# A Database Perspective on Geospatial Data Modeling

Agnès Voisard and Benoit David

**Abstract**—We study the representation and manipulation of geospatial information in a database management system (DBMS). The geospatial data 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 among these objects. In addition, it allows easy manipulation of nonbasic 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.

**Index Terms**—Spatial abstract data type, aggregate/disaggregate function, complex object, extended-relational model, object-oriented model.

## 1 Introduction

Many new applications that deal with geospatial data, such as resource management, urban planning, meteorology, are characterized among other things by large amounts of data, both alphanumeric and spatial. They increasingly rely on *database management systems* (DBMS). Such DBMS have specific needs for querying, manipulating, and representing these data and for integrating geometric processing within their environment. 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 geospatial data modeling (for a comprehensive survey on features that a geospatial DBMS should provide, such as graphical user interface, spatial access method, or query optimization, see [1], [2]). The basic entities considered 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. Hence, 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 *geographic model* or *map model* as 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.

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 challenging issue is to embed both these operations on maps and on ADTs in the same geographic model. In addition, spatial ADTs must be closed under the operations one can apply to them. For previous work on spatial ADT, see [3], [4], [5], [6].

Another problem is derived from the fact that some general functions on maps correspond to sophisticated operations for which conventional database models and query languages 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 this 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 as the function to be applied on all elements is a user-defined function. Even when aggregation function 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).

• *A. Voisard is with the Computer Science Institute, Freie Universität Berlin, Takustr. 9, D-14195 Berlin, Germany.
E-mail: voisard@inf.fu-berlin.de.*
• *B. David is with the French Ministry of Transportation, SETRA, 46 avenue Aristide Briand, B.P. 100, F-92225 Bagneux Cedex, France.
E-mail: Benoit.David@equipement.gouv.fr.*

*Manuscript received 21 June 1997; revised 13 Jan. 1998; accepted 19 Feb. 1999; posted to Digital Library 25 July 2001.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 104728.*

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 relatively late that object composition has received attention (see [7], [8], [9]). The example above also illustrates the similarity with statistical databases [10], where a major problem is to define proper mechanisms for data aggregation across many dimensions. According to [11], 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 nonempty intersection. In [12], the authors make the bridge between the two disciplines by defining an aggregation technique over a hierarchy of (multiscale) partitions.

Some attempts have been made to define geographic models using standard database models. Some of them, such as [13], [14], [9], focus more on pure data modeling than on the detail of map operations. Others which deal with both data and operations often suffer from limits due to the chosen underlying database model or to restrictions imposed on their own model. The Geo-Relational Algebra [3] and Spatiarel [15] 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 [4], 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 [16]). The Rose Algebra [17], [18] 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 and rather ad hoc as it requires the introduction of new operators within SQL [19].

The considerations above show that handling geospatial data goes beyond traditional data handling. To summarize, the peculiarity of geospatial applications is due to:

- the existence of both nonspatial and spatial data, which implies the definition of spatial data types closed under the operations applicable to them,
- highly structured data with the notion of objects composition,
- the existence of user-defined operations (even though there may exist a kernel of universal operations), which requires an extensible underlying model, and
- combinations of functions that exist at both a low-level of abstraction, i.e., on the geometric type, and at a high-level of abstraction, i.e., on the maps, which leads in theory to second-order models. This is a challenging issue which is studied throughout the paper.

Our goal is not to define another geospatial model, but to study the implementation of a general geospatial model using two major commercial DBMS models as a support. We chose to base our reference model on the complex object model of [20]. This model is well-adapted to the representation of (complex) geographic objects and to their manipulation through the existence of user-defined functions and a high-level operator, called *replace*, which applies a function to all elements of a set.

This paper is organized as follows: In Section 2, we give as a basis a simple geographic data model, together with examples of operations that one is likely to apply to maps. We also present several representative queries on maps, which serves as references throughout the paper. In Section 3, we study the use of database concepts for implementing a geospatial data model. This study leads to the proposal of three classes of operators which represent a key point to DBMS extensibility. Finally, we express our geographic data 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 Geographic Data Model

The goal of this section is to present a geographic model *independent of any database representation*. Many existing conceptual models, such as the entity-relationship model [21], its recent extensions, UML, or IFO [22], could serve as a basis to express a geographic data model, as it was done in previous work [9]. Here, we take as an underlying model the complex object model of [20]. 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 [23] and all the true object-oriented models). Nevertheless, we believe that this concept is not essential for what we want to show, namely, geospatial data modeling (i.e., data structuring and querying, and no updates).

