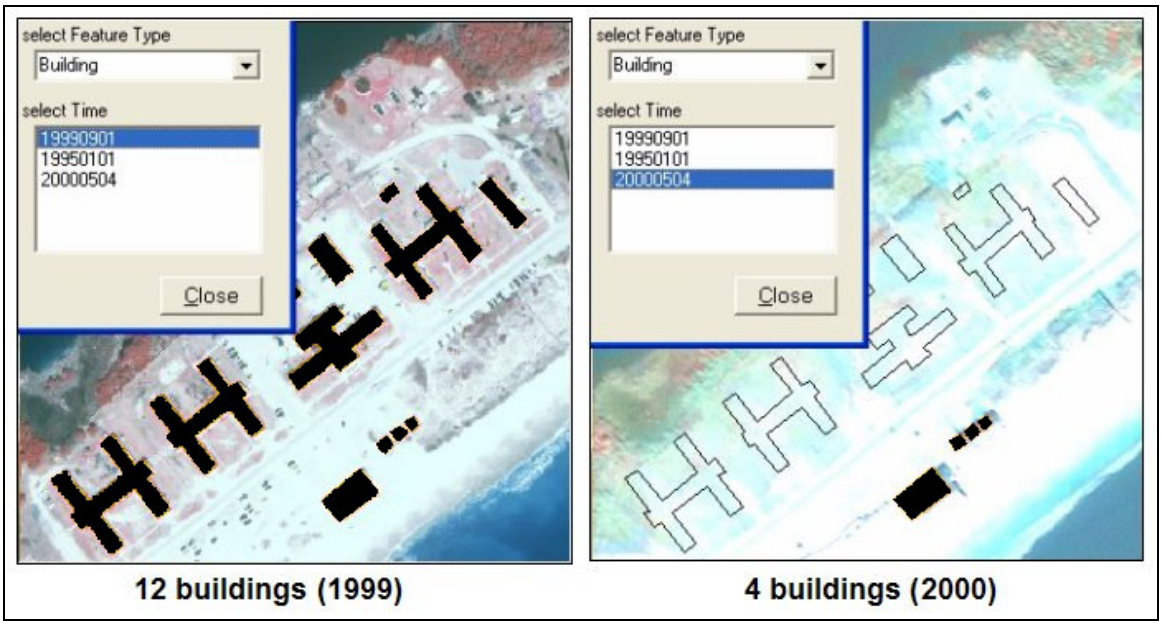


Figure 4.12 Temporal Differences of Buildings between 1998 and 1999

4.6 Summary

With the feature data model, a prototype system was developed in this chapter. Fifteen Classes in the data model were designed and used to build the system. The classes can be grouped into type, attribute, relationship, and geometry classes with the exception of the Feature and the Spatial Reference classes. Type classes include Feature Type, Attribute Type, and Time classes. Attribute classes include Spatial, Thematic, and Temporal Attribute classes. Relationship classes include Feature Type Relation, Feature Relation, Temporal Relation classes. Geometry classes include Point, Line, Polygon classes, which come directly from MapObjects classes. Spatial Reference class can contain the projection information.

With the derived classes, the developed system has three main parts: import/export module, display module, and database for feature storage. The import/export module can import

a shape file into a feature database and export parts of a feature database into a shape file. The display module is designed to visualize all information of features including geometry. Specifically, the tree viewer in the display module can effectively visualize the data structure of features because the feature data model uses hierarchical structures. For the storage of features, the feature database was schematically designed and implemented using tables in the relational database. Using the import/export module, existing spatial data such as a shape files can be easily imported into the feature database.

The developed system was applied for the visualization of geographic features with multi-temporal instances. In the study area, there was a hurricane that caused changes on the artificial structures directly. In reality, the hurricane destroyed many buildings in the study area so that those buildings disappeared between two source images with only a five month time difference. The changes in buildings on the coast area were effectively represented with one hypothetical situation and two real situations. Background image data were used to show the real situation in the study area.

CHAPTER 5

SPATIAL DATA GENERATION WITH RULE-BASED EXPERT SYSTEM

5.1 Introduction

One of the most important capabilities of a geographic information system (GIS) is to interpret and map data for solving spatial problems. Generally, researchers use a GIS interfaced with a statistics package for spatial analysis. One of the other approaches for the analysis of spatial data has focused on artificial intelligence (AI) techniques such as knowledge-based systems with GIS.

Artificial intelligence can be considered as tools that enable computers to be intelligent to emulate human thought to help in solving problems (Winston, 1992). The knowledge system in AI has structured knowledge about a field of expertise so that it can be used for data interpretation. A knowledge system is called an expert system because this system solves problems with the knowledge of experts (Carrico *et al*., 1989).

Although some AI techniques have been applied to generate or interpret spatial data with GIS, it is very difficult for a non-expert to develop and use a hybrid system combining GIS and AI techniques. The reason is most AI techniques require computer programming skills in order to apply knowledge to the system, which is not a simple process. However, a rule-based system is the most common technique in knowledge-based AI for a hybrid system with GIS because it has a very simple rule structure to derive solutions.

The basic unit of knowledge in the rule-based systems is the rule, defining a so-called ‘rule-based expert system’. A rule is a conditional test-action pair (if condition is true, then

action) and can be programmed using a natural language such as English, Japanese, and Korean. Therefore, once the rule-based engine (or interpreter) for the system is developed, the rule for an application can be easily constructed by a non-expert in computer programming (Graham, 1989; Naylor, 1989; Nikolopoulos, 1997).

In remote sensing and GIS, previous studies using rule-based expert systems focused on the automatic interpretation of aerial photographs or satellite images, or on the automatic mapping of spatial data for experts (Usery *et al*., 1988; Foschi and Smith, 1997; Wilson, 1997). However, the rule-based expert system can be applied to the interactive interpretation of aerial photographs for neophytes.

Automatic interpretation approaches with rule-based systems use the decision theoretic method for feature recognition on the aerial photographs. A decision theoretic method uses hierarchical decision-making procedures by inference rules incorporated into a tree structure (Estes *et al*., 1983). The inference procedure in the automated approach is very strict. Once the rule has been created, then there is no way to incorporate the situation information during the inference process of image interpretation. Therefore, the rules should be very detailed and specific to an application, such as soil mapping for a certain study area. Thus, it is very difficult to use the same rules for a new problem or different data set even with a similar problem. The rules should be complete for accuracy of the result of the automatic interpretation. A small mistake in the rule construction may produce large errors in the results of the interpretation. Therefore, the automatic interpretation is appropriate only for experts (the producers of the rules) for their specific application purposes.

Interactive approaches also use the inference rules with a tree structure but are based on the manual method of interpretation of aerial photographs and satellite images. The decision of

each node in the decision tree is made by the users based on the manual image interpretation using question and answering sequence, so the results of the inference procedure are very flexible according to the user's decision. The inference rules in the interactive approach only give guidelines or suggestions for the final solution for the decision, which also depends on the users' recognition of features in the images. Therefore, the rules are not necessarily very detailed and specific to an application. Instead, they can be broadly applicable to a group of similar problems such as aerial photo interpretation. Also, interactive methods are more focused on the user's aspect with existing expert's knowledge as a guideline for image interpretation.

Automated interpretation approaches for aerial photographs and satellite images are strict, application-specific methods, and focused on the producers' aspects. On the other hand, interactive interpretation approaches are flexible, broadly applicable methods, and focused on the users' aspects. The interactive method can utilize human recognition ability to include situation information. This manual approach for the basic recognition of features will improve the accuracy of the result of image interpretation. Therefore, the interactive method has been chosen for the inference procedure in this paper.

In this chapter, a rule-based expert system is developed for an interactive interpretation of images and to integrate this expert system with GIS for spatial data generation and mapping. The produced data will be directly saved into the feature database. This hybrid system is not oriented to a specific problem. Instead, it can be applied to solve various problems if the rules of the problems are already constructed in this system. In this paper, this system is applied to assist the interpretation and mapping of aerial photographs for an urban area.

The remainder of this chapter is organized as follows. First, the architecture of the integrated system of GIS and a rule-based expert system will be discussed. The next section

describes the system development process including both the system module and database design. This is followed by a geographic application to evaluate the performance of the integrated system. A summary is presented in the final section of this chapter.

5.2 Integration of GIS and a Rule-Based Expert System: System Architecture

An integrated system of GIS and a rule-based expert system has three main architectural components: a rule-based expert module, user interface for input/output, and a GIS module (Figure 5.1). The Expert module is composed of three components: the knowledge base, the inference engine, and the knowledge acquisition module. The knowledge base is stored and retrieved using a database. The GIS module includes a spatial data loader, a viewer, an on-screen digitizer, and a spatial data exporter. The GIS and expert modules are connected by the user interface for data input and output.

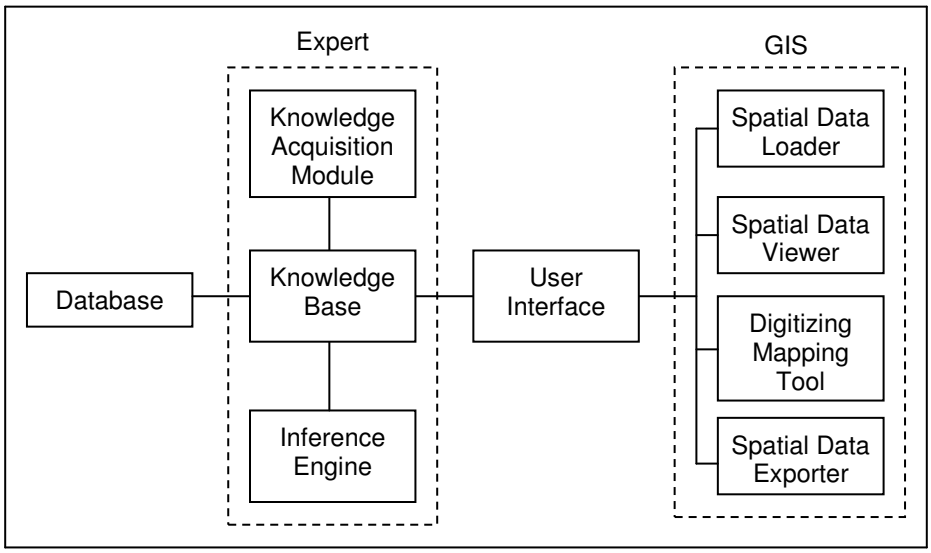


Figure 5.1 Architecture of Integrated System

In the expert module, the knowledge base contains domain specific knowledge that is used for problem solving. Knowledge is represented as rules in the form of 'If-then' constructs.

The inference engine is based on an inference rule and a search strategy, and contains algorithms (Watson, 1997). The knowledge acquisition module enables experts to store knowledge in the knowledge base or expert system to deduce new knowledge through a machine learning process.

For the inference engine in expert systems, three reasoning methods are generally used: rule-based, model-based, and case-based reasoning. A reasoning method should be selected carefully according to the application.

Model-based reasoning has some limitations (Davis and Hamscher, 1986). Current modeling technology cannot model and predict the subtle and complicated interactions in certain domains. Also, simple problems often do not have to be modeled using complex modeling technology. Therefore, model-based reasoning should be used when the structure and behavior of the device should be reasonably well known and simple enough to model, but complex enough that exhaustive simulation is impossible.

Case-based reasoning has advantages over model-based systems. Case-based reasoning can handle domains where problems are not fully understood because cases learn from experiences (Schank, 1982; Slade, 1991; Holt *et al*, 1997). Also, case-based reasoning has advantage over rule-based reasoning because it allows exceptions to rules and automatic knowledge updating mechanism from the given cases (Slade, 1991). Therefore, case-based reasoning may be the most enhanced reasoning methods for an expert system among current reasoning methods, which can be used for solving complex problems. However, in case-based reasoning systems, the knowledge is in the form of particular experiences (cases) rather than in the form of rules and it is often more difficult to gather case data.

Rule-based reasoning is more suited to solve a simple problem when it is difficult to gather cases (Ford, 1991; Althoff *et al*. 1994). The expert's knowledge can be more easily

implemented in rule-based reasoning systems than the other reasoning systems because rules can be constructed easily by a simple reasoning structure ('if-then') and a natural language. In this study, a reasoning method also will be used to solve simple problems. Therefore, the rule-based reasoning method was selected for the expert system module in this hybrid design.

However, there are some limitations in rule-based reasoning of this hybrid system. First, the maintenance of knowledge in a rule-based expert system is normally a manual process for further knowledge acquisition. Second, there are no exceptions to the rules. Rules need explicit knowledge, exact questions, and exact answers.

5.3 System Development

This hybrid system of GIS and a rule-based expert system consists largely of a main window system and a database. The main window system includes a knowledge acquisition module, a rule-based inference engine, a GIS mapping module, and user interface for data input/output as shown in Figure 5.1. A database is used for storage and retrieval of the knowledge base.

The knowledge acquisition module, inference engine, and user interface were built using Visual Basic 6.0 (Harris, 1997; Ryu, 1999; Stephens, 2000). Although AI shells are fast and convenient for developing simple expert systems, developing hybrid systems often requires complete recoding using an entirely different software package (Bramer, 1989). The GIS was built by customizing MapObjects Version 2 (ESRI, 1999a and 2001). MapObjects has many GIS facilities and can be extended and integrated easily with other systems using a conventional language such as Visual C++ and Visual Basic. The database for the knowledge base (prologue, questions and answers, and rules) is Microsoft Access 2002.

5.3.1 System Module Design

The main window system consists of an expert part, a user part, and a map part (Figure 5.2). The expert system module in the system architecture is implemented separately into expert and user parts. The GIS module is implemented directly into the map part.

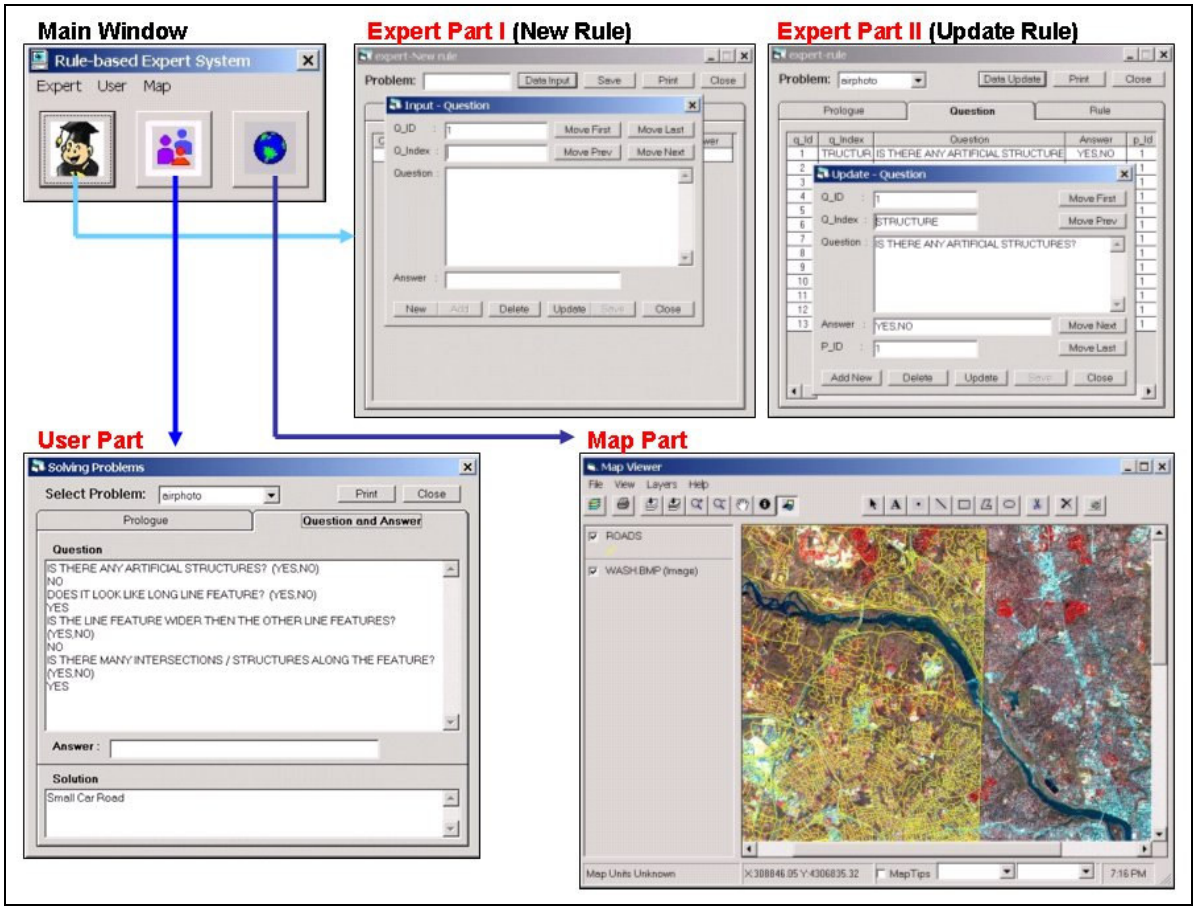


Figure 5.2 Main Windows of Hybrid System

The expert part is the knowledge acquisition module which constructs new knowledge (prologue, question, answer, and rule) or updates the existing knowledge by experts. The user part includes the user interface for the output of the knowledge base and the rule-based engine. The user part can be used to solve a specific problem from the existing knowledge in the database using an interactive question-and-answer sequence. The rule-based engine will generate

questions according to previous answers. Once the problem is solved through this interactive process, the result will be shown in the solution text box.

The map part is a GIS and deals with various vector (ESRI coverage, ESRI shape data, and computer-aided design (CAD) drawings) and raster (ERDAS Imagine files (*.img), ESRI Grid data, standard image data such as bitmap (*.bmp), gif (*.gif), jpeg (*.jpg), tiff (*.tif), and windows metafile (*.wmf)) data. The map part can be used to analyze spatial data visually, to draw a thematic map by on-screen digitizing on top of the imagery, and to generate vector data by exporting digitized data.

