TOWARDS AUTOMATION OF UPDATE PROPAGATION IN VECTOR-STRUCTURED SPATIAL DATABASES

Olajide Kufoniyi
Federal School of Surveying
P.M.B. 1024, Oyo, Oyo State, Nigeria
Email: kufoniyi@skannet.com.ng

IC Working Group IV/III.1

KEY WORDS: Data models, GIS, Revision, Spatial databases, Topology

ABSTRACT

In geo-information production, currency of the data plays a very important role in the reliability of the information. Database updating is an important aspect of GIS development because the terrain objects represented in the database are generally not static in time; thus the database should also respond to such object dynamics through "consistent" updating. By consistent, we mean that the structural and semantic constraints of the database must be enforced after each update. The paper focuses on the maintenance of data currency, while monitoring and enforcing data consistency. Algorithms for consistent update propagation in vector-structured databases are presented in the paper. The structural and semantic constraints for the vector data model used are formalised using topologic relationship as a tool for detecting and resolving integrity violation during updating of the database.

1.0 INTRODUCTION

Updating of geo-information has evolved from simple graphic map revision using analogue method and equipment, through digital map revision with the aid of computer system but using a spaghetti model and with intensive involvement of the human operator. The goal should be to update a structured database in a GIS with a high degree of automation, but this last stage is still very much confined to the research and development domain.

In this digital era when many processes are being automated to take advantage of technologic developments, updating has become a more complex operation. Developers are now focusing attention on the complete automation of updating procedures starting with change detection through the extraction of changed data, to database updating including editing and quality assessment. This paper focuses on the database updating aspect, addressing the issues of update propagation, topologic editing and consistency enforcement in vector-structured spatial databases. In this context, database updating denotes an operation involving (a) insertion of new data into the database, (b) modification of existing data, and/or (c) removal of obsolete data from the database. In a topologic vector data model, the representation of objects includes their interrelationships; thus an update on one object may affect another object(s), causing an inconsistent state. To forestall such inconsistency, an update "propagation" must be effected when any update operation is carried out. Intuitively, update propagation means the system should identify all objects that are affected by a single update and modify them or warn the human operator to ensure consistency.

The work is based on a 2.5D vector data model in which three generic types of spatial object are represented: point, line and area objects. Changes in the database will therefore result from the dynamics of these objects. Each terrain object is uniquely defined by its geometric and its thematic attributes. The dynamics of an object can thus be grouped into two basic aspects: thematic changes and geometric changes. The updating of the two aspects is presented in the paper with emphasis on the geometric aspects. The procedures for monitoring and enforcing consistency after each update are also presented.

2.0 THE WORKING DATA MODEL AND BASIC CONSTRAINTS

2.1 Data Model

To serve as framework, a conceptual data model was developed to represent a multi-valued terrain abstraction, especially when frequent spatial analyses across many map layers is envisaged. The proposed model is an object-based 2.5D data model for multi-valued vector maps (DMMVM) (Kufoniyi, 1995). Here, a vector map refers to a database representation of the terrain situation as points, lines, areas and bodies in which positional data are given in the form of coordinates of isolated points and the end-points of line segments.

A multi-valued vector map then refers to the vector-based representation of terrain objects from multiple map layers whereby two objects of the same geometric type may be spatially coincident. Two objects are said to be spatially coincident if they (partially) overlap in space. In this model, positions of objects are defined in a 3D metric space but embedded in 2D topologic space, i.e., a 2.5D model. This means that only surfaces of objects are represented such that a pair of X and Y coordinates must have a single Z value. The model was based on the formal data structure (FDS) for single-valued vector maps (Molenaar, 1989). In the 2D FDS, each object has a thematic component and a geometric component. In the thematic domain, the objects can be grouped into thematic classes in which each class has a specific attribute structure (Molenaar, 1993). In the geometric domain, the object types points, lines and areas are distinguished for a 2D terrain description, subject to a constraint that two objects of the same type may not be spatially coincident. The three object types are then completely described by a common set of two types of geometric elements (arc and node). The FDS was extended in this work to allow objects of the same type to be spatially coincident, to facilitate the use of a single structure for the representation of multi-layer geo-data.

