2003

Conceptual and application issues in the implementation of object-oriented GIS

Janette Elizabeth Bennett
West Virginia University

Conceptual and Application Issues in the Implementation of Object-Oriented GIS

Janette E. Bennett

**Thesis Submitted to the Eberly College of Arts and Sciences
at West Virginia University in partial fulfillment
of the requirements for the degree of**

**Master of Arts
in
Geography**

**Trevor Harris, Ph.D., Chair
Gregory Elmes, Ph.D.
Timothy Warner, Ph.D.**

Department of Geology and Geography

**Morgantown, West Virginia
2003**

Keywords: Geography, Geographic Information Systems, OOGIS, Schema, Geodatabase, ArcGIS

ABSTRACT

Conceptual and Application Issues in the Implementation of Object-Oriented GIS

Janette E. Bennett

The adoption of object-oriented technology for spatial data modeling is becoming a significant trend in GIS. This research explores the concepts of Object-Oriented GIS (OOGIS) and illustrates its versatility in two case studies. OOGIS provides a feature-based, intuitive representation of real world features. The study emphasizes the fundamental concepts of inheritance, polymorphism, and encapsulation in OOGIS and explores schema design, long transactions, and versioning. Further, the study discusses the advantages of OOGIS in the management and analysis of geospatial data. The case studies demonstrate both the conceptual basis of OOGIS and specific functionality including behavior, methods, versioning, long transactions and data locking. OOGIS demonstrates many advantages over the traditional entity-relationship model in database maintenance and functionality.

ACKNOWLEDGEMENTS

Thesis work can never be accomplished without the support of many people. To these people I am truly grateful. First and foremost, I would like to thank the members of my committee for their time, knowledge, insights, understanding and support. Without their guidance, I do not feel I would have produced a thesis of this quality. There are no words to describe my gratitude.

I would also like to thank the staff of the Geography Department. Donna Titus, Hope Stewart and Randy Crowe offered their continuous support and assistance. They have truly made the process of earning a graduate degree more enjoyable. I would also like to thank Carol Hando at the Dean's Office of the Eberly College of Arts and Sciences for her continuous support and encouragement. Additionally, I would like to thank the graduate coordinators of the Department of Geography: Dr. Robert Hanham and subsequently, Dr. Timothy Warner. Also, I would like to express my appreciation for the semesters of Graduate Assistantships that I was offered. The knowledge gained during those times is invaluable.

I would also like to thank the graduate students of the Department of Geography and Geology for your continuing support and wisdom. Each of you contributed something to me that was special, whether you knew it or not. I would also like to thank the West Virginia State GIS Technical Center for the opportunity to grow as a geographer by offering me summer employment during my graduate studies.

Finally, I would like to thank my parents, Joseph and Jacqueline McNeer, grandmother, Joan Hamilton, brother J. Tyler McNeer, husband Darren Bennett, and his family Lindy, Judith and Lori Bennett for seeing me through the good times and bad and offering encouragement every step of the way. I dedicate this thesis to my grandparents Harold Hamilton, William McNeer and Anelu McNeer. I know you are watching over all of us.

Table of Contents

List of Tables

List of Figures

Chapter 1. Introduction

The abstraction and conversion of features found in the real world into digital forms is a crucial component in the creation of information systems (Davis and Borges, 1994) and especially Geographic Information Systems (GIS). GIS represents the real world through the use of the data model, a core element of a GIS. To date, the most widely used data model used in GIS to represent real world features is the Entity-Relationship model. The Entity-Relationship model is "a logical way of describing entities and their relationships within a relational database" (Association for Geographic Information, 1999). An entity is a cartographic unit that cannot be broken down to any smaller unit. Relationships between entities are maintained within a data structuring system that uses collections of tables associated by common attributes. While this model is commonly used in the handling of geographic data, it does have some disadvantages and limitations that have led to the development of a new data model that emphasizes an object-based approach. The ability to manage these objects in a more effective manner, as well as associate the object with various behaviors, has made the object-oriented approach very attractive to GIS developers and users.

Object-Oriented GIS (OOGIS) does not have one generally accepted definition. Its origins stem from the field of computer science, where object-oriented programming languages were first developed and used. The GIS community has shown considerable enthusiasm for object-oriented technology, though the implementation of the technology has taken time. OOGIS has many advantages over current GIS software. In particular, OOGIS focuses on a feature based approach, where the emphasis is placed not only on the spatial component of the object, but also on the behaviors associated with the object.

The OOGIS data model thus has the potential to not only be a more accurate representation of the real world and also incorporate "intelligent" objects, that have associated behaviors and methods, which help to define the objects and the relationships between them. The recent movement toward feature-based GIS warrants an in-depth study of OOGIS, its concepts, strengths and weaknesses, and a review of how it is used in practice as demonstrated through two case studies.

1.1. Research Objectives

The two primary objectives of this research were to understand the conceptual basis of OOGIS and to explore its strengths and weaknesses through the implementation of OOGIS in a practical application. The first objective involved an in-depth exploration of the conceptual basis of OOGIS as it pertains to GIS system development. A comprehensive review of the conceptual base of object-oriented GIS was explored. Surprisingly, providing a definitive explanation of OOGIS proved difficult since there is no one generally accepted definition. This is a reflection of the evolution of OOGIS, whose roots stem from computer science. The concepts that make up the basis of OOGIS require a full understanding of a diverse terminology. Terminology pertinent to OOGIS are explored and examples of each concept are given in order to better understand the underlying principles of OOGIS.

The second objective was to use two case studies as a means of exploring issues involved in the implementation of OOGIS. Two projects involving the various concepts the distinguish OOGIS from its predecessors were selected because they contribute in different ways to an understanding of the concepts and the strengths and weaknesses of an OO approach to GIS. These case studies are the National Imagery and Mapping

Agency (NIMA) case study and the Federal Geographic Data Committee (FGDC) case study. A history of each case study was given and the specific tasks of each project that emphasize OO concepts were discussed. The case studies illustrate the general concepts of OOGIS, provide a comparison base to other GIS models and enable the benefits and limitations of OOGIS technology to be explored.

1.2. Research Questions

Four research questions stem from the overarching research objectives. They are:

1) What are the concepts and principles of OOGIS?
2) What OOGIS concepts are best illustrated in the case studies?
3) How does the OOGIS model compare with, or differ from, the Entity-Relationship model?
4) What are the perceived benefits and weaknesses of OOGIS?

1.3. Methodology

OOGIS concepts are reviewed in Chapter 2 by a literature search gained from a variety of different sources, including the Internet, journal articles, technical writings and books. The literature review provides the basis for the exploration of the fundamental concepts of OOGIS.

Two case studies were explored to exemplify three fundamental concepts of object-oriented GIS—encapsulation, polymorphism and inheritance. The projects were funded by the National Imaging and Mapping Agency (NIMA) and the Federal Geographic Data Committee (FGDC). In Chapter 3, the NIMA case study is discussed. The project began in 1998 and explores topics such as schema development, geodatabase, and methods. The FGDC project, discussed in Chapter 4, was undertaken in 1996-97 and emphasizes other OO concepts such as conflation, data locking, versioning, and the

benefits of a feature based approach to the framework national database. Conclusions to my thesis are presented in Chapter 5.

Chapter 2. Literature Review

2.1. Concepts and Terminology

There are several basic concepts underpinning OOGIS. An **object** is considered to be "a complex data structure that is capable of storing all of its data along with information about the necessary procedures to create, destroy and manipulate it" (Davis and Borges, 1994). Each object in an OOGIS contains not only information about its spatial location, but also its attributes and relationships with other objects. For example, a "house" object may have geographic information associated such as its street address. Its location in space could also be its coordinates in latitude and longitude. Attribute information about the house object might include its color, number of windows, or number of occupants. Relationship information about the house object could be the community the house is situated in, or the fire jurisdiction or voting district the house object is part of. Objects may be grouped logically to form **object classes,** and can also have subclasses. In addition, a class can be nested within another class. For example, several individual streets (objects) may be grouped to form a "roads" class. Roads, railroads and waterway classes can subsequently be grouped into a "transportation" class.

A **schema** is a collection of items that model some or all of a real world object. The schema is a critical component of object-oriented technology. A **database schema** is a group of related tables in a database that model features in the real world (Association for Geographic Information, 1999). A schema contains information such as data types, the hierarchy of the objects related within the OOGIS, and the relationships between objects. An established schema can assist in minimizing data errors by the use of validation methods.

There are three levels of schema design or schema modeling. The first level is the conceptual schema. A conceptual schema is developed before an actual computerized schema is developed or implemented and is therefore not limited to data types or relationships acceptable to database implementation. The second level schema is the logical schema, which reproduces the conceptual schema as a set of constructs supported by the database, such as classes and values. Finally, the physical schema determines how the logical schema is operationalized in the computer. OOGIS reduces the gap between the conceptual and the logical levels of schema modeling and the physical level of schema development is hidden from the end user (Wadembere, 2001).

Abstraction is at the heart of every GIS, not just an OOGIS. Abstraction is the main characteristics of a feature that can clearly differentiate it from any other feature and therefore absolutely define the boundaries of the feature, relative to the viewer's perspective (Booch, 1996). On other words, it is a simplified characterization of a real world feature that places emphasis on attributes that are significant and deemphasizes relatively insignificant properties.

There are three fundamental concepts that are common to every OOGIS—encapsulation, inheritance and polymorphism. **Encapsulation** represents information, methods and behaviors embedded within an object that are accessed via an external interface that allows users and other objects to interact with the object. The inner workings of the object are hidden from users and other objects. Access to the inner workings is granted only to the user creating the object, whether it is code or data that is embedded in the object. The creator has permission to create, destroy or manipulate the object.

Sub-classes may inherit both characteristics and changes from classes higher up in the data structure. The adoption process is known as inheritance. **Inheritance** is defined as the creation of an object class based on the properties of other object classes that are higher in the hierarchy of the object classes (Association for Geographic Information, 1999). A hierarchy is the structure of object classes or abstractions are ordered (Booch, 1996). This produces child and parent structures, where the child class "inherits" properties from the parent class. The ability to inherit properties makes for greater efficiency in the management of changes and updates in the database.

Objects in object classes need to be able to communicate with each other. While all objects receive instructions sent by user commands, in OOGIS only objects capable of undertaking the instructions have the ability to implement them. **Polymorphism** is the capacity for an object to respond to a single command that could be interpreted in several different ways by an object, depending on the nature or context of the object itself (Association for Geographic Information, 1999). For example, an object may receive the "calculate area" command. The message is sent to retrieve information for the requisite object, a polygon of some kind, which has the ability to carry out the instruction. Several different types of objects may receive this command, but each object would implement the command based on the nature of the object itself. Thus, if a feature that is circular receives the command, it will execute the calculation "area = πr^2". However, a different object of rectangular shape would execute the calculation "area = length * width". Despite the differing interpretations and calculations, only one command needs to be sent.

As the Entity-Relationship model recognizes and supports relationships between entities, so too does the OOGIS recognize and support relationships between objects. The relationships between objects are captured through topology. **Topology** is the relative location of spatial data independent of a Cartesian location (such as XY coordinates). Digital geographic data uses topological relationships such as connectivity and adjacency, which are usually expressed as relationships between nodes, links and polygons (Worboys, 1995). Topology varies depending on the type of object data structure. In a vector entity relationship, connectivity is defined explicitly by a database pointer between records that describes features linked in the real world (e.g. the junction of two gas pipelines). Topological relationships are built from simple elements into complex elements. In the vector data model, the elements include points, arcs, areas, and routes. Redundant data are eliminated because a single feature may represent more than one object in space. For example, an arc may represent a linear feature, part of the boundary of an area feature, or both. Topology in GIS is crucial because many spatial modeling operations require that the relationships between features be specified.

In OOGIS, each object is defined as a separate component, allowing users to create their own model as an extension of the basic object-oriented data model. Topology is built using an OOGIS, so that coincident lines can have references to multiple attributes. For example, if a road was also the edge of a tax parcel, the information about the road and the parcel could be maintained, edited or deleted separately from each other or, if the road was rerouted, the parcel and the road could be updated at the same time.

Versioning or **dataset versioning** provide solutions to common problems associated with multiple data versions, including the need to develop hypothetical or predictive models and the merging of datasets. Data versioning in OOGIS produces a logical copy of a dataset without cloning the entire dataset as is required in the Entity-Relationship (ER) model. Copies of the data can then be used in multiple situations without actually modifying the original dataset. This is accomplished by creating a copy only of the data being modified, not the entire dataset. Versioning in OOGIS greatly facilitates data updates and predictive modeling while at the same time avoiding data redundancy and procedural problems for the user (LaserScan, no date).

