MAP-OVERLAY WITHIN A GEOGRAPHIC INTERACTION LANGUAGE

Vincent Schenkelaars*
Erasmus University / Tinbergen Institute,
P.O. Box 1738, 3000 DR Rotterdam, The Netherlands,
Phone: +31 10-4081416, Fax: +31 10-4526177,
E-mail: Schenkelaars@cs.few.eur.nl.
and
Peter van Oosterom
TNO Physics and Electronics Laboratory,
P.O. Box 96864, 2509 JG The Hague, The Netherlands,
Phone: +31 70-3264221, Fax: +31 70-3280961,
Email: oosterom@fel.tno.nl.

The current generation of spatial SQL languages has still severe problems in specifying queries which contain complex operations. One of these complex operations is map-overlay with topological structured layers. In this paper an attempt is made to model the map-overlay operation into an object-relational query language. This query language is the formal part of a geographic interaction language. An example application of the concepts of this language is given which shows that map-overlay can be specified with relative ease. This paper also deals with the creation of topological structured layers.

1 Introduction

Previous research, by various authors [2, 6, 7], proved that the original relational model is not very suitable for a Geographic Information System (GIS). One of the main problems with the relational data model is that it lacks the geometric data types. A very similar problem occurs in the relational query language SQL. The extended relational data model solves the problem quite well, the extended SQL still has a number of remaining problems. The extended SQL approach has difficulties when dealing with more complex objects and operations, for example operations that apply to topological structured input sets. Examples of this type of operations are: shortest path in road network, network analyses, map-overlay, visibility analyses in 3D terrain, and computations of corridors. Since map-overlay is generally regarded as one of the crucial operations in a GIS, our attention is focused on modelling this operation

*This research was funded by a grant from TNO-FEL under AIO-contract number 93/399/FEL

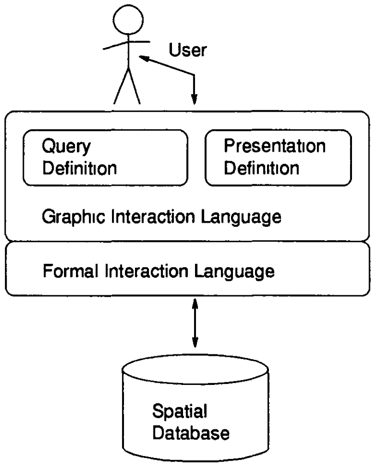

Fig. 1: Structure of the Interaction Language

in the formal part of the interaction language. We propose an *object-relational* formal syntax for this map-overlay operation.

Our final goal is to develop a general geographic interaction language. This interaction language is not meant to be yet another extended SQL. It consists of two layers. The first layer is formed by the object-relational model. The extended relational algebra, which contains all required spatial types and operations, describes the formal side of this geographic interaction language. On top of this formal language a more graphic, user-oriented language will be defined. This graphic interaction language performs two major tasks. First of all, it takes care of easy definition of queries and operations. The second task of this graphic interaction language is the definition of the presentation of query results [7]. Issues which are involved in the presentation are: shape, size, color, shading, projection, transformation, etc. All these factors can be defined interactively with the presentation language in a graphic way. Figure 1 visualizes the structure of the interaction language. More information about our interaction language can be found in [13].

2 Map Layers

An important principle in GIS is the layer concept. In this concept the geographic data is stored into layers. Each layer describes a certain aspect of the modeled real world [4]. Using map layers is a *natural* technique to organize the data from possibly different sources.

Two distinctive types of layer organizations can be identified: thematic and structured layers. The most common type is the *thematic* layer [4]. For each theme on a map, a separate layer is specified without a topological structure. In order to solve user queries containing features of more than one layer, a spatial join operation has to be performed. Elements of both layers are individually combined. The output of such combination is usually a set of object pairs. Each pair consists of an object of both

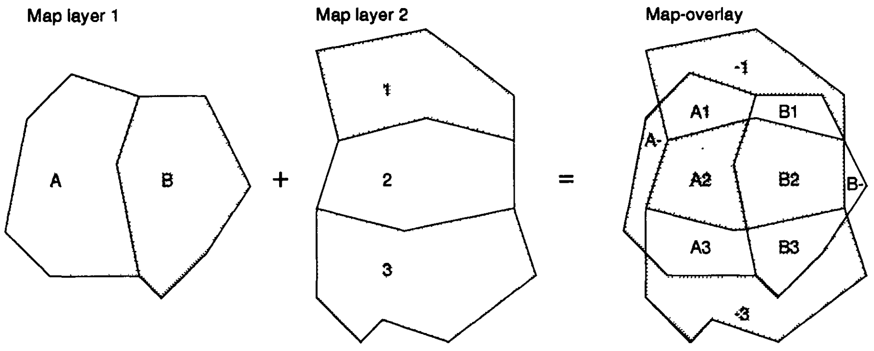

Fig. 2: Map-overlay computation and label propagation

layers.

