INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center St. • Suite 600 • Berkeley, California 94704-1198 • (510) 643-9153 • FAX (510) 643-7684

# Mapping Conceptual Geographic Models onto DBMS Data Models

Agnès Voisard* Benoît David†

TR-97-005

March 1997

**Abstract**

We study the representation and manipulation of geographic information in a database management system (DBMS). The conceptual geographic model that we use as a basis hinges on a complex object model, whose set and tuple constructors make it efficient for defining not only collections of geographic objects but also relationships between them. In addition, it allows easy manipulation of non-basic types such as spatial data types. We investigate the mapping of our reference model onto major commercial DBMS models, namely a relational model extended to abstract data types (ADT) and an object-oriented model. Our analysis shows the strengths and limits of the two model types for handling highly structured data with spatial components.

*Institut für Informatik, Freie Universität Berlin, D-14195 Berlin, Germany, voisard@inf.fu-berlin.de

†Institut Géographique National, 2 Avenue Pasteur, BP 68, F-94160 Saint-Mandé, France Benoit.David@ign.fr

# 1 Introduction

Many new applications that deal with geographic data require powerful modeling tools. Such applications, e.g., resource management, urban planning, meteorology, are characterized by large amounts of data, both alphanumeric and spatial. They increasingly rely on *database management systems* (DBMS). A challenging issue is to capture a maximum of semantics of such applications while representing both their data and associated operations in typical DBMS data models. This paper focuses on geographic data modeling (for a comprehensive survey on features that a geographic DBMS should provide, such as graphical user interface, spatial access method, query optimization, see [Güt94a]).

The basic entities that we consider in these applications are collections of *geographic objects*. In the sequel, we refer to these collections as *maps*, although this term might evoke a different notion to the GIS community, namely a frozen representation (on the screen for example) of what is denoted map here. A geographic object (such as a county) is derived from an entity of the real world. Intuitively, it has two kinds of components: alphanumeric ones (e.g., its name and population) and spatial ones (e.g., a polygon). As customary in this domain, we refer to the spatial component of a geographic object as a spatial *abstract data type* (ADT). In our study we only deal with vector representations: raster objects are not considered further. The spatial objects can have (not exclusively) 0 (points), 1 (lines) or 2 dimensions (regions). A third dimension is not considered here, nor is a possible temporal component of a geographic object. We denote as *geographic model* or *map model* a representational model that handles geographic data in general. These models can be conceptual or can be expressed by means of the logical model of a DBMS. Whatever model is chosen, the following properties must hold for efficiently representing geographic objects:

- Independence between higher (the maps) and lower (the geometry) levels of abstraction.
- Extensibility, i.e., the ability to add new operations on maps or on spatial ADTs.

One problem in the definition of a geographic model results from the existence of application-dependent functions on maps that rely on operations on spatial ADTs. A major issue is to embed both these operations on maps and on ADTs in the same geographic model. In addition, spatial ADT must be closed under the operations one can apply to them. For previous work on spatial ADT see [Güt88b, SV89, JKO+89, GNT91].

Another problem is derived from the fact that some general functions on maps correspond to sophisticated operations for which conventional database models are deficient. Consider the following example of aggregation. Suppose that a map of states is to be derived from a map of counties, which includes computing the total population of each state by summing up the populations of all the counties that belong to that state. To perform that operation, one needs not only some implicit geometric union on the spatial part of counties, but also a general operation that allows, for each state, to compute its population. This kind of operation, which correspond to aggregate functions in the relational model, are not always possible in the geographic models proposed so far. Even when they exist,

they are usually not defined in a clean way (partly due to the lack of formalism in the definition of aggregate functions), and are based on *ad hoc* solutions. Furthermore, the existing approaches sometimes mix the two levels of abstraction (map and geometry), and some database concepts appear at the geometry level, which leads to models difficult to understand (see Section 2).

The previous example on the map of states shows a feature that frequently occurs in GIS applications, namely the possibility of representing a composition of objects (e.g., a state is composed of counties). In the GIS area, it is only recently that object composition has received attention [Wor94, HT96].

The example above also illustrates the similarity with statistical databases [Mic91], where a major problem is to define proper mechanisms for data aggregation (across many dimensions). According to [Len93], a database is a statistical database if it contains the following three kinds of data: *microdata* (primary data, e.g., census data), *macrodata* (grouped or aggregated data, cross-classified by a set of categorical attributes) and *metadata* (data about data). In that respect a spatial database is a statistical database. However, the statistical database community looks for a standard set of statistical operators, mainly based on aggregation/disaggregation. The spatial database community is aiming at finding a set of *spatial* operators, for which the concept of aggregation/disaggregation will also play an important role in the case of space partitioning. It is likely that these two sets will have a non-empty intersection. In [RS95], the authors make the bridge between the two disciplines, by defining an aggregation technique over a hierarchy of (multi-scale) partitions.

Our goal is not to define another geographic model, but to study the implementation of a simple geographic (conceptual) model using two major commercial DBMS models as a support. We chose to base our conceptual reference model on the complex object model of [AB95]. This model is well adapted to the representation of (complex) geographic objects and to their manipulation through a high level operator called *replace*, which applies a function to all elements of a set.

Some attempts have been made to define geographic models using standard database models. However, they often suffer from limits due to the chosen underlying database model or to restrictions imposed on their own model. The Geo-Relational Algebra [Güt88b] and Spatiarel [Dav91] are based on extended relational models and cannot gather regions in a clean way (gathering is the first step before realizing the geometric union of several regions). In [SV89], an algebra for manipulating maps is defined using a complex object model. However, the set constructor is restricted to spatial values, and the model is nothing other than a powerful relational model extended by sets of spatial values. The definition of regions incorporates the notion of set, which should exist at the database level rather than at the ADT level for a cleaner separation (for a discussion, see [SVP+96]). The Rose Algebra [GS93b, GS93a] has a clean approach with a clear separation of levels due to the introduction of an Object Model Interface that allows one to make the connection between a geographic model and a database model. However, this solution may seem complex as

it requires second-order signatures [Güt93]. We will show that operations on maps can be expressed in a simple manner if some general features are provided in the underlying database framework.

This paper is organized as follows. In Section 2, we give as a basis a simple conceptual geographic model, together with examples of operations that one is likely to apply to maps. We also present several representative queries on maps, which will serve as references throughout the paper. In Section 3, we study the use of database concepts for implementing a conceptual geographic model. This study leads to the proposal of three classes of operators which represent a key point to geographic DBMS extensibility. Finally, we express our conceptual model using two major database models, namely a relational model extended to ADT, and an object-oriented one. Following our operator classification, we discuss their advantages and drawbacks while implementing a geographic model with full functionalities.

## 2 A Conceptual Geographic Model

The goal of this section is to present a conceptual geographic model independent of any database representation. Many existing conceptual models, such as the entity-relationship model [Che76, Mar88], its recent extensions [EN89], or IFO [AH87], could serve as a basis to express a geographic model. In this paper we take as an underlying model the complex object model of [AB95]. This model offers all the features required for geographic information handling, both at the representation level (powerful information structuring) and at the manipulation level (existence of an algebra and of high level operators). Recall, however, that our goal is not to define yet another map model, but to consider the major modeling features needed for handling geographic data efficiently. The complex object model we use is of great interest in this context although it does not consider object identity like other complex object models do (e.g., FAD [BBKV87] and all the true object-oriented models). Nevertheless, we believe that this concept is not essential for what we want to show, namely simple geographic data modeling (i.e., data structuring and querying, and no update).

In GIS, it is common to make the distinction between space-based (or field-based) and feature-based (or entity-based) models [Cou92]. In the first case, the entities of interest are regions within the space; attributes (such as temperature or elevation) are associated with them. In the latter case, the primary objects are geographic entities with which is associated a spatial attribute. Recent work has been done to integrate both approaches, as for instance in [GS93b, HT96]. These approaches are based on a mapping of conceptual geographic models onto geometric layers. The model proposed in this paper belongs to the latter category, hence we focus on "conceptual" geographic entities.

This section starts with a description of our base complex object model. We then present our map model as well as the reference schemas. We detail the notion of spatial types and of spatial objects and we give a taxonomy of their operations. Finally, after giving a description of map operations, we study the mechanisms of aggregation and disaggregation on maps, as it turns out to play a major role in map manipulation.

## 2.1 Complex objects

Using a database model for representing geographic information allows us to embed both alphanumeric and spatial data in the same general model, and to use common data modeling operations. A complex object model such as [AB95] is well-suited for this problem. Complex objects are obtained from atomic objects using the *set* and *tuple* constructors. These can be applied arbitrarily deep, contrary to the relational model, where each of them is used only once, or even to non-first normal form models such as Verso [Ver86, AB86], NF2 [JS82], where they have to be alternated. This interesting feature allows one to express compositions of geographic objects in an elegant way, which is not the case for most of the geographic models already proposed (for that matter, the model of [SV89] can be seen as an "extended relational model").

Abiteboul and Beeri assume a finite set of domain names and an infinite set of names, called attributes. Elements of the domains are called atomic values. Types are constructed from domain names, attributes and the set and tuple constructors. if $D$ is a domain name, then $D$ is a type; if $T_1, ..., T_n$, are types, and $A_1, ..., A_n$ are attributes not used in any of them, then $[A_1, ..., A_n]$ is a *tuple* type; if $T$ is a type and A an attribute then $\{A : T\}$ is a *set* type ; if $T$ is a type and A is a name not used in it then $A : T$ is a *named* type, with name $A$. Tuple and set types are created by applying the corresponding constructors to named types. Objects are defined as instances (values) of a type. Both a calculus and an algebra are proposed for expressing queries, and the equivalence between those two is shown.

To remain closer to previous work in this area, we chose to consider only the algebra in this paper. The following operations are defined: *set operations* (union, difference, intersection), *cross product* (creates a set of tuples), *powerset* (which creates the set of all subsets of a set), and *filter operations*: tuple-collapse (collapses each tuple of tuples of a set into a flat tuple), set-collapse (collapses a set of sets into a set). Other operations with intuitive semantics such as *rename* and *extend* are also proposed. Finally, this algebra allows to consider user-defined functions and predicates (as "interpreted" functions and predicates), which are of great interest in map manipulation.