A geometric data type, the m-dimensional container, or simply m-container, where $m \in \{0,1,2\}$ was introduced to model spatial coincidence among objects of the same geometric type. The 0-container represents spatially coinciding J point objects from J layers; a 1-container represents (part of) K line objects from K layers and a 2-container represents (part of) L area objects from L layers, where L is the maximum number of layers and J and K may each be less than or equal to L. By introducing the container data type, overlapping sections across the layers are uniquely identified such that they have their own individual geometric data and non-spatial data, apart from those inherited from the overlapping objects. They can then be maintained and manipulated by the DBMS just like single objects thereby improving spatial analyses in GIS.

Using graph theory as a mathematical tool, the three container types are then represented by the topologic primitives arc and node. A node defines one 0-container and/or beginning or end of an arc, while an arc defines (part of) one 1-container and/or (part of) a boundary of a 2-container. The arc is defined by one start node and one end node, and a node is defined by a coordinate triplet X,Y and Z. The geometry of the map is thus represented by a planar graph G(N,A) where A is the set of arcs of the graph and N is the set of nodes. Each m-container C, $m \in \{0,1,2\}$, is then a subgraph of G such that the geometry of C is represented by $G_c(N_c,A_c)$ where $N_c \subset N$ and $A_c \subset A$. For a 0-container, $A_c = \emptyset$. Thus eight basic geometric data types are defined to represent geo-data from multiple map layers, namely area, line, point, 2-container, 1-container, 0-container, arc, and node.

2.2 Basic Topologic Relationships

The basic topologic relationships among the three object types area, line and point, and among the geometric primitives arc and node were formalised (see Kufoniyi, 1995) using the 9-intersection model (Egenhofer and Herring, 1992) as interpreted for graph-structured vector maps (Molenaar et al, 1994). Algorithms were then defined for detecting the occurrence of any of the elementary relationships for any object combination. The algorithms can be translated to topologic operators and used as a tool for detecting violation of and enforcing geometric constraints. To maintain consistency, the topologic operators will serve as detectors of inconsistencies. The return value of the operator will trigger the relevant rule that will enforce consistency if violation occurs. The rules that enforce the geometric consistency were also presented in (Kufoniyi, 1995). The relationships were uniquely coded as the decimal conversion of the binary number that corresponds to its relationship e.g., relation r511 (111111111 in binary digits) represents a tuple where all nine intersections are non-empty. The nine intersections are derived by evaluating the set intersections between the boundary, interior and exterior of one object and the boundary, interior and exterior of the second object. When the intersection is empty, it is interpreted as bit value 0 while a non-empty intersection is interpreted as bit value 1.

2.3 Structural and Semantic Constraints

Update operations must be performed on the database in a manner that maintains the structural and semantic constraints of the database. The structural constraints relate to the geometric structure of the data model while the semantic constraints relate to the application making use of the model, i.e., application-dependent integrity rules.

2.3.1 Structural constraints

The structural constraints and the rules for their enforcement during updating were analysed under five headings:
(a) Consistency rules for the geometric primitives arc and node.
(b) Consistency rules for the geometric structure of m-container types.
(c) Consistency rule for the planarity of the map (Euler constant)
(d) Consistency rules for the structure of object types
(e) Consistency rules for the functional relationships among the data types (node, arc, m-container, and object).
Some of these are summarised below (details can be found in Kufoniyi, 1995).

