INFORMATICA, 2005, Vol. 16, No. 4, 1–14
© 2005 *Institute of Mathematics and Informatics, Vilnius*

ORVPF – the Model and its DNC Implementation

Hong ZHOU
Department of Mathematics, Saint Joseph College
West Hartford, CT 06117, USA
e-mail: hzhou@sjc.edu

Shahram RAHIMI, Raheel AHMAD
Department of Computer Science, University of Southern Illinois
Carbondale, IL 62901-4511, USA
e-mail: rahimi@cs.siu.edu

Marcin PAPRZYCKI
Computer Science Institute, Warsaw School of Social Psychology
03-815 Warsaw, Poland
e-mail: marcin.paprzycki@swps.edu.pl

Yufang WANG, Maria COBB
Department of Computer Science and Statistics, University of Southern Mississippi
Hattiesburg, MS 39406, USA
e-mail: ywan2@orca.st.usm.edu, maria.cobb@usm.edu

Received: October 2004

Abstract. Vector Product Format (VPF) based databases store geographical data in a relational framework, where individual VPF files are arranged hierarchically in a directory tree structure. Access and update of the VPF data can become difficult due to fragmentation of data among multiple tables. This paper presents an object-oriented model for the management of a VPF database, which provides easy access and automatic update for the VPF data, and is compatible with ESRI ArcView. This model has been successfully implemented in Java, for Digital Nautical Charts (DNC).

Key words: vector product format, ORVPF, digital nautical charts.

1. Introduction

Vector Product Format (VPF) is a specification developed by the U.S. Defense Mapping Agency (DMA) for storing geographical data in a relational framework (DOD, 1996). According to the VPF specification, geographical features are stored in ASCII or binary files and organized hierarchically using a directory structure. A number of existing geographical databases, including Digital Nautical Charts (DNC), World Vector Shoreline (WVS), and others, implement the VPF specification. Each of these VPF-based products has different sets of geographic features and attributes. DNC (NIMA, 1997) is a popular

VPF implementation that is being used in our current research project (see Acknowledgement). In this project we attempt at developing an autonomous geographical data management system which utilizes mobile software agents to find conflicts (or updates) among distributed geographic databases, analyze these conflicts (or updates) and employ conflation methods to resolve them. The very first requirement for this system to work is to provide geographic data to the agents in a format that the mobile agents can effectively utilize in the conflation process. Since the outline of the project requires us to support the DNC implementation of the VPF format, it was necessary to provide the system with mechanism by which software agents could easily read and update information stored in DNC databases. Since Java is used as the primary programming language in this project, we chose a Java-based object-oriented approach to model the VPF database and implement the DNC dataset. As our model is object-oriented while the underlying implementation is the relational VPF database, we refer to the proposed approach as the Object-Oriented Relational VPF (ORVPF).

To better understand the implementation details and intricacies described later in the paper, we start with a brief description of the VPF and the DNC database. We then present the difficulties related to reading and updating of the DNC database without using an object-oriented database management system. We follow with a description of the object-oriented structure of the ORVPF model followed by the overview of the implementation. The paper concludes with a discussion of future work regarding this project.

1.1. *VPF and DNC*

As mentioned earlier, VPF is a general specification for storing geographical data that is widely used in various geographic software products. DNC, an extension of VPF, is a database designed primarily for marine navigation and produced by the National Geospatial-Intelligence Agency (NGA). Its aim is to provide a digital reproduction of paper charts. In DNC, all VPF data is stored as relational tables in individual VPF files.

Fig. 1 depicts the hierarchical structure of the VPF data that are arranged hierarchically in a directory tree structure in order to organize the relationships between the components of the database. VPF's root directory is the *database* which is composed of a collection of libraries represented by the library subdirectories. A *library* defines a geographic boundary and its scale, and is further divided into *coverages*. Each *coverage* contains data for logically and spatially organized groups of geographic features that are topologically related. The coverage directory contains files describing its feature classes and individual feature attributes. All location and topology information is further subdivided among tile subdirectories within each coverage. Each tile represents a spatial subregion within the library boundaries. There are four types of features defined in VPF: point feature, line feature, area feature, and text feature. The four types of features are composed of five types of geographic *primitives*: entity node, connected node, edge, face, and text.

