Managing Multiple Representations of Georeferenced Elements

C. Bauzer Medeiros
DCC-IMECC-UNICAMP
CP 6065
13081-970 Campinas, SP, Brazil
cmbm@dcc.unicamp.br

M.J. Bellosta, G. Jomier
LAMSADE
Université Paris-Dauphine
75775 Paris, France
name@lamsade.dauphine.fr

Abstract

This paper presents a framework for the management of multiple representations of georeferenced elements in a GIS environment. This solution is presented from a database perspective, and is based on extending the DBV *version approach with view facilities. Given this environment, users can build consistent views of their modelled world, creating and combining different representation frameworks. The environment dissociates the logical dimension from the physical dimension, liberating users from concerns about implementation details.*
Keywords: database, versions, views.

1 Introduction

Geographic Information Systems (GIS) are software that manipulate *georeferenced data* – data about geographic phenomena associated with their spatial relationships and location on the terrestrial surface. The term *georeferenced entity* in this paper refers to any real world element which is georeferenced (whether a human artifact or a natural phenomenon).

The definition of proper database support for GIS covers a vast range of issues, from the user interface level down to the storage system level [10, 20]. Some problems fall into the domain of database management systems (DBMS) – e.g., query optimization – whereas others are handled at the end-users' side – e.g., application modelling. Some issues, however, present a challenge to be met at different levels by both end-users and database researchers. This paper is concerned with one such issue, namely, that of *multiple representations*. This corresponds to the need of maintaining, in the GIS, different descriptions of the same georeferenced entity.

As pointed out in [26], this issue occurs in every application domain, since it results from the fact that the same problem can be modelled in different ways (thereby producing distinct database schemas) and the same entity can be stored according to distinct views and standards (thereby resulting in different instances on a database).

In the case of GIS, this problem is aggravated by the fact that geographic reality is multi-faceted and the way it is perceived depends on the user needs. This corresponds, from the end-user side, to handling the issues involved in modelling all facts in the world, discretizing and storing them into data files. This often requires that different representations be handled simultaneously incurring thereby into problems such as establishing links and consistency among representations of a given area.

From a database point of view, handling multiple representations of a given georeferenced entity must be treated at several levels: interface (presentation), logical modelling, schema specification, query processing, storage structures. At all these levels, several open issues arise, concerning the management of distinct representations: data replication, consistency maintenance, entity identification and multiplicity of behaviors of a given entity.

This paper presents a database-centered environment to solve some of these problems. This approach is based on extending the specific version management approach [3, 4], called DBV, with views.

We face the problem from a database perspective and we are not concerned with how the user has arrived at the different representations. Our main goal is to analyze the problem of managing these representations, once they have been stored in a database.

The remainder of this paper is organized as follows. Section 2 briefly discusses on some issues involved in the management of multiple representations, and comments on previous research in the area of database support for them. Section 3 introduces an example that is going to be used throughout the paper to illustrate our solution. Section ?? briefly outlines the DBV approach, extending it with views. Section 5 presents our solution for database support for multiple representations, which is based on the DBV approach with views. Finally, section 6 presents conclusions and directions for future work.

2 Multiple Representations - a Brief Overview

The problem of multiple representations in geoprocessing is related to the fact that users must model and store data about the world for different types of application. This requires sampling and discretizing reality, and therefore implies several open issues, notably those involving data quality [5, 8]. As remarked in [27], even though the state of a given entity may remain unchanged, its representation and relationships with other entities may vary according to user perspectives.

[27] defines diversity in representations as a result from variations in users' requirements. Though this is certainly true, we prefer to emphasize that different dimensions exist through which representations may evolve – e.g., resolution, time, model, user point of view, and others. We analyze briefly some of these possibilities.

Resolution refers to the level of abstraction with which the world is represented. For instance, a road network may be considered as an atomic entity (high level of abstraction) or as a set of sub-entities, containing crossings and bifurcations (lower level of abstraction). Resolution variation, as pointed out by [24], is often considered as synonym to scale change. However, we prefer to see scale changes as an implementation for multiple resolutions.

Varying representations along the *time* dimension corresponds to the temporal evolution of an entity – e.g., modifying the road network, adding and eliminating road branches. Here, even when the resolution is kept constant, one entity (older representation) may be replaced by different entities (after update).