5.3.2 Database Design

A relational database structure is used to save the knowledge for problems. The database includes a problem table, a question table, and a rule table (Figure 5.3). The problem table contains the name and prologue for all problems. The question table contains both questions and answers for a problem. The question is symbolized using simply one word for easy processing in the rule-based inference engine. The rule table contains the rules for a problem.

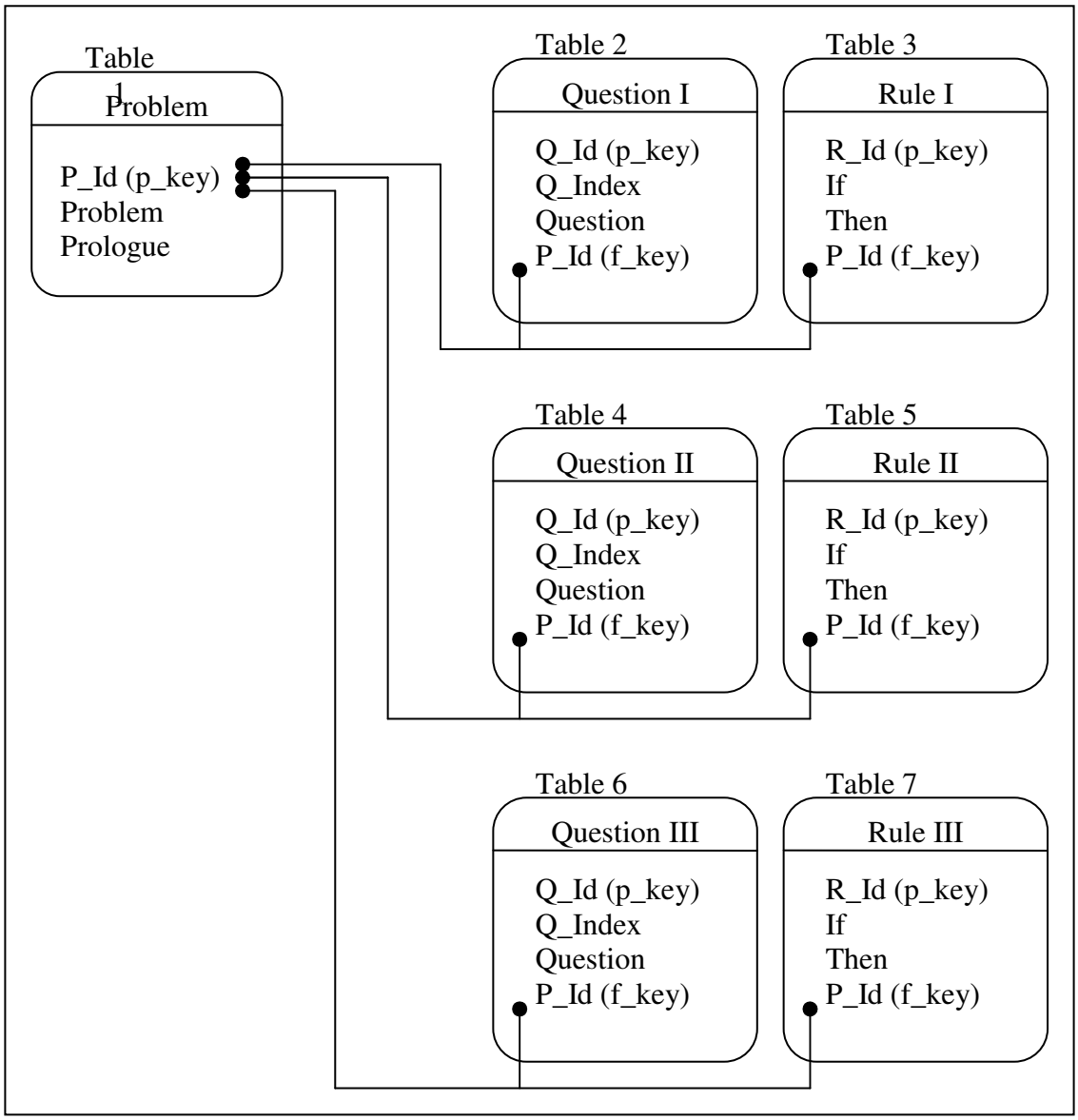


Figure 5.3 Table Structure and Relations in the Database

Each problem in the problem table is related to the question table and the rule table for that problem. If a new problem is created, the problem will be added in the problem table. Then, the question and rule tables for that problem will be created automatically in the database.

5.4 Application: Interpretation of Aerial Photographs

This hybrid system of GIS and a rule-based expert system is applied to the interpretation of aerial photographs for an urban area. Rules and symbols are constructed from the key elements for land use/land cover interpretation of a black and white aerial photograph.

5.4.1 Rule Construction

Rule construction requires three steps: decision tree construction, symbolization, and rule construction. A decision tree is constructed by the existing expert knowledge in the application domain.

In aerial photography, the seven basic elements for interpretation are tone, texture, pattern, shape, shadow, size, and situation (Avery and Berlin, 1992). Tone is the density of brightness. It is a record of light reflection from the land surface on the photo. Texture is a frequency of tone change within the photographs. Pattern is the spatial arrangement of objects. Shape is the general form of an object. Shadow reveals shape and height of an object. Size of particular object is related to that of others. Situation is the position of an object in relation to other objects in the immediate vicinity.

Each of these elements is used to construct rules in a decision tree (Figure 5.4). For example, if there are artificial structures such as buildings or houses, this area may be used for urban land use/land covers such as commercial, industrial, schools, residential, and transportation. In the urban uses, some categories are separated from others using shape and situation. The forest types (deciduous, mixed, or evergreen) can be separated by the tone. In total, fourteen land use/land cover categories are included in developing a rule: residential, commercial,

industrial, highway, road, railroad, evergreen forest, mixed forest, deciduous forest, school, golf course, park, water, and bare ground.

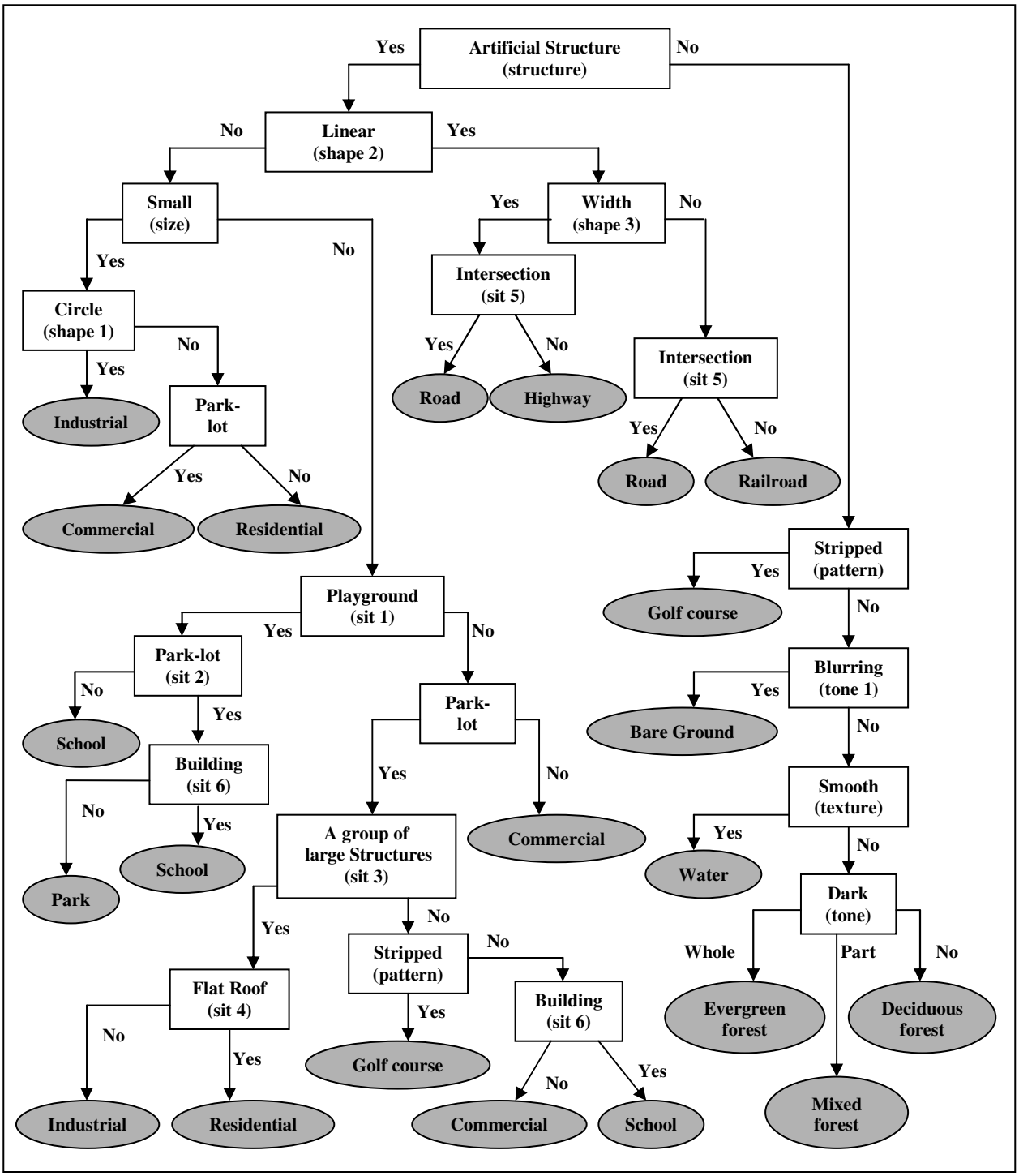


Figure 5.4 Decision Tree for Rule Construction

The most difficult categories in rule construction for the interpretation of aerial photographs are highways and railroads because of the low resolution of the image for visual interpretation. However, highways are wider than railroads or small roads. Also, railroads can be separated from small roads between residential areas because railroads have few intersections but roads have many intersections.

Symbol	Question and Answer
STRUCTURE	Question: ARE THERE ANY ARTIFICIAL STRUCTURES? Answer: YES,NO
SIZE	Question: IS THE STRUCTURE SMALL? Answer: YES,NO
SHAPE1	Question: IS THE STRUCTURE CIRCLE? Answer: YES,NO
SITUATION1	Question: IS THERE PLAYGROUND ADJACENT TO THE STRUCTURE? Answer: YES,NO
SITUATION2	Question: ARE THERE LARGE PARKING LOTS ADJACENT TO THE STRUCTURE? Answer: YES,NO
SITUATION3	Question: ARE THERE MANY LARGE STRUCTURES GROUPED? Answer: YES,NO
SITUATION4	Question: DOES THE STRUCTURE HAVE FLAT ROOF? Answer: YES,NO
SHAPE2	Question: DOES IT LOOK LIKE LONG LINEAR FEATURE? Answer: YES,NO
SHAPE3	Question: IS THE LINE FEATURE WIDER THAN THE OTHER LINE FEATURES? Answer: YES,NO
SITUATION5	Question: ARE THERE MANY INTERSECTIONS / STRUCTURES ALONG THE FEATURE? Answer: YES,NO
PATTERN	Question: IS IT STRIPED IRREGULARLY WITH DARK AND WHITE TONE? Answer: YES,NO
TEXTURE	Question: IS THE SURFACE OF THE FEATURE VERY SMOOTH? Answer: YES,NO
TONE	Question: IS THE TONE DARKER THAN OTHER SAME FEATURES? Answer: WHOLEY,PARTLY,NO
SITUATION6	Question: IS THE PARKING LOT SMALLER THAN THE BUILDING? Answer: YES,NO
TONE1	Question: IS IT BRIGHTER THAN OTHERS (EVEN BLURRING)? Answer: YES,NO

Figure 5.5 Symbolization of Rules Decision Tree

Each node in the decision tree can be symbolized as a question and each link in the decision tree can be symbolized as an answer for the knowledge base (Figure 5.5). For example, the first node in the decision tree is ‘Artificial Structure’ and its two links are ‘Yes or No’ in Figure 5.4. This node and its links are symbolized as ‘STRUCTURE’ in Figure 5.5. This symbol will be used to construct rules and interpreted as a question with possible answers when the inference engine is running.

Each rule for each solution case will be generated according to the decision tree using symbolized questions and answers (Figure 5.6). Constructed rules are saved in a database as a knowledge base for the application.

If	Then
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ YES AND SHAPE1 EQ NO AND SITUATION2 EQ NO	Residential
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ YES AND SHAPE1 EQ NO AND SITUATION2 EQ YES	Commercial
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ YES AND SHAPE1 EQ YES	Industrial
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ NO AND SITUATION1 EQ YES AND SITUATION2 EQ NO	School
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ NO AND SITUATION1 EQ YES AND SITUATION2 EQ YES AND SITUATION6 EQ NO	Service(park)
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ NO AND SITUATION1 EQ NO AND SITUATION2 EQ NO	Commercial
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ NO AND SITUATION1 EQ NO AND SITUATION2 EQ YES AND SITUATION3 EQ NO AND PATTERN EQ NO AND SITUATION6 EQ NO	Commercial
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ NO AND SITUATION1 EQ NO AND SITUATION2 EQ YES AND SITUATION3 EQ NO AND PATTERN EQ YES	Golf Course
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ NO AND SITUATION1 EQ NO AND SITUATION2 EQ YES AND SITUATION3 EQ YES AND SITUATION4 EQ NO	Residential (Apartment)
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ NO AND SITUATION1 EQ NO AND SITUATION2 EQ YES AND SITUATION3 EQ YES AND SITUATION4 EQ YES	Industrial
STRUCTURE EQ YES AND SHAPE2 EQ YES AND SHAPE3 EQ YES AND SITUATION5 EQ NO	Highway
STRUCTURE EQ YES AND SHAPE2 EQ YES AND SHAPE3 EQ NO AND SITUATION5 EQ NO	Railroad
STRUCTURE EQ YES AND SHAPE2 EQ YES AND SHAPE3 EQ NO AND SITUATION5 EQ YES	Road
STRUCTURE EQ NO AND PATTERN EQ YES	Golf Course
STRUCTURE EQ NO AND PATTERN EQ NO AND TONE1 EQ NO AND TEXTURE EQ YES	Water
STRUCTURE EQ NO AND PATTERN EQ NO AND TONE1 EQ NO AND TEXTURE EQ NO AND TONE EQ WHOLEY	Evergreen Forest
STRUCTURE EQ NO AND PATTERN EQ NO AND TONE1 EQ NO AND TEXTURE EQ NO AND TONE EQ PARTLY	Mixed Forest
STRUCTURE EQ NO AND PATTERN EQ NO AND TONE1 EQ NO AND TEXTURE EQ NO AND TONE EQ NO	Deciduous Forest
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ NO AND SITUATION1 EQ YES AND SITUATION2 EQ YES AND SITUATION6 EQ YES	School
STRUCTURE EQ YES AND SHAPE2 EQ YES AND SHAPE3 EQ YES AND SITUATION5 EQ YES	Road
STRUCTURE EQ YES AND SHAPE2 EQ NO AND SIZE EQ NO AND SITUATION1 EQ NO AND SITUATION2 EQ YES AND SITUATION3 EQ NO AND PATTERN EQ NO AND SITUATION6 EQ YES	School (University)
STRUCTURE EQ NO AND PATTERN EQ NO AND TONE1 EQ YES	Bare Ground

Figure 5.6 Rules using If-then Structure

5.4.2 Processing Steps

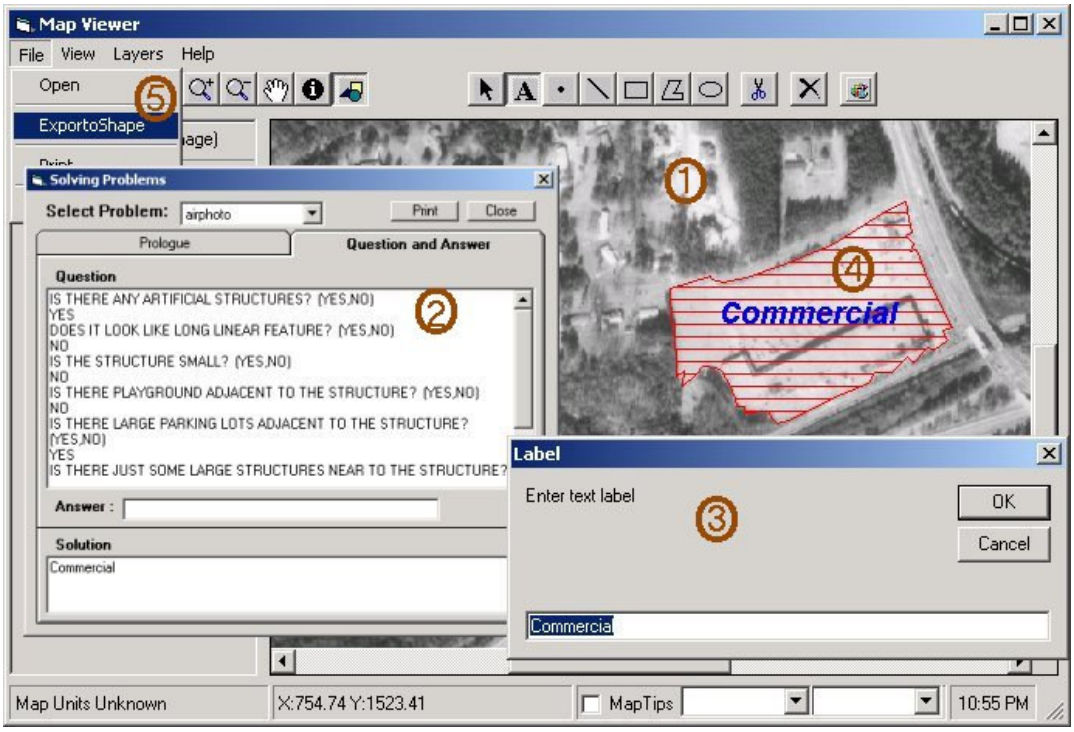


Figure 5.7 Processing Steps for Land Use using Expert System

To map spatial data, the system requires five steps: feature selection, solving the problem, solution & suggestion, labeling, and exporting (Figure 5.7). First, a user selects and digitizes a feature during the interpretation of an aerial photograph. Second, the user chooses the 'airphoto' interpretation problem in the user part, and then the inference engine will generate the first question about the feature selected. The user should answer this question. The inference engine, then, will generate the next question. Third, when the condition for a certain feature type is satisfied through this interactive question-and answer sequence, the inference engine will suggest a feature type. Fourth, the suggested feature name can be labeled on the digitized vector data. Last, the thematic data digitized on the image can be exported as vector data (shape file format). During export, the labeled feature name will be saved automatically as an attribute of the vector data.