We focus here on a high level operation called *replace*. It is similar to the *mapcar* operation of Lisp-like languages. This iterator applies a function to a set of objects. Note that it does not increase the power of the language. The parameter given to the replace operation is called the *replace specification*. The effect of replace is:

$$replace < f > (m) = \{f(t) | t \in m \wedge f(t) \text{ is defined } \}.$$

The nest and unnest operators of N1NF algebras [JS82, Ver86], suitable to gather several values of the same type in a single set (case of the nest), is not part of the algebra, but Abiteboul and Beeri show their simulation using replace, rename and select.

## 2.2 A model based on geographic entities

In the following, we define a thematic map as a collection of homogeneous geographic objects. A geographic object is usually derived from an entity of the real world. It has a spatial part as well as an alphanumeric one called its *description*. As an example, a river is characterized by its name, its flow (both representing its description), and its geometry (a polyline). A geographic object can be composed of other geographic objects, such as a river composed of branches. In this case, a river is considered as a complex object whose branches are atomic objects. Another example is the well-known state-county hierarchy, where states are composed of counties, and counties are in turn composed of cities and rural areas. Below is a generic definition of a map schema; because of the lack of subtyping in the underlying model, we simply give the abstract syntax of a map description.

```
thematic-map = { geographic-object }
geographic-object = [ <Ai:Ti>, SpatialAtt: Spatial ] /* atomic */
                  | [ <Ai:Ti>, Mi: thematic-map]     /* complex */
```

`<Ai:Ti>` denotes an enumeration of named types (the description). `Ti` denotes a basic type (integer, real, boolean). In the following we refer to the `Ai`'s as alphanumeric attributes. `SpatialAtt` denotes the name of a spatial attribute whose type is `Spatial`, which is detailed in Section 2.3.

The representation of geographic objects above permits to associate a geometry with only atomic objects. One then avoids duplicating the geometry of (complex) objects which can be inferred using the *propagation* mechanism [EF89] from the geometry of objects that compose them (possible recursion). Other attributes such as population may be propagated within such nested maps using the same mechanism. In the state-counties partition, the geometry and the population only appear at the lowest level, i.e., at the county level. The county map is hence considered as an "atomic thematic map", whose elements are atomic geographic objects. The notion of atomicity is useful when describing the explicit manipulation of the geometry of a map. This leads to the map-oriented following definitions:

```
atomic-geographic-object = [ <Ai:Ti>, SpatialAtt: Spatial ]

thematic-map = {atomic-geographic-object } /* atomic thematic map */
             | {Mi: [ <Ai:Ti>, Mi': thematic-map]}  /* complex or nested map */
```

As an illustration, let us consider two particular map schemas: states (`States`) and highways per state (`StateHighwayNetwork`), which will serve as references throughout this paper. The term *network* denotes a map whose spatial part consists of 1-dimensional objects only.

```
States:{State: [StateName: string,
                Counties: {County: [CountyName: string,
```

```
                                    Population: integer,
                                    MainSpokenLanguage: string,
                                    Geometry: Spatial]} ]}
```

`States` is a map, i.e., a set of geographic objects (`State`) whose one component is again a set of geographic objects, or map (here `Counties`), composed in turn of geographic objects (`County`). One can define an object "USA" or "France" with type `States`. Because of composition, the geometry has been factorized, i.e., it only appears at the `County` level (atomic map): the geometry of `State` or `States` are implicit.

```
StateHighwayNetwork:{Highway: [HighwayName:string,
                               StateName: string,
                               Elements: {RoadElement:[MaximumSpeed: integer,
                                                       AverageTraffic: Real,
                                                       Edge:Spatial]}]}
```

`StateHighwayNetwork` is the map of highways per state in which each basic highway portion (`RoadElement`) is an atomic geographic object. As in the previous example, the geometry appears only at the lowest level (in the `RoadElement` object).

## 2.3 Spatial types

In spatial databases, spatial data types are usually defined as ADTs, i.e., encapsulated types together with operations. At implementation time, one can define spatial indexes on spatial ADTs. In this paper, we do not elaborate on the notion of ADT in general. For more information on this topic see [CW85, LZ74, SRG83, Sto86]. A spatial object is an instance of a spatial type. It can have 0 (point), 1 (line) or 2 (zone) dimensions. In [DV93], we compared different definitions of spatial ADT and proposed one based on spatial values (heterogeneous geometric figure). Common geometric operations (union and intersection) and topological operations (interior and closure) can be applied on this type. We assume that the `Spatial` type above follows the same definition. Of course, this is only a possible spatial model. Other spatial models such as raster or topological [EFJ90, EF91] are also widely used.

### 2.3.1 Defining spatial types

Defining spatial types means defining both their structures and the operations applicable to them. Nevertheless, it is a challenging task as it must meet at least the following criteria:

1. *Visibility of the internal structure of ADT and encapsulation.*
   The ADT definition (or more precisely, its interface) must be rich enough so that many applications can access an object of the type without frequently "disencapsulating" the type. Indeed, if a type is not appropriately defined, one might need to access its internal structure, hence to violate its encapsulation. A typical example is the access to holes of polygons in applications dealing with objects of such types (e.g.,

lakes on islands if a lake is defined as a polygon). If holes (i.e., lakes) are not directly accessible, users will constantly have to access the internal structure of the polygons containing them (i.e., polygons representing the islands).

2. *Closure of the type under operations applicable to them.*
   One may then be tempted to define a general type, such that it remains closed under at least a set of common operations. Having however a type too general means encompassing a large number of cases and not taking advantage of the power of typing. Thus the challenge is to define the smallest domain for appropriate types. As we will see further, object-oriented models with their notion of inheritance and overriding are useful to overcome this difficulty. Consider, for example, the intersection operation, which is often used in GIS applications. The intersection operation does not preserve the dimension of geometric figures. The intersection of polylines (dimension 1) can be either a line segment (dimension 1) or a set of isolated points (dimension 0) or even the union of these two, One may surmount this problem by defining a different intersection operation such that it preserves the dimension of its operands (as the *regularized intersection* in [Til80]). In [DV93], this approach was chosen and two cases of intersection were defined, a geographic intersection that preserved dimensions and a regular geometric intersection. A general spatial type was defined as a geometric union of 0-, 1- and 2- dimensional figures, which can easily be implemented by ADT (such as in [JKO+89] in the closely-related domain of graphics databases) or object-oriented classes.

3. *Independence of the type w.r.t. the DBMS data model.*
   This concerns the use of particular features of a DBMS in the (encapsulated) ADT definition, that might lead to models lacking genericity. An example is [SV89], where the set constructor appears at the geometric level (a region is defined as being either an elementary region or a set of elementary regions, in order to meet the requirement on closure).

### 2.3.2 Operations on spatial ADT: A classification

Trying to give an exhaustive list of ADT operations is obviously not realistic, as many operations are application-dependent. Providing an application designer with a framework able to handle a large variety of such operations is more sensible. To our knowledge, although there exists a great variety of operations on spatial types, there is no standard classification of them. Many criteria for a classification can be of interest, which include the type of the arguments or the semantics of the operations (consequently leading to categories such as geometric operations, operations using a metric, or topological operations). Below is a classification that takes into account the signature of the operations on spatial objects, which are denoted `SO` (of type `Spatial` from the previous section). Each operation takes one or many spatial objects as arguments, and returns a spatial object, a boolean, a number, or a set of spatial objects. With this classification there is no distinction between geometric and topological operations. We do not describe here the structure of a `SO`: whether it is defined as an infinite set of points, a set of polygons, etc. is not relevant here. Note also that we distinguish spatial objects from *sets* of spatial objects although this is not absolutely

necessary as a spatial ADT could encompass the notion of set. Yet making such a distinction at this point is important for describing later a way to embed these types into a DBMS (Section 3).

- `(SO,SO) -> SO`
  These operations are binary (set) operations such as union, intersection, difference, which can be easily generalized to n-ary operations.

- `SO -> Bool ; (SO,SO) -> Bool`
  These predicates are test predicates, such as for adjacency or for the inclusion of an object in another one. objects.

- `SO -> n ; (SO,SO) -> n`,
  (where `n` denotes a real). These operations use a metric. Examples include length, perimeter and area of an object, or the distance between two objects.

- `SO -> SO`
  These operations are geometric and topologic transformation of objects such as rotation or change of scale, shape or position.

- `SO -> {SO}`
  Such operations include the decomposition of objects into their non-connected parts. They could be defined within an ADT if the type is general enough to handle the notion of set. In this case, the set is transmitted from the ADT to the database where the spatial operation is performed, as we shall see further. In addition, the database must be able to handle a set when returning the result of the operation.

- `{SO} -> SO; {SO} -> {SO}; {SO} -> n`
  These operations are performed on collections of objects. Examples include the computation of the center of a set of points, the Voronoi diagram, or the shortest distance between spatial objects.

For more information about spatial operations the reader is referred to [Peu84, KW87, Güt88b, LT92], and for a description of geometric algorithms to [PS85, GS95].

### 2.4 Examples of enduser operations

We now give several examples of simple operations that an enduser is likely to apply to maps. Some of them can be found in [SV89, GS93a]. The corresponding figures are placed at the end of this paper. For each operation we give its signature, in which `map` is the type `thematic-map` described in Section 2.2 and `Ai` a collection of alphanumeric attributes. To avoid any confusion with well-known algebraic relational operations, we chose to prefix operation names with "map" when there might be ambiguities. Most operations seem trivial at first sight, but expressing them in a database model is sometimes difficult, as we will see. Note that this set of operations does not pretend to be complete: someone may come up with new map operations that are not part of our list. Completeness would have to be defined with respect to another system, which does not make sense in this case as there is

no reference set of map operations. However, one could show that the set {map-projection, map-selection, map-union, map-overlay, merger, map-decomposition} is minimal.