Multiple thematic layers can be stored into one *structured* layer. These kind of layers are organized with a topological structure. The use of a topological structure removes a large amount of redundant information. Each edge is only stored once and contains references to the polygons on each side of the line. Each polygon only stores a reference to its defining edges. Although this topological structure stores map information more efficient, solving queries which contain features of more than one structured layer becomes more awkward. The combination of two of these layers is not performed separately on each element of the layers but on the layer as a whole. The result of this combination is a new structured layer.

This paper concentrates on the latter type of layers: the topological structured layer. It may be clear that the combination of two ore more of these structured layers is far more complex compared to the spatial join of two relatively simple thematic layers. We focus on the specification of the map-overlay process in a formal query language.

3 Map Overlay

The input of a map-overlay operation consists of two or more topological structured layers of edges only, in case of a linear network, or edges and faces, in case of a polygonal map. The output of the process is a new topological structured layer in which the attribute values of the new edges and faces are based on the attribute values of the elements of the input layers. Figure 2 shows the map-overlay process.

Note that the spatial domains of the different input layers do not have to match exactly. The map-overlay is usually computed in three logical phases [8]. The first step is performed at the metric level and computes all *intersections* between the edges (line segments) from the different layers. Followed by a reconstruction of the *topology* and assignment of labels or *attribute* values in the next two steps [18]. Several algorithms have been developed for computing the map-overlay:

- brute force method;
- plane-sweep method [1] (including several variants);

- uniform grid method [9];
- z-order-based method [11];
- R-tree-based method [15].

A problem related to map-overlay is the introduction of sliver polygons. Solutions for this have been presented in [3, 19]. Certain parameters have to be specified (e.g. minimum face area) in the map-overlay process.

When inserting a new line into a topological layer, several situations can occur: touch, cross, overlap, or disjoint from the lines already present in the structure. Geometric computations are used to determine the actual situation. Because computers have only finite precision floating-point arithmetic [10, 12], *epsilon*-distance computations have to be used. The actual epsilon value has to be given (by default) for every topological layer.

4 Extended Relational Approach

This section describes an attempt to model the map-overlay process directly into a relational query language. Since we use the extensible relational database management system (eDBMS) Postgres [14] in our GIS research, we used its query language Postquel as a framework. It is clear that the modeling can be done in other eDBMS's in a very similar way.

First, at least two layers have to be created. This can be done by creating a relation for each layer. The code fragment below shows the statements. The first line creates a parcel layer with owner and value information. The other layer contains soil information. Note that the layers contain explicit polygons and there is no topological structure.

Code Fragment 1

```
create layer1 (name=text, shape=POLYGON2, owner=text, value=int4)

create layer2 (id=int4, soil=text, location=POLYGON2)
```

With this layer definition, it is possible to specify the map-overlay operation as is shown in the next code fragment. The shape of the objects of the new layer are defined as the intersection between an object from `layer1` and an object from `layer2`. The attribute value `newval` is calculated from attribute value `soil` of `layer2`, `value` of `layer1`, and the area of the new polygon. Note that some functions are used in the calculation of the value of `newval`.

Code Fragment 2

```
retrieve into layer3 (oldname=layer1.name,
    newshape=Intersect(layer1.shape, layer2.location),
    newval=SoilRef(layer2.soil)*layer1.value*AreaPgn2(newshape))
```

We now have specified map-overlay in relational terms. The number of input layers does not have to be limited to two, but can be increased to any number. However, for each number of layers to be combined an `Intersect` function with the appropriate number of parameters has to be defined. Other attributes can be specified at will.

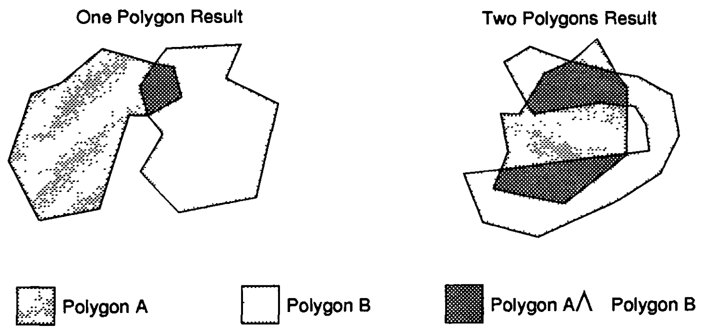

Fig. 3: Intersection of two polygons

They can be copies of old attributes or functions applied to attributes of any layer. This is a very flexible and elegant way to specify the propagated attribute values in the map-overlay operation.