Geometric Primitive Constraint #1 (GPC_1): Nodes must not overlap (i.e. no duplication of nodes)
Geometric Primitive Constraint #2 (GPC_2): Arcs must not intersect (node must be created at the point)
Geometric Primitive Constraint #3 (GPC_3): Arcs must not overlap
Geometric Primitive Constraint #4 (GPC_4): An arc must be defined by only two adjacent node.
0-Container Constraint #1 (0CC_1): The node defining the geometry of a 0-container exists.
1-Container Constraint #1 (1CC_1): For each 1-container, a simple and elementary path must exist.
1-Container Constraint #2 (1CC_2): The length of the path in 1CC_1 must be ≥ 1.
2-Container Constraint #1 (2CC_1): For all $n_i \in N_L$ $degree_L(n_i) = 2$
Point Object Constraint: The 0-container defining the object must be created.
Line Object Constraint: A simple and elementary path must exist between the beginning and end of the chain of 1-containers defining the object
Area Object Constraint: The 2-containers representing the object must be connected and non-overlapping.
Euler constant for (connected or disconnected) planar graph: $n \quad a + f = k + 1$ must be enforced where k = no. of component graphs, n = no. of nodes, a = no. of arcs and f = no. of faces (or polygons).

Constraints for Thematic Component of Objects

The thematic data of individual objects are application-dependent. However, the following mandatory constraints must be enforced during updating:
(a) Cyclic classification hierarchy is not permitted.
(b) The classification must be complete, i.e. all objects must be classified.
(c) The classes must be mutually exclusive, i.e., each object must belong to only one class in each layer, but if one object appears in more than one layer, the object can be classified in the other layer as well.

2.3.2 Semantic constraints

In general, semantic constraints are application-dependent and they are both spatial (e.g., two parcels must not overlap) and non-spatial (e.g., a lessee cannot transfer his right on a leasehold beyond the period of the lease). In the spatial domain, they are mostly a group of forbidden relationships between pairs of objects. These constraints are also important in spatial databases because the database may be geometrically consistent but semantically inconsistent. For instance, it is topologically in order for a line object to cross an area object if the geometric constraints are fulfilled. But this relationship may or may not be consistent, depending on the *meaning* of the two objects.

This makes it difficult to provide a generic algorithm for resolving semantic constraints. But a monitoring procedure can be formulated for those that are topologic in nature, especially between pairs of objects. The monitoring strategy being proposed here is to use topologic relationships as alerters. The strategy is for only semantic constraints that are geometric in nature, and it is based on the assumption that the database is structurally consistent. Some of these constraints are for individual objects, e.g., size and shape constraint. These can be monitored and enforced by the editing routines of the data acquisition system or with functions defined as part of the topologic editor. A scheme for monitoring the constraints between object pairs during update propagation is as follows:

- Translate the constraint to a set of expected topologic relationships ($\{E\}$) which will maintain the consistency of the map. In practice, this can be done for all semantic constraints (topologic) in the application and stored in the database.
- On-line derivation of the actual relationship (A) by the system during insertion using the topologic relationship operator (see Kufoniyi, 1995)
- System checks for membership of A in the set E
- If $A \notin \{E\}$ then violation has occurred and the system should warn the user. The user can then interactively resolve the violation either by using pre-defined rules (if applicable) or by taking any other action.

3.0 UPDATE PROPAGATION

The updating of the thematic and geometric aspects of a spatial object are analysed in the following sections with emphasis on the geometric aspects. Examples of the procedures for propagating the geometric updates are also presented in pseudo codes.

3.1 Updating Thematic Data of an Object

Here, the thematic attributes of an object being application-dependent are restricted to its thematic class label. Thus only the class label will require updating, as in the reclassification of an object (e.g., an area object "parcel" changing from class "vacant" to class "built-up"). It is assumed that the DBMS of the implementation platform will have facilities for simple updating in which propagation is not required. When a new object comes into existence, simple insertion of its class label (and other thematic attributes if any) can easily be done by the DBMS tools. The same holds when an object comes to the end of its life span, in which case its thematic information is simply deleted.