The issue of representations in *modelling* of data is related to the nature of phenomenon being modelled. The canonical example is the discussion between field and object representations [6]. Topography data for building the road, for instance, is commonly represented using a field (continuous) model, whereas the road itself obeys a network (object) spatial representation.

Multiple representations reflecting users' *points of view* occur when distinct users want to represent a given entity according to their needs: the road is represented differently for cartographic, engineering, and traffic flow optimization purposes. These different representations will exist even if the nature of the phenomenon does not change, the time is fixed and so is the level of resolution.

We stress some commonalities among these representation variations. Some representations can be *computed* from existing data (e.g., in cartographic generalization [15, 21]) while others require *storage* of distinct data records (e.g., in temporal evolution). *Materialized* (as opposed to *computed*) representations correspond to data that have been actually stored in the database. In database terms, computed representations can be compared to views, while materialized representations may be treated as versions (of real world geographic elements).

Our solution combines both versions and views, thereby facilitating the joint management of both types of representation. It extends the GIS related work of [18, 19], presenting and detailing a framework which allows the management of multiple representations through versions and views. This solution is based on the DBV version approach [3],[9].

Versions are a means of storing different states of a given entity, thereby allowing the control of alternatives and of temporal data evolution [14, 13, 12, 16]). Views in databases are usually defined as the result of a query. Views may be stored (materialized), but in general it is understood that versions correspond to stable data, whereas views are generated from stable data and are usually temporary. The issue of views in relational databases is understood, but in object-oriented databases is still a matter of research [1, 17, 25]).

Existing view and version mechanisms, if taken in isolation, are not sufficient to allow satisfying GIS users requirements for multiple representations. This is true, for instance, in the multiple representation case. So our solution composes views and versions in order to satisfy representation management.

3 Illustrative Example

This section describes a short example, which will be used throughout the text to illustrate the proposed solution, as well as the theoretical framework adopted in the paper. We just provide enough detail to let the reader understand the mechanism presented and omit the complete database schema specification, given paper space limitations.

We adapt an example described in [24]. Consider a region with a set of roads which needs to be repre-

sented differently according to two user group requirements: cartographers and traffic flow engineers. A cartographer will need a detailed description that will respect distances, relative positions and spatial relationships among the roads and neighbor geographic accidents (e.g., hydrography, vegetation). Traffic engineers require manipulating the road data using a directed graph representation with weights, but without concern for scale or material. Furthermore, they need details about traffic flow (e.g., speed limits).

3.1 The data model

We adopt a multi-level view of georeferenced data modelling, similar to [2]. The world is modelled as a set of object classes, called *georeferenced classes*. A *georeferenced object* is an instance of a georeferenced class. It has a spatial and a non-spatial (descriptive) component. The spatial component describes the object's geometric and topologic properties, varying according to the representation chosen. A georeferenced object may be associated to different representations according to application purposes. Each representation corresponds to a distinct database class. The advantages of dissociating an entity's representations from the entity itself are discussed in different contexts [23, 27, 11, 24].

Representations are built from a set of primitive classes to describe object's geometries. These primitive classes are rooted at a class called *Geometry*, which has subclasses *Point*, *Line*, *Polygon*. These classes are the basis for building complex representation descriptions.

A georeferenced class is typically defined in a high level way as

```
Class Geo-class
type tuple (non-spatial1: t1,
            non-spatial2: t2,
            non-spatialn: tn,
            geo-components: set(Geo-class),
            location: Geometry
method public equal(Geo_class o1) : Boolean,
     public display(),
     public element( Geo_class o1): Boolean),
      ...
end
```

The *non-spatial* components correspond to non-spatial attributes; *geo-components* are objects from other georeferenced classes that compose the *Geo-class*; *Location* contains the representation descriptions, which may vary in scale, time frame etc. The *element* method checks if a georeferenced object is composed of the *o1* geo-component.

3.2 Example database schema

In our simplified world, we assume that the only georeferenced classes of interest are *Road, Vegetation, Soil, City*. The database must also contain a class called *Network* which is defined as a list of lines and over which are defined methods and constraints appropriate for specifying networks and their topologic properties. Class *Network* has two subclasses, *Cart-representation* and *Weighted-graph*. The first one is used for representing the road network for cartographic purposes, and contains methods such as *change_scale* and *change_weight*. The second is used for the traffic flow requirements, and contains methods such as *compute_flow* and *change_weight*. The *change_weight* method is a method of *Network* class redefined on its two subclasses.