In GIS, it is common to make the distinction between space-based (or field-based) and feature-based (or entity-based) models [24], [25]. In the first case, the space is seen as a continuous domain with different values over different places. Hence, the entities of interest are regions, lines or points within the space, and 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 [17], [14]. These approaches are based on a mapping of geographic models onto geometric layers. The model proposed in this paper belongs to the latter category, hence our focus on the notion of geospatial entity.

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 Underlying Complex Object Model

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 (sometimes called structured object model in the literature) such as [20] is especially 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 (first the tuple constructor and then the set constructor to create a relation), or even to nonfirst normal form models, such as Verso [26], [27], and NF2 [28], 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 [4] can be seen as an "extended relational model"). The fact that repeated use of these constructors is allowed can be explored to express the composition of geographic objects with arbitrary depth.

In [20], 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 $T_1, \ldots, T_n$ are types and $A_1, \ldots, A_n$ are attributes not used in any of them, then $[A_1 : T_1, \ldots, A_n : T_n]$ is a *tuple* type; if $T$ is a type and $A$ an attribute not used in it, 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 $A$. Hence, tuple and set types are created by applying the corresponding constructors to named types. Objects are defined as instances (values) of a type. Both calculus and 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, and 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) and 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 enables us to consider user-defined functions and predicates (as "interpreted" functions and predicates), which are of great interest in map manipulation.

This approach also offers a high-level operation called *replace*. It is similar to the *map* 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 [28], [26], 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 Map Model

In the following, we define a *thematic map* as a *collection of homogeneous geographic objects*. A thematic map, sometimes called a layer, is in the literature [13], [14]. A geographic object is usually derived from an entity of the real world. Examples include rivers, cities, countries, or highways. Such an object has a spatial part and an alphanumeric one called its *description*. For instance, a river is characterized by its name, its flow (both part of 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 common example is the 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. The genericity required for a general map schema implies a higher level of abstraction. We do not describe, for instance, the structure of a state or of a river object here. Rather, we describe the structure of a map in general. This will be useful in the sequel while defining operations on maps. Note that because of the lack of subtyping in the underlying model, we simply give the abstract syntax of a map description. We use a simplified[1] metasyntax *à la* Backus-Naur Form with symbol ":" for type assignment and with constructors { } (set), [ ] (tuple). `Ti` denotes a basic type (e.g., integer, real, Boolean). In the following, we refer to the `Ai`'s as alphanumeric attributes. `<Ai:Ti>*` denotes an enumeration of named types, i.e., the description. `S` denotes the name of a spatial attribute whose type is `Spatial` (see Section 2.3). Comments are prefixed by \\.

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

1. We tried to give the most possible compact version in order to avoid confusion. Nevertheless, a more expanded version that extracts the kinds of components (nonspatial and spatial) would have been more correct here.

**Remark on map nesting vs. map atomicity.** The definition of the geographic objects above can be seen as a tree whose leaves are atomic geographic objects and nodes complex geographic objects. This distinction is important as it permits us 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 [7] from the geometry of objects that compose them (possible recursion). Other attributes such as population or even position may be propagated within such nested maps using the same mechanism. In the country-state-county partitioning, 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 map atomicity is useful when describing the explicit manipulation of the geometry of an entire map. This leads to the following map-oriented definitions:

```
<atomic-geographic-object>
      ::=  "[" [<Ai ":" Ti>*], S ":" Spatial "]"
<thematic-map>
     ::=  "{" <atomic-geographic-object> "}"
            \\ atomic thematic map | "{" Mi ":" "["
          [<Ai ":" Ti>*], Mi' ":" <thematic-map> "]"
            "}"   \\ complex
```

To illustrate the definition above, 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 one-dimensional objects only.

```
States:{State: [StateName: string,
 Counties: {County: [CountyName: string,
    Population: integer, MSL:
 string, Geometry: Spatial]} ]}
```

`States` is a thematic map, i.e., a set of geographic objects (`State`) of same type, whose one component is again a set of geographic objects, or thematic map (here `Counties`), composed in turn of geographic objects (`County`) having a name, a population, a main spoken language (attribute `MSL`), and a geometry. One can define an object "USA" or "France" having type `States`. Because of composition, the geometry has been factorized, i.e., it only appears at the `County` level (atomic thematic 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 `Road-Element` object).