5.4.3 Urban Mapping and Result

The study site is Athens-Clarke County, GA. The USGS digital orthophoto quarter quadrangle (DOQQ) data in 1993 is used for the interpretation and mapping. In the study area, four subsets (2km x 2km) of the DOQQ have been generated and used for land use/land cover mapping (Figure 5.8). The interpretation of black and white DOQQ images was completed only by the inference rules of the expert module using the interactive procedure, and the land use/land cover maps have been generated by the GIS module in this hybrid system (Figure 5.9).

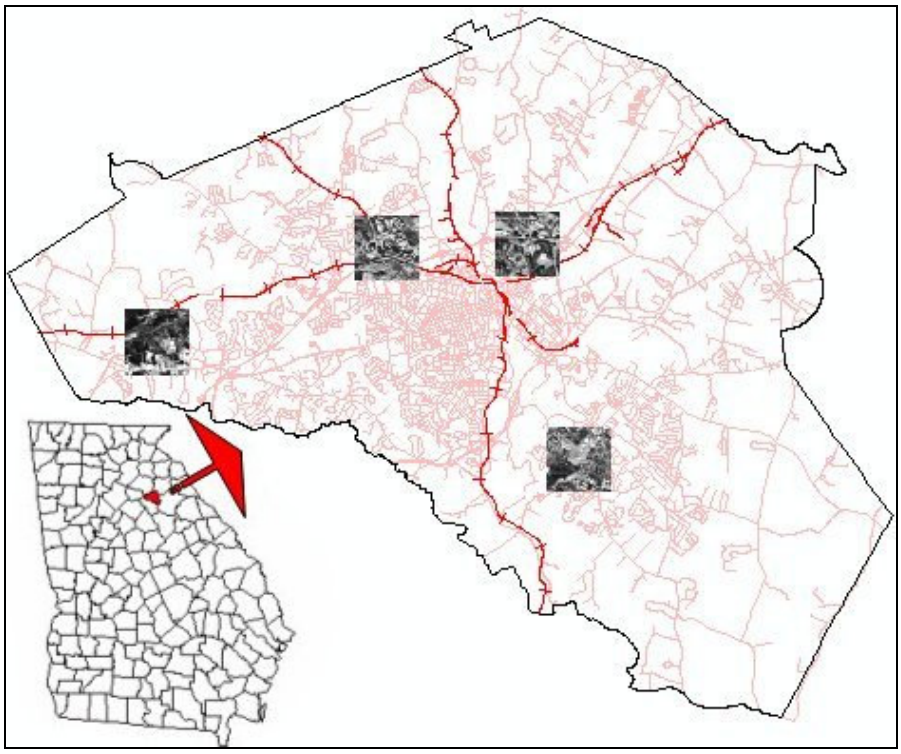


Figure 5.8 Study Area for Expert Mapping

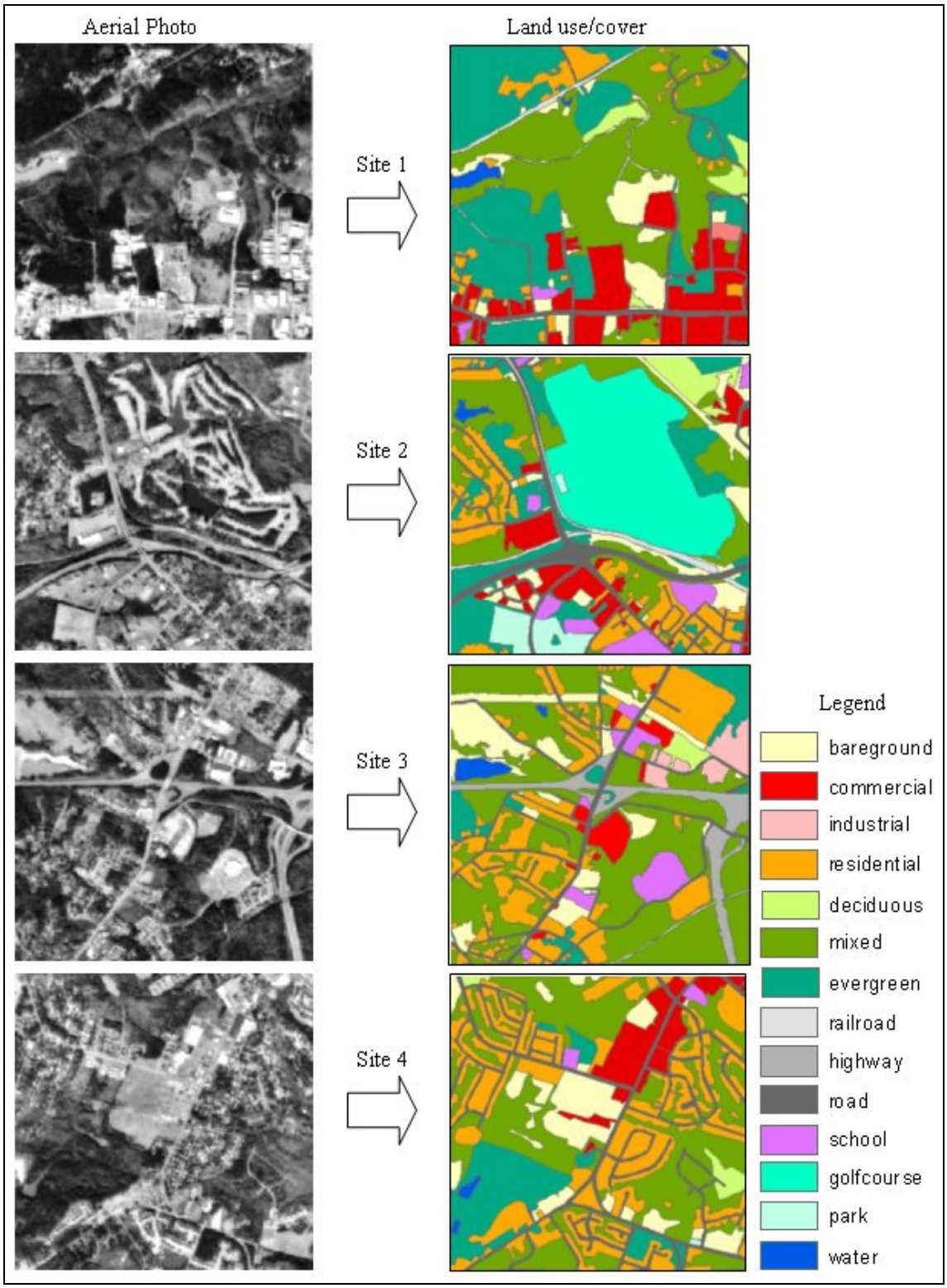


Figure 5.9 Land Use/Land Cover Mapping from Aerial Photographs

Accuracy assessment of the resulting land use/land cover maps was conducted to evaluate the usability of the rule. Forty sample sites were selected for each land use/land cover category using the stratified random sampling method (Table 5.1). The original DOQQ image and the land use data that were generated by the planning department of Athens-Clarke County were used for the reference data.

Table 5.1 Accuracy Assessment of Classified Land Use/Land Cover Data

		Reference Data														
		A	B	C	D	E	F	G	H	I	J	K	L	M	N	Sum
Classified Land Use / Cover	A	36				1					3					40
	B		36	3		1										40
	C	1		38							1					40
	D				39						1					40
	E	2				37				1						40
	F	1					37	1			1					40
	G	1			3			36								40
	H	1							38		1					40
	I									39	1					40
	J									2	38					40
	K	1									1	38				40
	L										1		38	1		40
	M	4									2			34		40
	N	1													39	40
	Sum	48	36	41	42	39	37	37	38	42	50	38	38	35	39	560

A: Bare Ground, B: Commercial, C: Industrial, D: Residential, E: School,
F: Highway, G: Road, H: Railroad, I: Evergreen Forest, J: Mixed Forest,
K: Deciduous Forest, L: Golf Course, M: Park, N: Water

Overall Accuracy = 93.39 % and Kappa Index value = 0.9288

The accuracy is 93.39 % in overall accuracy with a kappa index of 0.9288. Almost all errors occur in the boundary area between different land use/land covers. Land use/land cover mapping is executed by a novice user in GIS so that the errors will be reduced by advanced digitizing skill. Therefore, the rules developed in this study may be valid and used for the other applications.

5.5 Summary

In this chapter, a hybrid system of GIS and a rule-based expert system was discussed. This system consists of three main parts with a database: expert, user, and map. The expert and user parts are the rule-based expert system, and the map part is a GIS.

The expert part is used to construct new knowledge (prologue, questions and answers, and rules) or update existing knowledge. The knowledge (prologue, questions and answers, and rules) in this system can be constructed easily by an expert using a natural language (English). The user part is employed to solve a specific problem for which the rules are constructed in the system. Novice users who are not good at programming or a particular problem domain can derive a solution to a specific problem easily from the constructed rules through the interactive question and answer sequence. The questions will be generated by the rule-based engine using existing rules.

The map part is used to generate spatial data. Various vector data formats, such as ESRI coverage, shape files, and CAD drawings and raster data formats, such as ESRI Grid, Geotiff, and others can also be loaded in the map part. The map part can also be used for visual interpretation of image data.

To test the applicability, the developed system has been used for the interpretation of aerial photography by a neophyte user. If users cannot identify a certain feature during the interpretation of aerial photographs, they can consult the rule-based expert module in the user part of this system through the question and answer sequence. The questions will be generated automatically by each of the user's answers.

The rules were constructed for the aerial photograph of the study area using basic interpretation elements. Using this hybrid system, fourteen land use/land cover categories are

interpreted and mapped easily from black-and-white DOQQ images. The mapped land use/land cover data were exported to a shape file and evaluated for accuracy. The accuracy of mapped data is over 93 % in both overall accuracy and kappa index so that the rules constructed in this study are valid for the interpretation of black-and-white aerial photographs. Therefore, users can consult this hybrid system of GIS and a rule-based expert system during mapping or digitizing in order to produce spatial data.

CHAPTER 6

SYSTEM INTEGRATION FOR FEATURE-ORIENTED GIS (FOGIS)

6.1 System Architecture for FOGIS

In this chapter, the feature data viewer in Chapter 4 and the input procedure in Chapter 5 have been integrated into a complete system. The procedures of the integrated system include data generation, creation of a feature database, and visualization of the features in FOGIS. The system architecture of the feature data viewer has been extended for FOGIS (Figure 6.1).

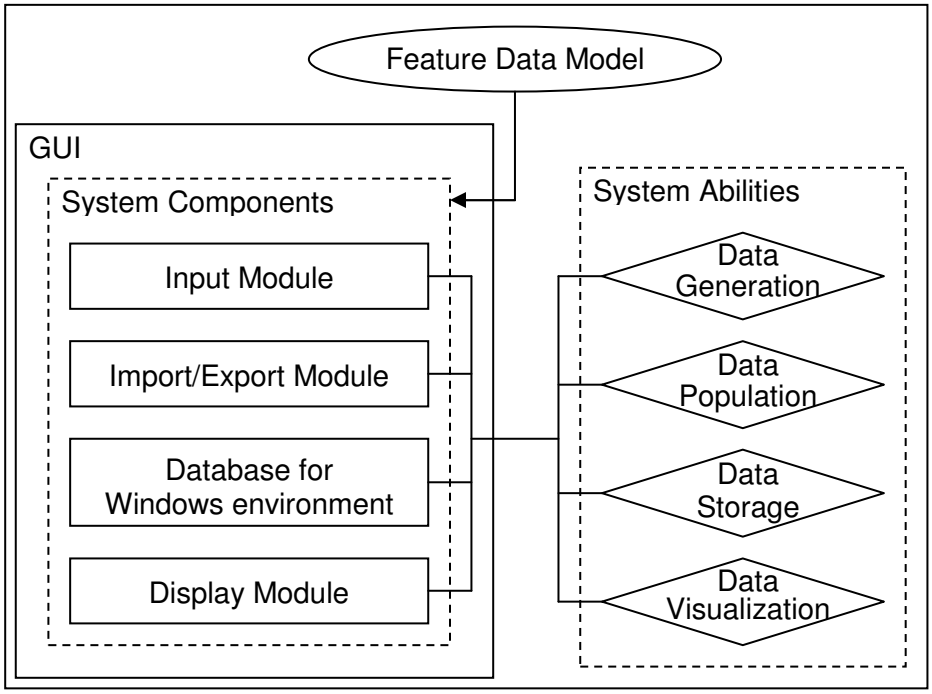


Figure 6.1 System Architecture for FOGIS

6.2 System Integration for FOGIS

For the system integration, a new main interface has been designed, which includes the input module, the import/export module, the tree viewer, and the display module. For data

storage, the feature database designed and implemented in Chapter 4 has been directly used. Figure 6.2 shows the main user interface of each module in the integrated FOGIS.

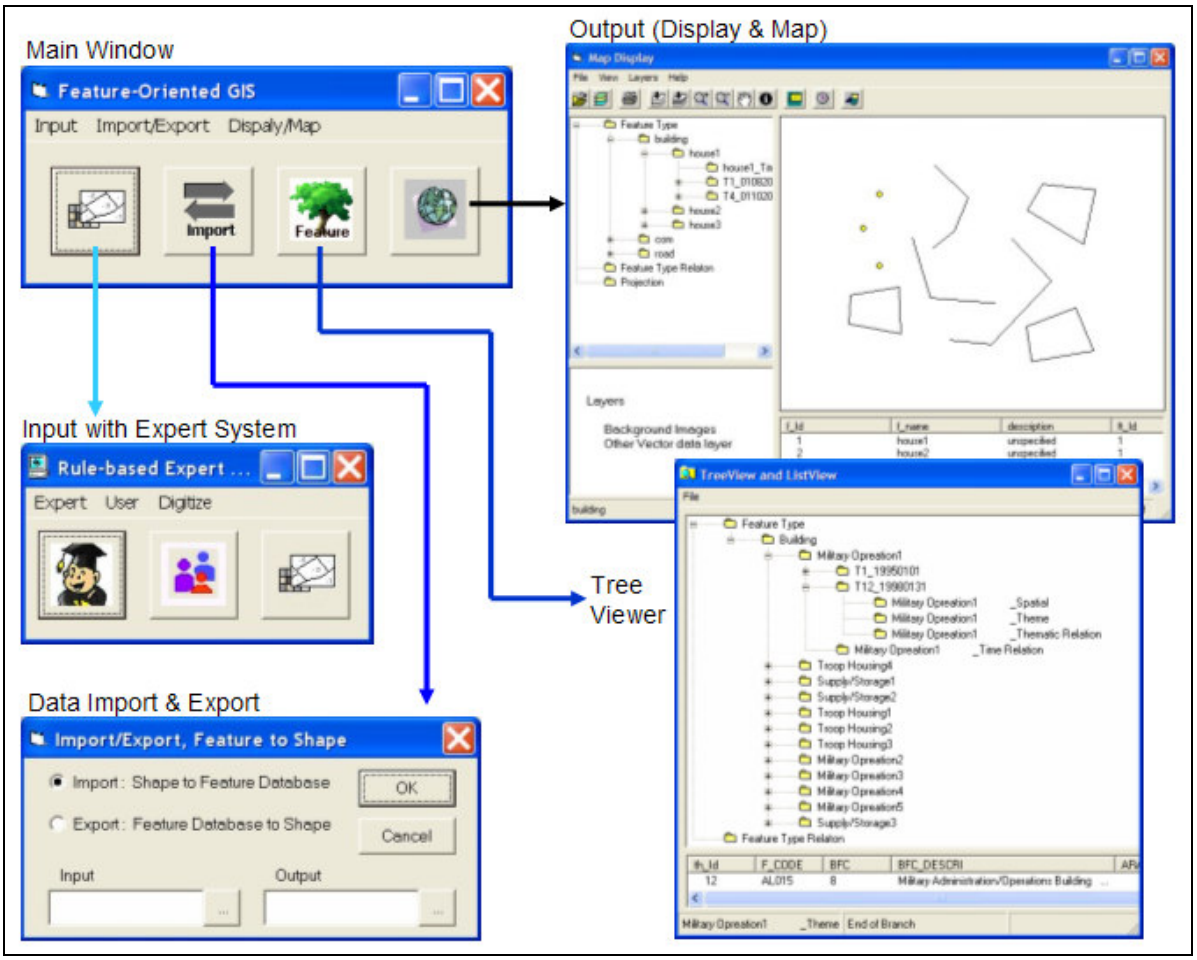


Figure 6.2 User Interfaces of FOGIS

6.2.1 Feature Population Method in the Input Module

The input procedure in Chapter 5 is primarily focused on spatial data generation as a common layer-based data format such as a shape file with a new data generation method by a rule-based expert system. However, the input procedure in the integrated feature-oriented system should have a method of data population directly into the feature database. Therefore, the input module in Chapter 5 has been modified for populating newly generated data into a feature

database. Also, the input module includes the projection module for setting or entering the proper projection information of the generated data into the feature database.