Although the relational approach is simple and straight forward, this 'solution' has several severe drawbacks:

a. It is based on explicit polygons, not on a topological data model. As is stated before, there are clear advantages in topologically structuring the layers. It is possible to create topological structured layers in a relational model, but specifying the map-overlay process becomes impossible. Since one does not longer have explicit polygons, it is necessary to execute a query for each polygon to get its defining edges. This construction does not fit in the relational algebra.

b. It assumes that each pair of intersection polygons result in at most one polygon, in general this is not the case. Any number of polygons can be returned as result of the intersection; see righthand side of figure 3. Therefore, a new type of polygon with disconnected parts must be defined. It is clear that this is not desirable.

c. This method does not work for the map-overlay of two linear network layers. Intersections of lines return in general points and not lines. One could define the intersection function in such way that it returns the resulting line parts, but then again a new polyline type with separated parts must be defined.

d. It is not possible to use an efficient plane-sweep algorithm, because the intersection function operates on pairs of individual polygons. This is not efficient.

The common cause behind these problems is that map-overlay should be applied to complete layers and not to the individual polygon instances making up the whole layer structure. In order to be able to do this, the concept of complex objects is needed. The topological structured layer is considered to be a complex object with its own intersection operation. So there seems to be a need to move away from the pure relational model and adopt some object-oriented concepts. The next section describes what is needed to specify the map-overlay operation in an object-relational model.

5 Object Relational Approach

This section described our new approach to model the map-overlay process in a more object oriented approach. The next subsection describes the way to define a topological layer structure. Section 5.2 describes the actual creation of the topological layer, while section 5.3 deals with the final map-overlay operation specification.

5.1 Topological Layer Definition

To solve the problems associated with the extended relational approach, we need to create a complex object `layers`. A fully topological structured layer contains nodes, edges, and faces. To make the model simple, we assume in our examples that the nodes are stored in the edges, and that only faces have labels (attributes). Before the topological structure of a layer can be created, we need to define some prototypes. This is done in the first two lines of the next code fragment. These prototypes are essentially the same as ordinary relations, but they can not contain any data. They form the framework links between faces and edges.

Code Fragment 3

```
define prototype faces (id=oid, boundary=edges.id[])

define prototype edges (id=oid, line=POLYLINE2, left=faces.id,
    right=faces.id)

create layers (layer_id=unique text, boundaries=prototype edges,
    areas=prototype faces)

define topology _layers_topol on layers using polygonal (boundaries,
    areas)

define index _layers_bdy on layers.boundaries using
    rtree (line polyline_ops)
```

The prototypes `faces` and `edges` define the basic attributes of a face and an edge respectively. Both have an attribute `id` of type `oid`[1], which contains a unique identifier for each edge and face. The `faces` prototype has an attribute `boundary` which is a variable length array (indicated by the square brackets) of `id` values of the prototype edges. This concept of references forms the actual link between the two prototypes. Note that the `faces` prototype has a forward reference to the `edges` prototype. This forward referencing is only allowed in the definition of the prototypes. These have to be defined before the actual `layers` relation can be created.

The creation of the `layers` relation is quite straight forward. The layer has a unique identifying name and *references* to the edges and the faces member relations. These two relations are not directly visible for the user. The only way a user can retrieve individual edges or faces is through the layers relation similar to the object-oriented concept of class and member class. The edges and faces can therefore be in one layer only.

[1]oid stands for object identification

The next line in code fragment 3 registers the topological structure in the DBMS. This line will enhance the `append`, `replace`, and `delete` statements whenever these statements deal with a topological structured layer. This enhancement of these statements is similar to the enhancement of the same statements when defining an index on a relation attribute. The syntax is also similar to the index definition syntax in Postgres as can be seen in the last line of this code fragment. The two parameters between the parenthesis relate to the attributes of the layer which contain the edges and the faces. A optional third parameter could be the epsilon value; see section 3 The keyword `polygonal` denotes that the topological structure contains edges and faces. The possible topological structures are:

- `full_polyhedral`: A three dimensional topological structure with nodes, edges, faces, solids;
- `polyhedral`: A three dimensional topological structure with edges, faces, solids;
- `full_polygonal`: A two dimensional topological structure with nodes, edges, and faces;
- `polygonal`: A two dimensional topological structure with edges, and faces;
- `network`: A two dimensional topological structure with nodes and edges.