## 2.3 Spatial Data Types

In spatial databases, spatial data types are usually defined as ADTs, i.e., encapsulated types which are associated 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 [29], [30], [31], [32], [33]. A spatial object is an instance of a spatial type. It can have 0 (point), 1 (line), or 2 (zone) dimensions. A set of spatial objects is usually referred to as a layer. In [34], 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 Type `Spatial` above follows the same definition. Of course, this is just a possible spatial model. Other spatial models such as raster or topological models, see [35], [36], [37], are also widely used. Puppo and Dettori [38] define a mathematical model of maps based on abstract cell complexes, which fulfills topological and metric consistency rules. Erwig and Schneider [39] propose a spatial model for handling vague regions (based in particular on the fuzzyness of boundaries). A recent proposal by the Open GIS Consortium [40] describes a complete geospatial model which is likely to become a standard.

### *2.3.1 Major Issues in Spatial Types Definition*

Defining spatial types means defining both their structures and the operations applicable to them. 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 it.* One may then be tempted to define a general type such that it remains closed under at least a set of common operations. However, a too general type implies 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 [41]). In [34], 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 zero-, one-, and two-dimensional figures, which can easily be implemented by ADT (such as in [5] in the closely-related domain of graphics databases) or object-oriented classes.

3. *Independence of the type with regard to 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 [4], 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 (see, for instance [42], [43], [44], [40], [45]), there is no standard classification of them. The reader will find in [43] a complete list of geometric operations based on the types of the arguments. Many criteria for a classification can be of interest. They 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. Here, we do not describe 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 later describing 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's object.
- `SO -> n ; (SO,SO) -> n`, (where `n` denotes a real). These operations use a metric. Examples include the *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 nonconnected 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. Some examples include the computation of the *center* of a set of points, the *Voronoi* diagram, or the *minimal distance* between spatial objects.

For more information about spatial operations, the reader is referred to [46], [47], [3], and [48] and for a description of geometric algorithms to [49].

## 2.4 Examples of End User Operations

We now give several examples of simple operations that an end user is likely to apply to maps. Some of them can be found in [4], [18]. For each operation, we give its signature, in which `map` corresponds to Type `thematic-map` described in Section 2.2 and `Ai` corresponds to 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. Note that this set of operations does not claim 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

1. **Map Projection**. The signature of this operation is `map` $\times$ `Ai` $\rightarrow$ `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). If S denotes a spatial type and m denotes an instance of the map type, this operation is expressed as $\pi_{Ai,S}(m)$ in the relational algebra. 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 (Fig. 1,

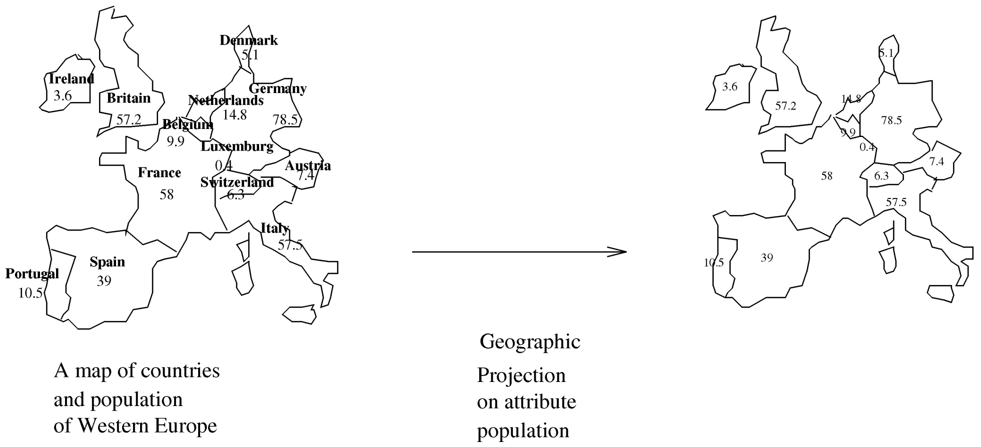

Fig. 1. Map projection.