VPF provides four levels of increasingly complex topology from level 0 to level 3, with level 3 representing full topology. VPF utilizes a winged-edge topology structure

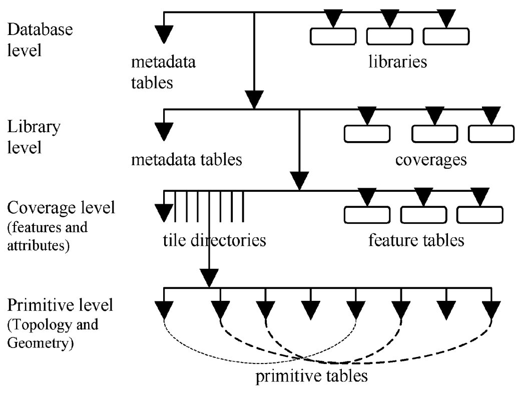

Fig. 1. The VPF hierarchical, directory based structure.

to represent the comprehensive level 3 topology (DOD, 1996). In winged-edge topology, the topology information is maintained in tables of face, edge, ring and node. Each edge, face, ring and node contains information regarding its own topology and its neighboring primitives if needed. For example, an edge stores its own start and end nodes and its neighboring edges and faces. Every node stores the face that it is part of, and so on (Cobb *et al.*, 1998a). Winged-edge topology is one of the most complex components that are required to be implemented in this work.

1.2. *Motivation behind the Object-Oriented Implementation of VPF/DNC*

The relational structure of VPF is quite complicated because of its treatment of spatial topology, i.e., the inter-relationships among features and primitives. These inter-relationships cause even a small update to the VPF database to result in updating multiple tables. At the level of coverage directory, the VPF structure provides a number of many-to-many join tables to relate features and geographical primitives. Other join tables support textual notes pertaining to multiple spatial features and brief textual descriptions of non-spatial attributes. As a derivative of VPF, DNC is also hierarchical and contains complex inter-relationships between features and primitives. In our experiments we utilize a DNC library (A0713060 in DNC07), which is 1.47Mb in size and contains 3498 geographic features. This library consists of 732 files and 52 directories to describe the location, topology, and other attributes of features. Given the high degree of interdependency among features and geographical primitives, managing even simple changes is non-trivial. In fact, even reading data to construct a single feature becomes difficult as the pertinent information is spread among many tables.

While a relational model, as implemented in DNC, is believed to be too restrictive for many spatial data applications (Egenhofer and Frank, 1989), the object oriented technology is argued to be well suited for the complex VPF/DNC data modeling (Shaw *et al.*, 1996). In fact, Naval Research Laboratory (NRL) at the Stennis Space Center and the University of Florida's GeoPlan Center developed ODNC, which is a prototype OO viewer/editor for the Digital Nautical Chart (Shaw *et al.*, 1996). Further, built upon the

ODNC prototype, an OO database (OVPF) for four different VPF products was designed (Chung *et al.*, 1995; Shaw *et al.*, 1996). ODNC was initially implemented in SmallTalk. The original design was later translated to Java and became the starting point for the NRL's web-based GIS database (Cobb *et al.*, 1998b). Our goal in developing the ORVPF was to provide a complete data management system for the VPF/DNC, a system that is flexible for future extensions, platform independent, compatible with ArcView, and allowing direct access. Finally, we believe that our system will further the path to interoperability between geo-databases and software agents that are to operate on them, and to the goal of open source.

In our design we have decided to concentrate on the following benefits of object-oriented design:

- ORVPF models every geographical feature, primitive, attribute, and higher level directory as an object. The inter-relationships among these geographical components are implicitly built in ORVPF objects, and thus require no explicit processing. This makes the data access and update much easier and intuitive.
- The object-oriented model of ORVPF allows it to be easily integrated into other applications and provides great flexibility for future development. Furthermore, as we will show, ORVPF objects can be easily made compatible with other applications such as ESRI ArcView.

2. ORVPF Model

The object-oriented principles of identity, encapsulation, inheritance and polymorphism (Jacobson *et al.*, 1999; Rumbaugh *et al.*, 1999) empower an object-oriented data model with capability to handle topological and other relationships among spatial feature objects directly and in a simple fashion (Shaw *et al.*, 1996). We follow these basic principles to analyze the VPF data and to design the ORVPF model.