Figure 6.3 shows the process of populating features from digitized data into the feature database through the projection module. First, the features are digitized from the proper image source and labeled with appropriate feature type and feature name. Second, the projector module is used for setting the proper projection information for the input data. Third, the time information of the data is set with the 'save' user interface. Finally, digitized data and the projection information are populated into the feature database. The features do not have to be one type of geometry (point, line, or polygon). Also, various feature types such as buildings, roads, and so on can be digitized and populated into the feature database.

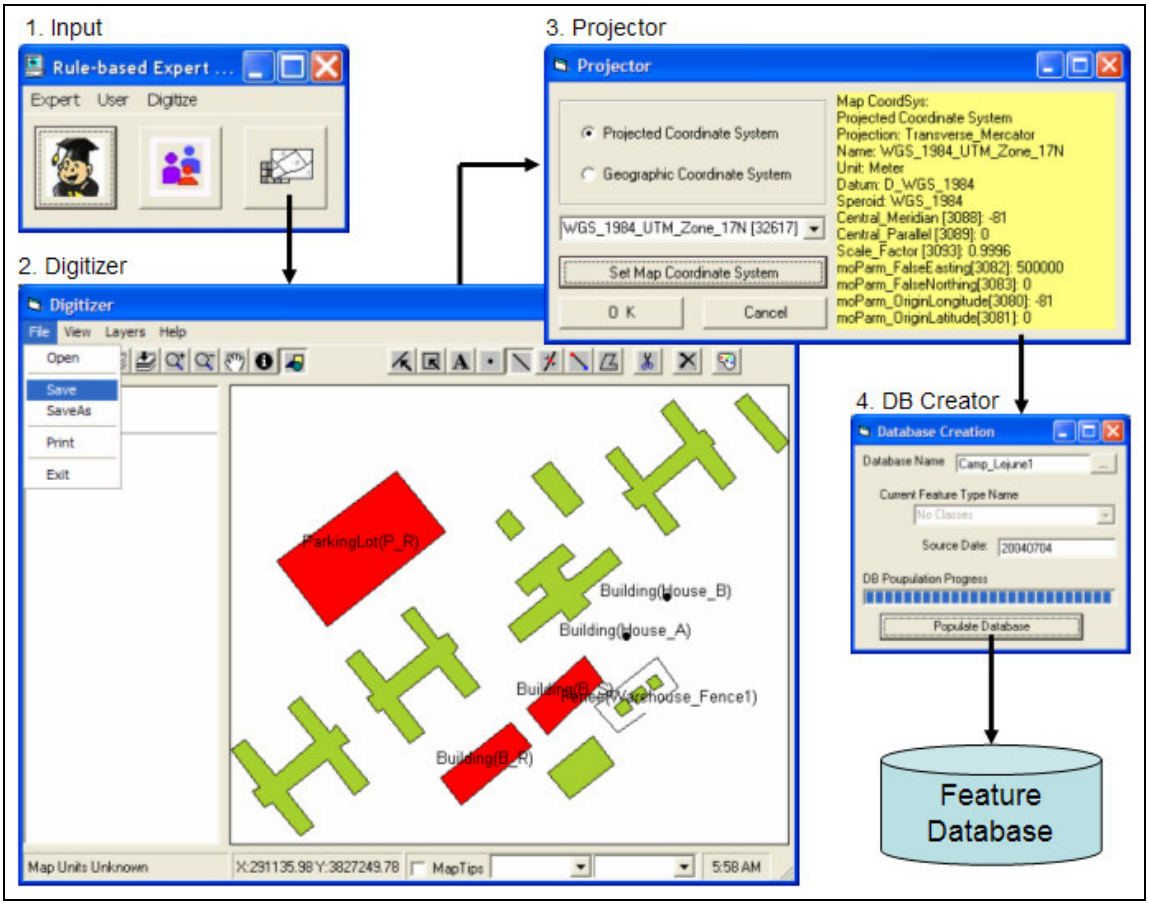


Figure 6.3 Feature Generation Procedures in the Input Module

6.2.2 Data Interoperability with Import/Export Module

With the Import/Export Module, shape file format spatial data can be imported into a feature database and exported from the database. Figure 6.4 shows both the importing and exporting processes. The import process from the selected shape file to the selected feature database requires the main import/export interface and the schema constructor for a feature database. The export process from the selected feature database to the newly created shape file requires the main import/export interface and the attribute selector.

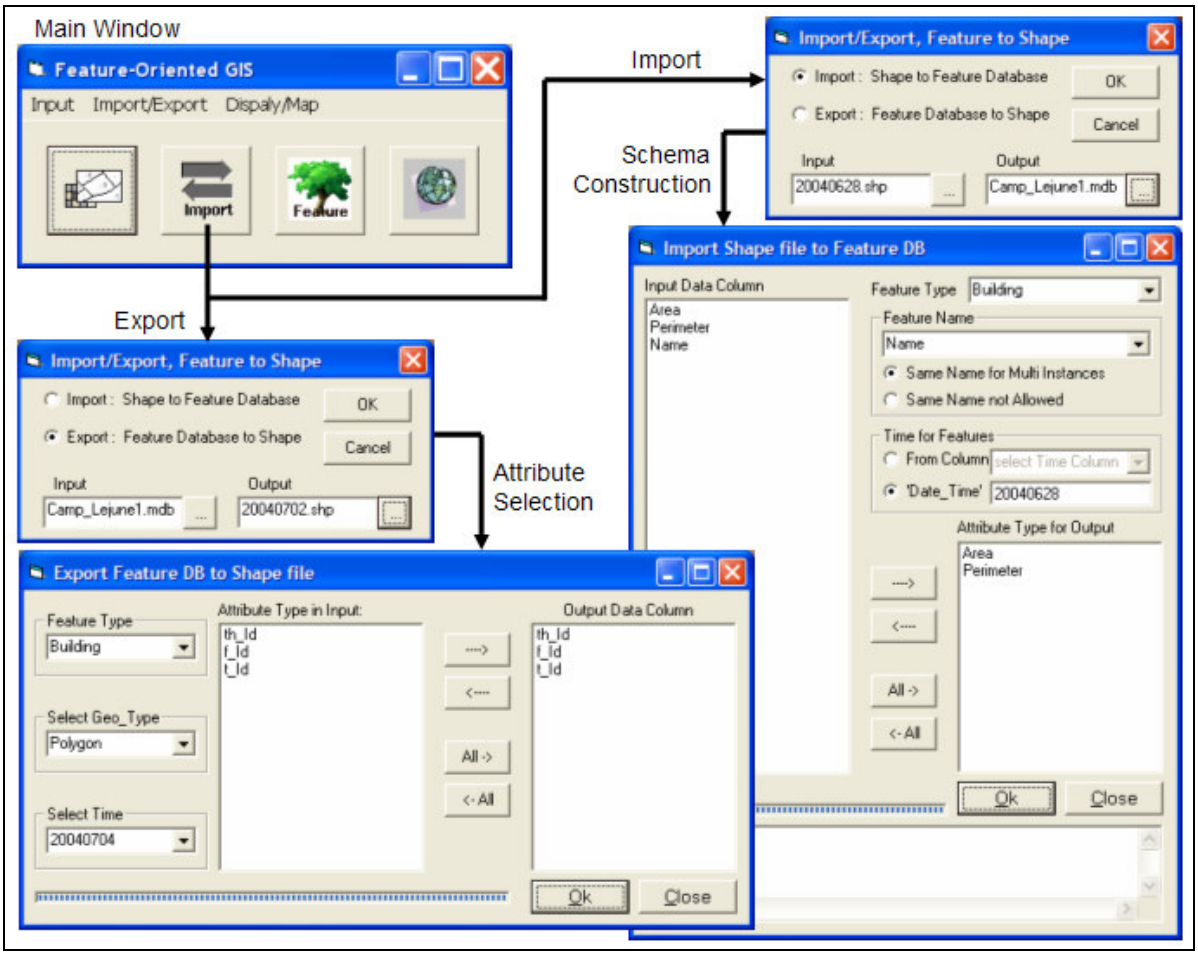


Figure 6.4 Import/Export Procedure in FOGIS

The import option in the import/export is selected, and then the shape file format will be set automatically for input and the Microsoft access database format as a feature database will be

set for output. Once the file has been selected, the schema constructor will be used for populating features into a feature database from the spatial and attribute data in the input shape file. For schema construction, the feature type should be defined for the feature type class in feature data model, which comes usually from the shape file name.

Also, the feature name and the time information should be set for both the feature class and the time class. Finally, the attribute columns in the shape file should be chosen for the thematic attribute type class and the thematic attribute class. If the input file has no projection information, the projector (Figure 6.5) will be used for populating projection instances in the spatial reference class. All these classes are populated, and then the feature database will be populated from the populated classes on the feature data model.

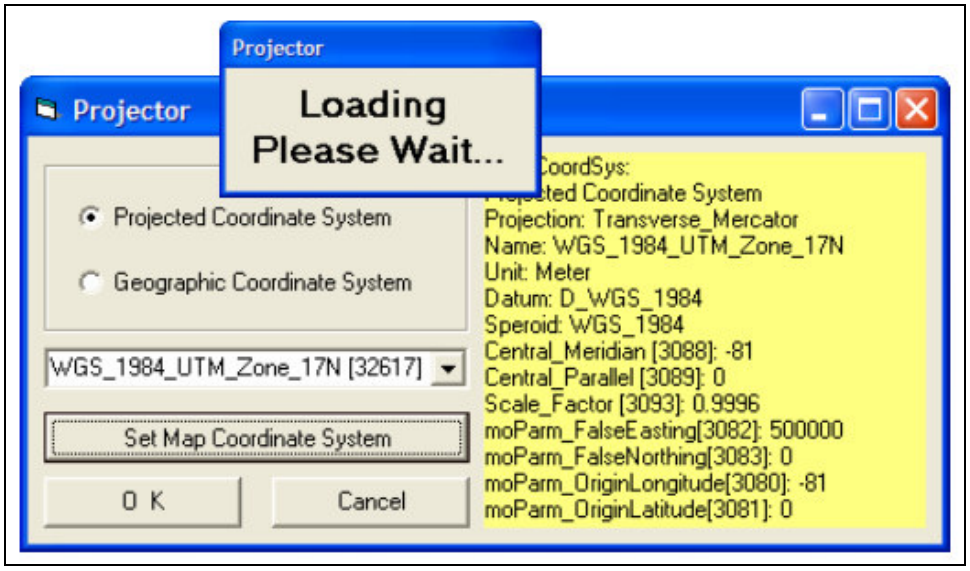


Figure 6.5 User Interface for Projection Selection

When the export option in the import/export is selected, the Microsoft access database format as a feature database will be set automatically for input and the shape file format will be set for output. Once the file has been selected, the attribute selector will be used to select features and related attributes from the feature database for the spatial and attribute data in the output

shape file. For attribute selection, the feature type should be chosen for selecting feature instances. The geometry type of the features should be selected because the shape file only supports one geometry type for one layer.

The time attribute also has to be selected because the shape file usually contains data for one time spot. Once these options are selected, the columns for the output shape file should be selected from the feature attributes. The attribute types in the feature database will be directly converted into the column name in the output shape file. The feature names of the selected feature type will be directly converted as name column in the output file. Also, the projection information in the spatial reference class of the feature database will be converted into the projection file of the shape file.

6.2.3 Feature Database Structure

The imported features will be populated in the classes in the feature data model and then entered into a feature database. Once the building features are populated, the resultant feature database will have tables that cover classes in the feature database design (see Figure 4.6). The tables for building features are shown in Figure 6.6.

All the name of tables related to a certain feature use the feature type name such as building_spatial, building_theme, building_time, and so on. The attribute types are populated as the column name in the corresponding table. The table structures are very simple so that some of the thematic attributes can be directly modified into the theme table of the corresponding feature. If a thematic attribute needs to be added to a table, it can also be added directly into the theme table by checking the feature instance id of the attribute in the feature type table and the column name as attribute type.

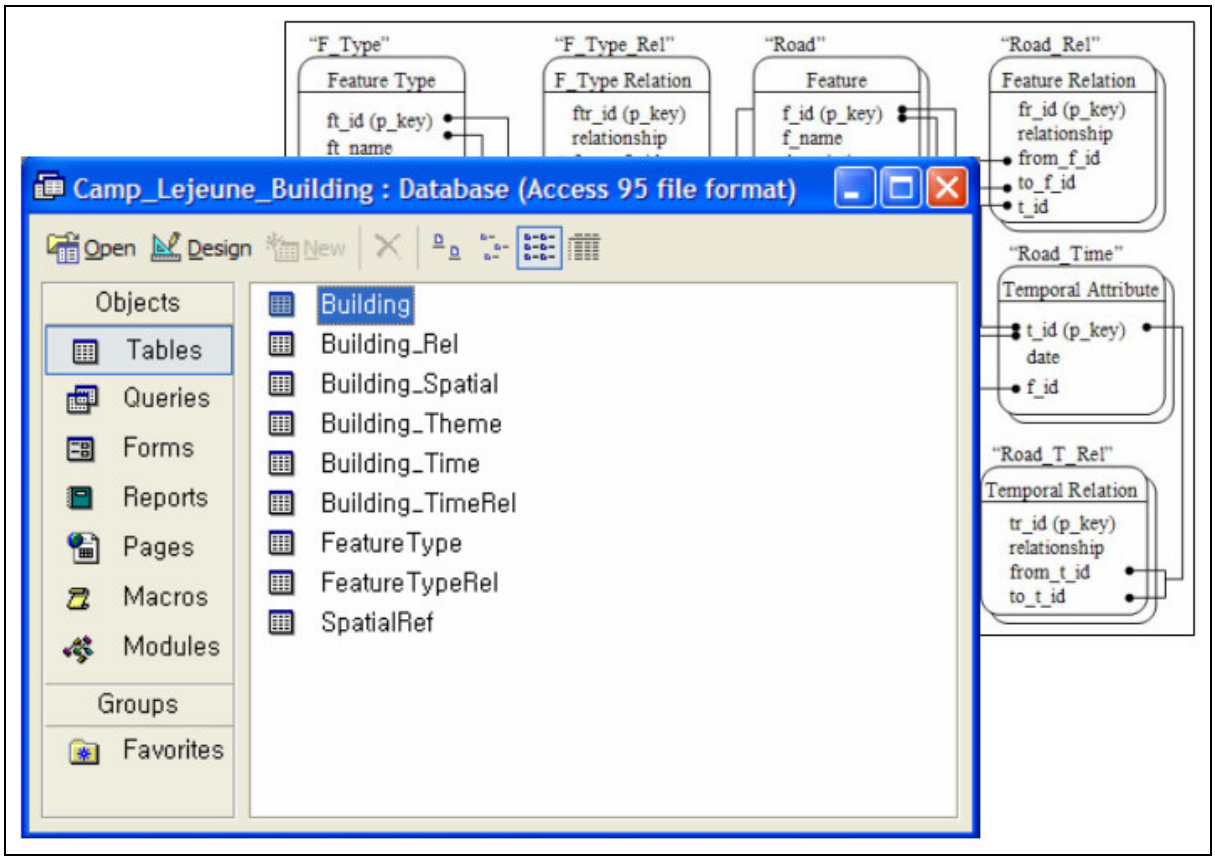


Figure 6.6 Table Structure in Feature Database from Feature Data Model

With the simple table structure, the developed system does not require an attribute editing module because attribute editing can be accomplished by accessing a feature database directly. Unfortunately, the current system has no automatic method for populating relationships such as a feature type, feature, or temporal relationships. Therefore, all these relationships have to be populated by editing a feature database.

6.2.4 Feature View with Tree Viewer and Display Module

The tree viewer (Figure 6.7) has been included as one of the main modules in the system for a clear view of the tree structure of objects (all data related with features) in the feature database. The tree viewer includes the tree view section on the top and the attribute view section

on the bottom. In the tree view section, all classes in a feature database are displayed as nodes: feature type, feature instances, all dimensions of feature instances and relationships, and the spatial reference information. These classes have been hierarchically organized in the tree view section for an effective visualization of data structure in the feature database. If any node in the tree view section is selected, the attribute under the node will be shown in the attribute section. In the attribute view section, the column name will be populated directly from the instances of the attribute type class.