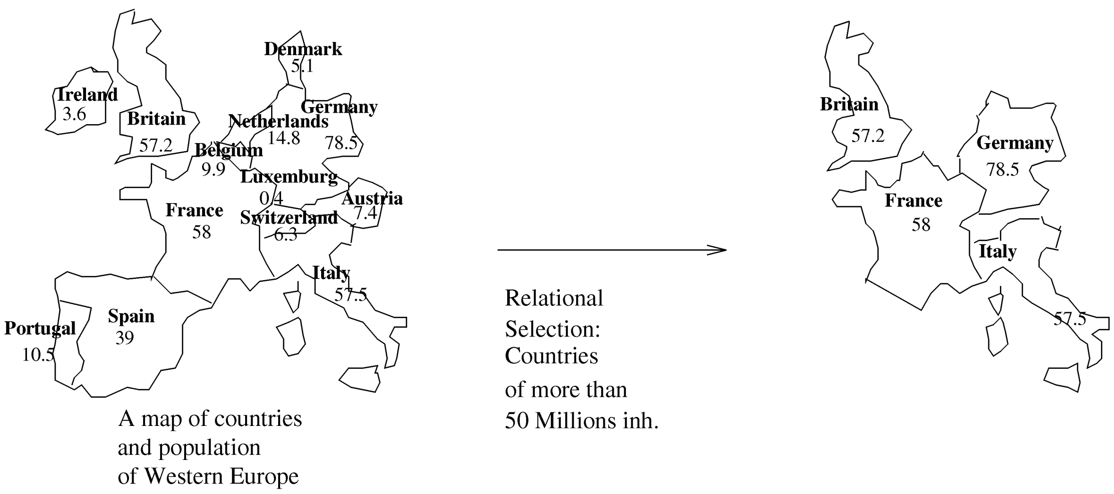

Fig. 2. Map selection.

left). When a *map projection* is applied to the population, the country names are eliminated (right hand side of Fig. 1). 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.

2. **Map Selection**. The signature of this operation is `map` $\times$ `condition` $\rightarrow$ `map`, where `condition` is a predicate on one or many alphanumeric attributes (`condition` = `p(Ai)`). It is similar to the relational selection and can be expressed as $\sigma_{p(\mathrm{Ai})}(m)$. On the previous map by applying a *map selection* to the countries whose population is greater than 50 million inhabitants, we obtain the map on the right hand side of Fig. 2.
3. **Map Union**. This operation (signature `map` $\times$ `map` $\rightarrow$ `map`) consists of performing the union of sets of geographic objects having the same schema. This operation is similar to the relational union, hence if $m_1$ and $m_2$ are two map instances, the result is denoted $m_1 \cup m_2$. In the example of Fig. 3, 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 of 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 first to be defined.
4. **Spatial Selections**. As opposed to the previous selection, the following ones refer to a *spatial* property and they are useful for region and point queries. Their signature is `map` $\times$ `SO` $\rightarrow$ `map`. `SO` is, for instance, an area drawn by the user on the screen (e.g., a rectangle) or a point.

- **Windowing**. Windowing returns the geographic objects of a map whose intersection with a given area is not empty (Fig. 4). Let $w$ be such an area, $e$ an element of a map (tuple in relational-like models), and $\bullet$ the intersection of spatial objects.[2] Then, windowing is expressed as

2. Hence, $w$, $e.S$, and $(e.S \bullet w)$ have the same (spatial) type.

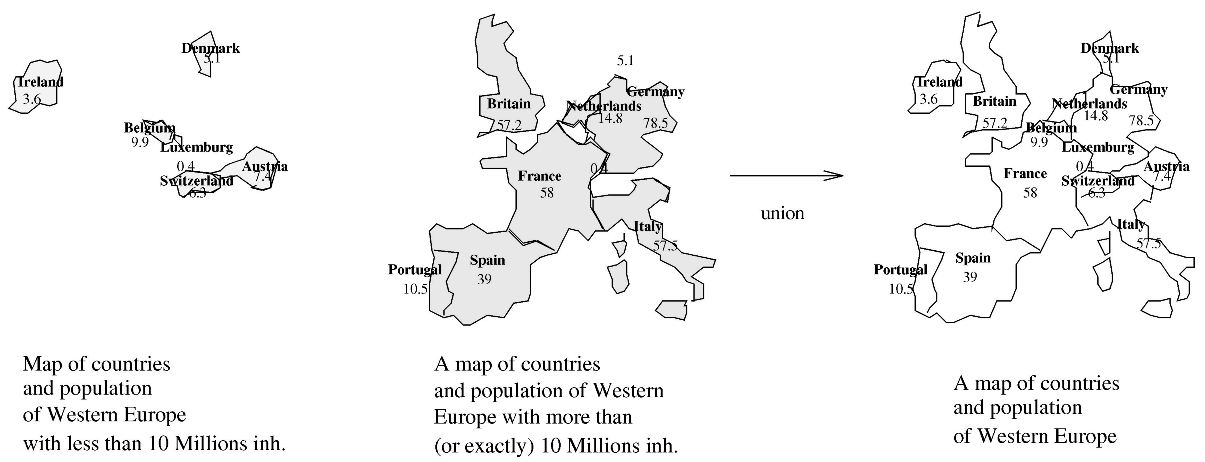
Ireland 3.6
Denmark 5.1
Belgium 9.9
Luxemburg 0.4
Austria 7.4
Switzerland 6.3
Britain 57.2
Netherlands 14.8
Germany 78.5
France 58
Italy 57.5
Spain 39
Portugal 10.5
union
Map of countries and population of Western Europe with less than 10 Millions inh.
A map of countries and population of Western Europe with more than (or exactly) 10 Millions inh.
A map of countries and population of Western Europe