5.2 Topological Layer Creation

The definition of the framework structure of the layer is now complete. It is good to realize that the topological structured layers itself have still to be created. The following code fragment shows how this could be specified in the formal database language.

Code Fragment 4

```
define prototype faces2 (name=text, owner=text, value=int4)
    inherit faces

append layers (layer_id="parcels", areas=prototype faces2,
    boundaries=prototype edges)

append l1.boundaries (polyline="(....)"::POLYLINE2)
    from l1 in layers where l1.layer_id="parcels"

replace l1.areas (name="....", value="....", owner="....")
    from l1 in layers
    where PointInPolygon ("(x,y)"::POINT2, current))
          and l1.layer_id="parcels"
```

The first line of code fragment 4 defines the additional attributes of the areas in this layer. The `inherit` keyword allows `faces2` to be used wherever `faces` can be used. In this way the generic topological structured layer can be extended to have arbitrary number of attributes.

The next line creates the new empty layer. Only the `layer_id` attribute value has to be provided. The append statement checks the type of `faces2` against the original definition of `faces`. Note that the original definition of edges is used as the `boundaries` attribute. If one would need to have boundaries with additional attributes, a new prototype has to be defined in a similar way as is done for `faces2`.

Now the layer is defined and ready to receive its defining edges. The third line in code fragment 4 is executed for every edge in the layer. It is an append into the member relation of the layer where the edges are stored. The location of this relation is stored in the `edges` attribute of the layer. This append has some extra functionality due to the definition of the topology structure. Whenever an edge E1 intersects an edge E2 already in the layer, both edges are split by the append operation. Edge E2 is removed from the layer, the resulting parts of the splitted edge are appended to the layer one by one. Note that in case of edges with additional attributes, both parts of the splitted edge get the original attribute values. While appending edges, faces are being formed. Those faces are stored in the faces relation of the layer in a similar way. After all the edges are added to the layer we have a layer with all edges and faces, but the faces have no labels yet; the additional attributes have to be given values. This is done in the last line of code fragment 4. Each area is checked whether it contains the location. The keyword `current` refers to the area which is checked at that time.

Note that the described process above creates a topological layer from scratch. When one has a data set which contains topological references as is for example the case in DIGEST data [5], this process can be simplified by defining the topology after the areas and lines are inserted in layer. When inserting the edges and faces in this case, no extra functionality is needed in the append. When the topology is defined at the end, it provides the additional append functionality at that time. The definition of the topology triggers also a topology check process on the already inserted data. This to ensure that the topology structure of the data is valid. When new edges are added to the layer, the append statement will take care of the topology maintenance. It also contains an object id manager. This manager will keep object id's unique and insures referential integrity of the topological structured layer.

5.3 Map Overlay

Once the layers have been created, they can be manipulated as layer objects; that is as *complex objects*. Since complex objects can be handled in the same way as ordinary objects, one can write an intersection function which is executed on the layer as a single object. The intersection of two layers is exactly what map-overlay is. The next code fragment shows how this can be specified in the query language. We assume that the map-overlay function is registered in the database.

Code Fragment 5

```
append layers (layer_id="combined layer")

retrieve (count=overlay(l1,l2,new_layer,"FaceAttrSpecStr",
    epsilon, sliver))
    from l1, l2, new_layer in layers
    where l1.layer_id="parcels" and l2.layer_id="soil" and
          new_layer.layer_id="combined layer"
```

Before the map-overlay can be executed, we need to provide a structured layer in which the map-overlay result can be stored. This is done in the first line of code fragment 5. Note that we do not need to initialize the areas and boundaries sets. This initialization is done in the `overlay` function. Now we are ready to compute the resulting layer. The map-overlay is now nothing more than a retrieve using a user-defined function. Since a function cannot be called without having a variable to receive its result, some extra information can be returned from the overlay function. In this case the number of areas in the new layer is returned. The `FaceAttrSpecStr` contains for each attribute of the areas in the new layer a specification string. Each part of the specification string contains the name of the attribute and the expression which specifies the value of the attribute. Each expression is similar to the expression in code fragment 2. This total string is parsed by the overlay function.

The result of the map-overlay operation is stored in a new layer. The coordinates of all the edges in the new layer are redundantly stored. However, since some layers may be regarded as temporary layers, a user has two option to remove the redundancy. The user can either remove the new layer after studying the result, or one or more of the original layers is no longer useful and can therefore be removed.