### 2.4.1 Operations on sets of geographic objects

#### ◇ Map projection

The signature of this operation is `map` × `Ai` → `map`, where `Ai` is a collection of alphanumeric attributes of `map`. It gives back a map whose description is made of the `Ai` attributes and whose spatial part is unchanged. Note the difference with a standard relational projection where the spatial part is projected out (hence the "map" prefix here). Let us illustrate this operation and consider the map of the Western European countries with their name and population. As explained previously, each country is a geographic object, and the name of the country together with its population represents its description (Figure 2, left). By applying a *map projection* to the population, we eliminate the country names (right hand side of Figure 2). In the previous section, we made the distinction between an atomic or basic map (a map with explicit geometry) and a complex map with implicit geometry. Map projection easily applies to a basic map. In case of a complex map, the geometry has to be "pulled up" from the lower level by using aggregation, as detailed in the next section.

#### ◇ Map selection

The signature of this operation is `map` × `condition` → `map`, where `condition` is a predicate on one or many alphanumeric attributes (`condition` = p(`Ai`)). It is similar to the relational selection. On the previous map, by applying a *map selection* to the countries whose population is greater than 50 millions inhabitants, we obtain the map on the right hand side of Figure 3.

#### ◇ Map union

This operation (signature `map` × `map` → `map`) consists in performing the union of sets of geographic objects having the same schema. It is similar to the relational union. In the example of Figure 4, we consider two maps, one with the countries of Western Europe having less than 10 million inhabitants, and the other one with the countries having more than (or exactly) 10 million inhabitants. The relational union consists in gathering in a single map the geographic objects coming from these two maps. The final map is thus composed of all countries of Western Europe. If one wants to perform the union of heterogeneous maps (different schemas), then a common schema has to be defined first.

#### ◇ Spatial Selections

As opposed to the previous selection, the two following ones refer to a *spatial* property and they are useful for region and point queries. Their signature is `map` × `SO` → `map`. `SO` is for instance an area drawn by the user on the screen (e.g., a rectangle) or a point.

- **Windowing**
  By applying windowing, one gets the geographic objects of a map whose intersection with a given area is not empty. In Figure 5, the area is a rectangle drawn on the screen by the user. But the argument could have any shape, for instance a circle. Consider the following query [SV89]: "Get all countries located less than 100 kilometers from a given point". The windowing of the initial map with circle of radius 100 kilometers and center given by the user is performed. A special case of windowing is selection-by-pointing, or point query, which consists in selecting a single geographic object on a displayed map. For this operation, the area argument is reduced to a point, or to a very small region that permits a certain approximation.

- **Clipping**
  This operation extracts the portion of a map located within a given area (Figure 6). As opposed to windowing, the geometry of the result corresponds exactly to the intersection between the geometry of the geographic objects and the geometry of the area.

### ◇ Map overlay

This operation (signature `map` $\times$ `map` $\rightarrow$ `map`) is very common in GIS applications. It computes a new map from several overlaid maps. New geographic objects are created in the resulting map: their geometry is computed by applying the intersection operation to the geometry of the participating geographic objects, and their description is a combination of the participating descriptions. In Figure 7, two maps with a different partitioning are drawn: the map of Western European countries (whose description is the country name) and the map of the repartition of families of spoken languages over Western Europe. The result of the overlay is a map whose geographic objects are areas characterized by both the country they belong to and the family of language spoken in the area. The maps considered do not have necessarily the same surface. In this case, the resulting surface is the intersection of the participating surfaces (regions). Note also that clipping can be seen as a particular case of map overlay, if a map is created from the geometric argument (a simple map with only one geographic object having no description).

### ◇ Merger

The merger operation performs the geometric union of the spatial part of $n$ geographic objects that belong to the same map, under a condition given by the enduser. (Figure 8). As it operates within a single map its signature is `map` $\times$ `condition` $\rightarrow$ `map` (observe the difference with the map union, which takes as arguments several maps and gathers them into a single one.) Merger relies on the concept of object aggregation (see further) and on the `union` operation on set of spatial objects. According to the condition, it partitions the input set, similarly to a SQL "group-by". Below is an example of the merger operation using our reference schemas (Query 1).

**Notation regarding the queries:**

1. From now on we omit the type of attributes when it is not essential for understanding.
2. Applying a function `f` to a value of attribute `A` is denoted `f(A)` in the explanations below. Creating an attribute `B` whose value is the result of applying `f` to `A` is denoted `B <- f(A)`. This is an abbreviation for the composition of the *extend* operation and projection on all attributes but `A`.

---

**Query 1:**
**From the map of Western Europe, build the map of language regions using the main spoken language of each county** (*merger*)
This query shows an aggregation on the spatial part of counties. The evolution of the schema during its execution is given below. The counties whose main spoken language is the same are first grouped and their geometry is gathered in a set (nest). Then the geometric union is applied to each set (function `union-set` on spatial ADT).

0. Initial Schema

```
States:  {State:  [StateName,
                   Counties:
                       {County:[CountyName,
                            Population,
                            MainSpokenLanguage,
                            Geometry]}]}
```

1. Gather all the counties of all states in a single set (map projection on `County` and set-collapse):

```
States':  { County:[CountyName,
                    Population,
                    MainSpokenLanguage,
                    Geometry]}
```

2. Project on `MainSpokenLanguage` and `Geometry` and `nest` on `Geometry`:

```
States'':  { County':[MainSpokenLanguage, G:{Geometry}]}
```

3. Apply `union-set` to set G (G' ← `union-set`(G)):

```
States'':  { County'':[MainSpokenLanguage, G']}.
```

---

In [SV89, GNT91, GS93a], a somehow comparable operation called *fusion* is proposed. Its semantics are slightly different since the union of the geometry of geographic objects is applied *if they have the same value for a given attribute*. For instance, if the fusion attribute is crops, two fields growing corn on a crops map are merged to form a single field. However, this is a particular case which illustrates the need for a more general merger operation that allows one to gather several geographic objects under a general condition given by

user-defined predicates. In the complex algebra we use, we can always express this via a composition of operations (cross product followed by relational selection on the predicate in question, followed by an aggregation and a geometric union). We cannot express the general case because a second order signature tool is missing.

With such an operation, a combination of alphanumeric attributes (such as a sum of populations in Figure 8) can be considered. New values are assigned to (alphanumeric) attributes as described in our queries.

### ◇ Map decomposition

The map decomposition operation creates independent geographic objects from their non-connected geometric components. Similarly to merger, its signature is `map` $\rightarrow$ `map`. Figure 9 gives an example of the decomposition of a map. Take for instance the simplified geometry of France (the same reasoning applies to other countries having islands): it is composed of two non-connected polygons, the bigger one corresponding to the mainland and the smaller one corresponding to an island, Corsica. After the decomposition of map $m$, the polygon of Corsica is the spatial part of a new geographic object.

Map decomposition turns out to be a complex operation as (i) the non-connected parts of a geographic object have to be isolated, and (ii) attributes and attribute values of the newly created objects have to be specified (here, the name "Corsica" has to be entered, and the population either removed or set to some unknown value). In any case, there must exist a way for restructuring the objects, possibly extend their description, and for computing new values for some attributes.

This operation can become quite complicated in some GIS applications when new alphanumeric values require sophisticated computing. For instance, such a value can be a ratio that depends on the values of other attributes of either the same object (e.g., the area of a spatial attribute value) or of all the objects of the map (e.g., the average of one attribute). This requires the existence of a high level function, say $g$, which operates on each element $e$ (geographic object) of a map $m$ to assign a value to a (possibly new) attribute $A_n$, as follows ($e.SpatialAtt$ is not indispensable here as it behaves as any other attribute. Nevertheless, given its importance in geographic applications, mentioning it better illustrates our discourse.):

$$e.A_n = g(e.A_i, e.SpatialAtt, f(m.A_j)), \forall e \in m$$

In other words, the new attribute value is a function ($g$) that is a combination of (i) the value of an attribute $A_i$, (ii) the value of the spatial attribute $SpatialAtt$ and (iii) possibly a function applied to the attributes of all the elements of a map such as average (here $f(m.A_j)$).

### 2.4.2 Operations based on geometric algorithms

Even though some operations presented above make intensive use of spatial type functions, most of them belong to what can be considered as an algebra on maps: they take as argument one or several maps and return a map. However, it turns out that some applications require specific operations whose output does not belong to the map domain anymore. As a simple case, take selection-by-pointing which returns a single geographic object. A geographic object can be of course stored in a map as a singleton, but this is not very elegant. Below is an example of query whose execution requires such a specific operation (Query 2).

---

**Query 2:**
**Find the shortest path between Paris and Berlin**
First, the Paris and Berlin coordinates have to be located on the network (provided that we do not have maps of cities). This can be performed as follows. From Counties, extract counties of Paris and Berlin, and for each one get a representative point such as its center (apply a geometric function `center` on each county's geometry).

Then the query execution consists in calling the `shortest path` algorithm on the network (external function), with the two end-points.

---

We do not want to elaborate on graphs here: the reader will find in [Güt94b] an approach for embedding graphs in databases, together with algorithms. What we want to show through this example is the existence of two independent features: (i) the possibility of defining and calling elaborate functions on the geometry (such as `shortest path`), and (ii) the possible heterogeneity of the result of such queries, which can be for example a map (a sub-network in this particular case) or even a spatial object (the union of lines composing the path).

### 2.4.3 Operations based on geometric algorithms and descriptive attributes

A problem may arise in the input structuring when such specific functions require data from the description of geographic objects. On the previous example, suppose that one wants to find out the average time it takes by car between the two given cities (function of the maximum speed on a segment, the average traffic and the length of a segment). The shortest path operation is applied between these two points, with a cumulative sum of average traveling times per road segment. This example shows the case of a function that is more than a pure geometric operation. Indeed, the special shortest path algorithm invoked in this case takes as input not only the geometry of all the segments (in order to obtain the network), but also attribute values of the description for each segment. Therefore one must be able to specify such an input, which is feasible in our model by applying composition of functions.

As another example, take the allocate function, which is a common function on networks in

GIS (see for instance [ESR89]). The allocate function takes a network consisting of edges and nodes and a set of centers as arguments. Its purpose is to allocate each edge and node of the network to a center by considering a resource demand. In practice, resource demand is for instance the number of students who live along an edge or on a node in a street network. Each center has a resource capacity which is the total number of resources which can be supplied to or from a center to meet the demands along edges and nodes. Resource capacity can be for instance a school's capacity. A similar case occurs with "Voronoi with weight", that is a special case of Voronoi (which is a function on the geometry) that computes a new map not only from geometric constraints but also from alphanumeric ones.