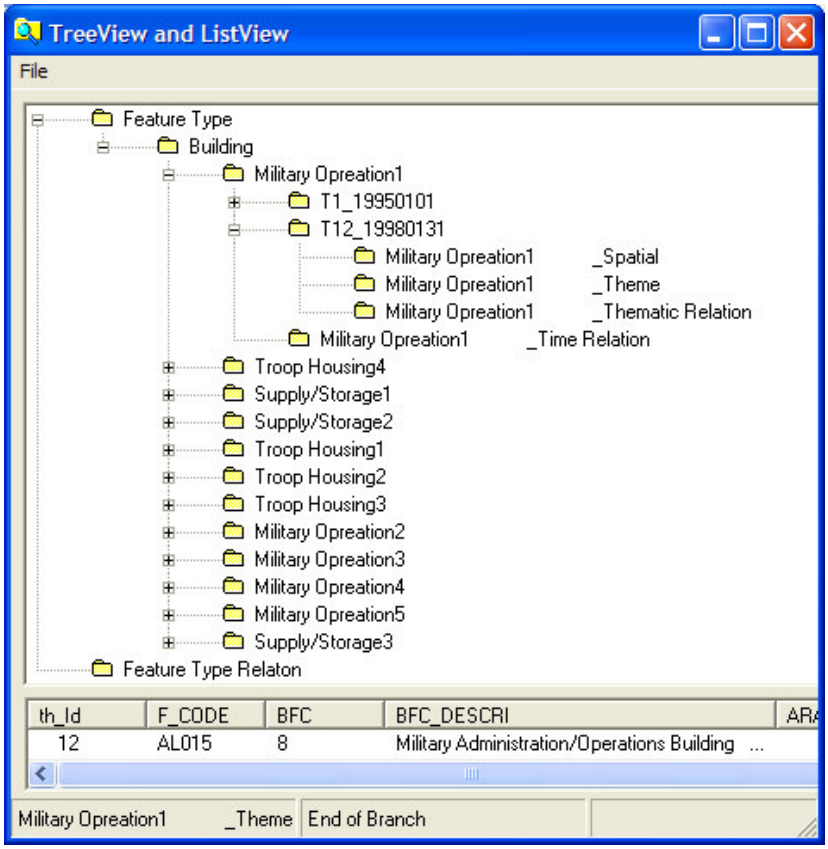


Figure 6.7 Tree Viewer for Feature Data

The data in the feature database under the feature data model has been well-organized hierarchically from the feature types to the detailed attributes of each feature under the feature type. This would be the major advantage of the use of object-orientation for representing features

with the feature data model. Also, the hierarchical view of feature information can make it easy to query any attribute of a feature by its name so that the feature data model has advantage over the layer-based model. In the layer-based model, the feature types such as roads, buildings, and so on are usually used for the name of the layers and only the layer name is shown in the user interface of the commercial software. The detailed feature names could not be directly shown without any query over the database linked to the geometry.

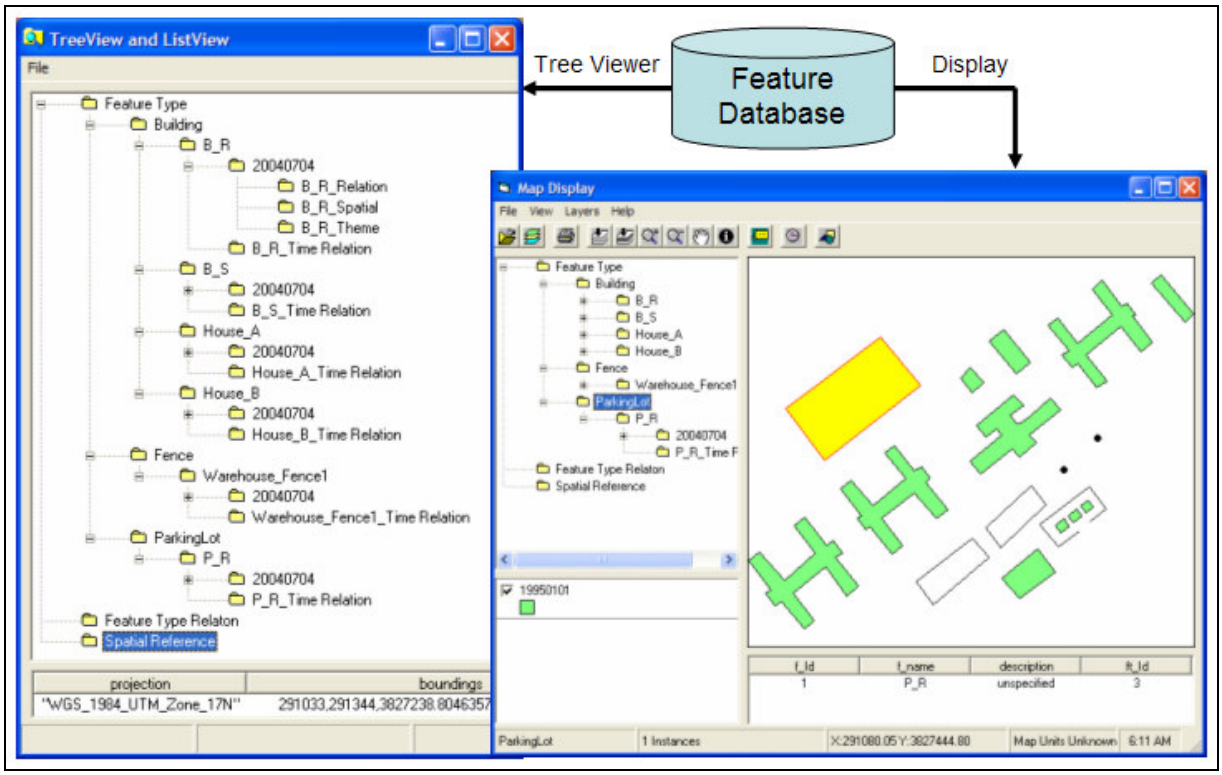


Figure 6.8 Tree Viewer and Display Module for a Feature Database

All information of a feature database can be displayed in the display module with necessary background data such as remotely sensed images and vector data. In Figure 6.8, the feature information populated in Figure 6.3 has been shown with both the tree viewer and the display modules. In the feature database, the three different feature types have been populated: Building, Fence, and ParkingLot. Under the ‘Building’ feature type, two different geometry

types of features have been populated: point and polygon. ‘Line’ type of feature has been populated under the ‘Fence’ feature type. Also, the clock icon on the menu bar in the display module can be used for time based query for features. Therefore, one feature database can contain multiple feature types and multiple time situations, and has no limitation on the geometry types of features.

6.2.5 Feature Query using Feature Relationship as Analysis Module

As one of the spatial analysis methods, the feature query tool has been added with the time query tool in the display module (Figure 6.9). For feature query, the feature type has to be chosen and then criteria selected from a feature based on a specific time. Finally the relationships of selected feature need to be selected.

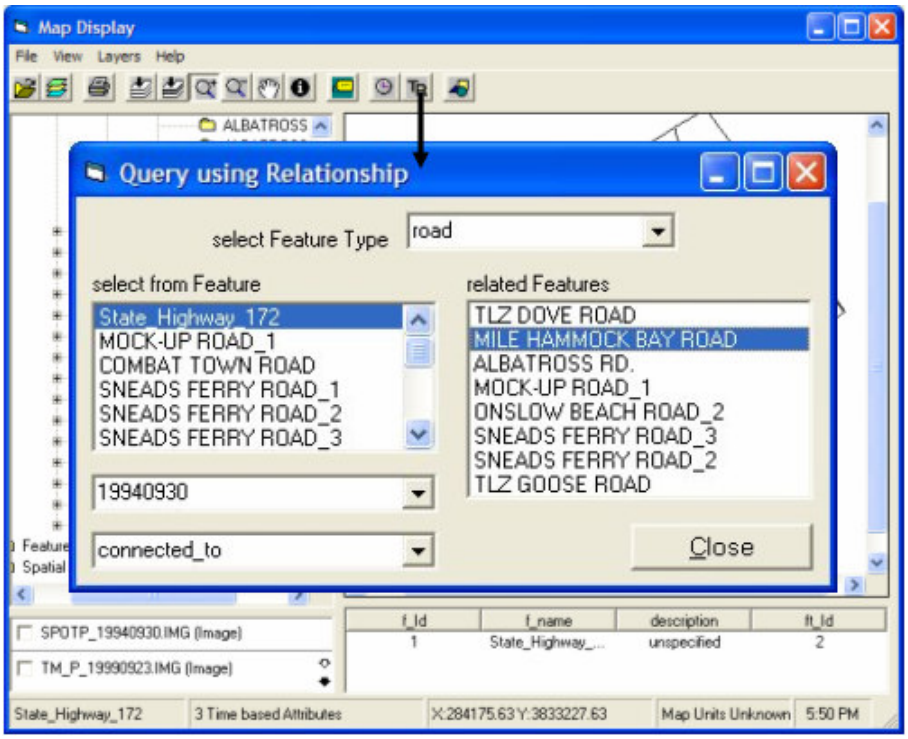


Figure 6.9 Feature Query using Feature Relationship

Once all options are selected, the resulting features will be displayed on the canvas in the display module. The resulting features will be displayed using different colors so that they can be differentiated from the input feature of the query. Each feature of the result query also can be differentiated from the others by selecting item in the related feature lists of the feature query tool. This query tool directly uses the feature relationship (thematic relationship) for spatial analysis.

6.3 Application I: Feature Management using Time

Coastlines are one of the most changeable natural features. To test the integrated FOGIS, coastlines have been chosen for visualization through time. The study area is the same area, Camp Lejeune Marine Corps Base in North Carolina that was used in the building change application in Chapter 4, but is focused on coastlines (Figure 6.10).

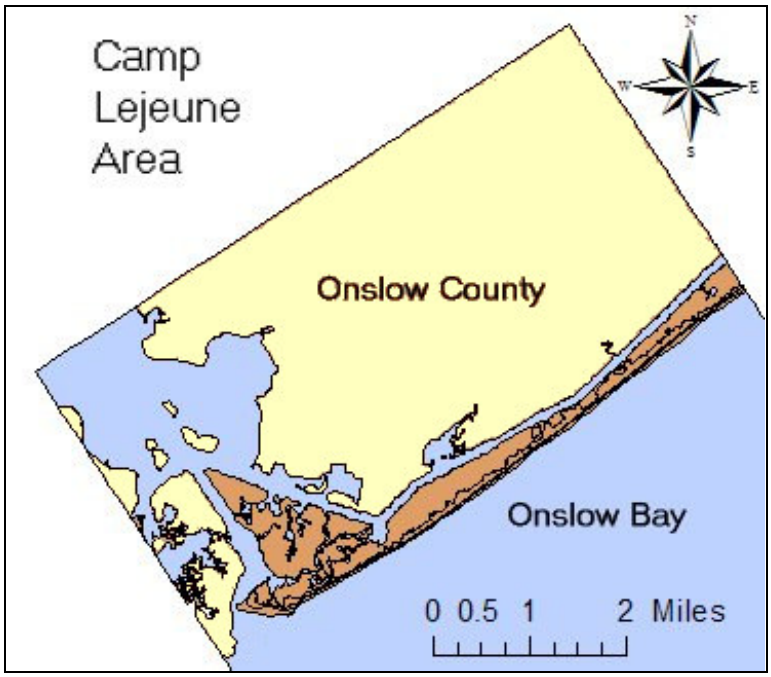


Figure 6.10 Camp Lejeune Study Area

For extracting coastlines in the study area, three remotely sensed images have been acquired: a SPOT Panchromatic image on Sept. 30, 1994, Landsat ETM plus Panchromatic image on Sept. 23, 1999, and Ikonos pan-sharpened image on May 4, 2000. Each of these data sets has different ground resolution: 10m in SPOT, 15m in Landsat ETM plus, and 1m Ikonos. Even though three source images have different ground resolution, the difference of their ground resolution will not have much effect in detecting the changes of coastlines because the changes are over 100m (Figure 6.11).

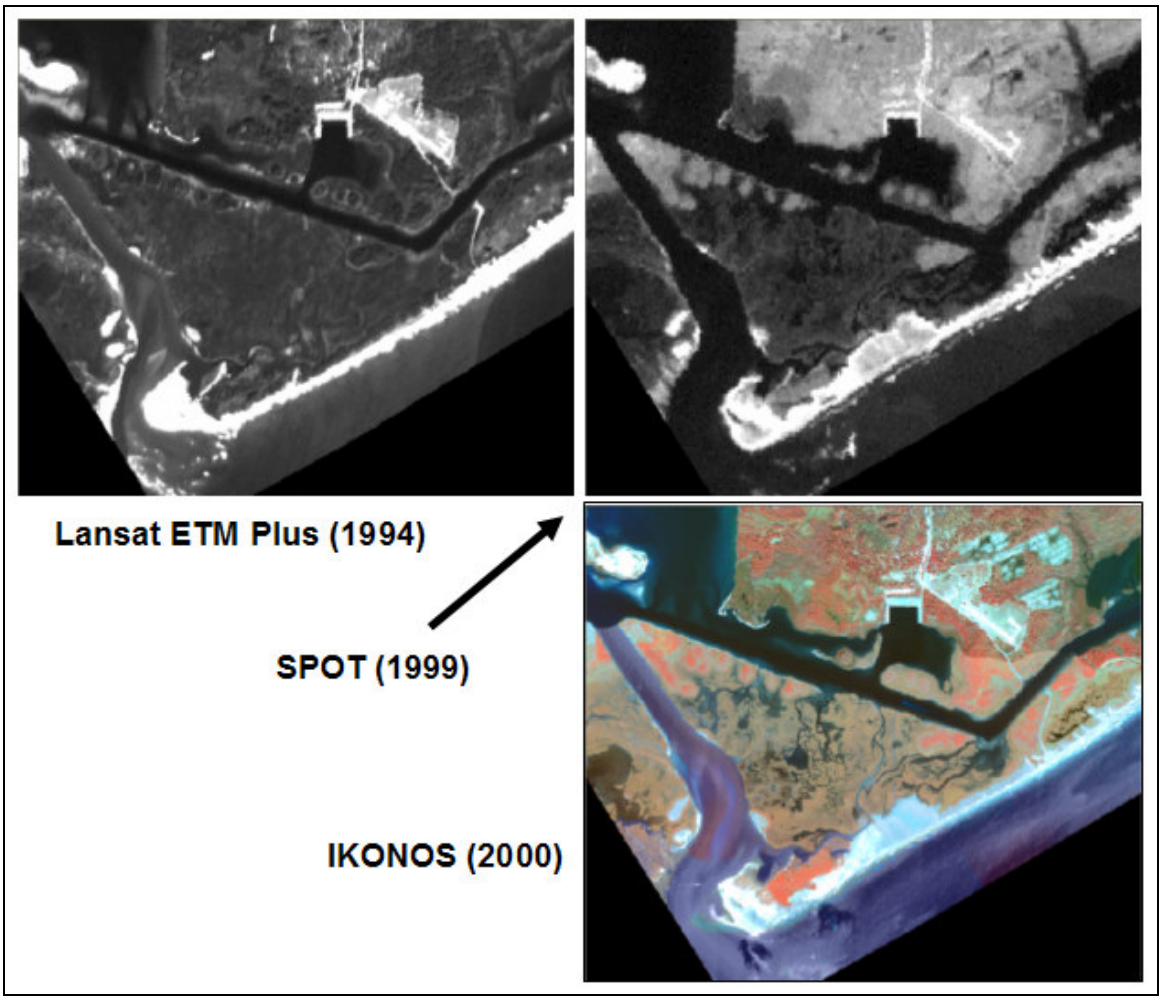


Figure 6.11 Coastline Changes in Different Image Source in Different Time

6.3.1 Processing Steps

From three different remotely sensed images, the coastlines have been extracted using the input procedure. First, the images have been loaded on to the canvas in the input module. Second, the coastlines are digitized on screen in each image. Third, once digitizing is finished, the feature has been named with a proper class name through the labeling user interface. Fourth, after naming the feature, it has been populated into a feature database. During population, the projector has appeared for choosing the projection information of the features digitized. Finally, if the projection was decided, then all the information with digitized features would be populated into a feature database with time information in the database creation interface (Figure 6.12).

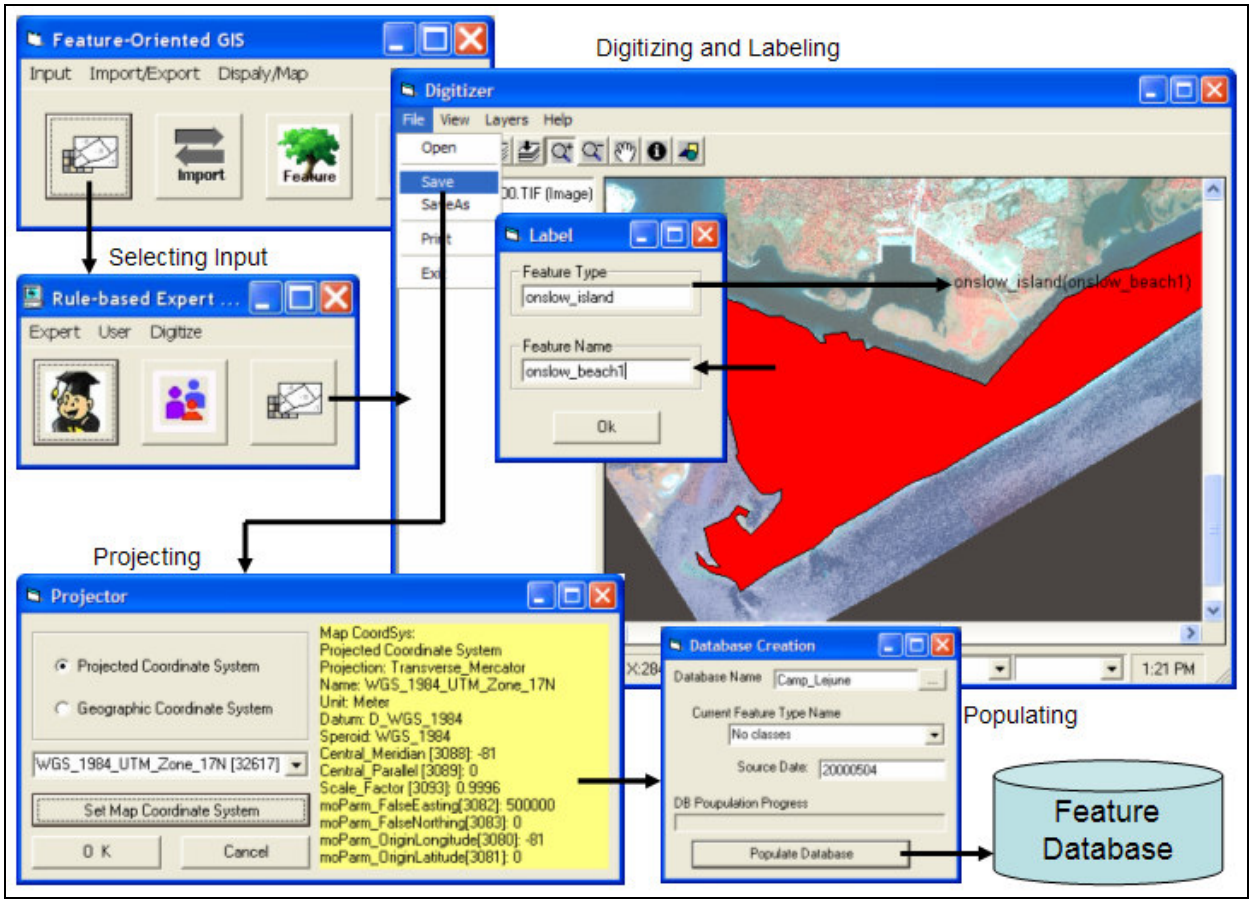


Figure 6.12 Input Procedure for Populating Coastline

Through these procedures, three coastlines from three different image sources have been extracted and populated as features into the feature database. If necessary, the digitized coastlines can be directly exported into shape file format data for other uses in the input user interface. Also, the coastlines in the feature database can be exported into shape file format using import/export module. If users are more comfortable using different digitizing methods, then they can digitize a data into a shape file format and import it into a feature database using import/export module. After populating coastlines, the feature database has been edited directly for populating the temporal relationship.