2.1. *General Structure*

The general structure of the ORVPF model is composed of two parts: the interface and implementations. The ORVPF interface defines the object-oriented VPF database management model. It specifies how we view the VPF hierarchical structure in the object-oriented form, together with the operations on the VPF data. Implementations facilitate actual data manipulation based on the specifications defined in the interface. Each existing implementation of VPF will result in different implementation of the ORVPF interface. For instance, this paper presents the specific DNC implementation.

Based on the hierarchical structure of the VPF, the ORVPF was designed as a tree model in which different levels of tree nodes represent different hierarchical levels in the VPF. The key idea in the ORVPF is to represent each VPF directory, file table, and table record as a Java object (an ORVPF node). Relationships between main ORVPF nodes and VPF hierarchical directories/files/table-records are illustrated in Table 1 (each ORVPF

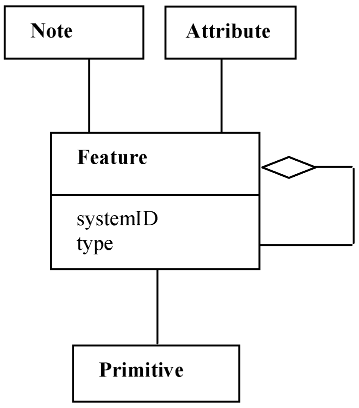

Fig. 2. The structure of an ORVPF Feature and its relationship with Primitive, Note and Attribute. Notice that the class relationship between Feature, Primitive, Note, and Attribute is aggregation and is also bidirectional association.

Table 1

The relationships between ORVPF classes and the VPF directories/tables/table-records. en: entity node; cn: connected node

ORVPF Class	VPF Table/Directory
Database	database
Library	library
Coverage	coverage
VirtualTile	tile/coverage
FeatureManager	feature Class
Feature	feature
AreaFeature	area feature
LineFeature	line feature
PointFeature	point feature
TextFeature	text feature
PrimitiveManager	primitive table
EntityNodeManager	en table
ConnectedNodeManager	cn table
EdgeManager	edge table
FaceManager	face table
TextManager	text table
Primitive	primitive
EntityNode	entity node
ConnectedNode	connected node
Edge	edge
Face	face
Text	text
Note	notes

class name starts with an uppercase letter while names of VPF directories/files/table-records are all in lower case).

In the VPF, not all libraries are tiled (physically partitioned). Thus, to make the ORVPF interface independent of the existence of tiles, a class called VirtualTile was introduced. Class VirtualTile is guaranteed to exist in every Coverage class regardless of whether the coverage is tiled or not. When the coverage is tiled, a VirtualTile object corresponds to the real tile in the VPF structure, while a VirtualTile object corresponds directly to the coverage for a non-tiled coverage. Therefore, by providing the class VirtualTile, the existence of tiles in coverage is transparent to users. Another important class in the ORVPF is called FeatureManager, which exists inside the Coverage class just as a feature class exists in the VPF coverage directory. A FeatureManager object specifically matches to a feature class (a feature table) and contains all the Features it is managing, provides methods to retrieve any specific Feature, and provides the ORVPF with the capability to update/modify a Feature object.

Since ESRI is the largest geographic tool vendor and our system is using ArcView as the Graphical User Interface, we decided to make ORVPF compatible with ArcView. To do so, ORVPF nodes should be transformed into Shapefiles for ArcView to display them. Shapefile is an ArcView GIS data set used to represent a set of geographic features (ESRI, 1998). It stores non-topological geometry and attribute information for the spatial features in a data set. The geometry for a feature is stored as a shape comprising a set of vector coordinates (ESRI, 1998). Shapefile defines its own data types such as Point, PolyLine, Polygon, MultiPoint, PointZ, etc, and can support point, line, and area features. The definition of a Shapefile feature is very close to the definition of a VPF feature (DOD, 1996; ESRI, 1998). This allows an ORVPF Feature to be matched to its counterpart in Shapefile. Shapefile does not support text feature as VPF does, however. Therefore, for simplicity, ORVPF does not define the Shapefile counterpart for an ORVPF TextFeature object. Table 2 shows: 1) the connection between ORVPF Features and Shapefile features, 2) the Shapefile data types that are used to represent the geometrical data of ORVPF features.