However, if hierarchic classification is implemented, then certain rules must be observed when updating the thematic classes. For class creation, the superclass has to be created first because its attribute structure will be inherited by its subclasses. The consistency rules for thematic data should also be enforced. The insertion of the attribute values of the individual objects belonging to the class can then be effected by the DBMS commands. When it is necessary to remove a class, an essential rule is that a superclass cannot be deleted unless all its subclasses have been deleted. And in general, no class should be deleted unless all its instances have been deleted.

3.2 Updating of Geometric Components

Changes in the geometric aspects of an object may lead to changes in the topologic relationships among the objects in the database. Thus care has to be taken to ensure integrity of the database during geometric updating of objects. In the work reported here, it is assumed that the boundaries of objects are well defined. The objects are also assumed to be correctly geo-referenced, using the same coordinate reference system and the same resolution.

In the working data model, the geometric characteristics of objects are represented by five data types namely, 2-container, 1-container, 0-container, arc and node. Thus geometric changes directly imply changes in the five data types. The updating can be analysed at three levels. The lowest level concerns the geometric primitives arc and node whose updating is triggered by an updating request from higher level data types by propagation. The next higher level concerns the m-containers, $m \in \{0,1,2\}$, whose updating will also be triggered by propagation from the updating at the highest level. At the highest/ user level are the individual terrain objects of types point, line and area, which actually trigger the database updating.

The expectation in automated database updating is therefore that the system receives an updating request at the object level, together with the necessary input data, and performs the propagation from the object level to the level of geometric primitives. In addition, for the database to be geometrically consistent, all the constraints defined in section 2.2 must be satisfied after each updating. Thus (automated) procedures must be provided to propagate the update while enforcing the constraints. In addition, in a multi-user database, locking device must be provided to make it impossible for any other user to interact with the database when update propagation is taking place. In the following sections, the update propagation involving the basic data types (area, line, point, 2-container, 1-container, 0-container, arc and node) are analysed at their respective levels, starting from the lowest level (node and arc), with each level serving as "building block" for the next higher level.

3.2.1 Updating of Geometric Primitives

At the lowest level of the update propagation path are the two geometric primitives. Their updating will normally be an indirect operation triggered by geometric updating at the next higher level. The consistency rules defined for the geometric primitives in section 2.2 should be enforced during updating of the primitives to maintain geometric consistency. The updating operations (resulting from insertion, deletion or modification) involving the two primitives were analysed. They will serve as the elementary operations into which the updating of the higher level data types can be decomposed. Example is given below for the insertion of an arc; other cases can be found in details in Kufoniyi (1995).

Inserting a new arc:
To maintain the integrity of the database when a new arc is inserted, the geometric primitive constraints GPC_i, $i \in \{1,2,3,4\}$ should be enforced through the consistency rules defined for them. For instance, when a new arc is inserted, the system must evaluate the arc's topologic relationship with each of the existing primitives in the database and apply the corresponding consistency rule for any relationship that violates geometric consistency.
The following is the algorithm Insert_Arc (in pseudo code) for inserting a new arc.

```
Algorithm Insert_Arc:
begin
    get new arc $a_i$ (and its properties: id, coords, 1-container)
    assign node numbers for its start and end nodes
    do while exists $n_j \in N|Degree(n_j)=0$ /* N = existing nodes with degree zero */
        determine Relation($a_i$,$n_j$) /* by coordinate geometry */
        if Relation($a_i$,$n_j$) = r092
            do GP_Rule_2
        endif
        if Relation($a_i$,$n_j$) = r284 /* by comparing coordinates */
            do GP_Rule_1
    end do while
    do while exists $a_j \in A$ /* A = existing arcs */
        determine Relation($a_i$,$a_j$) /* using coordinate geometry */
        Case r063 do GP_Rule_3
        Case r095 do GP_Rule_4
        Case r159 do GP_Rule_5
        Case r179 do GP_Rule_6
        Case r220 do GP_Rule_7
        Case r255 do GP_Rule_8
        Case r287 do GP_Rule_9
        Case r400 do GP_Rule_10
        Case r435 do GP_Rule_11
        Case r476 do GP_Rule_12
        goto end
    end do while
    determine new values for left and right relationship
    store $a_i$
    insert start and end nodes of $a_i$ using Insert_Node function
end
```