6.3.2 Mapping Result and Query Coastline based on Time

The mainland area has been imported into the feature database for the reference of the coast area. The resulting mapping is shown in Figure 6.13. The same 'onslow_beach1' feature has changed through time from 1994 to 2000.

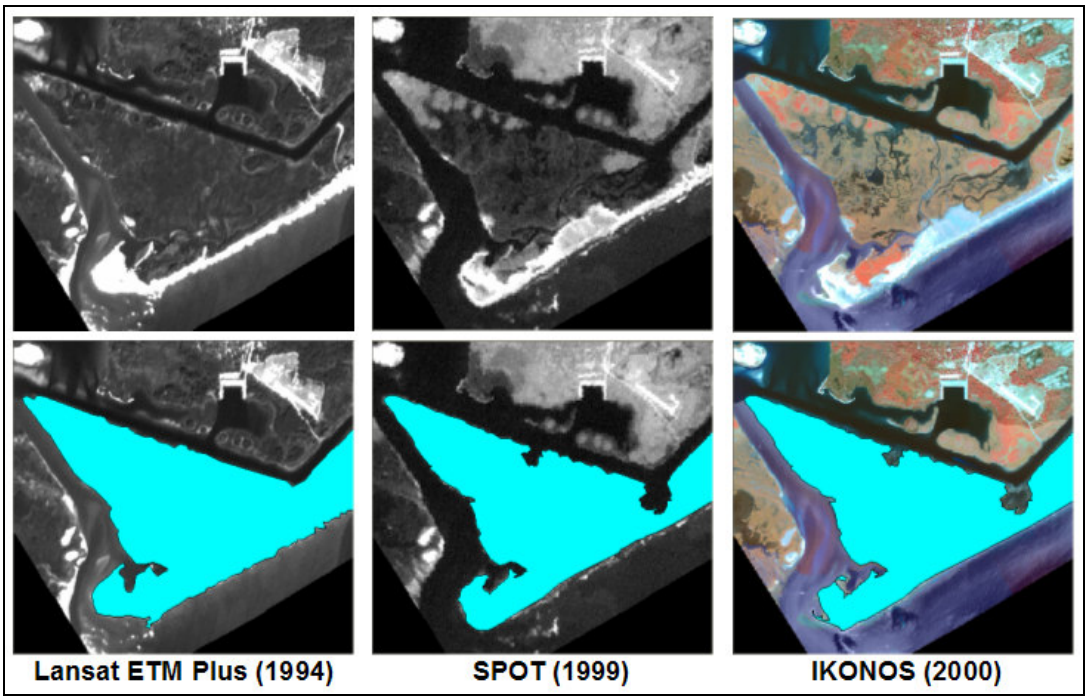


Figure 6.13 Coastlines Extracted from Three Different Time Image Sources

Figure 6.14 clearly shows the changes of coastlines through time. The difference of coastlines can be easily visualized by overlaying these lines in a conventional GIS. In a conventional GIS, these lines will be entered as different layers of which each layer will be referenced to a different time. Even though all three lines would be entered in a layer as lines instead of polygons, the lines would have different IDs so that the lines would not be the same feature. It could not be visualized under the layer-based model for the conventional GIS that the lines represent essentially the same feature.

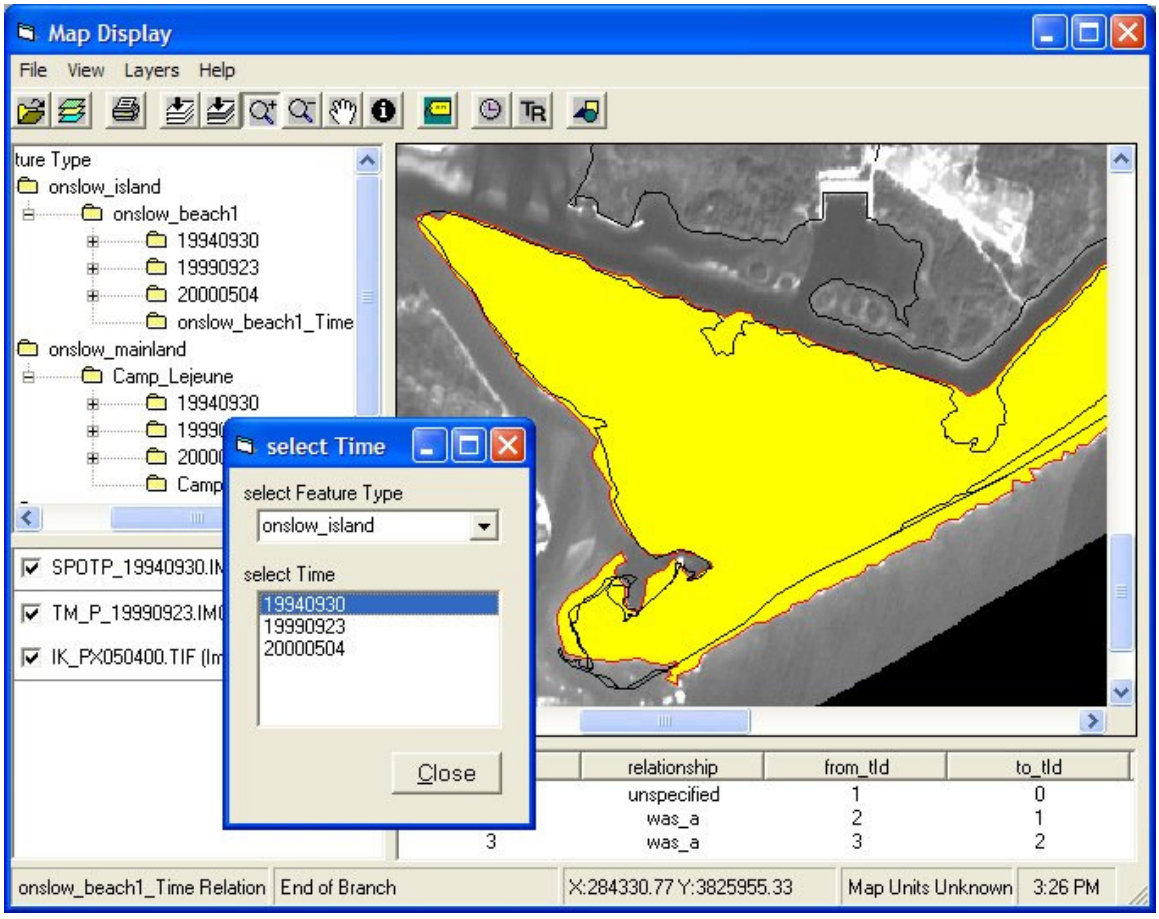


Figure 6.14 Multi-temporal Situation of Coastline Feature

Since three different lines are the same feature, they are populated as one feature ‘onslow_beach1’ with three different times in the feature database ‘Camp_lejeune’. In the

display module of FOGIS, the temporal differences of the coastline feature can be easily queried by selecting a specific time under the feature name in the tree viewer and selecting the feature type and time in the time query tool that can be activated by selecting the clock icon on the menu bar. Figure 6.14 shows the different coastlines are the same feature 'onslow_beach1' referenced on three different times under the feature type 'onslow_island'. The coastline in 1994 has been selected and easily distinguished from the other coastlines that are referenced to a different time. If multiple features under a specific feature type or all feature types need to be queried based on a specific time, the time query tool can also be used by selecting the specific feature type or all types with time.

6.4 Application II: Retrieval of Features using Thematic Relationship

The representation of thematic relationships is one of the important capabilities of the feature data model. The thematic relationship represents the relationships between feature instances so that it has been populated under the feature relationship class in the feature data model. The thematic relationship of a feature can be used to query for related features. The study area for this application is the same Camp Lejeune area, but is focused on roads in Onslow County area (Figure 6.15). The road data will be entered and used for a feature query using the feature relationship.

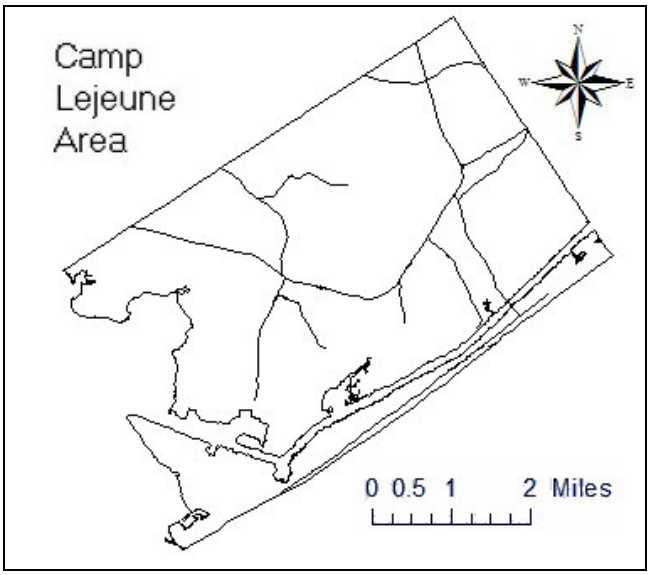


Figure 6.15 Roads in Camp Lejeune Study Area

6.4.1 Processing Steps

The existing road data in shape file format has been imported and populated into the feature database that was created in the previous application. The procedures for populating road features into the feature database are as follows (Figure 6.16). First, the import option and input file and output database has to be selected. Second, by clicking 'OK', the schema constructor will be activated, in which the feature type, feature name, time, and attribute types should be chosen and then click 'OK'. With the selected attribute types, the attribute data that has the same column name in the dbf file will be automatically populated as attribute instances under the attribute type in the feature database. Third, during populating features in the feature database, the projection information of the input data will be examined and compared with that in the database. If the feature database is newly created and the input data have no explicit projection information, the projector will be activated to select and populate projection information. Finally, the populated features can be visualized in the display module.

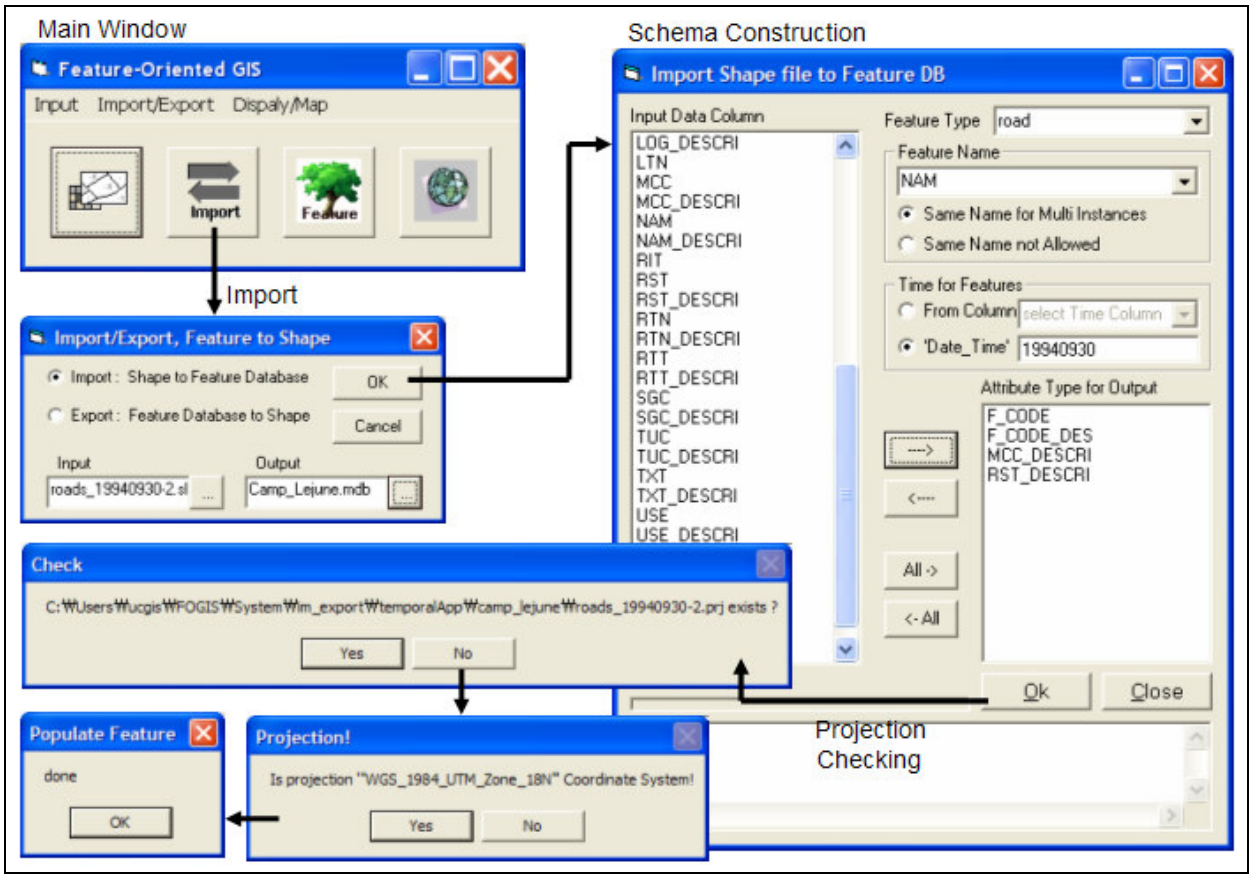


Figure 6.16 Importing Procedures for Populating Road Features

6.4.2 Roads Query based on Feature Relationship

After populating the features, the feature relationship between each road has been entered directly into the feature database 'Camp_Lejeune'. In this application, only 'connected_to' relationship has been used for roads that are connected to other roads in order to maintain the connectivity information between roads. In a conventional GIS, the connectivity information could be extracted from the topology using geometry so that all nodes have to be examined to complete connectivity between features. However, in FOGIS, feature relationships are directly used for searching connected features. This may reduce processing time for searching features based on connectivity.

In Figure 6.17, roads that are connected to the road ‘State_Highway_172’ have been queried based on a specific time (‘19940930’) and relationship (‘connected_to’) using the query tool that can be activated by selecting ‘TR’ icon on the menu bar in the display module. Figure 6.17 also shows the result of feature query based on the relationship. On the canvas in the display module, the criteria feature (road ‘State_Highway_172’) is represented with red color. The searched roads with selected relationship (‘connected_to’) are represented with cyan. If one of the searched roads is selected, then the road will be colored as yellow.