Shapefile coverage is a set of thematically associated data considered as a unit. It usually represents a single theme such as soils, streams, roads, or land use (ESRI, 1998). Correspondingly, we define the Coverage class in the ORVPF to match the coverage in Shapefile in Table 2. Though the ORVPF base class, VPFNode, defines the method to convert ORVPF objects into a Shapefile dataset, it is clear from Table 2 that currently this

Table 2
The relationship between ORVPF classes and ESRI Shapefile data types

ORVPF	Shapefile	Data Type
PointFeature	point feature	point
LineFeature	line feature	polyLine
AreaFeature	area feature	polygon
Coverage	coverage	

method cannot be implemented in Primitive classes in the ORVPF. Instead, the conversion between ORVPF Primitive classes and Shapefile data types are implicitly processed at the Feature level.

Summarizing, nodes in the ORVPF share some common properties which are represented in a class named VPFNode, which is inherited by all nodes at all levels. Common properties shared by all the nodes are:

- having a unique system ID,
- capability to locate parent node if there is one,
- method to find its child nodes if there are any,
- method to obtain its metadata if the metadata exists,
- method to convert the data concerning this node and its child nodes into a Shapefile format.

2.2. *Features and Primitives*

At the coverage level, the VPF provides other data tables and join tables to illustrate the topological relationships among features, notes, and attribute values. In the VPF, a feature is composed of attributes, notes, and a set of primitives that represent the geometrical data of the feature. However, topological relationships between a feature and its components are fragmented and stored in multiple relational tables (DOD, 1996; NIMA, 1997). From the perspective of object-oriented design, when a feature is constructed, its associated primitives, notes and attribute values should be automatically accessible. Therefore, in the ORVPF, we do not represent these topological relationships explicitly. Instead, they are embedded into the Feature class specification. Fig. 3 illustrates, in an UML notation (Scott, 2001), structure of an ORVPF Feature class and its relationship with Primitive, Note, and Attribute.

Fig. 3 also shows the many-to-many bidirectional association relationships among these classes. A Feature may refer to multiple Primitives, Notes and Attributes. Mean-

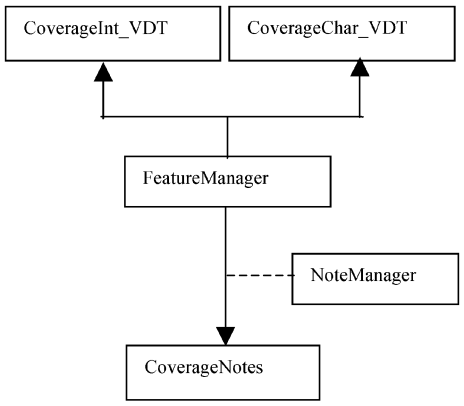

Fig. 3. The class relationships between FeatureManager, CoverageNotes, NoteManager, CoverageInt_VDT and CoverageChar_VDT in UML notation. It shows that NoteManager bridges the association relationship between FeatureManager and CoverageNotes.

while, any Primitive, Note, or Attribute may be referenced by multiple Features at the same time. Also, a Feature may contain other Features (complex feature (DOD, 1996)). It is important to point out that inside the ORVPF Feature, the Primitives must be arranged in a specific order. For example, AreaFeature and LineFeature contain a set of Edges. The first element in the set is the starting Edge. All the Edges are organized in an order that satisfies the winged-edge topology specification (DOD, 1996; NIMA, 1997). Such an ordered Primitive collection in the ORVPF Feature removes from the users the burden of finding the geometrical relationships among the Primitives.

In the VPF, geographic primitives refer to each other using cross-references in relational tables. For example, an edge primitive table (EDG) usually contains columns of ID, start node, end node, right face, left face, right edge, left edge and coordinates in order. This means that an edge contains information about its connected nodes, its left and right faces and its left and right edges. Meanwhile, a connected node primitive table usually has columns of ID, first edge and coordinates. This specifies that a connected node references an edge. Such primitive cross-references are represented by the primary/foreign key style of relational database. ORVPF models such low-level relationships among geographic primitives, and therefore its Primitive objects may cross-reference each other. For example, one Edge object references two other ConnectedNode objects (start node and end node), two other Face objects (left face and right face), and/or two other Edge objects (left edge and right edge). Also, the Edge object is referenced by other objects of ConnectedNodes, Faces, and/or Edges.