3.2.2 Updating of the m-Containers
At the next level of the update propagation path are the three m-containers: 2-container, 1-container and 0-container. Their updating will normally be an indirect operation triggered by updating at the highest level, i.e., of individual terrain objects. Like the geometric primitives, the updating of the m-containers should be automated as much as possible while enforcing the consistency rules defined for them. The updating operations involving the three types of m-container are analysed in Kufoniyi (1995) with the example of insertion of a new 2-container given below.

Inserting a new 2-Container:
To satisfy completeness of incidence in the DMMVM as a 2.5D vector map, all 2D segments (faces) must be classified, i.e., every closed polygon must be (part of) a 2-container; thus the addition of a new 2-container implies modification or deletion of an existing one. This means that insertion of a 2-container involves a combination of insert, modify and/or delete operations. It is thus a complex operation that may still require the use of semantic information as well as human intervention. For example, in a single-valued vector map, the new 2-container will be spatially coincident with one or more existing 2-containers. If after analysing the semantic information of those 2-containers it is found that they are part of area objects classified as "vacant" (or "unclassified"), then the update propagation can proceed. But if any of those 2-containers is part of a "real" terrain object then the human operator's decision is required as to whether that existing object should be deleted or modified. Also, in the multi-valued situation, it must be determined first if the overlapping 2-containers are class-compatible, e.g., a cadastral parcel

cannot overlap a lake. Class-incompatibility can be predefined in a look-up table (LUT), for example, such that the system consults the table when a new 2-container is being inserted and, in general, during overlay computation to ascertain if the overlapping 2-containers represent compatible objects. The same situation holds for 1-containers and 0-containers. Detecting and forestalling such incompatibilities can be handled in two ways: (1) at the beginning of the update propagation or overlay computation, (2) immediately after the update/overlay computation. The first approach implies that the checking routine should be part of the algorithm being used for the computation, but it will require more interaction with the human operator (except if an LUT is provided) during the updating. The second approach allows the use of any available overlay computation algorithm, after which the consistency module can verify compatibility. The algorithm provided here uses the first approach, but this part can be transferred to the end during implementation if the user so decides.

If the insertion is allowed, the existing 0-containers topologically contained by the 2-containers should be identified (computationally) in order to make this relationship explicit. The system should continue the update propagation by inserting one arc of the 2-container at a time using the Insert_Arc algorithm. Some existing arcs may need to be deleted, while some may require modification.
The generalised algorithm for inserting a new container is defined below.

```
Algorithm for inserting a 2-container: Insert_2-container
begin
        do for each new 2-container c2
        get $G_{c2}(N_{c2},A_{c2})$
        do 2CC_Rule_1 /* consistency rule for 2-container */
        select all existing 2-containers OC2 for which oc2 $\in$ OC2| Relation(c2, oc2) $\in$ {r179, r220, r400, r435,
        r476, r511} /* using computational geometry e.g., any polygon intersection algorithm */
                for each oc2
                select area object $AO_j$ for which Partof[$AO_j$,oc2]=1
                if map is single-valued
                        determine thematic class of $AO_j$
                        if class $\notin$ {"vacant", "unclassified"}
                        notify user /* decision $\in$ {interactive editing?, next oc2}  */
                else next oc2
                        endif
                else
                  if map is multi-valued
                        determine compatibility between $AO_j$ and new area object represented by c2 /* LUT or
                        user decision */
                        if class incompatible
                        notify user /* decision $\in$ {interactive editing?, next oc2} */
                else next oc2
                        endif
                  endif
        display all 2-containers EC2 $\ni$ $\forall$ ec2 $\in$ EC2 is Relation(ec2,c2) $\in$ {r179, r220, r400, r435, r476, r511}
        perform interactive updating /* modify neighbouring 2-containers which are affected by the new one and
        insert arcs and nodes of c2; using update algorithms of lower level data types */
        for each ec2 $\in$ EC2 where Contains[ec2,c0]=1 /* c0 = an existing 0-container */
                determine the new 2-container nc2 $\ni$ Contains[nc2,c0]=1
                if nc2 exists, store relation as property of nc2 and property of c0
        next ec2
next c2
end
```