Fig. 3. Map union.

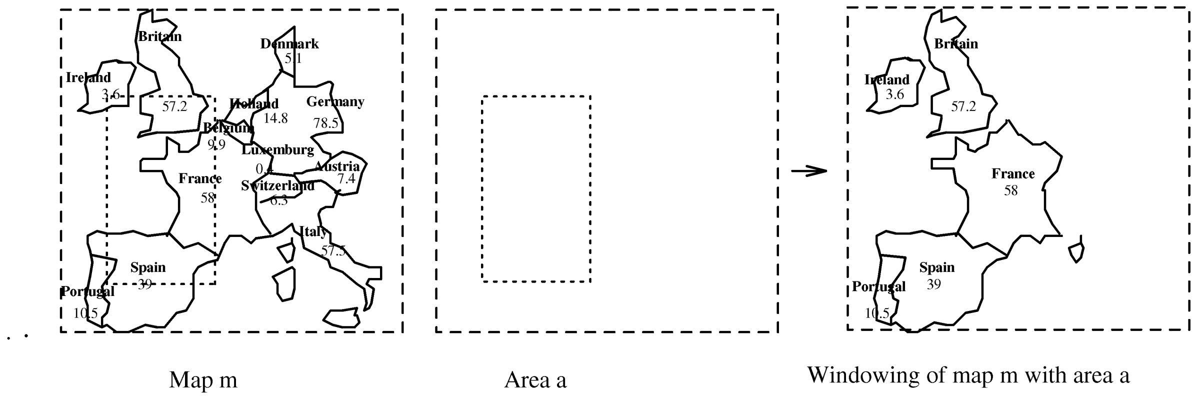

Fig. 4. Map windowing.

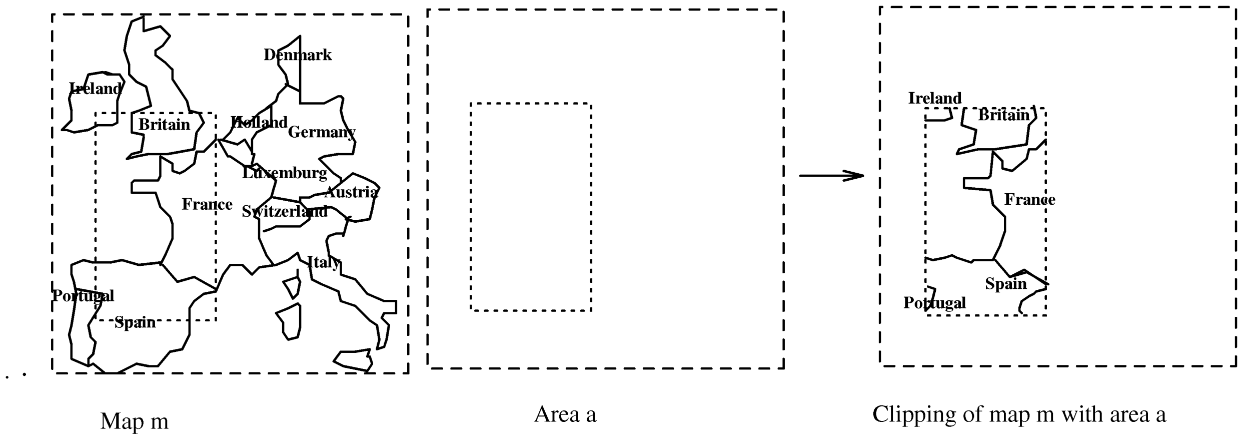

Fig. 5. Map clipping.

$\sigma_{e.S \bullet w \neq \emptyset}(m)$. In Fig. 4, 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 [4]: "Get all countries located less than 100 kilometers from a given point." The windowing of the initial map with a 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 of 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 (Fig. 5). 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.