2.3. *ORVPF Actions*

There are basically five types of actions that may be performed on the ORVPF database: reading, modification, deletion, creation, and importing. Reading is implicit in ORVPF and is addressed separately in Section 3. Since the current ORVPF implementation is designed primarily to support our geospatial data conflation process, we do not feel an immediate need to support direct Feature creation at this time. Instead we support Feature importing, which, in fact, is an action of Feature creation.

The three types of actions that involve ORVPF Features (modification, deletion, importing) may result in update of topological relationships between Features, Primitives, Attributes, and Notes. The following method of FeatureManager deals with the topological updates when a feature is updated (in Java notation).

```
public boolean update(Feature f, int action, TopologyManager topologyManager).
```

When a Feature is required to be updated, its FeatureManager is called for the action. ORVPF specifies that a FeatureManager object requires a TopologyManager object to execute the update action and complete the update. In the ORVPF, TopologyManager is an interface; how it performs the updates and topology validations is not specified and is left to implementation. By allowing different implementations of the TopologyManager, ORVPF provides a flexible framework to include also other geographic feature update procedures.

3. DNC Implementation

Our first implementation of the ORVPF is for the DNC database. At this time we were able to implement bridge between the relational structure of the DNC and the OO based ORVPF. However, the implementation of independent feature importing and Shapefile conversion is still in progress, hence we will not discuss them here.

3.1. *TableImpl and RowsetImpl Classes*

The first step in our implementation was to build classes that could properly read and update any table files of the DNC database. We developed two classes: TableImpl and RowsetImpl, where the RowsetImpl class models a relational record as a Java object and provides ways to retrieve and modify column values in a record. The TableImpl class models any relational table as a Java object and is in fact a collection of RowsetImpl objects. All RowsetImpls in TableImpl are organized in the order specified by the DNC table file. Additionally, TableImpl provides methods for retrieving any RowsetImpls from its RowsetImpl collection. Upon the request for an update (a write action), TableImpl writes back all its information into the original physical file, according to the original format. These two classes constitute the base for our object-oriented implementation of the DNC.

With TableImpl and RowsetImpl classes completed, Feature, Primitive, and various manager classes are successively being implemented. There are five different implementations for the five ORVPF Primitives (EntityNodeImpl, ConnectedNodeImpl, FaceImpl, EdgeImpl, and TextImpl). Correspondingly, there are five implementations for the five different primitive managers (EntityNodeManagerImpl, FaceManagerImpl, EdgeManagerImpl, TextManagerImpl and ConnectedNodeManagerImpl,). Each manager class is a wrapper of the TableImpl, and each primitive is a wrapper of the RowsetImpl. The Primitive objects are constructed when the manager classes read in the table information and construct the RowsetImpl objects. However, the cross-references of other Primitives inside a Primitive object are dynamically produced based on the information stored in the wrapped RowsetImpl object.

The implementations of the four types of features are also designed as wrappers of RowsetImpl objects. For instance, AreaFeatureImpl is constructed around a RowsetImpl object which represents a record in an area feature table. Since an area feature has to refer to multiple primitives (faces and edges) and has various attributes, the construction of AreaFeatureImpl has to build up references based on the information stored inside the wrapped RowsetImpl object. Different from the constructions of PointFeatureImpl and TextFeatureImpl, construction of AreaFeatureImpl and LineFeatureImpl follows the winged-edge topology algorithm (DOD, 1996; NIMA, 1997).

3.2. *Note and Attribute*

DNC provides data tables and join tables at the coverage level to describe notes, attribute values, and their topological relationships with features. In the ORVPF, these tables are not modeled explicitly. Instead, the relationships between notes, attribute values, and features are modeled implicitly in the Feature class, so that each Feature class has references to associated Notes and Attributes. Therefore, in the DNC implementation, upon the construction of various Feature classes, these relationships have to be developed. To the best of our knowledge, each DNC geographic feature contains at most one note. Thus, in our DNC implementation, each Feature class only references at most one Note.