Vegetation and *Soil* locations are described in terms of *Polygon* geometries. *City* is represented both as *Polygon* and *Point*, depending on the scale desired. Scales are managed according to class *Scale*.

The *Road* class, which is manipulated by both types of users, would be defined as:

```
Class Road
    type tuple (road_name: string,
       loc: list(tuple(s:Scale, rep:Network)))
    method public length (r:Road.loc): integer,
       public change_speedLimit(limit :Speed)
end
```

Method *length* computes the lenght of a road and is computed based on a representation and scale. Polymorphism allows different representations to appear in the list of locations *loc*. For instance, the *change_speedLimit* method applied to an object of class *Road* will be implemented as a *change_weight* invocation over its representation which may be an instance of the *Weighted-graph* class or the *Cart-representation* class.

4 The DBV approach

The database version approach has beeen developed to provide a way to implement efficiently versions in general purpose DBMS. One of its main features, which distinguishes it from other approaches, is the distinction between the logical level, seen by users, and the physical level, managed by the system and hidden from users.

At the logical level, the multiversion database is seen as a set of *DataBase Versions*, or DBV, identified by a database version identifier and composed of versions of object and class. Each DBV represents a consistent description, or version, of the modelled world, and may be associated with one or several dimensions corresponding

to the versioning semantics of the application. As a version of object/class may have the same value in several database versions, the conventional concept of "version of object/class" is split in two new concepts: a *logical version* of an object/class represents the version of this object/class as it is seen by users in a given database version; a *physical version* of object/class is used by the system to store the value of all the logical versions of this object/class having the same value. A database version is composed of one logical version of each object/class, identified by a pair (object/class identifier, database version identifier). DBVs are created by logical copying, called *derivation*, and may evolve independently from each other. This means that a DBV may be updated or deleted without side effect on other DBVs. Moreover user may work simultaneously on several DBVs : comparing their content, copying logical versions from one DBV to another one, etc.

At the physical level, the version mechanism, described in [3], allows to manage efficiently as many DBVs as needed.

Such a multiversion database enables the simultaneous management of distinct alternative scenarios, as well as handling georeferenced feature evolution through time. It can be used to handle some aspects of multiple representations (notably through time - [19]).

In the upper part of Figure 1, the multiversion database is composed of four database versions : Dv_2 is derived from Dv_1 by versioning the schema; Dv_3 is derived from Dv_2 by versioning the data.

5 Support of multiple representations in GIS

The use of views to support multiple representations has been recently proposed by [22], in the context of CAD applications, where many of the problems existing in GIS can be found (e.g., issues of scale and point of view). The authors, however, ignore the possibility of permanently storing new database versions, and of combining and manipulating sets of representations.

We have shown in [18, 19] how the DBV mechanism can be used to keep track of the temporal evolution of georeferenced entities: each database version Dv_i corresponds to a distinct state of the world, which can be manipulated by the user independently from the other states. This is one specific case of multiple representation handling, where time is the varying factor. The same framework can also be extended to other multiple representation situations, associating each Dv_i with a distinct representation need. In the example of section 3, this means that several sets of Dv's coexist in the database: one set for managing the representation needs of cartographers, and another for traffic engineers. Each set represents the variation of the phenomena over time.

Though this allows handling different representation dimensions, it does not provide the user with enough facilities for managing them in a GIS. In this section, we show that by enhancing a multiversion database with views, we can provide users with an environment where they can adequately manage an entity and its representations, as well as work simultaneously with several representations.

Nowadays, the problem of handling multiple representations is solved by the users through storing entities in separate files. In our model, this can be handled by storing the data in a multiversion database. Consider the example described in section 3 in a multiversion database. Each Dv_i corresponds to a specific representation of the world, where all objects are described in the same scale, projection and time frame. Consider teams of engineers and of cartographers that want to manipulate data about the area where the road network lies. Given a Dv, it will contain a *Road* object represented either using *Cart-representation* or *Weighted-graph*.