There are several scenarios associated with versioning that a user may encounter, the most important of which is known as a long transaction. A **long transaction** occurs when a user wants to make continuous changes to a dataset over a long period of time (e.g. over the course of days or longer) without prohibiting other users from using the existing data. In a georelational data model, this is very difficult to achieve without significant data duplication of large amounts of data. OO technology provides a solution to this problem through the branching and merging of datasets. A **branch dataset** is at the heart of versioning for it is the copy of the dataset that maintains a relationship with the original dataset. The user can edit the branch dataset, though there may be limits set by the dataset creator as to which elements of the dataset can be manipulated. These limits are defined by **data locking**, the procedure by which "database systems can prevent conflicting access to data when multiple users are making requests to the data" (Association for Geographic Information, 1999).

2.2. Data Models

The set of constructs used by the schema is determined by the data model. A data model "provides a collection of constructs for describing and structuring applications in the database" (Worboys, 1999). There are two general categories of spatial data models: vector data models and tessellation data models or, specifically, the raster data model. There are also database models, including the georelational data model and the Entity-Relationship models that are also significant in the development of the object-oriented approach. These models do not cover as broad a spectrum as raster and vector data models, but they are valuable in conceptual terms when attempting to understand OOGIS.

2.2.1. The Vector Data Model

Vector data models represent space as a series of unique geographically referenced entity-defined units of points, lines, and polygons and, in certain instances, pixels (Burrough and McDonnell, 1998). Representations of real world phenomena, such as fire hydrants, streets and buildings that are recorded in an XY coordinate system can be mapped to geometry of points, lines, or polygons. A point is a geographic feature of zero-dimension that can be identified or located with one set of Cartesian coordinates. A point feature could be a utility pole on a city map, or a city on a national map, depending on the scale. A line feature is a one-dimensional feature and implies that there are at least two pairs of Cartesian coordinates (nodes). A line feature has no width, except as denoted in the attributes. A line feature could be an electric utility line on a city map or a road on a national map. A polygon in its simplest form is an enclosed two-dimensional homogeneous area and could represent a building footprint or a state boundary. All

higher order features are constructed from points. Nodes define line segments, and line segments define polygons.

Vector data models are usually referenced using Cartesian coordinates. Vectorized features are static and are unchanging and currently do not contain information about variations in time or space. There are two ways to represent a polygon in a vector system: by recording the coordinates of its boundary or topologically by defining the area in terms of the coordinates of a surrounding polygon.

2.2.2. The Raster Data Model

Tessellation models comprise a second broad category of spatial data models in which the two-dimensional geometric surface is divided into rectangular units, or pixels. The size of the pixel varies depending on the resolution required to represent the spatial variation of an attribute in a given coverage (Burrough and McDonnell, 1998). Raster data models are ideal for applications such as change detection analysis, an analysis of feature change by the change in pixel digital number (DN) value by acquiring data of the same geographic area at different times. While change detection can be accomplished using vector data, raster data is more versatile over a large area.

2.2.3. The Entity-Relationship Data Model

The Entity-Relationship (ER) model is based on a representation of space as a collection of basic objects or entities that have spatial relationships (Chen, 1976). An entity is the data model representation of a unique feature that exists in space. Each entity may have a set of attributes, stored in a separate table. An entity resembles an object found in an OOGIS. However, entities and objects are distinctly different. One of the main differences between an object and an entity is that an entity does not exhibit

encapsulation. An entity does not have behavior associated with it. A relationship is an association that exists between defined entities. All entities or relationships that have similar attributes are grouped together to form an entity set or a relationship set.

There are four major phases to Entity-Relationship modeling: the identification of entities, the identification of relationships between the entities, the identification of entity attributes, and the derivation of tables from the previous three stages (Heywood, *et al.*, 1998). Each entity in a database must have distinct characteristics that are usually described by use of a noun, for example 'farmer', 'farm' and 'mall'. Each entity has a set of characteristics or attributes that is unique, such as an address, known as the unique identifier. The unique identifier makes it possible to distinguish one entity from another. The domain of an entity is the set of possible values for each of the attributes associated with the entity. The relationships between entities are described using verbs. For example, the farmer 'resides on' the farm and the farm 'is next to' the mall.

There are four types of relationships that can exist in an entity relationship model: one-to-one, one-to-many, or many-to-many and many-to-one (Figure 1). An example of a one-to-one relationship is a house located on one parcel of land in a subdivision. In most cases, only one house is permitted, or can be built, on a residential parcel in a housing development. An example of a one-to-many relationship would be a water utility company and its customers. In general, one water company serves many customers and one water main could serve many houses. A many-to-many relationship example would be a roadway system where many different people travel to different destinations using different route selections.

The Entity-Relationship model uses a Relational Database Management Structure (RDBMS) and the Entity-Relationship diagram (Figure 1) helps identify what tables are to be included RDBMS. When there is a one-to-one relationship, tables for each entity can be joined into one table or be kept in separate tables. With a one-to-many relationship, two tables are necessary with a common field that allows a relational join. This is known as the key field or primary key. In a many-to-many relationship, all tables should be kept separate to enable the relational join. If there are duplicate fields, the tables may need to be broken down to avoid data redundancy (Worboys, 1995). After the tables have been selected, the necessary attributes must be determined and the ER model may be constructed with its table definitions containing details of feature attributes, such as name, size and domain (Heywood *et al.*, 1998). After these steps have taken place the database can be implemented.

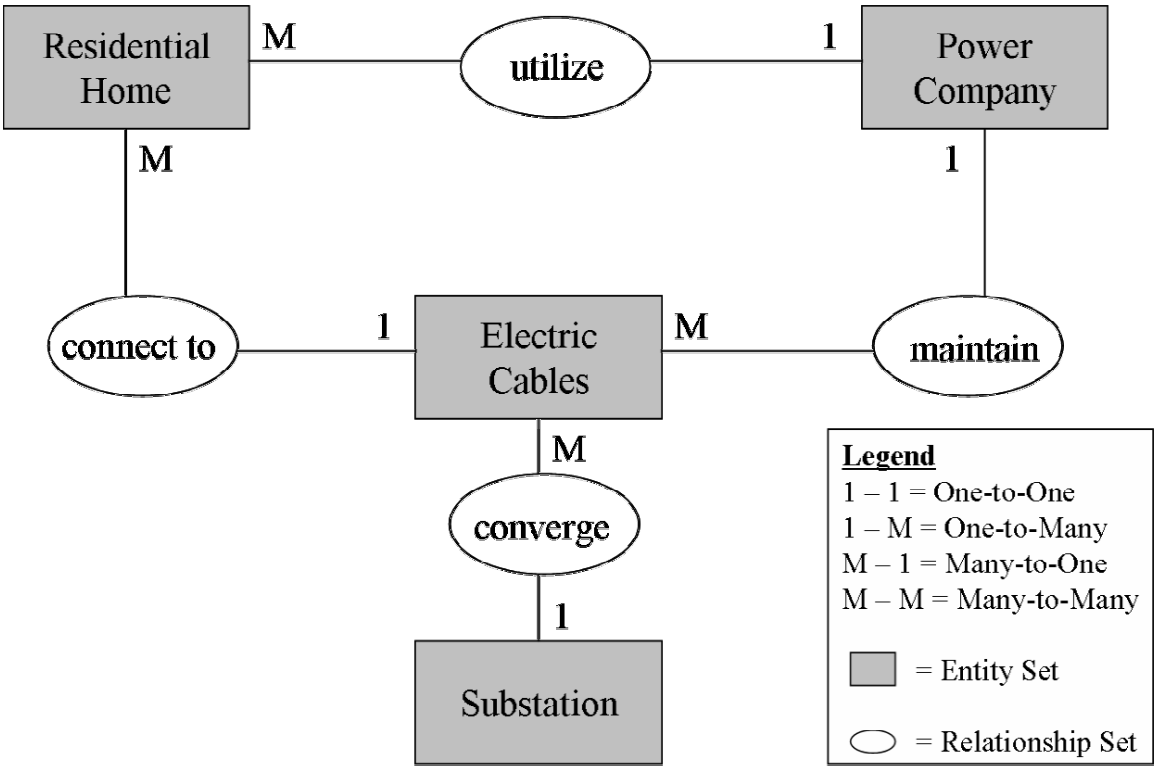

Figure 1. Example of an Entity-Relationship diagram (Chen, 1976)

In Figure 1 for example, the entity set 'residential home' incorporates the relationship set 'utilize' on another object, 'power company'. Connecting lines drawn between the entity and relationship sets designate relationships. Figure 1 displays several types of relationships. Many objects of one type can be related to one type of another object. In this example, many residential homes use one power company to supply their homes with electrical power. The entities 'power company' and 'electric cables' illustrate a one-to-many relationship. One power company maintains many electric cables.

2.2.4. The Georelational Data Model

The georelational model is based on the vector data model, utilizing points, lines and polygons as geometric feature types (Burroughs and McDonnell, 1998). Features are stored as graphical data elements that are referenced to a coordinate system. The geometric data is stored in a binary file. Systems using the georelational model may also divide the entire project into separate files, however the software handles the files as one large seamless map file (Korte, no date). The georelational model stores attributes in relational tables whereby each record has a unique identifier and descriptive information about each feature. The georelational model uses topology. Once topology is built within the model, relationships such as how linear map features are connected and how polygons are bounded, are described. The topological structure of the georelational model allows a GIS to query any spatial feature to determine connectivity, boundaries, and adjacency. In reality, water lines and power lines do not intersect. When using the georelational data model, the storage of different feature types in separate layers prevents

a situation that would be impossible in reality, such as a water main and a power line connecting.

The schema is important in the georelational data model. A relational schema does not include the data itself, but rather describes the structure of a relation, such as its attributes, the attribute domains, and restrictions placed on the data. The relation schema is created when the database is first established and remains static for the duration of time the database is in existence. In reality, a relation is dynamic, changing frequently as data are added, modified or removed from the database. A set of relations and data is known as a relational database and may have restrictions, such as coded values or ranges of values placed on it when created.

Distinct from the database schema, a database is a collection of actual data, which can be shared by many users of a system that has the capacity to define, access, retrieve, manipulate and display the data within it (Worboys, 1995). The evolution of the database structure is crucial to understanding the object-oriented database structure.

2.3. Database Structures

A database structure is a logical method of data organization within a system that is suitable for data storage and manipulation. In particular, spatial databases have unique forms of data structure that facilitate access to spatially referenced data (Worboys, 1995). Several database structures preceded the adoption of an object-oriented database structure.

2.3.1. The Hierarchical Database Structure

The hierarchical database structure was the first database structure to evolve and was based on a parent and child hierarchical structure involving one-to-one or one-to-many relationships. The hierarchical database provides a user with a quick and convenient way to access data. This database structure is most beneficial when used on well-structured data and is mainly implemented in the environmental sciences because it uses a model similar to soil classification systems and other physical data applications. The hierarchical database structure was popular because of the ease of understanding the data structure, record updating, and database expansion. The hierarchical database structure can be particularly beneficial in the organization of data in a mass storage system if all possible queries to be performed on the database are known beforehand. While this is possible with systems such as banking databases or library systems, environmental data and other data types, whose queries are more exploratory, cannot adhere to the fixed hierarchy of this data structure. Many users consider this form of database structure to be too inflexible and not appropriate for the handling of geographic data. In addition, the difficulty in using hierarchical data structures is as much due to the enormous size of the index files, which must also be maintained, as to these other concerns. Certain attribute values must be repeated several times, which causes data redundancy and leads to increased storage space and greater latency costs in accessing the data. Furthermore, search patterns in the hierarchical structure are very structured and rigid and the data is difficult to modify and update.

2.3.2. The Network Database Structure

Network database structures ease some of the rigidity of the hierarchical database structure by allowing the data to pass in other directions than what is already established in the taxonomy of the data structure. One of the main problems resolved by the evolution of the network database structure is that of data redundancy. The network structure is most beneficial when relations between features can be specified prior to the building of the database structure. However, network database structures tend to be very large due to the copious number of pointers that have to be built, maintained and updated every time there is a change in the database.

2.3.3. The Relational Database Management Structure

Relational database management structures (RDBMS) were first established for GIS in the 1980s and have been the database structure of choice in GIS for the majority of commercial GIS. This database structure can take many forms, but the simplest form has neither pointers nor hierarchy. The relational data structure stores data as records or tuples where tuples are unique data records that contain every attribute value (Worboys, 1999). Tuples are grouped in two 2-dimensional tables called relations. Each relation is stored as an independent file. The pointers of the hierarchical database structure are exchanged for the controlled data redundancy via identification codes that identify records in each file by a unique key. The user defines the parameters for a data query in a relational database and the program uses relational algebra to produce new tables for relations that do not already exist (Worboys, 1999).