CoverageNotes, NoteManager, CoverageInt_VDT, and CoverageChar_VDT classes are developed to construct the relationships between Features, Notes, and Attributes. At the Coverage level, there is one CoverageNotes object, multiple NoteManager objects, one CoverageInt_VDT object, and one CoverageChar_VDT object. The CoverageNotes object contains all the notes inside the VPF coverage. Each NoteManager object corresponds to a VPF notes join table. The CoverageInt_VDT class models the integer value description table, and the CoverageChar_VDT class models the character value description table. These class objects are constructed before Feature objects. When a FeatureManager object is constructed, it first creates the associated NoteManager so that it can link each Feature with the referenced Note later. Then, upon construction of Feature classes, the Feature class automatically builds up references to records in CoverageNotes, CoverageInt_VDT, and CoverageChar_VDT so that proper Note and Attributes can be attached to them. The relationship among FeatureManager, CoverageNotes, NoteManager, CoverageInt_VDT, and CoverageChar_VDT is illustrated in Fig. 3.

3.3. *Order of Object Construction*

Construction of objects is a process of reading DNC data and then constructing corresponding ORVPF objects. As the ORVPF is a tree structure, the object construction is thereby a top-down process in which the information pertaining to the Database object is read and constructed first. Upon the construction of the Database object, Library objects in the Database object are constructed. However, no Library object is populated with any real data until its method load() is executed. The purpose of not populating a Library object with real data at construction time is to save memory, since a DNC library usually takes tens of megabytes of space.

Because there exist cross-references among Features, Primitives, Notes and Attributes, the order by which these objects are constructed matters greatly. In a Coverage object, VirtualTile objects are constructed first. The reason is that Primitives in VirtualTiles are needed for the construction of Features. Inside each VirtualTile, the order by which Primitives are constructed does not matter since cross-references among Primitives are dynamically provided at request. Following the VirtualTile, CoverageChar_VDT, ConverageInt_VDT, CoverageNotes can be constructed in any order. Finally, FeatureManagers are constructed. NoteManager associated with the FeatureManager is constructed after the construction of the FeatureManager is initiated, but before the Features of the FeatureManager are constructed.

3.4. *TopologyManager*

TopologyManager classes are used to perform the topology update when a Feature is to be updated. There are three types of actions upon Features that require TopologyManager: modification, deletion, and importing. Feature importing is a process that involves Primitive creation, Feature creation, and topology updates. Currently, implementation of Feature importing is still in progress. Therefore, in this section, we only discuss Feature modification and Feature deletion.

3.4.1. *Feature Modification*

Modifications on a Feature object may involve Note, Attribute, or Primitive modification. When a Note in a Feature (say Feature F) is modified, say Note A is modified such that it becomes A', A' is compared to all the Notes that exist in the CoverageNotes. If a Note that is the same as A', exists then a simple reference switch is performed along with the updates of the NoteManager associated with the FeatureManager that manages Feature F, i.e., F is referencing A' instead of A as its Note. Meanwhile, if A is not referenced by any other Features besides F, it would be deleted. However, if there is no existing Note that matches A', then the old Note, A, has to be checked whether it is being referenced by other features besides F or not. Also, if no other feature references Note A, then Note A is simply modified so that it is now A'. Nevertheless, if there are other features referencing A, then a new Note is generated and Feature F is re-linked to this new Note. Fig. 4 demonstrates how a Note modification is processed in our DNC implementation. The same procedure applies to the modification of the integer and character value Attributes.

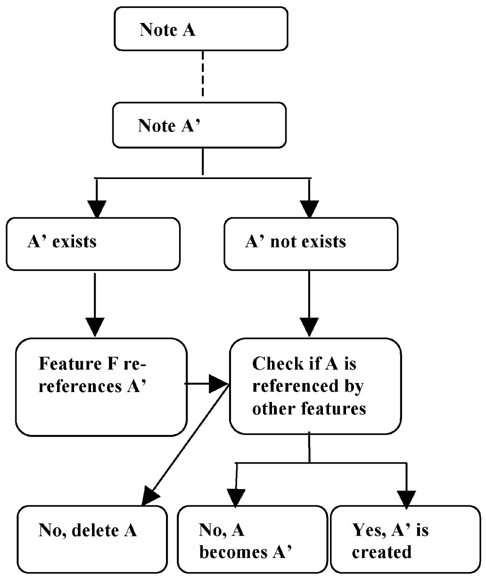

Fig. 4. The algorithm used to implement the modification of a Note in a Feature.