All the examples above, and many others such as a kriging algorithm, show that although some operations might look like pure geometric functions at a first glance, they sometimes require values from the description as well. They are applied on maps as a whole. The crucial point is to be able to specify the input and the output of these functions. As shown previously, the input is more than a simple map as it concerns a structuring of its elements, subparts of their description and of their spatial component.

## 2.5 Aggregation/disaggregation

Although *aggregation* and *disaggregation* are meant as database features, their role is essential in map manipulation as illustrated above (e.g., merger and decomposition). The basic case of *aggregation* is the following. Consider the schema: $\{[A : \alpha, B : \beta]\}$, and suppose that a function `f` such as `sum, union, ...` is to be applied to attribute $B$. The signature of `f` is $\{\beta\} \rightarrow \beta'$. To express that correctly, a first step consists in gathering the values of $B$ in a set $C$ as follows: $\{[A : \alpha, C : \{B : \beta\}]\}$ and then to apply function `f` to the created set ($D \leftarrow$ `f(`$C$`)`), where $D$ is a new attribute name: $\{[A : \alpha, D : \beta']\}$. When the language used is SQL and $\beta$ is an ADT, the set is not created explicitly. Only the attribute on which function `f` is to be applied is given (e.g., "`Select` Name, `sum (`$B$`) from` ... `where` ..."). In the case of *disaggregation*, a function `g`: $\{\beta\} \rightarrow \beta'$ is considered. Similar schema transformations can be defined.

Aggregate/disaggregate functions are defined on collections of attributes values, i.e. on *sets* of values in a database context, such as a aggregate function `sum` defined on integers. Therefore a set has to be introduced for (re-)structuring before or after applying the function. This is not a problem in the complex object model we use. However, it is difficult to introduce aggregate functions in the standard relational model because of the non-explicit existence of *sets of values*. Hence we see again the power of explicitly manipulating sets arbitrarily deep. The two following queries are examples of the use of aggregate/disaggregate functions.

**Query 3:**
**Create the map of states from their counties with sum of their population (*aggregation*)**

This query shows how to aggregate data according to the hierarchy defined by the data schema. Its execution requires the following steps:
For one state ("replace"), do the following: (i) compute the total of population of all its counties (function `sum`), (ii) perform the geometric union over the regions of all its counties (function `union` on the spatial attribute). The schema evolution is the following:

0. Initial schema (we give up the county names and languages )
   `States: {State: [StateName, Counties: {County:[Population, Geometry]} ]}`

1. Restructuring (for each state, projection on both the population and the geometry of its counties)
   `States': {State: [StateName, P:{Population},G:{Geometry} ] }`

2. Treatment: Apply `sum` to populations (`P' ← sum(P)`) and `union` to regions (`G' ← union(G)`), where `P'` and `G'` represent new attribute names
   `States'': {State: [StateName, P', G']}`.
   As stated before, all these operations can be expressed using *replace* and ADT operations.

**Query 4:**
**Decompose counties into connected parts and distribute the population proportionally to the area (*"disaggregation"*)**

This query shows the ability to disaggregate a spatial value into a set of spatial values and a particular case of allocation. First, we need the function `decompose` that returns all the connected parts of a single region in a set of values of type `Spatial`:

$$\texttt{decompose}(R : region) \rightarrow R' = \{R_i\}, with(\cup R_i = R)\ and\ (R_j \cap R_i = \emptyset)$$

The query execution is then the following: (i) apply `decompose` to the geometry of all counties ("replace", resulting in a set of regions G' for the geometry, (ii) create a new geographic object for each element of G' ("disaggregation") with a population value computed withfrom the ratio of the respective area.

0. Initial Schema (we focus on the population and geometry of counties)
   `Counties:{County:[Population,Geometry]}`
1. Add attribute G' by applying `decompose` to Geometry (G' $\leftarrow$ `decompose`(Geometry))
   `Counties':{County:[Population,Geometry,G':{Spatial}]}`
2. For each value $R_i$ (attribute name Geometry, type `Spatial`) in $R'$ (attribute name G', type set of Spatial), compute a new tuple having as spatial attribute `Geom` and as new population attribute `P'` $\leftarrow$ `(County.Population*area(`$s$`)/area(County.Geometry))`, where `county.Population`,
   `county.Geometry` represent respectively the population value and the geometry of the county that has to be decomposed, and `area` a function on spatial ADT that takes and ADT and returns its area (a real):
   `Counties'':{County:[Population,Geometry,G':{[P':`
   `real,Geom:Spatial]}]}`
3. Project out Population and Geometry and collapse the inner set and tuple:
   `Counties''':{County:[P', Geom]}`

The informal presentation of maps and operations above points out the necessity of having (i) operations on the description (the non-spatial part), as well as (ii) high level operators, such as iterators and operators for restructuring information, in order to apply operations to collections of geographic objects and to parts of them. Furthermore, it also shows intuitively that the "higher level" (maps) processing relies on operations defined

on the "lower level" (the spatial part of the geographic objects, i.e., the geometry). For instance, the merger operation presented above needs the geometric union defined on spatial types. However, this is not enough as operations such as some specific operations on the geometry described above should be defined easily. Hence we see a need to combine general operators and user-defined functions. As far as previous approaches are concerned, note that the operations above were not all treated in [SV89] as it was not the focus of this approach. In [GS93a], only a particular case of the merger operation, the fusion, is considered, and queries 1 and 4 are treated using specific operators introduced for this particular purpose.

# 3 Implementing Conceptual Geographic Models in a DBMS

We now address the problem of defining a logical model that corresponds to the conceptual geographic model of Section 2. If we focus on logical aspects of map manipulation as opposed to physical aspects, it appears from Section 2 that a DBMS should provide the following features for representing and manipulating geographic information:

1. A possibility to define a spatial data types with operations defined on them.
2. Constructors for considering collections of objects and semantic links among objects (e.g., hierarchies of geographic objects): for instance, the *set* constructor for a collection of geographic objects and the *tuple* constructor for describing the structure of an object. In addition, in non-flat database models, the set constructor can be used for describing relationships between objects (as done in the previous section).
3. The possibility of refererring to the spatial part of an object, e.g., by a dot notation (for instance `California.Geometry` for a named object `California`, or more generally, `x.Geometry` for any geographic object variable `x`).
4. Operators for handling collections of data, no matter their nature (set of descriptions, of spatial objects, of geographic objects, etc.) and which allow restructuring.

This section starts with a classification of the required operators within a standard database system, whatever its underlying model is. Following this classification, we then study how to match the requirements with DBMS data models. Two major database models are considered, namely a relational one extended by ADTs and an object-oriented one. We illustrate our discourse through the examples of Section 2. We end up with a study, for both of these data models, of (i) the implementation of the operators, (ii) the representation of geographic data, and (iii) the formulation of the reference queries.

## 3.1 Three classes of operators

From the operations and queries of the previous section, three classes of operators applicable to geographic objects can be extracted, according to the kind of objects they apply to: (i) ADT or sets of ADTs, (ii) sets of objects independent of each other (i.e., object after object), and (iii) collections of objects as a whole, i.e., combinations of geographic objects, functions of spatial objects, functions of decriptions or subpart thereof, etc. In [Güt88a], operators

are divided into six major groups. The classification criterion is different from ours as it corresponds to their type of output in a relational environment. Our approach is orthogonal to the one of [Güt88a] and is oriented towards the study of geographic DBMS extensibility. More precisely, our three classes are the following:

1. **Class I: User-defined functions .**
   We take as examples operators on spatial ADT, but this class applies to any (non-standard) data type. Apart from the simple case where a function takes one or many ADT as arguments and gives back a value of a basic type (e.g., `distance`: ADT X ADT $\rightarrow$ real), input and output of these operators can be [1]:

   - *ADT X ADT* (input) and *ADT* (output): Class I.a
     e.g., geometric `intersection`.
   - (*set of ADT* and *ADT* (Class I.b)
     i.e., aggregate functions on ADT such as `union` (sometimes denoted `sum`).
   - *ADT* and *set of ADT* (Class I.c)
     i.e., "disaggregate" functions on ADT such as `decompose` (or `connected-parts`).
   - *set of ADT* and *set of ADT* (Class I.d)
     such as the `voronoi` operator which takes a *set of points* as arguments and returns a *set of regions* (note that the set of points could also be a simple ADT, with the set notion incorporated in it. ).

2. **Class II: Database (generic) algebraic operators.**
   While executing these operations (e.g., projection and selection) objects (tuples) are considered sequentially and independent of each other. Thus we shall call the processing of data "linear". In our context, these operations are simply expressed using one *replace*. We cannot describe the general mechanism here because we would need second order tools (for describing functions of functions). Hence, when using this simple although powerful underlying model, operations must be described case by case, as we did above. We could have chosen other attractive base models in the domain of complex objects, as for instance FAD [BBKV87]. In this approach, the user is provided with the possibility of defining the abstraction of functions and predicates, which allows to express anonymous functions (denoted by `fun x y ....`), which can be used in turn as argument to other functions. It is very similar to the lambda calculus definition of functions. The possibility of expressing such functions establishes a bridge between databases and programming languages.

3. **Class III: Specific database operators with "non-linear" processing.**
   The input or output of these operators (e.g., `shortest path`) is more complex than a simple set of ADT as illustrated throughout our examples. Thus they cannot be defined on ADT. They are not part of the regular set of algebraic operations either, because they are not generic: if the application changes, they may have to be modified as well. Hence, they correspond to some other class of user-defined operators. The

[1] Only the most common classes are represented here.

input and output structures are "frozen" and the structuring before and after processing is done by the programmer using database operators of Class II. For defining this kind of operators, programming languages are of major interest. From a data model point of view, such an operator is only defined by its signature. In other words, only the interface to the algorithms is relevant.

As an example, take the shortest path operation on the `StateHighwayNetwork`, and consider a weight on each edge it consists of. This weight can be a function `f(maximum-speed, traffic)`. The algorithm takes as input (i) the begin and end points, and (ii) the network itself, that is a set E of edges together with their weight, whose schema is `(edge:spatial,weight:real)`. It might return a value which is a set E' of type `edge:spatial`. The algorithm itself is not relevant here and can be seen as a "black box".

Figure 1 summarizes these three classes.