The relational database structure has certain advantages and disadvantages. The structure of the relational database is very flexible and can handle the demands of most

unscripted queries using Boolean logic and other mathematical operations. Adding and deleting data in the database is also easier with this structure because the user can add or remove tuples to the RDBMS. Entire tables may also be added or removed. A user can query several relational tables at once by joining tables based on common fields. This is especially important in cases where all the records in a table have the same number of attributes, but there is no natural hierarchy (Worboys, 1999). Unfortunately, if the relationships between the tables are complex and many joins are needed, the operation may take a substantial amount of computing time. Relational database structures have to be well designed in order to perform database searches while controlling the demands for substantial amounts of computing power.

2.3.4. Object-Oriented Database Structure

The hierarchical, network, and relational database structures were fundamental to the development of the object-oriented database structure. Although originally object-orientation started in the field of computer science, and specifically in programming, it was subsequently applied to databases and then to GIS (Environmental Systems Research Institute, 1999). An improved method of data handling in a database was necessary because of the problems of data redundancy and the computationally intensive search methods of previous database structures. In GIS, this was particularly so due to the complex spatial entities and the inability of previous database structures to extent beyond simple points, lines and polygons and focus more on what is being represented in the real world (Burrough and McDonnell, 1998). The object-oriented database structure recognizes that there are not only relationships between objects, but objects have behavior that was unaccounted for in previous structures. Object-oriented database

structures combine the speed of the hierarchical approach with the flexibility of the relational approach by organizing the data with an emphasis on the entity, as opposed to the functions being processed (Burrough and McDonnell, 1998). In the object-oriented database structure, data are characterized in terms of a set of unique objects that are organized into groups of similar objects (object classes) and the relationships between the objects and classes are made using explicit links.

The data in this database structure are encapsulated within objects or features that are defined by a unique identifier within the database (Worboys, 1999). One of the main aspects of the object-oriented database structure is the emphasis on the uniqueness of objects. The unique identifier does not change regardless of what changes are made to any attribute values associated with the object. However, if the feature is split into other features or combined with other features, the unique identifier is modified to compensate for the change in spatial information. The classes and instances are connected by pointers, which define the different relationships and hierarchy of the relationship structure. Where hierarchies are formed, different states and methods can be passed down to other objects through what is known as inheritance. Inheritance makes the characterization of object attributes and the retrieval of objects from the database more efficient. All data in an object-oriented database structure are specified once only and can be retrieved rapidly. Data in an object-oriented database structure can only be queried or modified if a request to carry out an operation, called a message, is sent to the object. The object response is dependent on its state. The polymorphic capability of the same message to be sent to different objects is particularly valuable.

2.4. Object-Oriented GIS Software

There are several commercially available GIS software packages that incorporate object-oriented technology. These packages include CARIS GIS software, LaserScan's Gothic LAMPS2, General Electric's Smallworld and ESRI's ArcGIS 8.x. The software used in this research is the ESRI software, ArcGIS.

2.4.1. CARIS GIS

CARIS GIS software was developed from research into computer-aided cartography more than twenty years ago and was used in land-use and natural resource management (CARIS, 2002). Currently, CARIS is a fully functional LIS/GIS software suite and comprises a combination of tools designed for data capture, editing, updating, manipulating and displaying geographic data. CARIS also has the capability to edit, query, analyze and visualize spatial data. Some highlights of the software include the ability to handle 2-, 3-, and n-dimensional data, support many database formats, easy customization of the working environment, and rigorous topological structure. With CARIS, full network and polygon topology is supported (CARIS, 2002).

2.4.2. LaserScan LAMPS2

LAMPS2 is a map production system based on an object-oriented spatial database (LaserScan, 2001). LAMPS2 demonstrates many of the advantages of object-orientation including powerful versioning and data locking capabilities, allowing multi-user update without restricting data access. Inheritance and behavioral methods are essential to LAMPS2 (LaserScan, 2001). Data validation is supported in that objects can verify modifications automatically using a user-defined rule base. Data structure and topology is created and constantly maintained during data capture and updates.

2.4.3. General Electric (GE) Smallworld Core

General Electric's Smallworld Core is database-driven and object-oriented in nature (General Electric, no date). Smallworld uses an object-oriented programming language for development, known as Magik. Some of the advantages of Smallworld Core are that it is scalable, has a tiered architecture, has the ability to handle versioning and long transactions, and it has virtually seamless data access with the ability to integrate many data formats (General Electric, no date).

2.4.4. Environmental Systems Research Institute (ESRI) ArcGIS

ESRI's ArcGIS uses the object-relational geodatabase. The system continues to support legacy Entity-Relationship models and the georelational model. While this software package has many new features compared to its previous releases, the focus of this study was on the ArcGIS object data model. Until the release of ArcGIS, ESRI's ArcInfo focused on the georelational data model in which the geometry, attribution and topology were stored in binary files and attributes in a relational database management system (Environmental Systems Research Institute, 1999). Using object-oriented technology, geodatabase has the ability to combine the properties of objects with their behaviors (Environmental Systems Research Institute, 1999).

Geodatabase is a key component to ESRI's ArcGIS. There are several different types of features that can be represented in geodatabase. Some of these include geographic features (objects with location) and network features (objects geometrically integrated with other objects). The object data model enables the user to define one or many relationships between objects and to implement rules that maintain the reference integrity between objects. Geodatabase is based on a schema created with ArcCatalog

and has the ability to establish a data rule base. Data may then be imported into the geodatabase under the guidelines set by the schema to help prevent the production of erroneous geographic data.

The ESRI ArcGIS software was used in my case studies. While the other software packages discussed earlier are capable of completing the tasks investigated in the case studies, ArcGIS is the most widely available software. The following chapters give a background to the case studies and illustrate how OOGIS is used in each study.

2.4.5. A Comparison of LaserScan LAMPS2 and ESRI ArcGIS

LAMPS2 is a true OOGIS, implementing OO data modeling using inheritance and methods, so that map features (such as forest, orchards, roads, and rivers) become objects with behaviors, providing versatile mapping and data analysis. It is a GIS that is supported by an object-oriented spatial database, called Gothic. LAMPS2 uses validation methods that enforce database integrity, such that objects can validate themselves whenever they are modified using user defined rules. The use of methods, data locking and versioning provided by LAMPS2 reduces the chance of erroneous data entering the system. The database structure and topology is created and constantly maintained during procedures such as data capture and update. LAMPS2 has the capability of integrating raster and vector data. All of the Gothic products share a common object database that allows efficient and transparent deployment of LAMPS2 data to web browsers and to desktop viewers (LaserScan, 2001).

LAMPS2 separates the tasks of data compilation and update from those of presentation and product generation, allowing the generation of multiple products while minimizing capture and update costs. Some of the benefits of LaserScan's LAMPS2 are

the reduction of cost by eliminating duplication in the parallel update of product ranges, the production of new products to meet user demands direct from a centralized database, the production of rich data products involving structure and relationships to fit modern standards, an increase in productivity by using production tools built just for a specific task (not CAD or GIS) and movement away from constraints of sheet lines and press delays by supplying on-demand mapping.

LAMPS2 boasts the advantage of truly continuous mapping with no sheet edges, including multi-user independent update without locking out other users. Also, LAMPS2 is supported by a versioned database with an integral rollback and recovery mechanism that efficiently handles database changes through time. The database only stores changes from one database version to the next, which makes editing easy to manage.

The fundamental difference between LAMPS2 and ArcGIS is that ArcGIS is not truly object-oriented, whereas LAMPS2 is a true OOGIS. ArcGIS is an object-relational GIS. The immediate predecessor to ArcGIS, ArcInfo, was based on the georelational model. While the current version of ArcGIS still fully supports the georelational model, it also supports the object-oriented data model for the production of "smart" databases. This object relational system is known as the geodatabase. "Smart" means that the geodatabase can coalesce the properties of objects with their behavior into one "smart" object. Rather than developing an entirely new data model, ESRI's geodatabase is an extension to the standard relational database technology. Because of this, the geodatabase is not a true OOGIS, but an "object-relational" system. The geodatabase allows the user to add behavior, properties, a rule base and relationships to the feature data. The geodatabase is flexible, allowing user defined features for meeting specialized

requirements. For example, if a water company were interested in pinpointing water main breaks, they can use the geodatabase with certain rules or behavior set up to determine the break in the water line network using information such as houses without water and intersections of water lines with water mains. The geodatabase has the capability of handling topologically integrated feature classes, similar to the coverage model in ArcInfo. However, ESRI has expanded the geodatabase model by making it accommodate support for complex networks and relationships between multiple feature classes.

Chapter 3. The NIMA Project

3.1. Background

Several years ago, the National Imagery and Mapping Agency (NIMA) embarked on a program to convert from paper map production to digital production. As a part of this transition, NIMA sponsored a project by the West Virginia University Department of Geology and Geography to develop an innovative approach to automated feature extraction (AFE) from satellite imagery (Desai *et al*., 1999). The result was the creation of an object-oriented database of features based on commercial satellite imagery products.

The NIMA project had three main components: 1) the automated and semi-automated recognition and mapping of roads, rivers and forests from satellite images and Digital Terrain Elevation Data (DTED), 2) the automated conversion of raster features to vector features, and 3) the development of methods within an object-oriented GIS to automate the import, topological structuring, attribution, and export of Foundation Feature Data (FFD) (Desai *et al*., 1999). There were two case study sites for which data was provided: Santiago, Cuba and Camp Lejeune in the United States. Line features comprising roads and rivers were extracted from the satellite imagery using two different techniques. Roads were extracted through a procedure that began with the application of an edge detection algorithm, applied to satellite imagery. The edge image was used to develop a friction surface, from which optimal routes were generated between end points of the road system delineated by the user. The width of the road was determined automatically based on a cross-sectional spectral profile, perpendicular to the road (Desai *et al*., 1999). River features were identified and extracted using both satellite imagery

and the DTED. Rivers that were recognizable in the imagery were mapped directly. Rivers that could not be identified in the satellite imagery were mapped from the DTED, focusing on areas of relatively steep topography. However, if a river was too small to be identified from the satellite imagery, and the topography was to gentle to use the DTED, the river was mapped based on the identification of pixel with the spectral attributes of the river and associated vegetation, which were then connected with automated route selection methods (Desai *et al*., 1999).

The second main component of the NIMA project focused the conversion of raster feature data to a vector format. The conversion process was completed using an automated raster-to-vector conversion software package. This research component had several parts including:

- The ability to input and convert raster imagery of varying spectral and spatial resolution to vector format
- The ability to generate line datasets for road and river features
- The capture of raster cell values as attributes of the vector output
- The functionality to allow spline smoothing of the vector linework in batch mode
- The filtering and thinning of vector linework in batch mode
- The capability of raster gap-jumping with designated snap tolerance levels
- The capability to import and export in common data formats
- The automation of the entire system (Desai *et al*., 1999)

The algorithms used in the raster-to-vector conversion process assisted in recognizing limitations in the vector feature data, given the need to meet Foundation Feature Data (FFD) standards (Desai *et al*., 1999). FFD are a set of spatial data features that are captured to form a baseline set of features for a project.

The third component of the NIMA project, and the part of interest for my work, focuses on the use of object-oriented technology to integrate vector features using the Foundation Feature Data rule base. The FFD rule base contains attribution and accuracy

requirements as well as portrayal criteria (Desai *et al*., 1999). The NIMA sponsored research initially used LAMPS2 and Gothic DBMS by LaserScan. Following ESRI's release of their object-oriented software, ArcGIS geodatabase, the remainder of the project was completed using the geodatabase. ArcGIS was used because of the availability of the software and my previous experience with the software.

3.2. Development of the Schema

The schema for the NIMA project was originally developed using LaserScan's LAMPS2 OOGIS. During the first year of the project, it was discovered that schema mapping between LAMPS2 and ESRI software was possible. When the transition from LaserScan to ESRI occurred in the project, the schema remained intact. For this thesis, the schema development of the roads and rivers data layers was emphasized. A portion of this case study was focused on the development of the geodatabase for the encapsulation of the features used in the project. The layers imported into the geodatabase comprised the roads, rivers, and woodlands layers. The woodlands and orchards features were not developed further and emphasis was placed on developing the geodatabase for the roads and rivers layers.

Three feature classes for roads, rivers and woodlands were defined for the Santiago, Cuba dataset, shown in Figure 2. Each feature class is composed of one or more feature datasets that share common physical characteristics. For example, a feature class can be created called "water," which can then contain the feature datasets "rivers," "lakes" and "estuaries". For each feature dataset, attribute data can be assigned. For example, attribute data can include road surface type or hydrological category for a river.