Modification of Primitives of a Feature is a much more complicated process. There are two types of Primitive modifications that are under consideration. They are: Primitive deletion and Primitive coordinate modification. For Primitive deletion, firstly, it is deleted from the Primitive collection of the Feature(s). Secondly, it is checked to see if there are other Features that are referencing this Primitive. If there is no other feature that references it, this Primitive is deleted from its Primitive manager; otherwise, this Primitive is only updated so that the Feature is no longer in the referencing Feature collection of this Primitive. Currently, the DNC implementation provides two options for Primitive coordinate modification, Option one simply forces the coordinate modification of the Primitive; option two checks whether this Primitive is referenced by other features or not. If there are no other references, modification is performed; otherwise, such coordinate modification cannot be processed unless all referencing Features are requesting the same Primitive coordinate modification.

3.4.2. *Feature Deletion*

Feature deletion requires not only the deletion of the actual Feature, but also modification of referenced Primitives. When a Feature is deleted, it has to be deleted from the referenced-Feature collection of Primitives it references. Every Primitive this Feature references must be checked before deleting the Feature. Also, if this Feature is the only Feature that references a Primitive, then this Primitive is deleted from its Primitive manager. Our approach in implementing feature deletion in our DNC model follows the algorithm presented in (Chung *et al.*, 1995).

4. Discussion

It is a challenging task to build an object-oriented framework over a relational file structure. ORVPF demonstrates the promising insights that the VPF data management system can be modeled and implemented with object-oriented technology. Moreover, ORVPF, as a standalone geospatial data management system, is a valuable tool for the GIS community. In this model, the tree based structure of the regular VPF data is transformed to a similar tree structure of classes and objects by using object reference and inheritance features of Java. This transformation provides much easier access to complex VPF data using object-oriented techniques. Another advantage of the ORVPF implementation is that it is compatible with the popular ESRI ArcView package. An ArcView user can import an ORVPF object and work with it as a Shapefile.

There are several areas of development that the next version of the ORVPF will address. In allowing the conversion of ORVPF Features to Shapefile data types we will implement conversion from Shapefiles to ORVPF feature objects which would further aid development of an interface between ArcView and the ORVPF. Such an interface could allow ArcView to directly modify and save ORVPF features without any intermediate conversions required on the user's part. Another important issue that ORVPF needs to address is conversion of ORVPF objects into an XML format. As XML has become the standard for data transportation and ORVPF is being developed in such a way to be accessed by mobile agent working in a distributed environment, occasionally it may be necessary for mobile agents to carry data in the XML format (e.g., for the purpose of inter-operability).

Though the DNC implementation is currently used in our project, it is yet to be completed. Primitive creation and Feature importing are the two major areas that are currently being implemented. Also, in the current DNC implementation, complex feature is not supported because we did not find any complex features in our DNC database and hence did not feel an immediate need for it.

5. Acknowledgement

This work is partially supported by the project "Intelligent Database Agents for Geospatial Knowledge Integration and Management" which is funded by National Imagery and Mapping Agency (DoD), University Research Initiative Award # NMA201-00-1-2004.

References

Arctur, D., E. Anwar, J. Alexander, S. Chakravarthy, M. Chung, M. Cobb and K. Shaw (1995). Comparison and benchmarks for imports of vector product format (VPF) geographic data from object-oriented and relational database files. In *Proc. Fourth Symp. Spatial Databases*. Springer-Verlag, New York. pp. 368–384.

Blaha, M.R., and W.J. Premerlani (1993). Object-oriented concepts for database design. In *5th Annual Software Technology Conference*, Salt Lake City, Utah, USA.

Chung, M., M. Cobb, K. Shaw, and D. Arctur (1995). An object-oriented approach for handling topology in VPF products. In *Proc. GIS/LIS 95*, Vol. 1. ASPRS, Bethesda, Maryland. pp. 163–174.