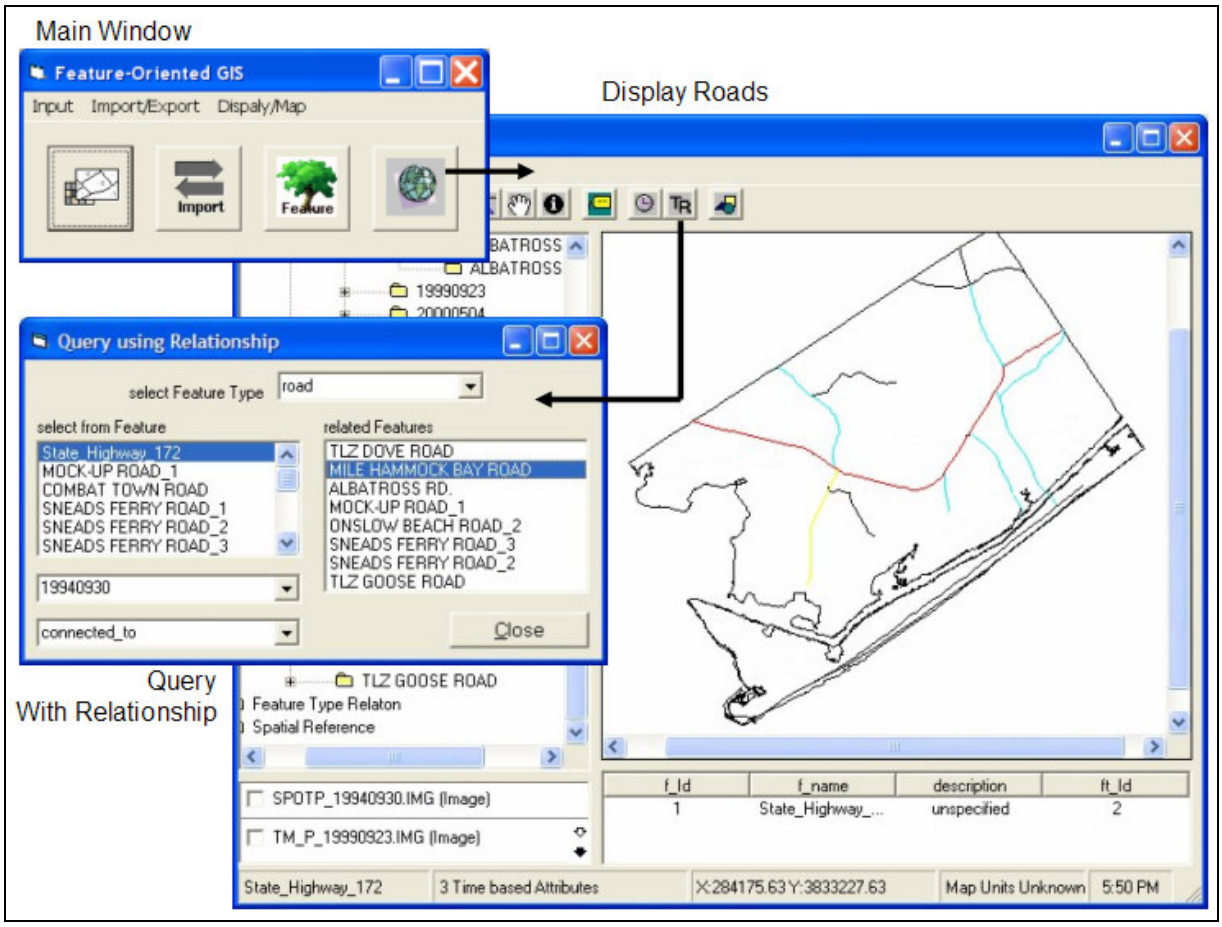


Figure 6.17 Road Query using Feature Relationship (Thematic Relationship)

6.5 Summary

In this chapter, the input procedure in Chapter 5 and the import/export and display module in Chapter 4 have been integrated to provide for a complete procedure from data generation to the visualization of features stored in feature database. The input module has been modified to be incorporated with a feature database. The projector and database creator has been added and the input module can populate multiple types of classes and multiple geometry types for each class at one time. It may increase the efficiency in the digitizing process because each feature type and geometry type can be separated into different layers in the conventional layer-based model approach.

In the import/export module, the export procedure has been designed to produce proper shape files from parts of the feature database. The shape file usually contains one geometry type based on a certain time. The export user interface, therefore, contains the selection for feature type, geometry type, and time. One or more attributes for the selected feature type in a feature database can be exported by an attribute selector.

For feature visualization, the tree viewer has been added. Using a tree structure, the classes that are well-organized hierarchically can be visualized effectively. The name of a feature can be directly shown in the tree structure even with other classes. In the layer-based model, the feature types are usually used for the name of the layer so that detailed feature names could not be directly shown without query over the database. With the time query tool in the display module, a feature query tool as one of the spatial analysis tools has been added, which uses feature relationships (thematic relationships) to query features.

With integrated FOGIS, two applications have been conducted. The first application was multi-temporal representation of coastlines. Three different coastlines that are the same feature

have been extracted from different image sources at different times and populated into the feature database. By a time query in the display module, the differences of coastlines through time could be effectively visualized and compared with each other.

The second application was the feature query based on a thematic relationship. For this application, pre-existing road data in shape file format have been imported into the feature database. With the feature query tool using the feature relationship, the roads linked to the input road have been queried easily and visualized with different colors. Each road in the queried roads also can be identified separately with the others. This application shows the usability of feature relationship (thematic relationship) for spatial analysis directly.

CHAPTER 7

CONCLUSION AND DISCUSSION

7.1 Conclusion

The aim of this study was to develop a prototype FOGIS that fully implements a feature concept with object-oriented programming. Many current GIS use a layer-based data model that is geometry-oriented model, so they are limited to represent temporal aspects and relationships of geographic features. A feature should be represented more effectively in a feature data model, which represents geographic phenomena fully with three dimensions: space, theme, and time. A FOGIS was developed based on the feature data model with object-oriented programming, so geographic features could be represented more effectively.

Five major research themes addressed in this study were to construct a feature data model, to develop an input procedure, to use the procedure for a geographic application, to design and implement a FOGIS with a feature database, and to apply the FOGIS for spatial query. Also, five problems in the conventional GIS identified in this study were limitations on the representation of disjunctive thematic relations, the multi-temporal representation of a feature, the usage of multiple geometry types of features, the effective feature retrieval using its name, and the use of an expert system for GIS (Table 7.1).

Table 7.1 Relation between Research Theme and Solving Problems

Research Themes	Solving Problems
• To construct a feature data model (Chapter 3)	• Disjunctive thematic relations • Multiple geometry types usage • Multi-temporal representation
• To develop an input procedure (Chapter 5) • To apply input procedure (Chapter 5)	• Expert system with GIS
• To design and implement FOGIS (Chapter 4, 6) • To use FOGIS for spatial query (Chapter 4, 6)	• Feature retrieval by name

A feature data model, which was the first research theme, was constructed using a feature concept, abstraction methods of SDTS, and the key concepts of an object-orientation in Chapter 3. Geographic features in the feature data model had three aspects: spatial, thematic, and temporal dimensions. Each dimension had attributes and relationships. By constructing a feature data model, the first three problems identified in this study were solved.

Disjunctive thematic relations, which was the first problem identified in this study, could be effectively represented by the hierarchical structure of features in the feature data model. Also, thematic relations could be populated directly in the feature relationship class in the feature data model (see Figure 3.6). Multiple geometry types, which was the second problem identified in this study, could be used for the same feature class in the feature data model. For example, some buildings could be mapped as points and the others as polygons based on the scale of an application (see Figure 4.5). Multi-temporal situations of a phenomenon, which was the third problem identified in this study, could be entered in one feature. One feature could have multiple spatial and thematic attributes and relationships based on each time (see Figure 4.12).

A spatial data input procedure, which was the second research theme in this study, was developed using an interactive rule-based expert system in Chapter 5. The input procedure was

designed to generate spatial data for both conventional GIS and FOGIS. Mapped data in the input procedure could be directly populated as instances of the classes in the feature data model and exported in a shape file for the conventional GIS. The developed input procedure applied for airphoto interpretation, which was the third research theme in this study. Knowledge (symbols and rules derived from decision tree) for the expert system was constructed and tested for the interpretation of panchromatic DOQQ images. The result of land use/cover data, over 93 % in overall accuracy, showed the knowledge could be used for another application of the panchromatic image interpretation.

The number of GIS users has been increased continuously because of its data handling capability and the spread of spatial data. Almost all current commercial GIS software packages, however, lack utilities to help neophytes in GIS to generate and analyze spatial data, which is the fifth problem identified in this study. This problem was solved by the input procedure developed in Chapter 5, which used an interactive rule-based expert system with proper explanation in a step-by-step fashion (see Figure 5.7). The input procedure also provided a way to integrate an expert system with GIS.

With the feature data model, a prototype system and a feature database was designed and implemented in Chapter 4. Fifteen classes of feature data model were designed and used for the system implementation. The developed system has three main parts: import/export module, display module, and database for feature storage. The import/export module could import and export data between a shape file and a feature database. The display module was designed to visualize all information features: attributes and relationships on space, theme, and time. The tree viewer in the display module could effectively visualize the data structure of features. For the storage of features, the feature database was designed and implemented using tables in the

relational database (see Figure 4.6). The developed system was applied for the visualization of multi-temporal feature instances on the coast area and the changes in buildings through time were effectively represented (see Figure 4.12).

System architecture for FOGIS was designed including a prototype system in Chapter 4 and the input procedure in Chapter 5 and implemented for a complete procedure from data generation to the visualization of features, which was the fourth research theme in this study. Two spatial analysis tools using time and thematic relationship were developed in FOGIS. In FOGIS, all information of features could be visualized by selecting feature name in a tree viewer, which solved the fourth problem identified in this study. A feature's name is an identifier in FOGIS so that all attributes and relationships are directly linked to the name, which is used to retrieve all information of a feature.

To test FOGIS, two applications were executed, which was the final research theme in this study. The first application was multi-temporal representation of coastlines. Three different coastlines extracted from different image sources at different times were populated into the feature database and were effectively visualized and compared with each other by a time query tool. The second application was the feature query based on a thematic relationship. With the feature relationship query tool, the roads were easily queried and effectively visualized with different colors, which showed the usability of feature relationships (thematic relationships) for spatial analysis.

7.2 Contributions and Limitations

This study makes several contributions to the GIS research community. The most important is that it provides a conceptual framework and a methodology for building FOGIS or implementing the object-oriented paradigm in a GIS environment.

Second, this study also provides a methodology for integrating an expert system as a consulting tool for GIS, which gives users a guideline for working procedures. In this study, an expert system was integrated into FOGIS in order to give users a guideline for the interpretation of an aerial photograph using an interactive question-and-answer sequence. With the interactive sequence, the expert system generated questions, based on the user's answers, and finally suggested a solution.

Third, this FOGIS expanded current GIS capabilities for feature representation by providing methods of multi-temporal representation and the use of multiple geometry types for a feature. The changes of a feature could be effectively represented by multiple instances based on time. A feature could be mapped as multiple types of geometry based on the application scale. In FOGIS, two query tools were developed: time and feature relationship query tools, which also expanded the spatial analysis capabilities of current GIS.

Finally, FOGIS is compatible with conventional GIS analysis ability and continues the usability of conventional GIS, because this system can import and export spatial data with a shape file format. The input procedure of FOGIS can also generate spatial data for both conventional GIS and FOGIS.

However, this study has limitations. First, in a feature data model, various relationships can be effectively represented, but it may be difficult to identify all relationships of a feature. Second, there is no automatic method to identify relationships of features, which requires manual

effort. Third, there are no standard terms for representing the relationship of features. One relationship (‘Is A’) can be represented by different terms (‘Is A’, ‘Is-A’, and ‘Is_A’). With this inconsistency of feature relationships, the implementation of query procedure using the relationships would be difficult. Fourth, in FOGIS, analysis functions of GIS except two query tools were not implemented, but spatial data in this system could be still used in the conventional GIS environment by exporting a feature database. Finally, specific efforts toward optimizing code and sophisticating the visual interface were not considered. Thus the performance of this system could be improved and enhanced.

7.3 Further Study

For further study, first, the developed feature system can be applied for improving the performance of the geographic analysis. For the shortest-path algorithm in the feature space, the feature itself can be a node for the searching process instead of each point in geometry. A line representing roads needs at least two points. Even though all lines are composed only of two nodes, it will take more time to search a location along roads using geometric nodes than to search the location using road relationships. In the road relationships, road name is a node in the road networks. It could save significant processing time.

Second, features in three dimensions (x,y,z) could be represented effectively with feature relationships such as ‘inside_of’, ‘northeast_corner_on_2floor’ and so on for buildings in complex urban information system. With geometry, the topological structure would be very complex to represent features in three dimensional spaces. Even though there is a lack of standard terms for representing feature relationships, the feature relationships can easily

represent the features in three dimensions because it uses natural language that can be easily composed and understood.

Third, effective directional representation for hierarchical water or road networks could be possible with feature relationships for direction such as 'from_to', 'connected_to', and so on. A feature data model uses the hierarchical structure to represent features, which is one of the key elements of object-orientation. Even though the current feature data model in this study uses a hierarchical structure only for representing feature attributes and relationships, the hierarchical structure of a feature data model could be extended to represent the hierarchical structure among feature instances in the case of water or road networks.

Finally, various types of the feature relationships should be defined clearly because there are no standard terms for them. A researcher in geography may have the potential for this work because geography has well developed discipline in the geographic phenomena.

REFERENCES

Abadi, M., and L. Cardelli, 1996, *A Theory of Objects*, Springer, New York, 396 p.

Abbott, R., 1983, "Program Design by Informal English Descriptions", *Communications of the ACM*, Vol. 26, No. 11, pp.882-894.

Althoff, K., S. Wess, and M. Manago, 1994, "Reasoning with Cases-Theory and Practice", In Cohn, A.G., (Ed.), *Proceedings of the 11th European Conference on Artificial Intelligence August 8-12, Amsterdam, the Netherlands*, John wiley & Sons, Inc., Chichester/London/New York, pp.1-26.

Arango, G., 1989, "Domain Analysis: From Art Form to Engineering Discipline", *SIGSOFT Engineering Notes*, Vol. 14, No 3, pp.152-159.

Atkinson, M., P. Bailey, K. Chisholm, P. Cockshott, and R. Morrison, 1983, "An Approach to Persistent Programming", *The Computer Journal*, vol. 26, No. 4, pp.360-365.

Atkinson, M., F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik, 1989, "The Object-Oriented Database System Manifesto", *in Proceedings of the First International Conference on Deductive and Object-Oriented Databases*, Kyoto, Japan, December 4-6, 1989, pp.40-57.

Avery, T.E., and G.L. Berlin, 1992, *Fundamentals of Remote Sensing and Airphoto Interpretation*, 5th Ed, Macmillan Publishing Company, New York, 472 p.

Barrea, R., A.U. Frank, and K. Al-Taha, 1991, "Temporal relations in geographic information systems: a workshop at the University of Maine", *SIGMOD Record*, Vol. 20, No. 3, pp.85-91.

Bian, L., 2000, "Object-oriented representation for modeling mobile objects in an aquatic environment", *International Journal of Geographical Information Science*, Vol. 14, No. 7, pp.603-623.

Biggerstaff, T. and C. Richter, 1989, "Re-usability Framework, Assessment and Directions", *Software Reusability: Vol. 1, Concepts and Models*, ACM Press, New York, NY, pp.1-17.

Booch, G., 1994, *OBJECT-ORIENTED ANALYSIS AND DESIGN with Applications*, 2nd Edition, The Benjamin/Cummings Publishing Company, Inc., RedWood, California, 589 p.

Borgida, A., 1988, "Modeling class hierarchies with contradictions", *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data In SIGMOD Record*, Vol. 17, No. 2, pp.434-443.

Bramer, M., 1989, "Expert systems: where are we and where are we going?", In Forsyth, R., (Ed.), *EXPERT SYSTEMS: Principles and case studies*, 2nd Ed, Chapman and Hall, New York, NY, pp.31-53.

Carrico, M. A., J. C. Girard, and J. P. Jones, 1989, *Building knowledge systems*, McGraw-Hill Book Co., New York, 335 p.

Cattell, R.G.G., 1994, *Object Data Management: Object-Oriented and Extended Relational Database Systems*, Addison-Wisley, Reading, MA, 389 p.

Cattell, R.G.G., 1996, *The Object Database Standard: ODMG-93*, Morgan Kaufmann Publishers, San Francisco, CA, 184 p.

Chen, P.P.S., 1976, "The entity-relationship model-toward a unified view of data", *Association for Computing Machinery Transactions on Database Systems*, Vol. 1, No. 1, pp.9-36.

Coad, P., and E. Yourdon, 1990, *Object-Oriented Analysis*, Prentice-Hall, Englewood Clieffs, New Jersey, 233 p.

Codd, E.F., 1970, "A relational model of data for large shared data banks", *Communications of the Association for Computing Machinery*, Vol. 13, No. 6, pp.377-387.

Dahl, O.J. and K. Nygaard, 1966, "SIMULA – An Algol-based Simulation Language", *Communications of the ACM*, Vol. 9, No. 9, pp.671-678.

Dahl, O.J. and B. Myrhang, and K. Nygaard, 1968, SIMULA 67 Common Base Language, Norwegian Computing Centre, Forskningveien 1B, Oslo 3, Norway.

Davis, R., and W. Hamscher, 1986, "Model-based Reasoning: Troubleshooting", In Shrobe, H. and the American Association for Artificial Intelligence, (Eds.), *Exploring Artificial Intelligence*, Morgan Kaufmann publishers, Inc., San Mateo, CA, pp. 239-346.

Defense Mapping Agency, 1987, *Feature Attribute Coding Standard (FACS)*, internal report, Reston, Virginia: U.S. Defense Mapping Agency.

Digital Geographic Information Working Group, 2000, "Part 4. Feature and Attribute Coding Catalogue (FACC)", *The Digital Geographic Information Exchange Standard (DIGEST)*, Edition 2.1, http://www.digest.org/html/DIGEST_2-1_Part4.pdf, 28 p.

Egenhofer, M. J. and J.R. Herring, 1991, "High-level spatial data structures for GIS", In Maguire, D.J., M.F. Goodchild, and D.W. Rhind, (Eds.), *Geographical Information Systems: Principles and Applications, Volume 1*, Longman Scientific Publications, London, pp.227-237.

Environmental Systems Research Institute (ESRI), 1999a, *ESRI MapObjects™ Version 2.0*, Environmental Systems Research Institute, Inc, Redlands, California, 26 p.

Environmental Systems Research Institute (ESRI), 1999b, *Modeling Our World*, Environmental Systems Research Institute, Inc., Redlands, California.

Environmental Systems Research Institute (ESRI), 2001, *Getting Started with the Map Control Using Visual Basic*, Environmental Systems Research Institute, Inc., RedWood, California, 28 p.

Estes, J.E., E.J. Hajic, and L.R. Tinney, 1983, "Fundamentals of Image Analysis: Analysis of Visible and Thermal Infrared Data", In Colwell, R.N., (Ed.), *Manual of Remote Sensing, Vol. I,* 2nd Ed, Falls Church, Va.: American Society of Photogrammetry, pp.987-1012.

Ford, N., 1991, *Expert systems and Artificial Intelligence: An information manager's guide*, Library Association Publishing Ltd, London, 277 p.

Foschi, P.G., and D.K. Smith, 1997, "Detecting Subpixel Woody Vegetation in Digital Imagery Using Two Artificial Intelligence Approaches", *Photogrammetric Engineering and Remote Sensing*, Vol. 63, No. 5, pp.493-500.