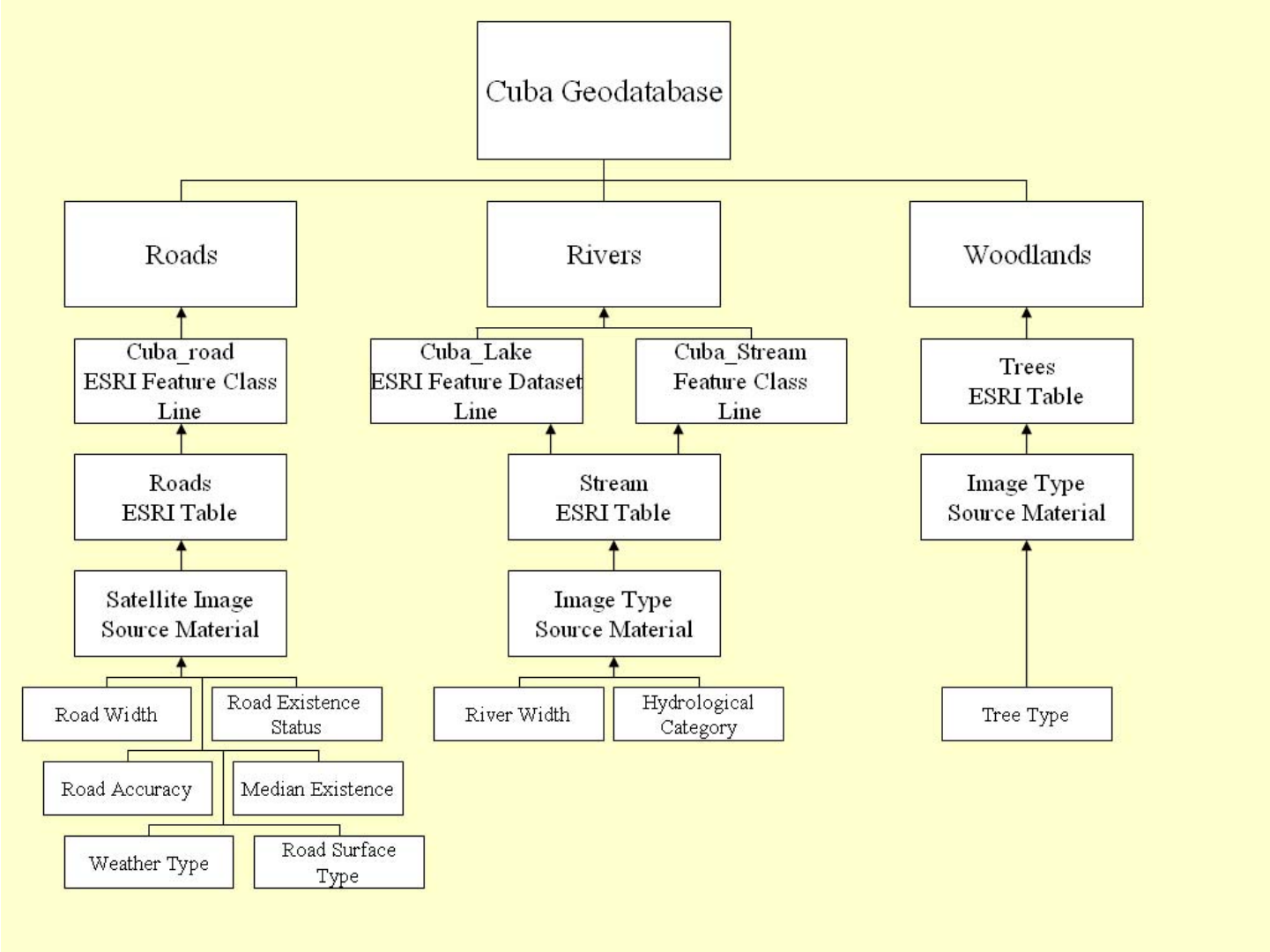

Figure 2. The Santiago, Cuba Geodatabase Schema

Domain attributes were developed for the object classes. Domain attributes consist of two data types—the range of values and coded values. A range of values comprise a minimum and maximum range of valid values established by the creator of the geodatabase. When a user enters an attribute value into a table, the method associated with the object reflexively checks the entry against the range of defined values. If the entry is out of the value range, the user is prompted to enter a valid value. The second type of domain attribute is the coded value. This is a numeric entry that represents a feature attribute. For example, a road surface type may consist of three types—paved, gravel, and dirt that can be represented with the numbers 1, 2 and 3. If another value other than those is entered into the system, the user will be prompted to enter a valid value. After the first year of the NIMA project, the data validation method for the road and river features had already been developed and provided an effective data entry validation process. Data validation in the geodatabase is in the form of domain attributes that can self-check entries in an attribute table, to ensure data accuracy. A range of values was set for the widths of roads and rivers. Coded values were established for attributes such as road surface type, with a value for "unknown" as well.

The development of the schema also illustrates the properties of inheritance, polymorphism and encapsulation. Once the creator of a geodatabase has established the schema, it can be locked so that users of the GIS cannot modify it. The interface can also be customized so that users only see what the creator of the geodatabase wants them to see. Also, as data is imported into the geodatabase, these datasets inherit the domain properties already established as it is imported into the geodatabase. For example, if the range of values were already set in the "rivers" data layer, then as a user exports a subset

of that dataset, the new data layer will inherit the behavior of the parent dataset. The geodatabase can also receive commands without being able to execute them as a stand-alone system, but can be linked to other software or hardware capable of executing the request. For example, if the user wants to calculate the area of the polygon data layers, the geodatabase does not understand the command, but the external system, in this case ArcMap, can check each dataset to look for polygon data layers and calculate the areas of each of the polygons within the polygon data layers.

3.3. Development of the Geodatabase

The schema contained the feature classes, attributes and methods appropriate for the FFD layers (Desai *et al*., 1999). The feature classes and attributes were created during the import of the vector data following the raster-to-vector conversion. Behavior methods were originally programmed in the LaserScan Lull programming language, but were later converted to Microsoft Visual Basic to be used with the ESRI software products. Reflex methods are executed automatically and determine the behavior of the feature classes, and can perform a number of tasks such as building topology and data import verification.

The ESRI geodatabase is feature-based, though not truly object-oriented. Rather it is object-relational. An object-relational model transforms geographical data between object and relational models and between the systems that support those approaches. It is capable of handling objects and entities. Object-relational is a transition step between entity-relationship and true object-oriented modeling. All of the object classes are contained within the geodatabase. The geodatabase can be treated as an all encapsulating

object, containing both spatial and non-spatial information. Geodatabase also supports polymorphism, in that only appropriate components produce a response to a request. For example, if the command "create centroid layer" was sent to geodatabase to operate on a polygon layer, the resulting layer would be a point layer. A point or line feature class would not complete the command and a centroid layer would not be created by either of these feature types. Likewise, the resulting layer must be a point layer, so the OOGIS would not produce a line or polygon layer.

Figure 3 illustrates samples of feature datasets, feature classes, tables and relationship classes. A feature dataset as defined by ESRI is "a collection of feature classes that share the same spatial reference" (Environmental Systems Research Institute, no date). Feature classes may also be stored outside a feature dataset, but these are referred to as standalone feature classes. To ensure a common spatial reference, however, feature classes must be contained within a feature dataset if they are to store topology (Environmental Systems Research Institute, no date). Feature classes "store geographic features represented as points, lines or polygons, and their attributes; they can also store annotation and dimensions" (Environmental Systems Research Institute, no date). Data may be converted to the geodatabase coordinate system upon import or the data may be converted before being imported into the geodatabase. Tables imported into a geodatabase may contain additional information, such as attributes or geographic information for a feature class. The tables stored in the geodatabase can contain vital information relevant to the feature datasets and classes, but not necessarily a spatial reference that can be displayed in a GIS.

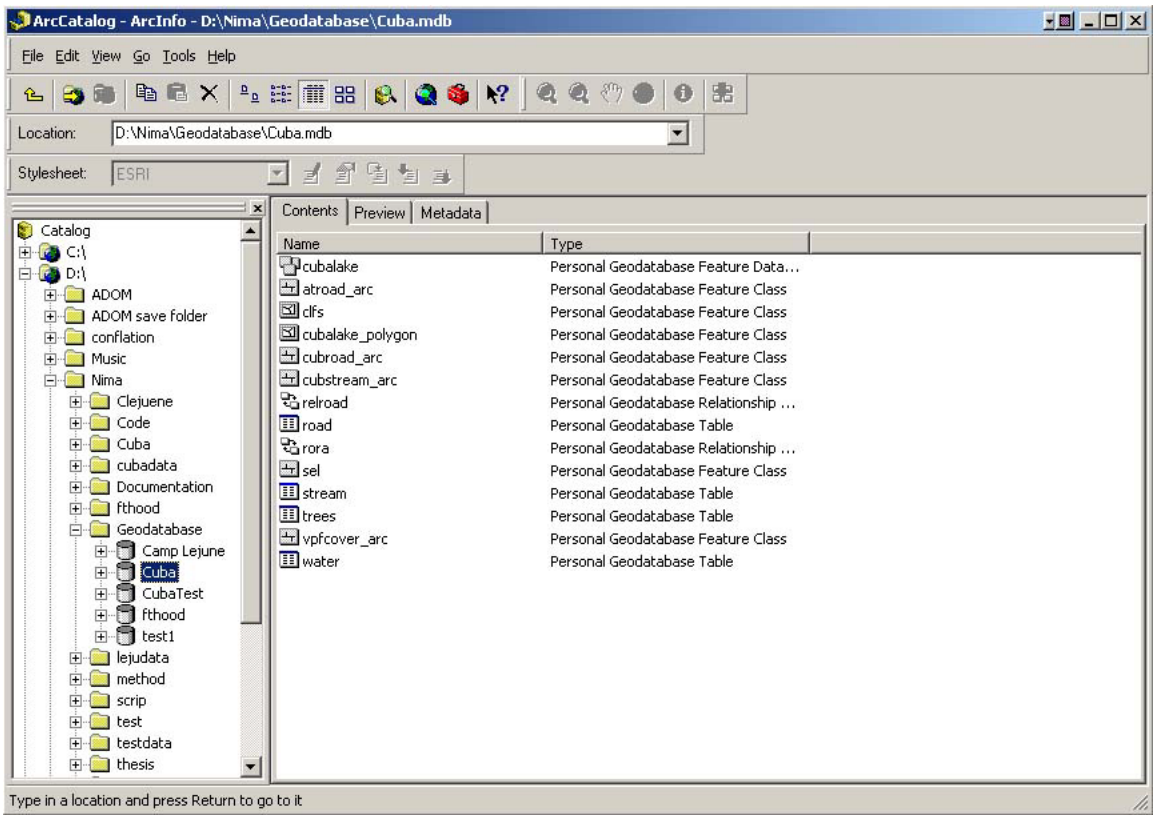

Figure 3. ESRI's ArcCatalog displaying the contents of a personal geodatabase

Relationship classes within the geodatabase reflect the relationship of objects in a real world system. As ESRI states, "a relationship can exist between spatial objects (features in feature classes), non-spatial objects (rows in a table), or spatial and non-spatial objects" (Environmental Systems Research Institute, no date). These relationships between objects are stored in the relationship classes. Unlike previous data models, such as the entity-relationship model, the object-relational data model allows for all objects, both spatial and non-spatial, to be related. Thus, the GIS organizes the geographic information more intuitively, including all information relevant to the area of interest.

3.4. Embedding Behavior and Methods

Methods are behaviors associated with an object and are an important aspect of OO technology. Three methods were developed in the NIMA project 1) the coverage to

geodatabase method, 2) the build and clean method, and 3) the domain validation method. The methods used in the NIMA project were developed in Microsoft Visual Basic 6.0 and a graphic user interface (GUI) was created. Some of the code was already developed by other research assistants working on this project (see Appendices A and B), some of the code was taken from sample code and I developed some of the code. My main contributions were to integrate the code into ArcGIS 8 and I discovered a sample code that would convert a coverage to a geodatabase feature dataset. The methods were designed to perform different functions, with the intention that they would evolve from a stand-alone program, to a call button in ArcGIS that would implement the method. Reflex methods differ from other methods because they are triggered automatically without the need for user input. While the coverage to geodatabase method and the build and clean method were not built into ArcGIS, the stand-alone applications illustrate the object-oriented concepts. The domain validation method was developed in ArcGIS and was implemented such that as data was entered into the attribute table of the object, errors were caught before such data modifications were accepted into the GIS. The Visual Basic source code for each of the methods can be viewed in Appendices A and B.