5.1 Multiversion View

However, GIS users do not normally want to work with the whole database. In particular, traffic engineers just need to consider *Road* and *City* classes, whereas cartographers require *Road, Vegetation* and *Soil*. Again, at present this is handled by combining data in separate files, from which users work. However, they cannot keep the links between such files and the underlying database.

Our framework, on the other hand, considers that each group of users may define a *view* over the multiversion database, which allows them to select the classes of interest, while at the same time keeping their links with the underlying database. This view, unlike all other views studied in the literature, is a *multiversion view*: it is constructed from the underlying Dv_i, each of which appears in the view as a separate *virtual database version*. We use the term *virtual database version* to denote that, in a multiversion view, each versioned database Dv_i corresponds to a view VDv_i. The middle part of Figure 1 shows this situation. The multiversion view appears to the user as a multiversion database, restricted to the entities of interest. In the figure, this view is composed of virtual database versions VDv_1 through VDv_3.

We now show how this can be done using the syntax of the O2 DBMS. In order to define a view in O2, users first define a virtual schema (class types and methods) and then specify its extension by means of queries. Relationships between view objects and database persistent objects are established by declaring which database

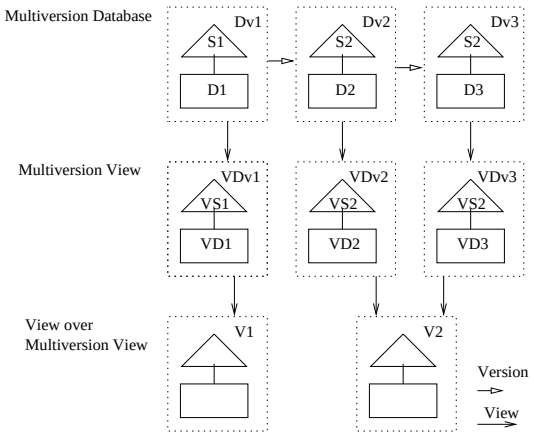

Figure 1: Modelling the user framework

persistence roots are used in the view.

We extend this notion to multiversion views, maintaining the same construction steps. Consider a multiversion database MDBV composed of database versions $Dv_1, Dv_2, \ldots Dv_j$. A multiversion view MV (view creation operation of type Va) is defined as

```
virtual schema multiversion view MV
    composed of (VDv1, ... VDVj)
    from multiversion schema MDBV
import class C1, C2;
import name RC1, RC2, O1;
```

Each virtual database versions VDv_k will have classes *C1, C2* in its schema.

In our example, a multiversion view for the traffic engineers, called $TE-View$, can be defined as

```
virtual schema multiversion view TE-View
    composed of (Road, City)
    from multiversion schema ROAD;
import class Point, Line, Polygon,
Weighted_graph, Network, Road, City;
/* persistence roots */
import name R_Road, R_City;
```

Engineers need only data about *City, Road*. Furthermore, only the weighted graph representation of *Road* will be used in the view. The extension of $TE-View$ can be built using queries:

```
virtual class VCity includes
    ( select * from R_City);
```

In view $VCity$, all database cities appear in each virtual view component. Alternatively, engineers may only be interested in Cities of a certain size:

```
virtual class VCity  includes
     (select x from R-City
     where x.population > 100,000);
```

The virtual *VRoad* class extension can be built in a similar way, restricting the *loc* representation attribute values to those that correspond to *Weighted-graph* representations.

5.2 View over Multiversion view

Each group of users (cartographers, engineers) can build its own multiversion view and work according to its needs. At this stage, users can start combining data, manipulating the entities of interest. This stage of the work can be modelled by stipulating that each group will build views *on top* of its multiversion view. If a multiversion view MV contains virtual database versions $VDv_1 \ldots VDv_j$, then basically two types of views can be built on top of MV:

- One view V_k defined over one single virtual database version VDv_k:

```
virtual schema DBV view Vk
     from VDv_k in MV;
/*import from virtual schema of view VDv_k*/
import class C1;
import name R-C1;
```

- One single view defined over several virtual database versions: $\{VDv_l \ldots VDv_m\}$:

```
virtual schema VDBV view Vk
     from (VDv_l ... VDv_m) in MV;
import class C1;
/*only the schema def. of VDv_1 is used*/
import class C2 in VDv_l;
```

These views may be built on top of a single or multiple virtual database versions VDv_i. The first case happens when, for instance, engineers select one VDv state on which they want to run simulations; in the second, the engineers want to analyze the evolution of the road through time, for a given representation. This is portrayed in the bottom level of the figure. View $V1$ is built on top of virtual database versions VDv_1; view $V2$ combines data from virtual database versions VDv_2, VDv_3.