Gatrell, A.C., 1983, *Distance and Space: A Geographical Perspective*, Clarendon Press, Oxford, 185 p.

Gatrell, A.C., 1991, "Concepts of Space and Geographical Data", In Maguire, D.J., M.F. Goodchild, and D.W. Rhind, (Eds.), *Geographical Information Systems: Principles and Applications, Volume 1*, Longman Scientific Publications, London, pp.119-134.

Graham, I., 1989. "Inside the inference engine", In Forsyth, R., (Ed.), *EXPERT SYSTEMS: Principles and case studies*, 2nd Ed, Chapman and Hall, New York, NY, pp. 57-83.

Graham, I., 2001, *OBJECT-ORIENTED METHODS: Principles & Practice*, 3rd Edition, Pearson Education, London, 864 p.

Guptill, S.C., K.J. Boyko, M.A. Domaratz, R.G. Fegeas, H.J. Rossmeissl, and E.L. Usery, 1990, *An Enhanced Digital Line Graph Design*, USGS Circular 1048, Restion, Virginia: USGS, 20 p.

Harmon, P. and B. Sawyer, 1999, *UML for Vusual Basic 6.0 Developers: Using Visual Modeler and Rational Rose 98*, Morgan Kaufmann Pub., San Francisco, CA, 383 p.

Harris, M., 1997, *Teach yourself: Visual Basic in 21 Days for Applications 5*, Sams Publishing Co., Indianapolis, Ind., 1248 p.

Holt, A., and G. L. Benwell, 1997, "Applying case-based reasoning techniques in GIS", *International Journal of Geographical Information Science*, Vol. 13, No. 1, pp.9-25.

Jacobson, I., 1992, *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, Workingham, England, 350 p.

James, P.E., and C.F. Jones, 1954, *American Geography: Inventory and Prospect*, Syracuse University Press, New York, 598 p.

Kochhar, N. and E. Lad, 1998, "Introduction", *Introduction to Oracle: SQL and PL/SQL*, Oracle Corporation, pp.I-1–I-32.

Lakoff, G., 1987, *Women, Fire, and Dangerous Things: What Categories Reveal About the Mind*, University of Chicago Press, Chicago, 614 p.

Loomis, M.E.S., 1992, "Object versioning", *Journal of Object-Oriented Programming*, Vol. 4, No. 8, pp.40-43.

McCarthy, J., 1960, ""Recursive Functions of Symbolic Expressions and their Computation by Machine, Part I", *Communications of the ACM*, Vol. 3, No. 4, pp.184-195.

McMaster, R.B., 1989, "Numerical Generalization in Cartography", *Cartographica*, Vol. 26, No. 1, pp.1-6.

Meyer, B., 1987, "Reusability: The Case for Object-Oriented Design", *IEEE Software*, Vol. 4, No. 2, pp.50-64.

Meyer, B., 1988, *Object-Oriented Software Construction*, Englewood Cliffs, NJ, 534 p.

Milne, P., S. Milton, and J.L. Smith, 1993, "Geographical object-oriented databases-a case study", *International Journal of Geographic Information Systems*, Vol. 7, No. 1, pp.39-55.

Moller-Jensen, L., 1990, "Knowledge-Based Classification of an Urban Area Using Texture and Context Information in Landsat-TM Imagery", *Photogrammetric Engineering and Remote Sensing*, Vol. 56, No. 6, pp.899-904.

Moore, J. and S. Bailin, 1988, *Position Paper on Domain Analysis*, CTA Incorporated, Laurel, MD.

Naylor, C., 1989, "How to build an inferencing engine", In Forsyth, R., (Ed.), *EXPERT SYSTEMS: Principles and case studies*, 2nd Ed, Chapman and Hall, New York, NY, pp.84-105.

Nikolopoulos, C., 1997, *Expert Systems: Introduction to First and Second Generation and Hybrid Knowledge Based Systems*, MARCEL DEKKER, INC., New York, 331 p.

National Oceanic and Atmospheric Administration (NOAA), 1999, "Hurricane Basics", *http://hurricane.noaa.gov/pdf/hurricanebook.pdf*, 19 p.

ONTOS INC., 1991, *ONTOS Developers Guide*, ONTOS Inc., Burlington, Mass.

Oosterom, P.V., and J.V.D. Bos, 1989, "An object-oriented approach to the design of geographic information systems", *Computer & Graphics*, Vol. 13, No. 4, pp.409-418.

Oracle Corporation, 1998, *Oracle8: Database Administration*, Vol. I, II, and III, Redwood Shores, CA.

Peuquet, D.J., 1988, "Representations of Geographic Space: Toward a Conceptual Synthesis", *Annuals of the Association of American Geographers*, Vol. 78, No. 3, pp.375-394.

Prieto-Diaz, R. and P. Freeman, 1987, "Classifying Software for Reusability", *IEEE Software*, Vol. 4, No. 1, pp.6-16.

Rentsch, T., 1982, "Object-Oriented Programming", *SIGPLAN Notices*, Vol. 17, No. 9, pp.51-79

Rhind, D.W., M.F. Goodchild, and D.J. Maguire, 1991, "Epilogue", In Maguire, D.J., M.F. Goodchild, and D.W. Rhind, (Eds.), *Geographical Information Systems: Principles and Applications, Volume 2*, Longman Scientific Publications, London, pp.313-327.

Robinson, A.H., R.D. Sale, J.L. Morrison, and P.H. Muehrcke, 1984, *Elements of Cartography*, 5th Ed., John Wiley & Sons, New York, 544 p.

Roman, S., 1997, *Concepts of Object-Oriented Programming with Visual Basic*, Springer-Verlag, New York, 188 p.

Rosch, E., 1978, "Principles of Categorization", In Rosch, E. and B.B. Lloyd, (Eds.), *Cognition and Categorization*, Halstead Press, New York, pp.27-48.

Ross, R., 1987, *Entity Modeling: Techniques and Application*, Database Research Group, Boston, MA, 218 p.

Rubin, K., A. Goldberg, 1992, "Object Behavior Analysis", *Communications of the ACM*, Vol. 35, No. 9, p.48-62.

Ryu, J., 1999. *Korean Visual Basic 6.0 using Bible*, Youngin.com Publishing Co., Seoul, Republic of Korea, 722 p.

Schank, R., 1982. *Dynamic Memory: A Theory of Learning in Computers and People*, Cambridge University Press, 234 p.

Shlaer, S., and S. Mellor, 1988, *Object-Oriented Systems Analysis: Modeling the World in Data*, Yourdon Press, EngleWood Cliffs, New Jersey, 144 p.

Shlaer, S., and S. Mellor, 1991, *Object Lifecycles: Modeling the World in States*, Yourdon Press, EngleWood Cliffs, New Jersey, 251 p.

Slade, S., 1991, "Case-Based Reasoning: A Research Paradigm", *AI MAGAZINE*, Vol. 12, No. 1, pp.42-55.

Stefik, M., and D. Bobrow, 1986, "Object-Oriented Programming: Themes and Variations", *AI Magazine*, Vol. 6, No. 4, p.40-62.

Stephens, R., 2000, *Visual Basic Graphics Programming*, 2nd Ed, John Wiley & Sons, Inc., New York, 712 p.

Tang, A.Y., T.M. Adams, E.L. Usery, 1996, "A spatial data model design for feature-based geographical information systems", *International Journal of Geographic Information Systems*, Vol. 10, No. 5, pp.643-659.

Tatsuyama, T., 1987, "Knowledge-Based Aerial Image Understanding Systems and Expert Systems for Image Processing", *IEEE Transactions on Geoscience and Remote Sensing*, Vol, GE-25, No.3, pp.305-316.

Taylor, D., 1991, *Object-oriented Technology: A Manager's Guide*, Addition-Wiley Pub Co., New York, 160 p.

Trainor, T.F., 1990, "Fully automated cartography: a major transition at the Census Bureau", *Cartography and Geographic Information Systems*, Vol. 17, No. 1, pp.27-38.

Usery, E.L., P. Altheide, R.R.P. Deister, and D.J. Barr, 1988, "Knowledge-Based GIS Techniques Applied to Geological Engineering", *Photogrammetric Engineering and Remote Sensing*, Vol. 54, No. 11, pp.1623-1628.

Usery, E.L., 1993, "Category Theory and the Structure of Features in Geographic Information Systems", *Cartography and Geographic Information Systems*, Vol. 20, No. 1, pp.5-12.

Usery, E.L., 1994a, "Implementation Constructs for Raster Features", *Proceedings of ASPRS/ACSM Annual Convention*, Reno, Nevada, pp.661-670.

Usery, E.L., 1994b, "Display of geographic features from multiple image and map databases", Proceedings, ISPRS Commission IV Symposium on Mapping and Geographic Information Systems, *International Archives of Photogrammetry and Remote Sensin*g, Volume 30, Part 4, Athens, Georgia, pp.2-9.

Usery, E.L., 1996a, "A Conceptual Framework and Fuzzy Set Implementation for Geographic Features", In Burrough, P.A. and A.U. Frank, (Eds.), *Geographic Objects with Indeterminate Boundaries*, Taylor and Francis, London, pp.71-85.

Usery, E.L., 1996b, "A Feature-Based Geographic Information System Model", *Photogrammetric Engineering & Remote Sensing*, Vol. 62, No. 7, pp.833-838.

Usery, E.L., 2000, "Multidimensional Representation of Geographic Features", *Proceedings of the XIXth International Society for Photogrammetry and Remote Sensing,* Vol. XXXIII, Part B4/3, Commission 4, pp.1092-1099.

Usery, E.L., G. Timson and M. Coletti, 2002, "A Multidimensional Geographic Feature System", In *GIScience 2002: The Second International Conference on Geographic Information Science*, National Atmospheric and Oceanic Administration, pp.279-282.

U.S. Geological Survey, 1997, *Spatial Data Transfer Standard (SDTS) – Part 1, Logical Specifications*, American National Standards Institute, New York.

Vckovski, A., 1998, *Interoperable and Distributed Processing in GIS*, Taylor & Francis, London, 230 p.

Wachowicz, M., 1999, *Object-Oriented Design for Temporal GIS*, Taylor & Francis, London, 118 p.

Ward, P. and S. Mellor, 1985, *Structured Development for Real-time Systems*, Three Volumes, Yourdon Press, Englewood Cliffs, New Jergey.

Watson, L., 1997. *Applying Case-Based Reasoning*, Morgan Kaufmann Publishers, Inc, San Francisco, CA, 289 p.

Wilson, P.A., 1997. "Rule-Based Classification of Water in Landsat MSS Images Using the Variance Filter", *Photogrammetric Engineering and Remote Sensing*, Vol. 63, No. 5, pp.485-491.

Winston, P.H., 1992, *Artificial Intelligence*, 3rd Ed, Addison-Wesley publishing Co, Reading, Mass., 691 p.

Wirfs-Brock, R., R. Wilkerson, and L. Wiener, 1990, *Designing Object-Oriented Software*, Prentice-Hall, Englewood Clieffs, New Jersey, 341 p.

Worboys, M., H.M. Hearnshaw, and D.J. Maguire, 1990, "Object-oriented data modeling for spatial databases", *International Journal of Geographic Information Systems*, Vol. 4, No. 4, pp.369-383.

Zaratian, B., 1998, *Microsoft Visual C++ 6.0 programmer's guide*, Microsoft Press, Redmond, Washington, 722p.

APPENDIX A

CLASS MODULES AND COLLECION CLASSES ON FEATURE IN FOGIS

1. ‘FeatureType’ Class Module for Single Feature Type

```
'local variable(s) to hold property value(s)
Private mft_id As Integer
Private mft_name As String 'local copy
Private mft_desc As String

Public Property Let ft_id(ByVal vData As Integer)
'used when assigning a value to the property,
    mft_id = vData
End Property

Public Property Get ft_id() As Integer
'used when retrieving value of a property,
    ft_id = mft_id
End Property

Public Property Let ft_name(ByVal vData As String)
    mft_name = vData
End Property

Public Property Get ft_name() As String
    ft_name = mft_name
End Property

Public Property Let ft_desc(ByVal vData As String)
    mft_desc = vData
End Property

Public Property Get ft_desc() As String
    ft_desc = mft_desc
End Property
```

2. ‘FeatureTypes’ Collection Class for Multiple Feature Type Instances

```
Option Explicit
'Declare a variable of type collection
Private mFtypes As New Collection

'Expose count property
Public Property Get Count() As Long
  Count = mFtypes.Count
End Property

'Wrapper for Item method
Public Function Item(Index As Variant) As CFeatureType
  Set Item = mFtypes.Item(Index)
End Function

'Wrapper for Add method
Public Function add(mft_id As Integer, mft_name As String, mft_desc As String) As
CFeatureType
  'Define a new CFeatureType class
  Dim mFtype As New CFeatureType

  With mFtype
    'set property
    .ft_id = mft_id
    .ft_name = mft_name
    .ft_desc = mft_desc
  End With

  'add FeatureType to FeatureTypes collection
  mFtypes.add mFtype
End Function

Public Sub Delete(ByVal Index As Variant)
  mFtypes.Remove Index
End Sub
```

3. ‘FeatureTypeRel’ Class Module for Single Feature Type Relationship

```
'local variable(s) to hold property value(s)
Private mftr_id As Integer
Private mftr_rel As String 'local copy
Private mfromft_id As Integer
Private mtoft_id As Integer

Public Property Let ftr_id(ByVal vData As Integer)
    mftr_id = vData
End Property

Public Property Get ftr_id() As Integer
    ftr_id = mftr_id
End Property

Public Property Let ftr_rel(ByVal vData As String)
    mftr_rel = vData
End Property

Public Property Get ftr_rel() As String
    ftr_rel = mftr_rel
End Property

Public Property Let fromft_id(ByVal vData As Integer)
    mfromft_id = vData
End Property

Public Property Get fromft_id() As Integer
    fromft_id = mfromft_id
End Property

Public Property Let toft_id(ByVal vData As Integer)
    mtoft_id = vData
End Property

Public Property Get toft_id() As Integer
    toft_id = mtoft_id
End Property
```

4. ‘FeatureTypeRels’ Collection Class for Multiple Feature Type Relationship Instances

```
Option Explicit
'Declare a variable of type collection
Private mFtypeRels As New Collection

'Expose count property
Public Property Get Count() As Long
  Count = mFtypeRels.Count
End Property

'Wrapper for Item method
Public Function Item(Index As Variant) As CFeatureTypeRel
  Set Item = mFtypeRels.Item(Index)
End Function

'Wrapper for Add method
Public Function add(mftr_id As Integer, mftr_rel As String, mfromft_id As Integer,
mtoft_id As Integer) As CFeatureTypeRel
  'Define a new CFeatureTypeRel class
  Dim mFtypeRel As New CFeatureTypeRel

  With mFtypeRel
    'set property
    .ftr_id = mftr_id
    .ftr_rel = mftr_rel
    .fromft_id = mfromft_id
    .toft_id = mtoft_id
  End With

  'add FeatureTypeRel to FeatureTypeRels collection
  mFtypeRels.add mFtypeRel
End Function

Public Sub Delete(ByVal Index As Variant)
  mFtypeRels.Remove Index
End Sub
```

5. ‘Feature’ Class Module for Single Feature

```
'local variable(s) to hold property value(s)
Private mf_id As Integer
Private mf_name As String 'local copy
Private mf_desc As String
Private mft_id As Integer

Public Property Let f_id(ByVal vData As Integer)
    mf_id = vData
End Property

Public Property Get f_id() As Integer
    f_id = mf_id
End Property

Public Property Let f_name(ByVal vData As String)
    mf_name = vData
End Property

Public Property Get f_name() As String
    f_name = mf_name
End Property

Public Property Let f_desc(ByVal vData As String)
    mf_desc = vData
End Property

Public Property Get f_desc() As String
    f_desc = mf_desc
End Property

Public Property Let ft_id(ByVal vData As Integer)
    mft_id = vData
End Property

Public Property Get ft_id() As Integer
    ft_id = mft_id
End Property
```

6. ‘Features’ Collection Class for Multiple Feature Instances

```
Option Explicit
'Declare a variable of type collection
Private mFs As New Collection

'Expose count property
Public Property Get Count() As Long
  Count = mFs.Count
End Property

'Wrapper for Item method
Public Function Item(Index As Variant) As CFeature
  Set Item = mFs.Item(Index)
End Function

'Wrapper for Add method
Public Function add(mf_id As Integer, mf_name As String, mf_desc As String, mft_id As
Integer) As CFeature
  'Define a new CFeature class
  Dim mF As New CFeature

  With mF
    'set property
    .f_id = mf_id
    .f_name = mf_name
    .f_desc = mf_desc
    .ft_id = mft_id
  End With

  'add Feature to Features collection
  mFs.add mF
End Function

Public Sub Delete(ByVal Index As Variant)
  mFs.Remove Index
End Sub
```

7. Other Class Modules for Single Object

‘SpatialRef’ Class Module for Single Spatial Reference

‘FeatureRel’ Class Module for Single Feature Relationship

‘AttType’ Class Module for Single Attribute Type

‘Spatial’ Class Module for Single Spatial Attribute

‘Theme’ Class Module for Single Thematic Attribute

‘Time’ Class Module for Single Temporal Attribute

‘TimeRel’ Class Module for Single Temporal Relationship

8. Other Collection Classes for Multiple Objects

‘FeatureRels’ Collection Class for Multiple Feature Relationship Instances

‘AttTypes’ Collection Class for Multiple Attribute Type Instances

‘Spatials’ Collection Class for Multiple Spatial Attribute Instances

‘Themes’ Collection Class for Multiple Thematic Attribute Instances

‘Times’ Collection Class for Multiple Temporal Attribute Instances

‘TimeRels’ Collection Class for Multiple Temporal Relationship Instances