Cobb, M., M. Chung, H. Foley, E. Petry and K. Shaw (1998a). A rule-based approach for the conflation of attributed vector data. *Geoinformatica*, **2**(1), 7–35.

Cobb, M., H. Foley, R. Wilson, M. Chung and K. Shaw (1998b). An OO database migrates to the web. *IEEE Software*, **15**(3), 22–30.

Department of Defense (DOD) (1996). *Military Standard*: *Vector Product Format*, MIL-STD-2407.

Egenhofer, M., and A. Frank (1989). Object-oriented modeling in GIS: inheritance and propagation. In *Proc. Auto-Carto 9, ASPRS*. Bethesda, Maryland. pp. 588–598.

ESRI Shape Technical Description (1998).

Jacobson, I., G. Booch and J. Rumbaugh (1999). *The Unified Software Development Process*. Addison-Wesley.

National Imagery & Mapping Agency (NIMA) (1997). *Digital Nautical Chart*, MIL-PRF-89023.

Rumbaugh, J., I. Jacobson and G. Booch (1999). *The Unified Modeling Language Reference Manual*. Addison-Wesley.

Scott, K. (2001). *UML Explained*. Addison-Wesley.

Shaw, K., M. Cobb, M. Chung and D. Arctur (1996). Managing the US Navy's first OO digital mapping project. *IEEE Computer*, **29**(9), 69–74.

H. Zhou received his BS degree (1988) in biology from Wuhan University and MS degree (1991) in biology from Xiamen University, P.R. China. He also received his MS degree (2000) in computer science and PhD degree (2004) in scientific computing from University of Southern Mississippi, USA. He once worked in Anabas, Inc. as a software engineer from 2001 to 2002 in California. He is currently an assistant professor of computer science in Saint Joseph College, Connecticut, USA. His research interests include bioinformatics, software engineering, distributed database, digital image processing and software agent technology.

S. Rahimi received his BS degree (1992) in computer engineering from National University of Iran, MS degree (1998) in computer engineering technology, MS degree (1999) in computer science, and PhD degree (2002) in scientific computing from University of Southern Mississippi. He worked as a visiting assistant professor in the University of Southern Mississippi from August, 2001 until August, 2002. He is currently an assistant professor of computer science in Southern Illinois University, Illinois, USA. His research interests include software agents, distributed computing and fuzzy logic.

M. Paprzycki received his MS degree (1986) in mathematics from Adam Mickiewicz University, Poland, PhD degree (1990) in mathematics from Southern Methodist University, Texas. He is currently an assistant professor of computer science in Oklahoma State University. His research interests include software agents, parallel and distributed computing.

Y. Wang received her BS degree (1988) in biology from Wuhan University, P.R. China, MS degree (2003) in computer science from University of Southern Mississippi. She is currently a doctoral student of scientific computing in the University of Southern Mississippi. Her research interests focus on computational biology.

R. Raheel received his MS degree (2003) in computer science from the University of Southern Mississippi. He is currently a doctoral student of computer science in Southern Illinois Univeristy.

M. Cobb received her PhD degree (1995) in computer science from Tulane University. She is currently an associate professor at the University of Southern Mississippi. Prior to that, she was an employee of the Naval Research Laboratory. Her research interests include spatial data modeling, uncertainty in spatial data, distributed object oriented systems.

ORVPF – modelis ir jo DNC realizacija

Hong ZHOU, Shahram RAHIMI, Raheel AHMAD, Marcin PAPRZYCKI,
Yufang WANG, Maria COBB

VPF formato (Vector Product Format) duomenų bazėse geografiniai duomenys saugomi reliaciniu pavidalu, o atskiri VPF failai sutvarkyti hierarchiškai į katalogų medžio struktūrą. Prieigą prie VPF duomenų ir jų pakeitimus yra sudėtinga realizuoti, nes duomenys išskirstyti skirtingose lentelėse. Straipsnyje pasiūlytas objektinis modelis, skirtas VPF duomenų bazei tvarkyti, kuris leidžia lengvai prieiti prie VPF duomenų ir juos automatiškai atnaujinti. Modelis yra suderinamas su ESRI Arcview programine įranga. Šis modelis buvo realizuotas Java kalba ir pritaikytas darbui su DNC (Digital Nautical Charts) duomenų baze.