6 Conclusion & Further Research

We have presented a way to model the important map-overlay concept into a formal query language. The following components are added to the Postquel language: `prototype`, `define topology`, and special `append`, `replace`, and `delete` statements. Besides these modifications in the DBMS (backend), also modifications to the geographic user interface (frontend) have to be made in order to visualize the topologically structured data [17].

An implementation of the suggested extensions in Postgres will be non-trivial, but partly comparable to adding a new access method [16]. An alternative is to use a true OODBMS as platform on which this spatial eRDBMS will implemented.

In this paper, we focussed on one specific type of topology, but as indicated, other types of topology structures can be supported as well. This will make the spatial eRDBMS a good generic basis for GIS-application, and also for other spatial applications; e.g. CAD systems.

References

[1] Jon L. Bentley and Thomas A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, C-28(9):643–647, September 1979.

[2] M.S. Bundock. SQL-SX: Spatial extended SQL - becoming a reality. In *Proceedings of EGIS '91*, Brussels, 1991. EGIS Foundation.

[3] Nicholas R. Chrisman. Epsilon filtering – a technique for automated scale changing. In *Technical Papers of the 43rd Annual Meeting of the American Congress on Surveying and Mapping*, pages 322–331, March 1983.

[4] Sylvia de Hoop, Peter van Oosterom, and Martien Molenaar. Topological querying of multiple map layers. In *COSIT'93, Elba Island, Italy*, pages 139–157, Berlin, September 1993. Springer-Verlag.

[5] DGIWG. DIGEST - digital geographic information - exchange standards - edition 1.1. Technical report, Defence Mapping Agency, USA, Digital Geographic Information Working Group, October 1992.

[6] Max J. Egenhofer. An extended SQL syntax to treat spatial objects. In *Proceedings of the 2nd International Seminar on Trends and Conce rns of Spatial Sciences*, New Brunswick, 1987.

[7] Max J. Egenhofer. Spatial SQL: A query and presentation language. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):86–95, February 1994.

[8] Andrew U. Frank. Overlay processing in spatial information systems. In *Auto-Carto 8*, pages 12–31, 1987.

[9] Wm. Randolph Franklin, Chandrasekhar Narayanaswami, Mohan Kankanhall ans David Sun, Meng-Chu Zhou, and Peter Y. P. Wu. Uniform grids: A technique for intersection detection on serial and parallel machines. In *Auto-Carto 9*, pages 100–109, April 1989.

[10] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, March 1991.

[11] Jack Orenstein. An algorithm for computing the overlay of k-dimensional spaces. In *Advances in Spatial Databases, 2nd Symposium, SSD'91, Zürich*, pages 381–400, Berlin, August 1991. Springer-Verlag.

[12] David Pullar. Spatial overlay with inexact numerical data. In *Auto-Carto 10*, pages 313–329, March 1991.

[13] Vincent Schenkelaars. Query classification, a first step towards a geographical interaction language. In *Proceedings of Advanced Geographical Data Modeling, AGDM'94*, Delft, September 1994.

[14] Michael Stonebraker and Lawrence A. Rowe. The design of Postgres. *ACM SIGMOD*, 15(2):340–355, 1986.

[15] Peter van Oosterom. An R-tree based map-overlay algorithm. In *Proceedings EGIS/MARI'94: Fifth European Conference on Geographical Information Systems*, pages 318–327. EGIS Foundation, March 1994.

[16] Peter van Oosterom and Vincent Schenkelaars. Design and implementation of a multi-scale GIS. In *Proceedings EGIS'93: Fourth European Conference on Geographical Information Systems*, pages 712–722. EGIS Foundation, March 1993.

[17] Peter van Oosterom and Tom Vijlbrief. Integrating complex spatial analysis functions in an extensible GIS. In *Proceedings of the 6th International Symposium on Spatial Data Handling, Edinburgh, Scotland*, pages 277–296, September 1994.

[18] Jan W. van Roessel. Attribute propagation and line segment classification in plane-sweep overlay. In *4th International Symposium on Spatial Data Handling, Zürich*, pages 127–140, Columbus, OH, July 1990. International Geographical Union IGU.

[19] Guangyu Zhang and John Tulip. An algorithm for the avoidance of sliver polygons and clusters of points in spatial overlays. In *4th International Symposium on Spatial Data Handling, Zürich*, pages 141–150, Columbus, OH, July 1990. International Geographical Union IGU.