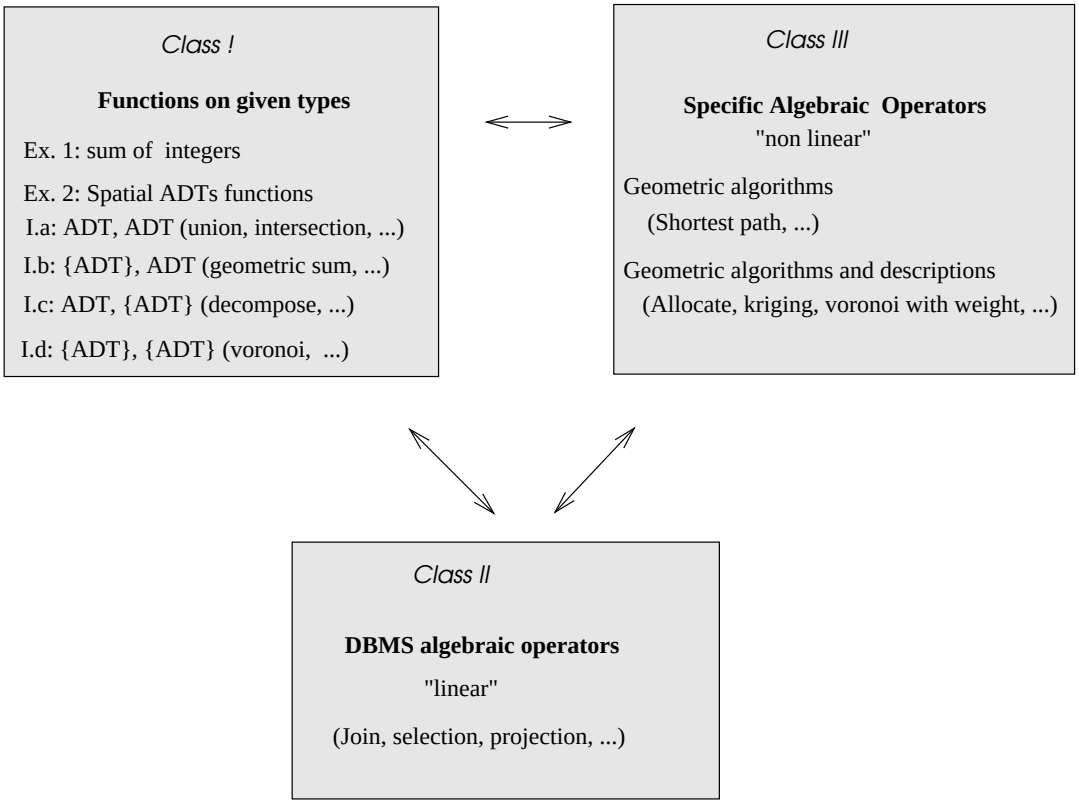



Figure 1: Three classes of operators in a geographic DBMS

## 3.2 Embedding the operators in a DBMS

The previous considerations lead us to consider the problem of extensibility in geographic DBMS (note that we are focusing on single systems rather than on distributed systems, even though some operators can be part of external libraries). More precisely, the problem is to find the minimal set of operators to be attached to each class. The tradeoff is the following: having Class III (specific GIS operators) completely independent of Class II (generic database operators) might lead to inconsistencies. On the other hand, the interest of having Class III separated relies on the fact that at implementation time, this set of operators can correspond to a module external to the database (expert system, statistics, ...). Then, merging Class III and Class II leads to a DBMS too specific. In the same way, it is now well known [SV92, GS93a] that Class I has to be independent from Class II for efficiency. It could be implemented by a specific geometric library.

One crucial problem is the communication between these modules, i.e. the combination of the three categories of operators. In a consistent framework all operators can be combined. For instance, map overlay is a combination of join and geometric intersection. More precisely, it is nothing other than a $\theta$-join [Ull88] with the geometric intersection test as the $\theta$ predicate. Class II belongs to the DBMS kernel. Take for instance Class I and the `decompose` operation, which takes a region and returns its non-connected parts in a *set*. The set constructor has to be known by Class I, which is not trivial. A similar problem arises for Class III, whose operators take *sets* as arguments. These modules must have an analogous notion (e.g., collection) that can be mapped to the database "set" concept. Such a mapping is realized both ways in the interface between the two modules.

## 3.3 Relational DBMS extended to ADTs

There have been several proposals to extend the original relational model with abstract data types [SRG83, SR86, Sto86, GCK+89]. This facility has often been used to add a spatial data type within a relational DBMS. ADT enable user-defined operations from Class I, but only from the first subclass "ADT x ADT and ADT" (Class I.a). The three other subclasses (Classes I.b, I.c, I.d) cannot be easily defined in the framework of the extended-relational data model because of its inability to express explicitly sets of values.

### 3.3.1 The operator classes

Some operations of subclass "set of ADT to ADT" (Class I.b) can be expressed as aggregate functions which are defined in relational query languages such as SQL. However, aggregate functions are not universally accepted in the relational model philosophy. As far as languages are concerned, aggregate functions are defined in an *ad hoc* manner, and extending the mechanism to abstract data types is therefore difficult. The two last subclasses of operations (i.e., classes I.c and I.d) cannot be expressed either in query languages such as SQL.

Most of the Class II algebraic operators are part of the relational algebra. In an

extended-relational system such as Sabrina-Objet [GCK+89], it is possible to use any function or predicate in the operators. Some operators such as *nest* or *unnest* do not fit into the relational context, and we will see below that a query such as Query 4 cannot be expressed.

Specific algebraic operators (Class III) cannot be generally introduced in extended-relational systems, but some systems can handle them by coupling the DBMS with a programming language (e.g., RAD [OH86] and its *transformation operations*). This drawback of the relational approach is particularly regrettable since many geographic operators such as `shortest path` have to be expressed as specific operators.

GeoSabrina [LPV93] is built on top of an extended relational system [GCK+89]. In this system, Class I.a operations are defined as (regular) ADT functions, Class I.b operations are considered as aggregate functions, and operations of Class I.c, Class I.d, and Class III cannot be implemented. Gral [Güt89] is based on an extended-relational approach that allows the designer to add new executable operators. However, due to the lack of the set constructor, Class I and Class III are merged.

### 3.3.2 Data representation

In contrast to a complex object model as the one used in Section 2, the relational model imposes flat tuples representation, which means that object composition cannot be expressed by nesting the set and tuple constructors. It is therefore necessary to split the representation in several relations and to keep a foreign key to be able to retrieve hierarchical information. Hence the `States` and `Counties` representation could be defined as follows:

```
create table States (StateName: string)
create table Counties (StateName: string,
                       CountyName: string,
                       Population: integer,
                       MainSpokenLanguage: string,
                       Geometry: spatial)
```

Similarly, the `StateHighwayNetwork` and `RoadElement` representation is the following:

```
create table StateHighwayNetwork (HighwayName: string,
                                  StateName: string)
create table RoadElement(HighwayName: string,
                         MaximumSpeed: integer,
                         AverageTraffic: real,
                         Edge:spatial)
```

### 3.3.3 Expressing queries

In an algebraic approach, expressing queries is equivalent to combining operators of the previous classification. However, we chose to express them below in SQL-like languages as

it corresponds to the standard enduser languages. We now express the queries of Section 2 in extended-SQL.

---

**Query 1:**
**Build the map of language regions using the main spoken language of each county** (*merger*)

```
select MainSpokenLanguage, sum(Geometry)
from Counties
group by MainSpokenLanguage
```

Note that this approach does not allow to keep the geometries in a set without running explicitly an aggregate function such as the geometric union.

---

**Query 2:**
**Find the shortest path between Paris and Berlin**

This query cannot be expressed in most SQL-based relational DBMS because an external function such as `shortest path` cannot be used. Such cases are usually handled by embedding SQL into a programming language such as C or C++, in which such algorithms are coded (they can also be coded in a more appropriate language such as a functional language called from C).

---

**Query 3:**
**Create the map of states from its counties with sum of their population** (*aggregation*)

```
select s.StateName, sum(Population), sum(Geometry)
from States s,  Counties c
where s.StateName = c.StateName
group by s.StateName
```

---

**Query 4:**
**Decompose counties into connected parts and distribute the population proportionally to the area** (*"disaggregation"*)

This query cannot be expressed because (i) disaggregate function such as `decompose` (Class I.c) cannot be expressed, and (ii) the unnest operator is not defined.

---

## 3.4 Object-oriented DBMS

In the past ten years, object-oriented DBMS (OODBMS) have received considerable attention from the database community. One of the goals of object-oriented DBMS (OODBMS) was to allow extensibility. This is generally done by allowing the use of a general programming language such as C or C++ to express the operations on objects stored in the database.

In the GIS community, some experiments were made to implement a GIS, or part of, on top of OODBMS such as $O_2$ [SV92], or ObjectStore [GR93]. Embedding typical GIS features within a so-called object-oriented GIS was studied as well [EF92]. For more information regarding the use of object-oriented system to handle geographic data, see [EF92, GL93, Wor94].

### 3.4.1 Defining spatial ADTs and operator classes

Operations of both Class I and Class III can be expressed using the general programming language of an OODBMS. Spatial ADT can be defined as a class in any OODBMS, and operations of classes I.a and I.c can easily be defined as methods on that class. It is necessary to define "a set of spatial ADT" either as a class or as a value in a OODBMS that makes a difference between object and value (such as $O_2$ [BDK91]). Let us take as an example the case of $O_2$, which has a clean data model and which was used as a basis for several other proposals [SV92, GS93a, DRSM93].

The definition of the spatial ADT corresponds to the following class definition:

```
class spatial
type ... /* not relevant here */
method
  area: real,
  center: spatial,
  intersect (arg: spatial): boolean,
  intersection (arg: spatial): spatial,
  union (arg: spatial): spatial,
  decompose: set(spatial)
end class;
```

User-defined operations may be defined as methods on a class if the first argument of the function is an object of the class. The declaration of `union` and `intersection` methods shows examples of class I.a operations. The `decompose` method realizes the disaggregation of a spatial value; it is an example of class I.c operations.