3.4.1. The Coverage Conversion Method

There were three parts to the coverage conversion method—the coverage to geodatabase method, the coverage to ESRI export or interchange file (.e00) method, and the coverage to ESRI shape file format (.shp) method. The coverage to geodatabase method was designed to save spatial data in a variety of formats. The method enables a coverage to be imported into the geodatabase and then be converted to interchange files

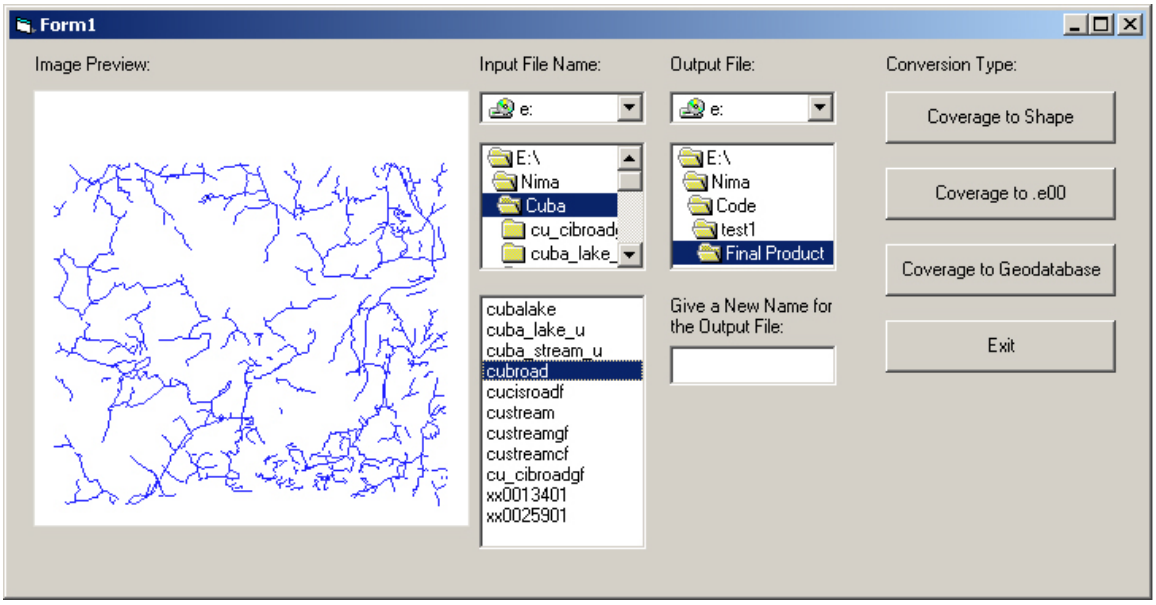

Figure 4. Custom interface to convert coverages to other file formats

and shape files (Appendix A). The stand-alone program interface is shown in Figure 4. When converting to a shape file, the user must not only enter an input and output file name, but must also specify whether the file to be converted consists of nodes, arcs, polygons or other features. A pop-up window prompts the user for this information (Figure 5).

Once a user enters the type of feature class to export from the coverage to the shape file, the conversion program then executes and creates either a point, line or polygon shape file in the directory and folder specified. If the user opts for the coverage to interchange conversion, the user simply specifies the input and output paths and file names and clicks the "coverage to .e00" button.

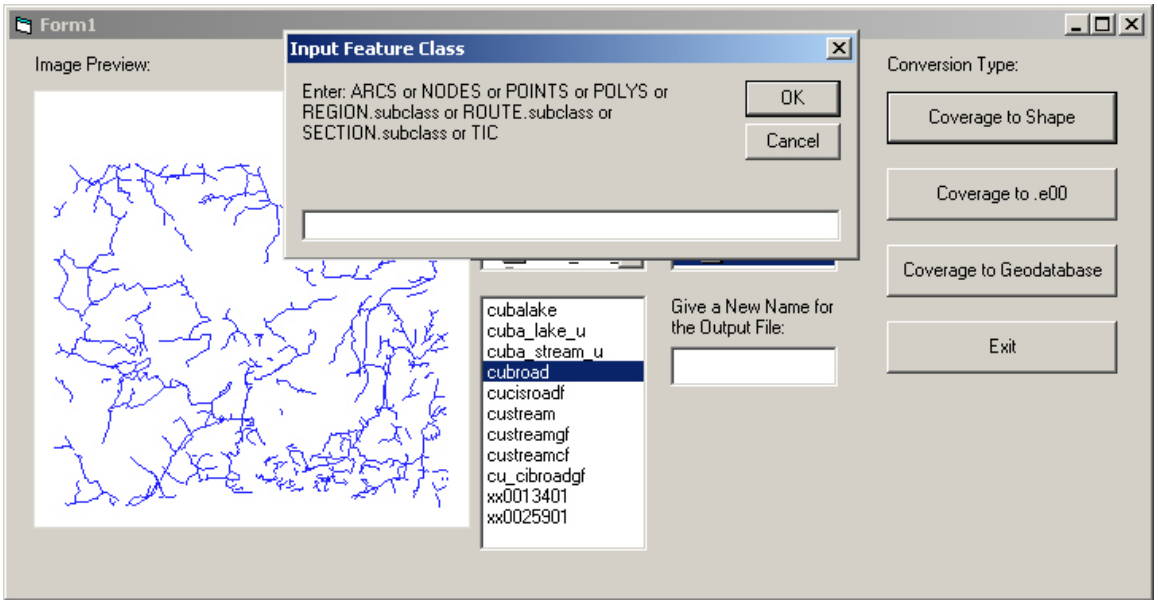

Figure 5. Pop-up window associated with coverage to shape file conversion

The coverage to geodatabase function is different from the other two options. Rather than creating a new file, it creates the geodatabase as well as producing a feature dataset within the geodatabase that contains the original coverage information. After the user selects the input and output paths and filenames, a pop-up window appears asking for the name of the geodatabase, as illustrated in Figure 6. Once the name of the new geodatabase is entered, the application then creates the geodatabase as well as the feature dataset.

If the coverage conversion method were developed as a reflexive method, each of these files would be created at the end of each ESRI ArcMap session and would be overwritten each time modifications were made so that versioning issues would not arise. Long transactions are capable using the coverage conversion method. A copy of a coverage could be created for modification while the functioning geodatabase remains intact and available for public use, so that when modifications are completed on part, or all, of the components of a geodatabase, the copy can then replace the existing geodatabase.

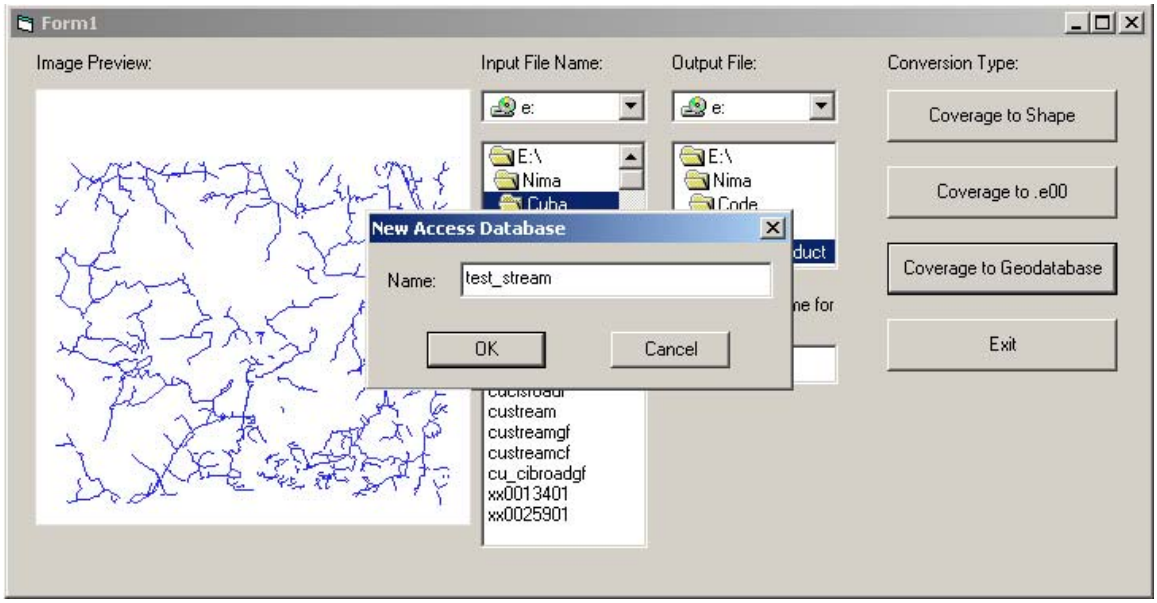

Figure 6. Coverage to geodatabase pop-up window

A further important aspect of embedding objects with their behavior is that behaviors can be inherited. As the coverage is converted to a geodatabase feature, it passes on its behaviors and attributes to the geodatabase domain properties. When a new coverage or other geographic data file is created within the geodatabase, it automatically takes on certain behaviors and attributes of the parent geodatabase. It is said to 'inherit' these behaviors and attributes.

3.4.2. The Clean and Build Method

The second behavior method relates to the clean and build method. This method also makes use of a GUI, which prompts the user to clean and build a specified layer and export it to a .vpf file or vector product format. The clean and build method has a similar GUI to the coverage to geodatabase method. Using this application, a user can display the geographic data to be cleaned and built (Figure 7).

The clean and build method exemplifies the concept of polymorphism in this case study. The 'clean' portion of the method performs operations such as connecting dangling nodes, removing sliver polygons and other similar error-reducing processes,

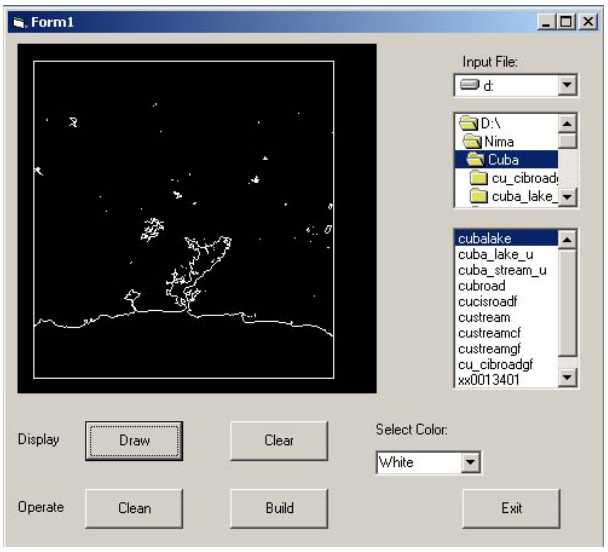

Figure 7. The clean and build method graphic user interface

based on a specific tolerance (Appendix B). The command is sent by the user and the function searches for a suitable 'target'. In this case, the target is a feature class. The target feature class does not have to understand what 'clean' is; only that it is applied to either a line or polygon feature class. If the coverage selected is found to contain errors, the coverage is 'cleaned'. For example, if a polygon data layer is going to be built from a line data layer, first the layer is cleaned to remove dangling lines or lines that do not end at an intersection with another line. Then, the polygon layer is built, creating areas from the line data layer.

3.4.3. The Domain Validation Method

The third method example is the domain validation method. This method automatically checks the validity of any new value entered into the system. An object within a feature class or table with valid values for all of its attributes is considered a

valid object. If one or more attributes of the object in a feature class or table is not valid then it is considered an invalid object. The object itself is not removed from the dataset if it is considered invalid, however the attribute modification process is halted until invalid values are corrected.

In ArcGIS, there are four types of validation rules. These are attribute domain rules, connectivity rules, relationship rules, and custom rules. Attribute domain rules are the primary focus of the domain validation method. Attribute domains are utilized to restrict values permitted in a particular attribute entry for a table, feature class or subtype. Every feature class or table can have a set of attribute domains that apply to the attributes. The attribute domains do not have to be exclusive to the feature class or table. Within geodatabase, attribute domains can be shared between feature classes and tables. One disadvantage of the attribute domains is that they do not have the ability to disallow null values. If a table or feature class is created within the geodatabase, the specific fields can be set to not accept null values. However, the acceptance of null values cannot be set universally, as with attribute domains.

As explained in section 3.2, there are two field types permitted within the attribute domains. A field may be set up to accept a range of values, with the user delimiting the minimum and maximum values. The other option is to have a set of coded values, with the user entering the valid values and a definition of what the value represents. The value range method is useful when the user is looking for actual numerical or value input. For example, in the NIMA project, the width of a road must be entered. Also in the NIMA project, the user must enter the type of surface on the roads. If the road is unimproved, the user enters a “1”. If the road is paved, the user enters a “2”. If the road surface type

is unknown, the user enters "99". After the value is entered, the method automatically checks the value in the cell to see if it is "valid". If the value is not within the range of values that is "valid" in the field, the method stops the data entry and informs the user that the value most recently entered is not valid and the system will not continue until a valid value is entered.