3.2.3 Updating the Elementary Objects

The propagation of geometric update initiated by the updating of a terrain object can be seen as a message carried by the object concerned into the database. The type of object and type of update will signify the message type and thus the actions to be performed by the system. This can be depicted by the following expression 3.1.

$$UP(OT, UT) \rightarrow \{PR_i : LPUT_i\} \oplus \{SR_k : LPUT_k\} \qquad (3.1)$$

where UP = Update propagation message
OT = Object type ($\in$ {point, line, area})
UT = Update type ($\in$ {Insert (I), Delete (D), Modify (M)})
$\rightarrow$ = Triggers
PR_i = Primary receiver (affected) data type (i.e., mandatory and definitely affected)
$LPUT_i$ = List of (alerted) propagated update types ($\in$ {I,D,M}) for data type i
SR_k = Secondary receiver (consulted/probably affected) data type k
$\oplus$ = Possibly affected path

Updating of an area object is given below as an example; others can be found in Kufoniyi(1995).

Updating Area Objects

Like point and line objects, an area object has two basic properties: thematic and geometric. Updating of the thematic data can be taken care of by DBMS commands during implementation. In the geometric domain, however, inserting, deleting or modifying a single area object may lead to further operations on other existing neighbouring area objects; thus updating of a single area object is a complex operation. The propagation of updating operations of inserting, deleting and modifying the geometric data of a single area object were defined in Kufoniyi (1995) with example of insertion of a single area object given here.

Inserting an area object:

The completeness of incidence constraint of the model implies that all two-dimensional objects must be classified (even if as class "unclassified"). Thus inserting an area object will always involve a combination of insert operations (for the new object) and possibly deletion and/or modification of existing area objects that are spatially coincident with the new object. The insertion of the area object's geometric data can be decomposed into insertion of a 2-container and the algorithm for inserting a 2-container used to insert this and propagate the insertion of its arcs and nodes. The insert algorithm for the 2-container already has facilities for modifying neighbouring 2-containers if they are affected. The propagation chain for inserting an area object is given in expression 3.1.1 below.

$$UP(AO, I) \rightarrow \{AO: I,D,M;\ 2C: I,D,M;\ AR: I,D,M;\ N: I, D\} \oplus \{0C: M\} \qquad (3.1.1)$$

This indicates that apart from inserting the area object, other area objects may have to be deleted or modified. The insertion of the new area object will then trigger the insertion of new 2-containers (2C) with possible deletion or modification of some existing ones, and similarly for arc (AR) and node (N) in the propagation chain. Some existing instances of the 0-container type may have to be modified if they are topologically contained by the new 2-containers of the area object. The updating operations to be performed on the existing area objects that are affected by the insertion of the new object cannot be determined by the system. For example, in a cadastral database, if a new parcel, represented as area object, overlaps with an existing one, the system can only notify the user while the user decides whether to reduce the existing one or the new one as one of the numerous possible decisions. This aspect has to be done interactively. The propagation of the insertion will proceed by regarding the area object as a new single 2-container. The following algorithm is proposed for inserting an area object.

```
Algorithm Insert_Area:
begin
        get properties of area object AO /* id, coords, thematic class, layer, */
        insert thematic data
        assign 2-container identifier → c2
        do Insert_2-container(c2)
        enforce AO_Rule_1
end
```

3.2.4 Handling Updating of Multiple Objects

Updating multiple objects of the same generic type:

In this case, the operation is handled by treating one object at a time, within a loop, using the appropriate algorithm.