For other user-defined operations, general functions have to be used. Unfortunately, function overloading is not permitted in the $O_2$ model. The following example shows how to define:

- A function that realizes the aggregation of spatial values (class I.b):

```
function union (arg: set(spatial)): spatial;
```

- A function that calculates the Voronoi diagram (class I.d):

```
function voronoi (arg: set(spatial)): set(spatial);
```

- A function that computes the shortest path (class III):

```
function shortestpath (begin: spatial,
                       end: spatial,
                       network:set(tuple (edge:spatial,
                                          weight:real))): set(spatial);
```

### 3.4.2 Data representation

The schemas defined in Section 2 can be expressed directly in the $O_2$ definition language [O2T93] by using the tuple and set constructors. For example the `States` schema can be defined as:

```
type States: set (State);
type State:  tuple (StateName: string,
                    Counties: set (County));
type County: tuple (CountyName: string,
                    Population: integer,
                    MainSpokenLanguage: string,
                    Geometry: spatial);
```

The named value containing the set of states may be defined by:

```
name states: States;
```

Similarly, the `StateHighwayNetwork` schema can be defined by:

```
type StateHighwayNetwork: set (StateHighway);
type StateHighway: tuple (HighwayName: string,
                          StateName: string,
                          Elements: set (RoadElement));
type RoadElement: tuple (MaximumSpeed: integer,
                         AverageTraffic: real,
                         Edge:spatial);
name stateHighwayNetwork: StateHighwayNetwork;
```

We only use set and tuple constructors (definition of types) and no class constructor. This means that no method can be defined on these sets and tuples and that values cannot be shared in the database. It is also possible to define corresponding classes. This would make no difference in the query language but would allow data sharing and method definition. However, for the sake of fairness when comparing this approach with extended-SQL, we do

not use methods for expressing queries in the following. Hence we need not consider the notion of classes and objects here. The presence of classes, objects and methods at the geometry level (`class spatial`) is justified by the extreme similarity between ADT of the extended-relational model and classes of an object-oriented model. As we will see further, the major difference of the two models concerns data manipulation at the map level.

### 3.4.3 Expressing queries

We now use the $O_2$SQL language [O2T93] in order to express the queries of Section 2. $O_2$SQL is a database query language whose syntax is close to SQL standard. It was a basis for defining OQL, the new SQL-like standard query language proposed in 1994 by *ODMG* (the OODBMS vendors consortium).

We now illustrate the use of $O_2$SQL to express the reference queries (all examples have been tested):

---

**Query 1:**
**Build the map of language regions using the main spoken language of each county** (*merger*)

```
group x in
     (select tuple (c.MainSpokenLanguage, c.Geometry)
      from s in states, c in s.Counties)
by (MainSpokenLanguage: x.MainSpokenLanguage)
with (Geometry: union (select p.Geometry
                       from p in partition))
```

---

**Query 2:**
**Find the shortest path between Paris and Berlin**

We assume the existence of a function `f` that computes the weight needed for the shortest path:

```
function f (averageTraffic: real, maximumSpeed: integer): real;
```

The shortest path is computed using the following expression:

```
shortestpath (Paris.Geometry->center,
              Berlin.Geometry->center,
              select(tuple(edge: r.Edge,
                           weight: f(r.AverageTraffic,r.MaximumSpeed)))
              from n in stateHighwayNetwork, r in n.Elements)
```

with `Paris` and `Berlin` considered as named objects (counties), and `center` returning a point corresponding to the center for a given Geometry.

---

**Query 3:**
**Create the map of states from their counties with sum of their population** (*aggregation*)

can be expressed using $O_2$SQL:

```
select tuple
( Name: s.StateName,
  Population: sum (select c.Population from c in s.Counties),
  Region : union (select c.Region from c in s.Counties))
from s in states
```

---

**Query 4:**
**Decompose counties into connected parts and distribute the population proportionally to the area** (*"disaggregation"*)

```
select tuple (Population: c.Population * e->area / c.Region->area,
              Region: e)
from s in states,
     c in s.Counties,
     e in c.Region->decompose
```

---

In summary, using "extended SQL" seems easier than using $O_2$SQL for expressing ag-

gregates (such as `sum` or `union`), thanks to the `group by` clause. However, the inability to represent sets in a relational-based model restricts the use of functions: Many GIS functions such as `decompose` or `Voronoi` are not expressible in most systems based on this model. This is not the case in an object-oriented approach, where all the operations defined in Section 2 can be expressed.

# 4 Conclusion

In this paper we studied the problem of modeling and manipulating geographic data in a DBMS environment. Geographic data (maps) can be decomposed into two levels of abstraction whatever the underlying database model is. A higher level, the *map level*, is constituted of geographic objects, and a lower level, the *geometric level*, corresponds to the spatial components of geographic objects. General operations on maps (e.g., map overlay) use low level operations on the geometry (e.g., intersection). In recent approaches [Güt88b, SV89, GN90, GNT91, Dav91, GS93a], the geometric level corresponds to the definition of spatial abstract data types (ADTs). Some of this work [Güt88b, Dav91] is based on the relational model extended to abstract data types. It suffers from several drawbacks, including poorness of data structures, embedding of geometric operations in high level operations, and lack of flexibility. The thematic-map model of [SV89] is based on a complex object model and emphasizes the separation between map level and geometric level. The originality of this approach is the association of the geometric operations with the data model through general constructs. However, a database model concept, the set constructor, still appears at the lowest level and is used intensively inside ADT manipulation, which leads to a model that is difficult to understand.

A useful notion that appears in many GIS operations is the one of of aggregation, with its dual, disaggregation. These two operations are used heavily in statistical databases. Aggregation allows one to apply a function to a set of values (e.g., the geometric union of several regions), hence switching from a given dimension to a lower one. Disaggregation allows one to get a set of values as a result of a function (e.g., the decomposition of regions). Unfortunately, aggregate/disaggregate functions are not well defined in the relational model (impossibility of manipulating explicitly sets of values), and only aggregate functions are considered in standard SQL.

Given these observations, it seems necessary to investigate precisely which database features are required for defining powerful geographic models. Our approach consisted in studying the mapping of a general geographic model onto DBMS data models. We first presented our reference model, i.e., a way of representing geographic objects, their spatial part (spatial ADT), and of manipulating them through general map operations. As an underlying database model, we chose [AB95] because of its power for representing and manipulating complex (geographic) objects. Through the set of queries we presented, the reader could get an impression of the type of functions one is likely to consider within a geographic DBMS.

We then studied the embedding of both the spatial ADTs and operations on maps into a database environment We proposed three classes of operators for manipulating geographic data: (i) user defined functions on ADT (e.g., geometric union), (ii) generic database algebraic operators with tuple-by-tuple processing such as selection and projection (which we called "linear processing"), and (iii) specific algebraic operators with "non-linear" processing (e.g., shortest path or allocate). We discussed the repartition of operations in these three classes. For instance, gathering user-defined functions and algebraic operations in the same class leads to a closed DBMS that is certainly too specific. We believe that considering these three independent classes is essential for flexibility and extensibility while designing geographic systems on top of existing DBMS.

We then took two database models as support: a relational model extended to ADT and an object-oriented one. In both cases, we studied (i) data representation (ADT and object composition), and (ii) query formulation (i.e., adequacy for defining and using the three classes of operators). The queries were formulated in SQL-like languages (generic "extended SQL" and $O_2$SQL respectively) for a better comparison of the two approaches.

It turns out that in practice, due to the suitability of SQL for some operations, queries based on aggregation (e.g., queries 1 and 2) are easily expressible in extended SQL (use of `group by`). On the contrary, their formulation in $O_2$SQL is more complicated because operations on sets have to be explicitly expressed. This is the price to be paid for a model handling sets, which allows on the other hand the expression of queries such as Query 4 (disaggregation).

As far as calls to specific operators (Query 2) are concerned, SQL alone is not sufficient. However, a query of that type can be expressed in an algebraic language (e.g., in Gral [Güt89]) although such a solution may lead to interfaces difficult to use. Finally, $O_2$SQL allows to embed both set handling and calls to specific operators in a unified way.

## References


[AB86] S. Abiteboul and N. Bidoit. Non First Normal Form Relations: An Algebra Allowing Data Restructuring. *Journal of Computer and System Sciences*, 1986. Extended abstract in Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1984.

[AB95] S. Abiteboul and C. Beeri. On the power of languages for the manipulation of complex objects. *The VLDB Journal*, 4(4):727–794, 1995.

[AH87] S. Abiteboul and R. Hull. IFO: A formal semantic database model. *ACM Transactions on Database Systems*, 12(4):525–565, December 1987.

[BBKV87] F. Bancilhon, T. Briggs, S. Khoshafian, and P. Valduriez. FAD, a Powerful and Simple Database Language. In *Proc. Conf. on Very Large Data Bases (VLDB)*, Brighton, 1987.

[BDK91] F. Bancilhon, C. Delobel, and P. Kanellakis, editors. *Building an Object-Oriented Database System: The Story of $O_2$*. Morgan Kaufmann, San Mateo, California, 1991.

[Che76] P. P. Chen. The entity relationship model - toward an unified view of data. *ACM Transactions on Database Systems*, 1(1):9, March 1976. Reprinted in M. Stonebraker, Readings in Database Sys., Morgan Kaufmann, San Mateo, CA, 1988.

[Cou92] H. Couclelis. People Manipulate Objects (but Cultivate Fields): Beyond the Raster-Vector Debate in GIS. *Lecture Notes in Computer Science, Springer-Verlag, Berlin*, 639, 1992.

[CW85] L. Cardelli and P. Wegner. On Understanding Types, Data Abstractions and Polymorphism. *ACM Computing Survey*, 17(4):471–522, 1985.

[Dav91] B. David. *Modeling, Representing and Managing Geographic Information: An Extended Relational Approach.* Ph.D. thesis, University of Paris VI, 1991. In French.

[DRSM93] B. David, L. Raynal, G. Schorter, and V. Mansart. GeO2: Why Objects in a Geographical DBMS? In D. Abel and B.C. Ooi, editors, *Advances in Spatial Databases (SSD'93)*, pages 264–276. Lecture Notes in Computer Science No. 692, Springer-Verlag, Berlin, 1993.

[DV93] B. David and A. Voisard. A Unified Approach to Geographic Data Modeling. Technical Report 9316, University of Munich (LMU), 1993. available at http://www.inf.fu-berlin.de/ voisard/pub.html.