3.5. Review of Object-Oriented Concepts in the NIMA Project

As demonstrated in the NIMA project, the geodatabase displays many of the fundamental components of object-orientation in GIS. The geodatabase is feature-based and has a schema that illustrates all of the data layers and their attributes and what types of data are valid for each attribute. The geodatabase combines all spatial and non-spatial data for a project into one file. Also, the user may define feature classes within the geodatabase into which datasets with similar characteristics may be imported. Both of these examples illustrate encapsulation. The use of methods exemplifies the strength of polymorphism. For each command to execute, a certain function is enacted depending on which type of data, point, line or polygon, receives the command. Finally, the third major component, inheritance, is exemplified through the application of parent characteristics to the newly created child datasets within the geodatabase.

Versioning and data locking is also embodied through the use of methods. The coverage conversion method creates several types of data layers. The original geodatabase layers remain intact and available to others to use while the layers created from the coverage conversion method can be edited and then imported into the

geodatabase once the edits are complete. Along with data locking, long transactions may be completed on the newly created layers without hindering access to the geodatabase.

Chapter 4. The FGDC Project

4.1. Background

The second case study used to exemplify OOGIS concepts focuses on a project sponsored by the Federal Geographic Data Committee (FGDC), concerning "National and Regional-Level Area Integrator Concepts Using Multi-Scale, Feature-Based Digital Transportation Data in West Virginia" (National Spatial Data Infrastructure, 1996). This project sought to use OOGIS to implement the feature-based DLG-F model in support of the National Spatial Data Infrastructure (NSDI) initiative. The FGDC project was undertaken as part of a National Spatial Data Infrastructure (NSDI) Competitive Cooperative Agreements Program that sought to demonstrate how cooperating agencies could support a national framework database concept by drawing upon data generated by local data producers. The specific feature-based transportation project is a cooperative effort between the West Virginia State GIS Technical Center, the Department of Geology and Geography of West Virginia University, The West Virginia State GIS Coordinator, The West Virginia Department of Transportation (WVDOT) Division of Research and Planning, and the United States Geological Survey (USGS) National Mapping Division (NMD). Initially, LaserScan OOGIS software was the preferred platform, though the project has subsequently adopted ESRI's geodatabase.

Importantly for the purpose of this project, the Framework concept focuses on the development of a National Spatial Data Infrastructure (NSDI) that "encompasses policies, standards and procedures for organizations to cooperatively produce and share geographic data" (Federal Geographic Data Committee, 1997). It was in this context that the West Virginia University (WVU) team sought to build on the capabilities of an

OOGIS approach to the national database concept. There are 17 federal agencies cooperating to create the NSDI, as well as participating organizations from state, local and tribal governments, and participants from the private sector and the academic community. Executive Order 12906 established the NSDI whose role was to encourage the development of "the technologies, policies and people necessary to promote sharing of geospatial data throughout all levels of government, the private and non-profit sectors, and the academic community" (Federal Geographic Data Committee, 1997). The NSDI sought to provide a structure for the practices and relationships between data producers and users that will promote data access, sharing and use in a comprehensive manner to enable decision makers to choose the 'best' available data (Federal Geographic Data Committee, 1997). The new procedures were to be accomplished through the following set of goals (Federal Geographic Data Committee, 1997):

- Provide a consistent means for sharing geographic data among all users.
- Produce significant savings in the cost of data collection.
- Enhance decision making.
- Reduce duplication of effort among agencies.
- Improve data quality.
- Reduce the cost related to geographic information.
- Facilitate accessibility of geographic data to the public.
- Increase benefits arising from using available geospatial data.
- Establish key partnerships with states, counties, cities, tribal nations, academia, and the private sector to increase data availability.

The main issue addressed by the NSDI was the development of a national database capable of handling data at multiple scales, accepting data from various producers, reducing data redundancy and duplication and allowing updates to the database while making the data continuously accessible to other users.

The research proposed by WVU was to address these issues through the use of OOGIS and to develop a technical process to enable multiple local data producers to contribute data to a regional database and enable update and edit capability without locking the entire database from users. Specifically, the study focused on transportation data generated by various regional and local data producers. OOGIS was explored as a way of enabling data producers to contribute to a national framework database. The FGDC project sought to produce a data environment and structure supportive of framework concepts that included:

- Multi-scale geometry in an OOGIS
- Feature-based unique identifier tracking (OOGIS)
- Transaction updates from contributors (versioning)
- Internet-based data dissemination (NSDI, 1996)

OOGIS was proposed as the GIS and data model of choice.

A supplementary proposal in 1999 titled, "Research and Design of Prototype Geospatial Data Transport and Update Capabilities in a Distributed Database Environment" provided additional focus on transactional updates and the creation of a distributed, multi-partner data maintenance process.

The FGDC case study illustrates the three fundamental principles of OOGIS, comprising polymorphism, inheritance and encapsulation, and yet also explores several other OO concepts, including versioning, long transactions and data locking as well as conflation and the use of unique identifiers. It is in the pursuit of OOGIS support of a framework national database and especially versioning that I focus on in this case study.

Versioning is a procedure that produces a logical copy of the data without cloning the entire dataset. The logical copies of the data can then be used in multiple situations

without actually modifying the original dataset in each instance (LaserScan, Inc., no date). Developing a central database that can support multiple users performing transaction updates is clearly an attractive proposition not least because of reduced data duplication and cost.

A long transaction in versioning enables a user to make changes to a dataset over a long period of time (e.g. days or longer) without restricting use of the original dataset to other users. Data locking is a mechanism by which database systems can prevent conflicting access to data when multiple users make requests to the data (Association for Geographic Information, 1999). Versioning and data locking enables multiple transaction updates and long transactions to be performed. When a user performs an operation on a dataset, a logical copy of the data being manipulated is made. The user data operations are then performed on the logical copy, leaving the original dataset intact so that others may use the same dataset. If a user wishes to manipulate a dataset over a long period of time, these long transactions are performed on the logical copy using data locking without affecting the original dataset. Versioning essentially sets a lock on the data while a long transaction is taking place and prevents conflicting access to the data through the generation of logical copies and leaving the original data intact until final data changes are sent to the regional database.

4.2. Digital Line Graph Data Models

The FGDC case study then was to develop a conceptual data model based on OOGIS and to focus this specifically on feature-based DLG-F transportation data model. The evolution of the DLG-F feature-based data model relies on the foundation built on

earlier models comprising the DLG-3 and DLG-E. The transition from DLG-3 to DLG-E and subsequently to DLG-F represents a move from vector data models to entity data models and subsequently to object data models. Since OOGIS uses feature-based data it was an obvious approach to apply to the development of a regional database for the NSDI.

4.2.1. The DLG-3 and DLG-E Models

The term "DLG-3" stands for Digital Line Graph, level 3. It is the term used for spatial data stored in vector form. There are eleven total record types in DLG-3 and these record types fall into the two broad categories: header records and data records. Record type is significant here in that data records conform to the characteristics of a vector data model:

- Node and area identification records—Contains qualitative and quantitative data that describes nodes and areas
- Node-to-line linkage records—Contains line segment internal identification numbers that are unique identifiers
- Area-to-line linkage records—Contain line segment internal identification numbers
- Line identification records—Contains quantitative data regarding lines, as well as internal identification numbers that associate each line with a starting node, ending node, left area and right area
- Coordinate string records—Contains coordinate points associated with each line in the dataset
- Attribute code records—Contain major-minor code attribute pairs that describe each line in the DLG-3 dataset
- Text records—Contain a descriptive text of the dataset

The DLG-3 model was entity based and the relative position of data in the data files is very important. Related node-to-line linkage records and attribute code records must follow each node identification record. Similarly, each area identification record must be followed by area-to-line linkage records and attribute code records and also line identification records must by followed by coordinate string records and attribute code records. Each of the identification records has associations with a record type. Failure to

create and maintain this order can result in data and file corruption. The DLG-3 model is dependent on its attribute table that is maintained separately from the spatial geometry. Overall, the model closely resembles the hierarchical database structure.

The Digital Line Graph – Enhanced data model was developed specifically for two-dimensional spatially referenced topologically structured vector data possessing features, attributes and relationships and reflects a feature-based perspective of the topologically structured vector-based data model.

Entities in the DLG-E model represent the digital characteristics of an entity and relationships provide the link between objects. For example, relationships can exist between elements that have topology, such as feature objects and spatial objects, and feature objects to other feature objects. Relationships can only involve two participants at a time using the DLG-E model. The DLG-E model closely represents the entity-relationship model, although the DLG-E model refers to data as "objects", they are not true objects in the sense used by OOGIS. The DLG-E model is a feature-based approach to data modeling that does not possess the fundamental qualities of the OOGIS data model such as encapsulation and inheritance. The DLG-E model separates the spatial information or geometry layer from the attribute table and the relationship classes, as in the Entity-Relationship model. Features in the DLG-E model are not truly "encapsulated" and inheritance is not possible in the DLG-E model. Since relationships are stored separate from the rest of the spatial and non-spatial information, the relationship information cannot be passed to a child class when it is created. Furthermore, there is no behavior associated with the DLG-E data model and the parent

data layers cannot pass along behavioral qualities to the child data layers created under them.

4.2.2. The DLG-F Model

In contrast to the DLG-3 and DLG-E models, the Digital Line Graph – Feature more closely reflects the OO data model. The model is feature-based, but emphasizes objects, rather than entities and their relationships. Objects created using the DLG-F model exhibit encapsulation, by the ability to store all of its attributes and behavior in one object, rather than as separate related files. DLG-F data can pass behavior to subsets of data or class object classes and polymorphic capabilities.

One of the most important features of the DLG-F model is the concept of permanent feature identifiers. The unique identifier, critical to OOGIS, employs the use of a numerical tag that is unique to the object with each feature instance. To avoid having to replace all features when updating data holding, users only update the features that were modified. This produces less error in the data because multiple versions of the data are eliminated. The unique identifier is also important because it can be implemented as the link between feature attribution and multiple scale feature spatial representations. The unique identifier is placed into a feature attribute table in the spatial data and can be used to access nonlocational data.

The DLG-F model also uses the concept of a data dictionary, which lays out the type of feature, its definition, the different feature classes associated with it, the coverage name, and the associated attribute table. The data dictionary also contains fields that are used to populate the attribute table and have properties associated with them such as attribute name, attribute type, data type, and attribute domain and definition.

DLG-F Feature:	Aircraft Facility			
Definition:	A location where aircraft can take-off and land, usually equipped with associated buildings and facilities.			
Feature Classes:	Point, Area			
Coverage Name:	TR_AIRPORTS			
Attribute Table:	TR_AIRPORTS.PAT			
DLG-F Attribute	**Arc/Info Item**	**Data Type**	**Attribute Domain**	**Definition**
Feature	FEATURE	Character 25	Alphanumeric	The name of the DLG-F feature.
Feature ID	FEATURE_ID	Character 10	Alphanumeric	Unique permanent feature ID.
Feature Type	FEATURE_TYPE	Character 25	Alphanumeric	The type of aircraft facility: Airport = fixed-wing aircraft Stolport = short take-off and landing aircraft Ultralight = small, ultralight aircraft
State FIPS Code	STATE_FIPS	Character 2	2-digit number	FIPS code of the state containing the feature.
County FIPS Code	COUNTY_FIPS	Character 7	(See table A-1 in appendix)	FIPS code of the county or counties containing the feature. (See table A-1 in appendix A for listing.)
FAA ID Number	FAA_ID	Character 4	Alphanumeric	The official ID assigned by FAA.
Facility Name	FACNAME	Character 42	Alphanumeric	Place name of the aircraft facility.
Facility Ownership	OWNNAME	Character 29	Alphanumeric	Ownership of the aircraft facility.
City of Location	CITYNAME	Character 26	Alphanumeric	Place name of the town or city nearest the aircraft facility.
County of Location	COUNTY	Character 21	3-digit integer	Place name of the county containing the aircraft facility.
Source	SOURCE	Character 32	Alphanumeric	Source data from which the feature was captured.
Editor	EDITOR	Character 32	Alphanumeric	Entity that performed feature updates.
Source Date	SOURCE_DATE	Date	Date	Date of the source data.
Feature Modification Date	FEAT_MOD_DATE	Date	Date	Date when feature was last updated.

Table 1. Example of DLG-F data dictionary entry (University of Georgia, Information Technology Outreach Services, 1998)

The definition may contain the coded values for the attribute domain if the field is set to accept coded values. Table 1 is an example of a data dictionary entry.

The DLG data models illustrate the evolution from a hierarchical structure to a feature-based model and subsequently to an object-oriented database structure. An OO DLG-F data model is fully encapsulated, thereby retaining information of several data layer types in one dataset. In Table 1 for example, the aircraft facility data layer has two types of data within it, a point layer and an area or polygon layer. With earlier DLG structures, the two types of data would have to be kept in separate files and the models were incapable of handling two separate types of data at the same time. In the DLG-F OO model, data is related to the object that also includes attributes and behaviors because behavior is embedded within each object. Validation methods can also be used.