Updating multiple objects belonging to different generic types:

These involve one or more point objects and/or one or more line objects and/or one or more area objects. The following strategy can be used to handle the operation:

- insert/delete/modify all the point objects (if any), one at a time, using the relevant algorithm for point object
- insert/delete/modify all the line objects (if any) using the relevant algorithm for line object
- insert/delete/modify all the area objects (if any) using relevant algorithms.

By grouping objects of the same type together, the same operation (e.g., Insert_Area) can be performed in a loop instead of changing from one operation to another in an ad-hoc manner.

4.0 CONCLUSION

In geo-information production, the cost of data collection has been said to be about seven to 10 times more than the cost of the hardware and software needed to establish the database (Peled, 1994). Thus it is very important that the accuracy and currency of the data should be reliable, such that the purpose for setting up the database can be fulfilled with profitable cost recovery. This paper aimed at contributing towards achieving this by providing algorithms for consistent automated update propagation.

Although geo-information updating includes change detection, data collection and database updating, the focus in the paper was on automated database updating under the assumption that the necessary changes have been detected and captured in readiness for input into a vector-structured database. Ideally, the defined algorithms should be translated into computer modules within an existing DBMS. However, since most of the operational DBMS are not capable of accepting user-defined rules and data types, the algorithms may have to be programmed in a high-level language and then coupled with the DBMS during implementation.

It should be noted that, although the final goal is to automate spatial database updating, some aspects will still have to be done interactively (i.e., require human intervention), aided by graphic visualisation. This limitation is related mostly to the area objects with respect to the correct decision to take when modifying the neighbouring area objects (of the same layer) effected by an area object that is being updated. After the update propagation, the consistency rules defined in section 2.2 should be verified again and certainly at regular intervals.

For implementation, the DMMVM has been translated into two prototype database structures, namely relational and object oriented (see Kufoniyi, 1995). If the relational prototype is used for an implementation, then the update propagation algorithms and consistency rules would have to be handled by a high-level programming language and coupled with the RDBMS since most operational RDBMSs are not capable of handling user-defined rules. If the object-oriented (OO) prototype is chosen, the algorithms have been defined as class methods, which should then be programmed using the programming language of the OO system.

Some of the proposed algorithms have been experimented in automated update propagation in single-valued vector-structured database (Kufoniyi et. al, 1993). They were also implemented in more details in multi-valued vector-structured database using a combination of C++ and Postgres DBMS. The algorithms were translated into C++ programs, while Postgres served as the extended RDBMS. Details of this experiment can be found in Kufoniyi (1995).

REFERENCES

Egenhofer, M.J. and Herring, J.R., 1992. Categorizing binary topological relationships between regions, lines and points in geographic databases. Technical report, Department of Surveying Engineering, University of Maine, Orono, 59p.

Kufoniyi, O, Molenaar, M. and Bouloucos, T., 1993. Topologic editing of relationally structured single-valued vector maps. ITC Journal 1993-4, pp 332-339.

Kufoniyi, O., 1995. Spatial coincidence modelling, automated database updating and data consistency in vector GIS. ITC Publication Series Number 28, ITC, Enschede, 206p.

Molenaar, M., 1989. Single valued vector maps a concept in GIS. Geo-Informations-Systeme, 2(1), pp 18-26.

Molenaar, M., 1993. Object hierarchies and uncertainty in GIS or why is standardisation so difficult. Geo-Informations-Systeme, 6(4), pp 22-28.

Molenaar, M., Kufoniyi, O. and Bouloucos, T., 1994. Modelling topological relationships in vector maps. In: Advances in GIS research Proceedings, 6th International Sysmposium on Spatial Data Handling, 1, pp 112-126.

Peled, A., 1994. Revision of digital maps and GIS databases. In: Int. Arch. of Photo. and Rem Sen. Vol. 30, Part 4, pp 268-272.