[EF89] M. Egenhofer and A. Frank. Object-Oriented Modeling in GIS: Inheritance and Propagation. In *Proc. Auto-Carto 9*, pages 588–598, 1989.

[EF91] M. J. Egenhofer and R. D. Franzosa. Point-set Topological Spatial Relations. *International Journal of Geographical Information Systems*, 5(2):161–174, 1991.

[EF92] M. Egenhofer and A. Frank. Object-Oriented Modeling for GIS. *Journal of the Urban and Regional Information Systems Association*, 4(2), 1992.

[EFJ90] M. J. Egenhofer, A. U. Frank, and J. P. Jackson. A topological data model for spatial databases. In A. Buchmann, O. Günther, T. R. Smith, and Y.-F. Wang, editors, *Proceedings of the 1st Symposium SSD on Design and Implementation of Large Spatial Databases*, volume 409 of *LNCS*, pages 271–286, Berlin, 1990.

[EN89] R. Elmasri and S. Navathe. *Fundamentals of Database Systems.* Benjamins/Cummings, Redwood City, 1989.

[ESR89] Inc. ESRI. ARC/INFO Network User's Guide, 1989.

[GCK+89] G. Gardarin, J.P. Cheiney, G. Kiernan, D. Pastre, and H. Stora. Managing Complex Objects in an Extensible Relational DBMS. In *Proc. Conf. on Very Large Data Bases (VLDB)*, 1989.

[GL93] O. Günther and J. Lamberts. Object-Oriented Techniques for the Management of Geographical and Environmental Data. *The Computer Journal*, 37(1):16–25,, 1993.

[GN90] M. Gargano and E. Nardelli. A Logical Data Model for Integrated Geographical Databases. In Raymond T. Ng, Peter A.; Ramamoorthy, C.V.; Seifert, Laurence C.; Yeh, editor, *Proceedings of the First International Conference on Systems Integration*, pages 473–481, Morristown, NJ, April 1990. IEEE Computer Society Press.

[GNT91] M. Gargano, E. Nardelli, and M. Talamo. Abstract Data Types for the Logical Modeling of Complex Data. *Information Systems*, 16(5), 1991.

[GR93] O. Günther and W.-F. Riekert. The Design of GODOT: An Object-Oriented Geographic Information System. *IEEE Data Eng. Bull.*, 16(3):4, September 1993.

[GS93a] R.H. Güting and M. Schneider. Realm-Based Spatial Data Types: The ROSE Algebra. Technical Report 141, Fern Universität, Hagen, Germany, 1993.

[GS93b] R.H. Güting and M. Schneider. Realms: A Foundation for Spatial Data Types in Database Systems. In D. Abel and B.C. Ooi, editors, *Advances in Spatial Databases (SSD'93)*. Lecture Notes in Computer Science No. 692, Springer Verlag, Berlin, 1993.

[GS95] R.H. Güting and M. Schneider. Implementation of the ROSE Algebra: Efficient Algorithms for Realm-Based Spatial Data Types. In M. Egenhofer and J. Herrings, editors, *Advances in Spatial Databases (SSD'95)*. Lecture Notes in Computer Science No. 951, Springer-Verlag, Berlin, 1995.

[Güt88a] R. H. Güting. Modeling Non-Standard Database Systems by Many-Sorted Algebras. Technical Report 255, Universität Dortmund, Germany, 1988.

[Güt88b] R.H. Güting. Geo-Relational Algebra : A Model and Query Language for Geometric Database Systems. In *Conf. on Extending Database Technology (EDBT '88)*, pages 506–527, 1988.

[Güt89] R.H. Güting. Gral: An Extensible Relational Database System for Geometric Applications. In *Proc. Conf. on Very Large Data Bases (VLDB)*, 1989.

[Güt93] R.H. Güting. Second-Order Signature: A Tool for Specifying Data Models, Query Processing and Optimization. In *Proc. ACM-SIGMOD Conf.*, 1993.

[Güt94a] R. H. Güting. An Introduction to Spatial Database Systems. *The VLDB Journal*, 3(4):357–399, 1994.

[Güt94b] R. H. Güting. GraphDB: A Data Model and Query Language for Graphs in Database. In *Proc. Conf. on Very Large Data Bases (VLDB)*, 1994.

[HT96] T. Hadzilacos and N. Tryfona. Logical Data Modeling for Geographic Databases. *Int. Journal on Geographical Information Systems (IJGIS)*, 1996.

[JKO+89] S.-J. Jiang, H. Kitagawa, N. Ohbo, I. Susuki, and Y. Fujiwara. Abstract Data Types in Graphics Databases. In T. L. Kunii, editor, *Proc. IFIP Conf. on Visual Database Systems*, B.V. (North-Holland), 1989. Elsevier Science Publishers.

[JS82] B. Jaeschke and H.-J. Schek. Remarks on the Algebra of Non First Normal Form Relations. In *Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, 1982.

[KW87] A. Kemper and M. Wallrath. An analysis of Geometric Modeling in Database Systems. *ACM Computing Surveys*, 1987(1):48–91, 1987.

[Len93] H.-J. Lenz. M3-Database Design: Micro-, Macro- and Metadata Modeling. In F. Faulbaum, editor, *SoftStat'93; Avances in Statistical Software*, Stuttgart, 1993.

[LPV93] T. Larue, D. Pastre, and Y. Viémont. Strong Integration of Spatial Domains and Operators in a Relational Database Systems. In D. Abel and B.C. Ooi, editors, *Advances in Spatial Databases (SSD'93)*, pages 53–72. Lecture Notes in Computer Science No. 692, Springer-Verlag, Berlin, 1993.

[LT92] R. Laurini and T. Thompson. *Fundamentals of Spatial Information Systems.* The A.P.I.C. Series, Number 37. Academic Press, 1992.

[LZ74] B. Liskov and S. Zilles. Programming with Abstract Data Types. ACM SIGPLAN Notices, 1974.

[Mar88] S. T. March, editor. *Entity-Relationship Approach.* Proc. of the Entity-Relationship conference, North-Holland, N-H, 1988.

[Mic91] Z. Michalewicz. *Statistical and Scientific Databases.* Ellis Horwood Publishers, London, 1991.

[O2T93] O2Technology. The $O_2$ User Manual, version 4.3. Technical report, $O_2$Technology, Versailles, 1993.

[OH86] S. Osborne and T. Heaven. The Design of a Relational Database System with Abstract Data Types for Domains. *ACM. Trans. on Software Engineering*, 11(3):357–373, 1986.

[Peu84] D.J. Peuquet. A Conceptual Framework and Comparison of Spatial Data Models. *Cartographica*, 21(4):66–113, 1984.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer Verlag, Berlin, 1985.

[RS95] P. Rigaux and M. Scholl. Multi-Scale Partitions: Application to Spatial and Statistical Databases. In M. Egenhofer and J. Herrings, editors, *Advances in Spatial Databases (SSD'95)*, pages 170–184. Lecture Notes in Computer Science No. 951, Springer-Verlag, Berlin, 1995.

[SR86] M. Stonebraker and L. A. Rowe. The Design of POSTGRES. In *Proc. ACM SIGACT-SIGMOD*, pages 340–355, 1986.

[SRG83] M. Stonebraker, B. Rubenstein, and A. Guttman. Application and Abstract Data Types and Abstract Indices to CAD Databases. In *Proc. of the Annual Meeting Database Week*, pages 107–113, 1983.

[Sto86] M. Stonebraker. Inclusion of New Types in Relational Data Base Systems. In *Proc. Intl. Conf. on Data Engineering*, pages 262–269, 1986.

[SV89] M. Scholl and A. Voisard. Thematic Map Modeling. In *Design and Implementation of Large Spatial Databases (SSD'89)*, pages 167–192. Lecture Notes in Computer Science No. 409, Springer-Verlag, Berlin, 1989.

[SV92] M. Scholl and A. Voisard. Object-Oriented Database Systems for Geographic Applications: An Experiment with $O_2$, 1992. in [53], pp. 585-618.

[SVP+96] M. Scholl, A. Voisard, J.-P. Peloux, L. Raynal, and P. Rigaux. *Systèmes de Gestion de Bases de Données Géographiques. Spécificités.* International Thomson Publishing, Paris, France, 1996. In French.

[Til80] R. B. Tilove. Set Membership Classification: A Unified Approach to Geometric Intersection Problems. *IEEE Transactions on Computers*, C-29(10), 1980.

[Ull88] J.D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 1. Computer Science Press, 1988.

[Ver86] J. Verso. VERSO: A Database Machine Based on non-1NF Relations. Technical Report 523, INRIA, 1986.

[Wor94] M. Worboys. Object-Oriented Approaches to Geo-Referenced Information. *Int. Journal on Geographical Information Systems (IJGIS)*, 8(4), 1994.