The DLG-F model can pass on characteristics and behavior to child datasets, a feature known as inheritance. Since the DLG-F model has a data dictionary with definitions of each attribute value, the information may be passed on to subsets of the data without having to physically reset the domain attributes. For example, if a user made a subset of aircraft facilities using an individual FIPS code, the resulting data layer would have all of the behavior of the original data layer, without having to redefine the behavior of each attribute.

In the context of the national database, an object-oriented data model clearly allows versioning to be used such that data modification and transaction updates can be undertaken without locking all other users out of the database. Versioning is extremely powerful in a multiple user GIS, such as that envisioned by the NSDI for the number of users creating, deleting and modifying data at one time could be in the hundreds to

thousands. Feature-based data stored in an object-oriented GIS has the ability to let each user access the dataset while another person is accessing the database. OOGIS enabled data with the capability of data locking and versioning is a powerful addition to the design of a national database.

Equally, the ability to incorporate multiple scale data that can share feature attributes through the unique ID is a further very effective capability of an object-based database. **Conflation** is the "process by which two digital maps, usually of the same area at different points in time, or two different thematic maps of the same area, may be matched and merged into one through geometrical and rotational transformations" (United States Geological Survey, 1999). Conflation is an involved process with a number of complex steps. The USGS is currently conflating the National Hydrography Dataset (NHD) for the entire United States such that the attribute and topological characteristic of feature data geometrically captured at one scale can be conflated with the spatial geometry of each water feature captured at a finer spatial scale.

The NSDI uses similar techniques to add multiple scale data to the NHD database through the use of a regional integrator. The regional integrator uses techniques similar to those of conflation to produce data at multiple scales. For example, a 1:100,000 road dataset could be generalized to 1:250,000 road dataset using a process similar to conflation for use in the NSDI. DLG-F is ideal for conflation and multiple scale use because of its versatility and powerful feature-based nature.

4.3. Review of Object-Oriented Concepts in the FGDC Project

The FGDC project exemplifies many unique characteristics and strengths of an object-based GIS. The feature-based DLG-F incorporated within an OOGIS has the capability to draw upon encapsulation, polymorphism and inheritance. The use of unique identifiers, data locking, versioning and conflation are powerful contributions to the national framework database provided by OOGIS. The use of a unique ID resolves issues of data redundancy and data error because it remains static, regardless of changes to the feature-based data.

Conflation, data locking and versioning are essential elements in a database of this magnitude and intended use. Since the users and contributors to such a national database would be many, the need to create, delete and modify data without locking out other users and contributors is vital. Feature-based data and an object-oriented database structure allow for simultaneous access to a dataset by creating logical copy of the data and permits conflation between data of differing spatial scales. A database that can draw upon the full range of OOGIS functionality would clearly be able to support the more complex organizational and user demands as envisioned by the NSDI. The limitations of the ER model are thus fully exposed in the instance.

Chapter 5. Conclusion

The data model is essential to any GIS. To date, the Entity-Relationship model has been the most commonly used model to represent real world features in a GIS. However, the object-oriented data model is a recent powerful alternative, and by all accounts, the data model of the future. Because of the advantages of objects over entities and the preceding chapters it is proposed that the object-oriented data model provides a more powerful abstraction of reality than the entity-relationship model and a more powerful model as well. The OO data model has the ability to manage data in a more intuitive manner by ordering data in a fashion similar to how a human would perceive and refer to the world. Thus people refer to roads, buildings and streams as features and objects, not by point or arc or polygon.

The object-oriented data model has some unique differences and advantages over the entity-relationship model. The OO data model is based on feature classes or a schema, where each feature type may have several attributes. Active object databases, such as geodatabases, allow feature-based data to be grouped into classes that portray similar entities. The classes facilitate the creation of new feature classes and only the differences need to be modified in the schema.

Features have their own behavior or methods encapsulated with the data, not in separate application programs. By defining each spatial feature as an object, users can determine how an object behaves in relation to other objects around it. For example, rivers cannot flow uphill. The definition of feature classes specifies what values and behaviors can be associated with an object. Methods or behaviors allow objects to respond differently to an instruction depending on its class and polymorphism provides a

mechanism for moving functionality out of applications and into class definitions stored with the object in the database (Sargent, 1999). The ability for different objects to interpret commands unique to the nature of that object is a very powerful tool.

The complex use of OOGIS in the case studies illustrates the power and benefits of the OO data model. Concepts such as methods, schema, long transactions and versioning cannot be accomplished using the ER model or the georelational model with any degree of efficiency or effectiveness. Since the two latter models do not associate behavior with data as the OO data model does, aspects of the NIMA project could not have been accomplished as well as they were by using the OOGIS.

The ER model, for example, would be structured such that entities (coverages) would participate in many different types of relationships. In any one-to-one relationships, attribute or data tables for each entity would be joined into one table or be kept separate. One-to-many relationships that require two attribute or data tables and a field that enables a relational join. In many-to-many relationships, all attribute or data tables would be kept separate (Heywood *et al.*, 1998). Once the attributes have been selected, the model would be created as well as a set of table definitions with details of the attributes, such as name, size and domain (Heywood *et al.*, 1998). In contrast, the OO data model has the capacity to identify relationships between features and data redundancy is minimized due to the relationships taking into account all of the factors described above without user intervention and possible user error.

The case studies exemplify the object-oriented data model as a feature-based model and a more intuitive representation of the real world. Similarly, since attribute domains cannot be implemented with the ER model, reflexive data validation cannot take

place because limitations cannot be set on field values. The ER model is not encapsulated and it keeps the data, attributes and relationships as separate entities. Since attribute domains cannot be created in the ER model, inheritance cannot take place and methods or behavior cannot to be passed to child classes.

The advantages of the OO data model are further illustrated in the FGDC case study that shows the power and versatility of the OO data model. The data dictionary could not be created using the ER model because behavior in the ER model cannot be associated with the data. Versioning would be extraordinarily difficult using the ER model. Since there are multiple tables and relationships involved with the ER model and creating a logical copy of only the data being manipulated could cause serious errors in the data and could lead to file corruption. Data locking and long transactions, which usually accompany versioning, would also be considerably more tedious and cumbersome procedure using the ER model. Data locking would have to take place for long transactions to prevent multiple edits of the same data or unnecessary data redundancy. Again, specific parts of tables would have to be restricted, which may, in turn, restrict the whole table and thereby, create accessibility problems for other users. If a long transaction were to take place, the data may be restricted for long periods of time, limiting the amount of data accessible to other users. Using the ER model would not meet the long-term goals of the NSDI in building a national database and may create more problems than the issues sought to be resolved.

There is a wide array of uses for OOGIS. As the trend in geographic data moves toward feature-based objects, the need for a system that handles data more intuitively becomes apparent. A data model that is “smart” and handles and organizes objects

representing the world in a feature-based approach is becoming increasingly important in GIS. Tools such as reflexive methods and versioning are made possible by the OO data model and represent significant advantages over earlier models. The OO data model is superior over the entity-relationship model and provides a development based from which to drive a new generation of GIS tools and applications. OOGIS is a powerful, reliable system in GIS technology and the wave of the future for geographic information systems.

References

Association for Geographic Information, 1999. GIS Dictionary. http://www.geo.ed.ac.uk/agidict/alpha.html. September 2000.

Booch, G., 1996. *The Best of Booch: Designing Strategies for Object Technology*. SIGS Books. New York.

Burrough, P. A. and McDonnell, R. A., 1998. *Principles of Geographical Information Systems*. Oxford University Press. Oxford, England.

CARIS, 2002. CARIS Revolutionizing Geomatics. http://www.caris.com. January 2001.

Chen P. P., 1976. The Entity-Relationship model - Toward a Unified View of Data. *ACM Transactions on Database Systems* Vol. 1, No 1: 9-36

Davis, C. A. and Borges, K. A. V., 1994. Object-Oriented GIS in Practice. http://www.odyssey.ursus.maine.edu/gisweb/spatdb/urisa/ur94070.html. May 2001.

Desai, C., Harris, T., Qin, P., Sun, X. and Warner, T., 1999. Automated Feature Extraction from Commercial Satellite Multi-spectral and Pan-chromatic Imagery. *Innovative Approaches to Satellite Imagery Feature Extraction, Vector Conversion, and Object-Oriented Geo-processing.* National Imaging and Mapping Agency contract number NMA202-98-K-1096.

Environmental Systems Research Institute, Inc., 1999. ArcInfo 8: A New GIS for the New Millennium. http://www.esricanada.com/support/resources/PDFs/AI8NewMill.pdf. September 2000.

Federal Geographic Data Committee, 1997. *Framework: Introduction and Guide*. Federal Geographic Data Committee. Washington, D.C.

General Electric, no date. GE Smallworld Core Spatial Technology: Overview. http://www.smallworld-us.com/english/products/spatial/Core.pdf. July 2001.

Heywood, I., Cornelius, S. and Carver, S., 1998. *An Introduction to Geographical Information Systems*. Addison Wesley Longman Limited. Essex, England.

Korte, G., no date. Trends in Spatial Database Technology. http://www.giscafe.com.GISVision/Review/GisDatabaseModule.html. December 2000.

LaserScan, Inc., 2001. Gothic Lamps2: Product Feature Overview. http://www.LaserScan.com/lamps2.htm. December 2001.

LaserScan, Inc., no date. The Significance of Object-Orientation for GIS. http://www.lsl.co.uk/papers/ooforgis.htm. December 2001.

National Spatial Data Infrastructure, 1996. National and Regional-Level Area Integrator Concepts Using Multi-Scale, Feature-Based Transportation Data in West Virginia. *Proposal Information Summary: (FY 96) NSDI Competitive Cooperative Agreements Program*.

United State Geological Survey, 1997. DLG-E to SDTS Document. http://mcmcweb.er.usgs.gov/sdts/emapoct_93/SDTSindex.html. January 2001.

United States Geological Survey, 1999 (a). FOD Maintenance. http://edcnts1.cr.usgs.gov/systest. January 2001.

United States Geological Survey, 1999 (b). Research and Design of Prototype Geospatial Data Transport and Update Capabilities in a Distributed Database Environment. *Cooperative Research and Development Agreement (CRADA)*.

United States Geological Survey, 2000. The National Hydrography Dataset.

United States Geological Survey, 2001. National Hydrography Dataset Conflation Procedures.

University of Georgia, Information Technology Outreach Services, 1998. DLG-F Data Dictionary—Transportation. http://www.gis.state.ga.us/edocs/framework/trans/DLG-F_Transportation_Data_Dictionary.htm. January 2001.

Wadembere, M. I., 2001. Object-Oriented Modeling. http://www.ismail.tripid.com/oomodeling.htm. January 2002.

Worboys, M. F., 1994. Object-oriented Approaches to Geo-referenced Information. *International Journal of Geographical Information Systems* 8 (4): 385-399.

Worboys, M. F., 1995. *GIS: A Computing Perspective*. Taylor and Francis, Inc., London, England.

Worboys, M. F., 1999. Relational Databases and Beyond. *Geographical Information Systems, Volume 1: Principles and Technical Issues*. Edited by M.F. Goodchild, P.A. Longley, D.J., Maguire, and D.W., Rhind.