We remark that these are the actual steps GIS users follow in order to perform different types of data analysis, but without support of versions or views. This forces them to manage each single database version created, which adds to the complexity of application development. In our framework, the links among Dv's and among virtual database versions are managed by the underlying version mechanism and its view complement. Thus, the steps for working in our environment that support users' work habits are: creation of the multiversion database; construction of multiversion views for the distinct user groups; and building views over the multiversion views.

Returning to the road example, the first type of view is similar to constructing a view over a non-versioned database [25], and will not be expanded here.

In the second type of view, creation specification assume that two virtual database versions represent two different temporal states of the road at times T1 and T2. Suppose that the engineers want to combine both representations in order to compare length differences. In our framework, this requires creating a view that will simultaneously contain two temporal representations of the road, for these times. Let these two representations be stored in virtual database versions named VDv_{T1} and VDv_{T2}. Schema and extension are defined as:

```
virtual schema DBV view T
     from (VDv_t1, VDv_t2) in MV;
/* Cities are not needed in this view */
import class VRoad  ;
import class Point, Line Polygon, Network,
Weighted-graph;
import name R-Road;

virtual class C includes
   (select tuple(x: a1.loc, y: a2.loc)
   from a1 in VRoad inDv VDv_t1,
a2 in VRoad inDv VDv_t2
   where a1.scale = a2.scale )
   attribute dif_length: integer in class C
         has values
         abs(self.x->length - self.y->length)
```

Class C does not correspond to any underlying database class, and contains attributes x, y (taken from the *loc* component of the virtual class *VRoad*, for the representations at $T1$ and $T2$). Furthermore, it contains an additional attribute *dif-length*, which shows the difference in the length of the two road representations. *Length* is a method defined for class *Road* in the schema of section 3. Notice furthermore that this only makes sense if both representations refer to the same scale.

6 Conclusions and Directions for Future Work

This paper discussed the problem of multiple representation of georeferenced entities, and presented a database-centered environment for managing them. This environment is based on a multi-level data model, and takes advantage of extending the DBV version mechanism with some view operators. This extension is in itself a contribution, since versions and views have so far been treated in isolated contexts by the database community.

The three main advantages of the framework proposed here are. It distinguishes between the logical problem of multiple representations from that of physically storing and managing them. It allows GIS users to specify their own representation framework, from which they can select the data entities of interest, without having to worry about other existing representations of the same elements. It provides a means for users to work simultaneously with several representation frameworks, without losing track of the original data layers.

The next steps in this research will contemplate the formalization of the view operators, and consider implementing the extension of the DBV mechanism in order to support the multiple representation paradigm.

In the DBV environment, the underlying DBMS handles the consistency of data. In a GIS framework, however, the problem is much more complicated, since entities are related to each other according to spatio-temporal constraints, which may or not be relaxed by the user, for a particular representation context. The multiple representation paradigm presents, in fact, an interesting challenge from this point of view. The issue of consistency among versions is part of an ongoing work at the LAMSADE and LAFORIA laboratories in France [7], and is not the main concern of this paper.

Acknowledgements

This work was partially supported by Brazilian grants from CNPq and FAPESP. The first author thanks M. Lagrange and L. Raynal from IGN-France for insights provided into the multiple representation problem.

References

[1] S. Abiteboul and A. Bonner. Objects and Views. In *Proc. SIGMOD Conference*, pages 238–247, 1991.

[2] G. Camara, U. Freitas, R. Souza, M. Casanova, A. Hemerly, and C. B. Medeiros. A Model to Cultivate Objects and Manipulate Fields. In *Proc 2nd ACM Workshop on Advances in GIS*, pages 20–28, 1994.

[3] W. Cellary and G. Jomier. Consistency of Versions in Object-Oriented Databases. In *Proc. 16th VLDB*, pages 432–441, 1990.

[4] W. Cellary, G. Vossen, and G. Jomier. Multiversion Object Constellations: a new Approach to Support a Designer's Database Work. *Engineering with Computers*, 10:230–244, 1994.