# Appendix: Figures

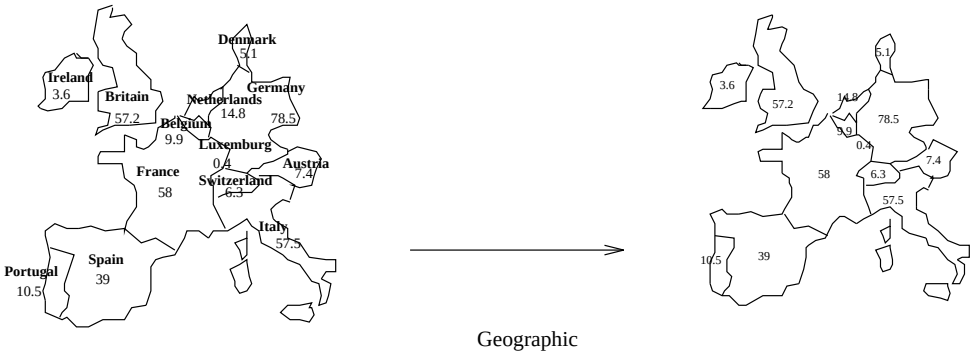



Figure 2: Map projection

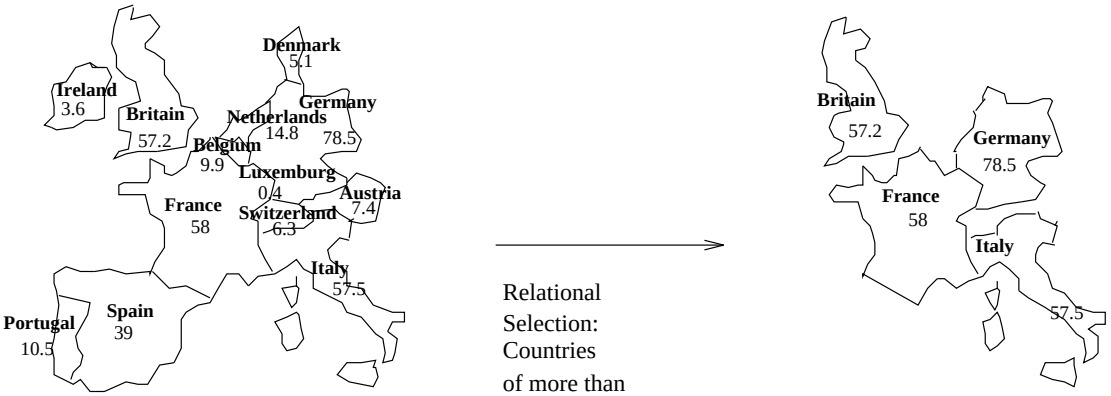
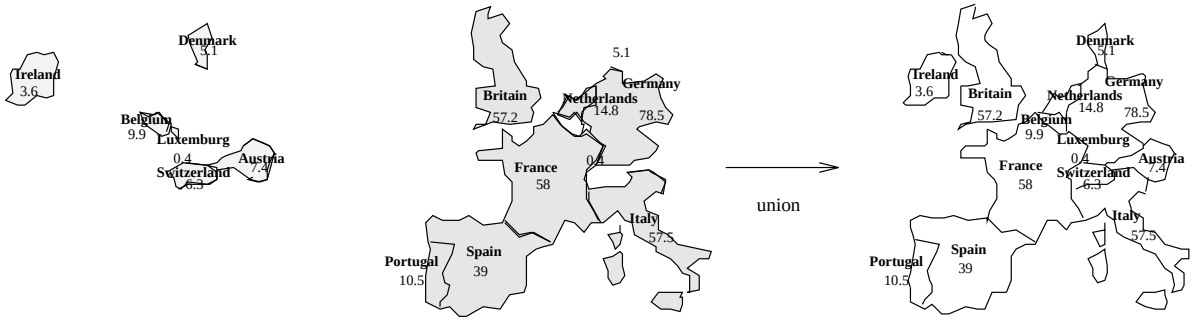



Figure 3: Map selection

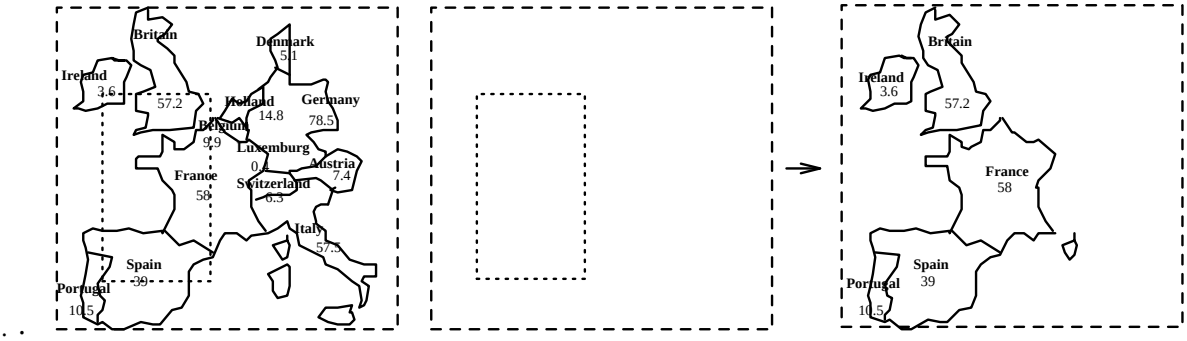



Map of countries
and population
of Western Europe
with less than 10 Millions inh.

A map of countries
and population of Western
Europe with more than
(or exactly) 10 Millions inh.

A map of countries
and population
of Western Europe

Figure 4: Map union

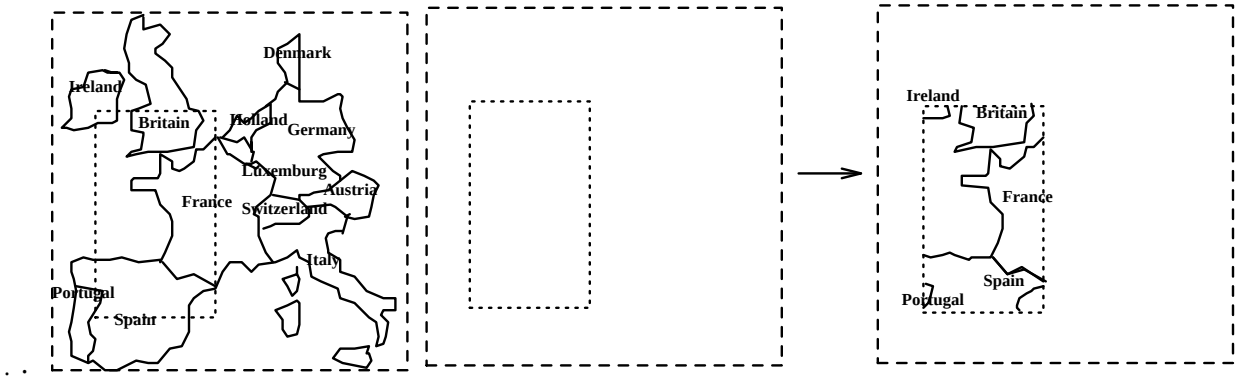



Map m

Area a

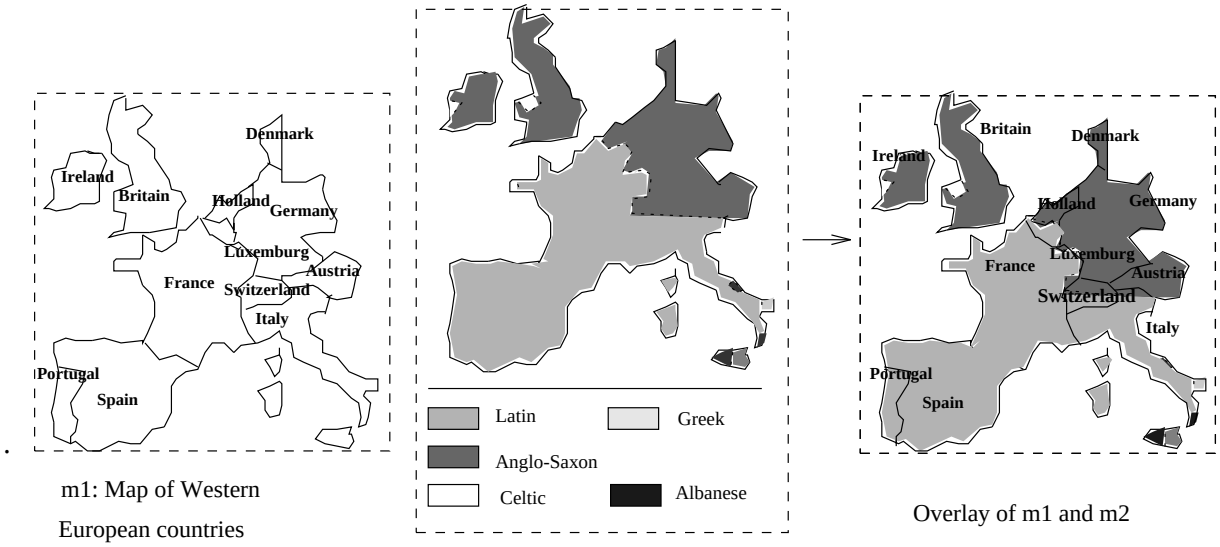
Windowing of map m with area a

Figure 5: Map windowing




Map m Area a Clipping of map m with area a

Figure 6: Map clipping

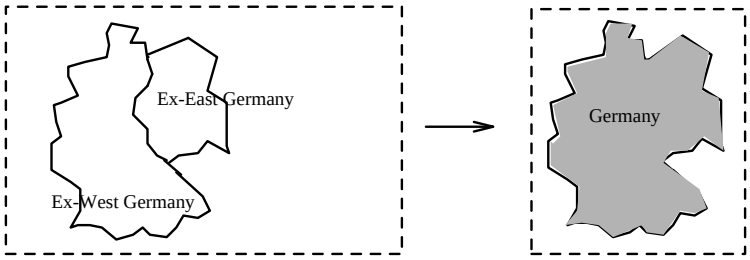



m1: Map of Western European countries

m2: Families of languages spoken all over Western Europe

Overlay of m1 and m2

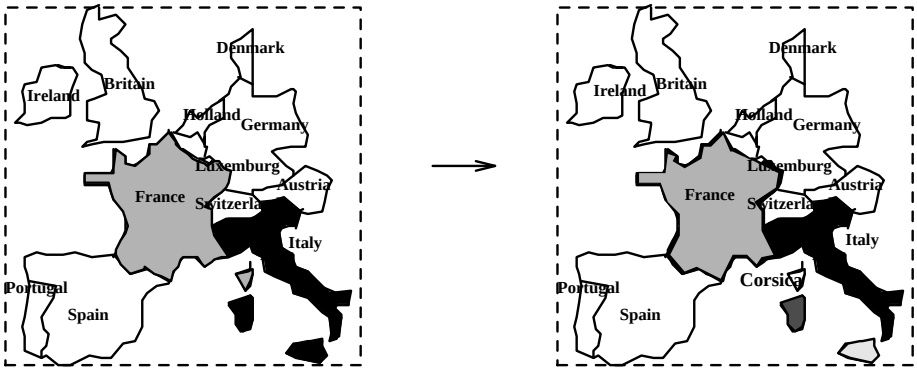
Figure 7: Map overlay




A map with two geographic objects

A map with one geographic object

Figure 8: Merging 2 geographic objects on a map




Map m

Decomposition of map m

Figure 9: Map decomposition