Appendix A – Coverage Conversion Method Code
(Code developed by Ping Qin, Xiaohua Sun and Janette Bennett)

```
Object = "{9FF55731-7ACD-11D0-89E2-080009A874FA}#1.0#0"; "arcplot.OCX"
Begin VB.Form Coverage_Conversion
   Caption         =   "Coverage Conversion"
   ClientHeight    =   5220
   ClientLeft      =   60
   ClientTop       =   345
   ClientWidth     =   10740
   LinkTopic       =   "Form1"
   ScaleHeight     =   5220
   ScaleWidth      =   10740
   StartUpPosition =   3  'Windows Default
   Begin VB.CommandButton cmdgdb
      Caption         =   "Coverage to Geodatabase"
      Height          =   495
      Left            =   8280
      TabIndex        =   15
      Top             =   1920
      Width           =   2175
   End
   Begin VB.CommandButton cmdexit
      Caption         =   "Exit"
      Height          =   495
      Left            =   8280
      TabIndex        =   14
      Top             =   2640
      Width           =   2175
   End
   Begin VB.ListBox List1
      Height          =   2400
      Left            =   4440
      TabIndex        =   13
      Top             =   2400
      Width           =   1575
   End
   Begin VB.TextBox Text1
      Height          =   375
      Left            =   6240
      TabIndex        =   8
      Top             =   2880
      Width           =   1575
   End
   Begin VB.DirListBox Dir2
      Height          =   1215
      Left            =   6240
      TabIndex        =   7
      Top             =   960
      Width           =   1575
   End
```

```
Begin VB.DirListBox Dir1
   Height          =   1215
   Left            =   4440
   TabIndex        =   6
   Top             =   960
   Width           =   1575
End
Begin VB.CommandButton cmde00
   Caption         =   "Coverage to .e00"
   Height          =   495
   Left            =   8280
   TabIndex        =   5
   Top             =   1200
   Width           =   2175
End
Begin VB.CommandButton cmdshape
   Caption         =   "Coverage to Shape"
   Height          =   495
   Left            =   8280
   TabIndex        =   4
   Top             =   480
   Width           =   2175
End
Begin VB.DriveListBox Drive2
   Height          =   315
   Left            =   6240
   TabIndex        =   3
   Top             =   480
   Width           =   1575
End
Begin VB.DriveListBox Drive1
   Height          =   315
   Left            =   4440
   TabIndex        =   2
   Top             =   480
   Width           =   1575
End
Begin Arcplot.Arcplot Arcplot1
   Height          =   4095
   Left            =   240
   TabIndex        =   0
   Top             =   480
   Width           =   4095
   _Version        =   65536
   _ExtentX        =   7223
   _ExtentY        =   7223
   _StockProps     =   0
End
Begin VB.Label lbloutname
   Caption         =   "Give a New Name for the Output File:"
   Height          =   495
   Left            =   6240
   TabIndex        =   12
   Top             =   2400
   Width           =   1695
End
```

```
   Begin VB.Label lblinput
      Caption         =   "Input File Name:"
      Height          =   255
      Left            =   4440
      TabIndex        =   11
      Top             =   120
      Width           =   1215
   End
   Begin VB.Label lbloutput
      Caption         =   "Output File:"
      Height          =   255
      Left            =   6240
      TabIndex        =   10
      Top             =   120
      Width           =   1215
   End
   Begin VB.Label lblconvert
      Caption         =   "Conversion Type:"
      Height          =   255
      Left            =   8280
      TabIndex        =   9
      Top             =   120
      Width           =   1455
   End
   Begin VB.Label lblimage
      Caption         =   "Image Preview:"
      Height          =   255
      Left            =   240
      TabIndex        =   1
      Top             =   120
      Width           =   1215
   End
End

Attribute VB_Name = "Coverage_Conversion"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Dim theInput As String
Dim theOutput As String
Dim result As New ESRIutil.Strings
Dim severity As Long

Public Property Get aicontrol() As Variant

End Property

Public Property Let aicontrol(ByVal vnewvalue As Variant)
    Dim Arcplot1 As Object
    Set Arcplot1 = vnewvalue
End Property
```

```
Private Sub cmde00_Click() 'Coverage to e00 button

    theInput = Dir1.Path & "\" & List1.List(List1.ListIndex) 'reads the
     input box
    theOutput = Dir2.Path & "\" & Text1.Text 'reads the output text
     entered by the user

    severity = Arcplot1.Command("arc export cover " & theInput & " " &
     theOutput, result) 'executes the conversion command

End Sub

Private Sub cmdexit_Click() 'Code for the exit button

    End

End Sub

Private Sub cmdgdb_Click()
 'This is from the ArcObjects samples that comes with Arc 8
  theInput = Dir1.Path & "\" & List1.List(List1.ListIndex)
  theOutput = Dir2.Path & "\" & Text1.Text

  Dim pPropset As IPropertySet
  Set pPropset = New PropertySet
  pPropset.SetProperty "Database", feature1

  Dim pOutAcFact As IWorkspaceFactory
  Set pOutAcFact = New AccessWorkspaceFactory
  Set pOutAcWorkspaceName = pOutAcFact.Create(Dir2.Path & "\",
feature1, pPropset, 0) 'This tells the computer where to put the new
geodatabase

  '+++ create a new feature datset name object for the output Access
feature dataset
  Dim pOutAcFeatDSName As IFeatureDatasetName
  Set pOutAcFeatDSName = New FeatureDatasetName

  Dim pOutAcDSName As IDatasetName
  Set pOutAcDSName = pOutAcFeatDSName

  Set pOutAcDSName.WorkspaceName = pOutAcWorkspaceName
  pOutAcDSName.Name = theOutput

  ' +++ now get the name object for the input coverage feature dataset
name.
  Dim pInCovWorkspaceName As IWorkspaceName
  Set pInCovWorkspaceName = New WorkspaceName
  pInCovWorkspaceName.PathName = Dir1.Path & "\"
  pInCovWorkspaceName.WorkspaceFactoryProgID =
"esriCore.ArcInfoWorkspaceFactory.1"

  Dim pFeatureDatasetName As IFeatureDatasetName
  Set pFeatureDatasetName = New FeatureDatasetName
  Dim pCovDatasetName As IDatasetName
  Set pCovDatasetName = pFeatureDatasetName
  pCovDatasetName.Name = List1
```

```
  Set pCovDatasetName.WorkspaceName = pInCovWorkspaceName

  '+++ now use the conversion function convert the coverage to an
Access feature dataset
  Dim pCovtoFD As IFeatureDataConverter
  Set pCovtoFD = New FeatureDataConverter
  pCovtoFD.ConvertFeatureDataset pCovDatasetName, pOutAcDSName,
Nothing, "", 1000, 0

  MsgBox "Coverage conversion complete!"
End Sub
Private Sub cmdshape_Click() 'This is the coverage to shapefile button

    Dim Feature As String

    theInput = Dir1.Path & "\" & List1.List(List1.ListIndex) 'sets the
     input coverage
    theOutput = Dir2.Path & "\" & Text1.Text 'sets the name of the
     output shape file

    Feature = _
    InputBox("Enter: ARCS or NODES or POINTS or POLYS or
REGION.subclass or ROUTE.subclass or SECTION.subclass or TIC", "Input
Feature Class") 'Pop-up box that prompts user for feature type

    severity = Arcplot1.Command("arc arcshape " & theInput & " " &
Feature & " " & theOutput, result) 'Creates the new shape file


End Sub

Private Sub Dir1_Change() 'Changes the input directory

    Dim result As New ESRIutil.Strings
    Dim Counter As Integer

    result.Clear
    List1.Clear
    Arcplot1.GetCover Dir1.List(Dir1.ListIndex), result
    For Counter = 0 To result.Count - 1
        List1.AddItem result.Item(Counter)
    Next Counter

End Sub

Private Sub Drive1_Change() 'Changes the input disk drive

    Dir1.Path = Drive1.Drive

End Sub

Private Sub Drive2_Change() 'Changes the output disk drive

    Dir2.Path = Drive2.Drive

End Sub
```

```
Private Sub Form_Load() 'retrieves information from disk drives and
directories

    Dim result As New ESRIutil.Strings
    Dim Counter As Integer

    result.Clear
    List1.Clear
    Arcplot1.GetCover Dir1.List(Dir1.ListIndex), result
    For Counter = 0 To result.Count - 1
        List1.AddItem result.Item(Counter)
    Next Counter

End Sub

Private Sub List1_Click() 'Displays the coverage selected in the input
box
    theInput = Dir1.Path & "\" & List1.List(List1.ListIndex)

    severity = Arcplot1.Command("clear ", result)

    severity = Arcplot1.Command("linecolor " & Color, result)
    severity = Arcplot1.Command("mape " & theInput, result)
    severity = Arcplot1.Command("arcs " & theInput, result)


End Sub
```

Appendix B – The Clean and Build Method Code
(Code developed by Ping Qin, Xiaohua Sun and Janette Bennett)

```
Object = "{9FF55731-7ACD-11D0-89E2-080009A874FA}#1.0#0"; "arcplot.ocx"
Begin VB.Form Form1
   Caption         =   "Form1"
   ClientHeight    =   6330
   ClientLeft      =   60
   ClientTop       =   345
   ClientWidth     =   7350
   LinkTopic       =   "Form1"
   ScaleHeight     =   6330
   ScaleWidth      =   7350
   StartUpPosition =   3  'Windows Default
   Begin VB.ComboBox Combo1
      Height          =   315
      ItemData        =   "jtest2.frx":0000
      Left            =   4560
      List            =   "jtest2.frx":0016
      TabIndex        =   12
      Top             =   5160
      Width           =   1335
   End
   Begin VB.CommandButton cmdexit
      Caption         =   "Exit"
      Height          =   495
      Left            =   5640
      TabIndex        =   11
      Top             =   5640
      Width           =   1215
   End
   Begin VB.ListBox List1
      Height          =   2010
      Left            =   5520
      TabIndex        =   10
      Top             =   2400
      Width           =   1575
   End
   Begin VB.DirListBox Dir1
      Height          =   1215
      Left            =   5520
      TabIndex        =   9
      Top             =   960
      Width           =   1575
   End
   Begin VB.DriveListBox Drive1
      Height          =   315
      Left            =   5520
      TabIndex        =   6
      Top             =   480
      Width           =   1575
   End
```

```
Begin VB.CommandButton cmdDraw
   Caption         =   "Draw"
   Height          =   495
   Left            =   960
   TabIndex        =   4
   Top             =   4800
   Width           =   1215
End
Begin VB.CommandButton cmdClean
   Caption         =   "Clean"
   Height          =   495
   Left            =   960
   TabIndex        =   3
   Top             =   5640
   Width           =   1215
End
Begin VB.CommandButton cmdClear
   Caption         =   "Clear"
   Height          =   495
   Left            =   2760
   TabIndex        =   2
   Top             =   4800
   Width           =   1215
End
Begin VB.CommandButton cmdBuild
   Caption         =   "Build"
   Height          =   495
   Left            =   2760
   TabIndex        =   1
   Top             =   5640
   Width           =   1215
End
Begin Arcplot.Arcplot Arcplot1
   Height          =   4335
   Left            =   120
   TabIndex        =   0
   Top             =   120
   Width           =   4455
   _Version        =   65536
   _ExtentX        =   7858
   _ExtentY        =   7646
   _StockProps     =   0
End
Begin VB.Label Label1
   Caption         =   "Select Color:"
   Height          =   495
   Left            =   4560
   TabIndex        =   13
   Top             =   4800
   Width           =   1215
End
```

```
   Begin VB.Label lblDisplay
      Caption         =   "Display"
      Height          =   255
      Left            =   120
      TabIndex        =   8
      Top             =   4920
      Width           =   615
   End
   Begin VB.Label lblOperate
      Caption         =   "Operate"
      Height          =   255
      Left            =   120
      TabIndex        =   7
      Top             =   5760
      Width           =   615
   End
   Begin VB.Label lblInput
      Caption         =   "Input File:"
      Height          =   255
      Left            =   5640
      TabIndex        =   5
      Top             =   240
      Width           =   1095
   End
End

Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Dim theInput As String
Dim theOutput As String
Dim result As New ESRIutil.Strings
Dim Severity As Long

Public Property Get aicontrol() As Variant

End Property

Public Property Let aicontrol(ByVal vNewValue As Variant)

    Dim Arcplot1 As Object
    Set Arcplot1 = vNewValue

End Property

Private Sub cmdBuild_Click() 'Builds the topology

    theInput = Dir1.Path & "\" & List1.List(List1.ListIndex)
    Severity = Arcplot1.Command("arc build " & theInput, result)

End Sub
```

```
Private Sub cmdClean_Click() 'Cleans the topology

    theInput = Dir1.Path & "\" & List1.List(List1.ListIndex)
    Severity = Arcplot1.Command("arc clean " & theInput, result)

End Sub

Private Sub cmdClear_Click() 'Clears the screen of the data display

    Severity = Arcplot1.Command("clear ", result)

End Sub

Private Sub cmdDraw_Click() 'Draws the layer selected in the input box

    theInput = Dir1.Path & "\" & List1.List(List1.ListIndex)
    Dim color As String

     color = Combo1.Text
    Severity = Arcplot1.Command("linecolor " & color, result)
    Severity = Arcplot1.Command("mape " & theInput, result)
    Severity = Arcplot1.Command("arcs " & theInput, result)

End Sub

Private Sub cmdexit_Click() 'Exits the application

    End

End Sub

Private Sub Dir1_Change() 'Changes the directory in the input box

    Dim result As New ESRIutil.Strings
    Dim counter As Integer

    result.Clear
    List1.Clear
    Arcplot1.GetCover Dir1.List(Dir1.ListIndex), result
    For counter = 0 To result.Count - 1
        List1.AddItem result.Item(counter)
    Next counter

End Sub

Private Sub Drive2_Change() 'Changed the drive of the output data layer

    Dir2.Path = Drive2.Drive

End Sub
```

```
Private Sub Form_Load() 'Loads the form of the application

    Dim result As New ESRIutil.Strings
    Dim counter As Integer

    result.Clear
    List1.Clear
    Arcplot1.GetCover Dir1.List(Dir1.ListIndex), result
    For counter = 0 To result.Count - 1
        List1.AddItem result.Item(counter)
    Next counter

End Sub

Private Sub drive1_change() 'Changes the drive of the input data layer

Dir1.Path = Drive1.Drive

End Sub
```