[5] N. Chrisman. *Geographical Information Systems - volume I*, chapter The Error Component in Spatial Data, pages 165–174. John Wiley and Sons, 1991.

[6] H. Couclelis. People Manipulate Objects (but Cultivate Fields): Beyond the Raster-Vector Debate in GIS. In *Proc International Conference on GIS - From Space to Territory: Theories and Methods of Spatial Reasoning*, Springer Verlag Lecture Notes in Computer Science 639, pages 65–77, 1992.

[7] A. Doucet, S. Gancarski, J. Jomier, and S. Monties. Integrity Constraints in Multiversion Databases. Submitted for publication, 1996.

[8] S. Faiz. *Modelisation et Visualisation de l'Information Qualité dans les Bases de Données Spatiales*. PhD thesis, Université Paris Sud, 1996.

[9] S. Gancarski and G. Jomier. Managing Entity Versions within their Contexts: a Formal Approach. In *5th International Conference,Database and Expert Systems Applications DEXA94*, pages 400–409, 1994.

[10] O. Gunther and A. Buchman. Research Issues in Spatial Databases. *ACM Sigmod Record*, 19(4):61–68, 1990.

[11] O. Gunther and W-F Riekert. The Design of GODOT: an Object-oriented Geographic Information System. *IEEE Data Engineering Bulletin*, pages 4–9, september 1993.

[12] W. Kafer and H. Schoning. Mapping a Version Model to a Complex-Object Data Model . In *Proc IEEE Data Engineering Conference*, pages 348–357, 1992.

[13] R. H. Katz. Toward a Unified Framework for Version Modelling in Engineering Databases. *ACM Computing Surveys*, 22(4):375–408, 1990.

[14] P. Klahold, G. Schlageter, and W. Wilkes. A General Model for Version Management in Databases. In *Proc XII VLDB*, pages 319–327, 1986.

[15] J-P Lagrange and A. Ruas. Etat de l'Art en Generalisation. Technical report, IGN-COGIT, April 1993.

[16] G. Landau, J. Schmidt, and V. Tsotras. Efficient Support of Historical Queries for Multiple Lines of Evolution. In *Proc. IEEE Data Engineering Conference*, pages 319–326, 1993.

[17] J-C. Mamou and C. B. Medeiros. Interactive Manipulation of Object-Oriented Views. In *Proc International IEEE Conference Data Engineering*, pages 60–69, 1991.

[18] C. B. Medeiros and G. Jomier. Managing Alternatives and Data Evolution in GIS. In *Proc. ACM/ISCA Workshop on Advances in Geographic Information Systems*, pages 36–39, 1993.

[19] C. B. Medeiros and G. Jomier. Using Versions in GIS. In *Proc. International DEXA Conference*, pages 465–474, 1994. Springer Verlag Lecture Notes in Computer Science 856.

[20] C. B. Medeiros and F. Pires. Databases for GIS. *ACM Sigmod Record*, 23(1):107–115, 1994.

[21] J. Muller. *Geographical Information Systems - volume I*, chapter Generalization of Spatial Databases, pages 457–475. John Wiley and Sons, 1991.

[22] H. Naja and N. Mouaddib. The Multiple Representation in an Architectural Application. In *Proc. DEXA Conference*, pages 237–246, 1995.

[23] J. F. Raper and D. J. Maguire. Design Models and Functionality in GIS. *Computers and Geosciences*, 18(4):387–400, 1992.

[24] P. Rigaux. *Interfaces Graphiques pour Bases de Données Spatiales: Application à la Représentation Multiple*. PhD thesis, CEDRIC - Conservatoire National des Arts et Metiers, 1995.

[25] C. Santos, S. Abiteboul, and C. Delobel. Virtual Schemas and Bases. In *Proc. EDBT*, pages 81–94, 1994.

[26] M. Scholl, A. Voisard, J. Peloux, L. Raynal, and P. Rigaux. *SGBD Géographiques – Spécificités*. International Thompson Publishing, Paris, 1996.

[27] R. Subramanian and N. Adam. The Design and Implementation of an Expert Object-Oriented Geographic Information System. In *Proc. 2nd International Conference on Information and Knowledge Management-CIKM*, pages 537–546, 1993.