5. **Map Overlay**. This operation (signature `map` $\times$ `map` $\rightarrow$ `map`) is very common in GIS applications. It

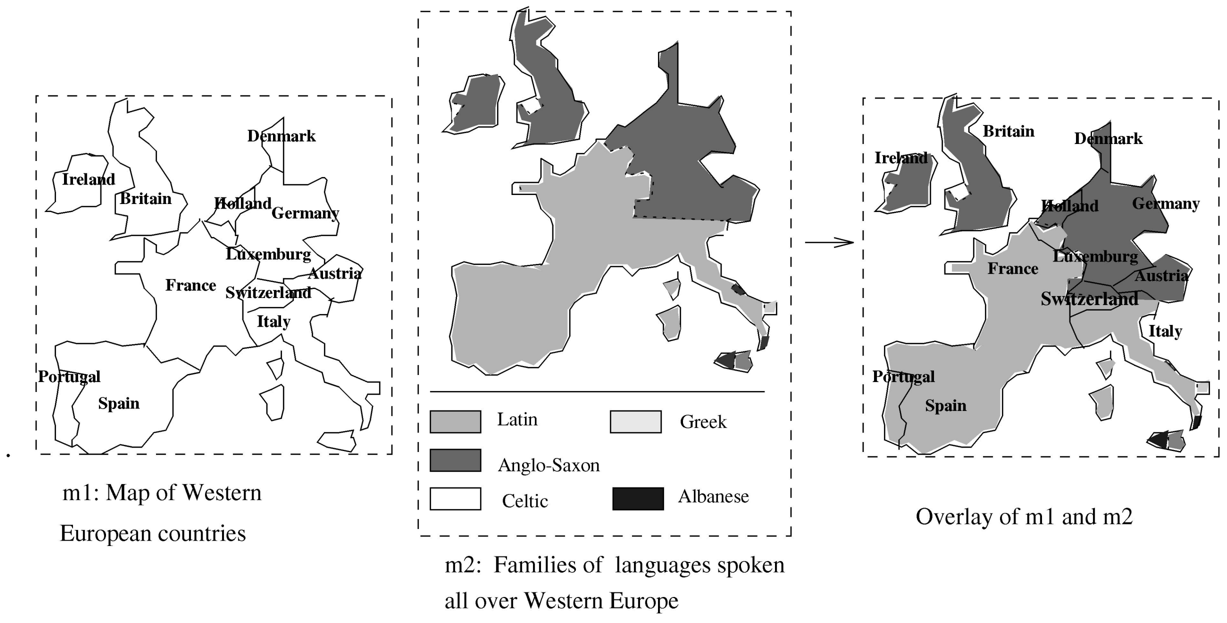

Fig. 6. Map overlay.

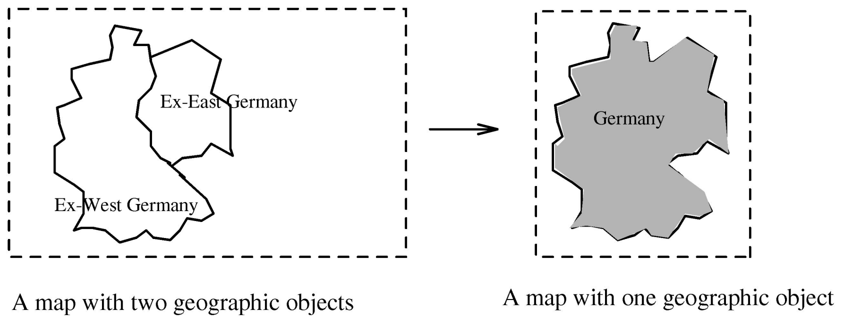
Ex-East Germany
Ex-West Germany
Germany
A map with two geographic objects
A map with one geographic object

Fig. 7. Merging two geographic objects on a map.

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. If $\bowtie_G$ is a spatial join (i.e., a relational join whose predicate is defined over the spatial domain), $m_1$ and $m_2$ are maps, then the overlay is expressed as $m_1 \bowtie_G m_2$. In Fig. 6, 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 distribution 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 languages spoken in the area. The maps considered do not necessarily have 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 when a map is created from the geometric argument (a simple map with only one geographic object having no description).

6. **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 end user (Fig. 7). As it operates within a single map, its signature is `map` $\times$ `condition` $\rightarrow$ `map` (observe the difference with the map union, which takes it as arguments between several maps and gathers them into a single one.) A merger relies on the concept of object aggregation (see further) and on the `union` operation on sets of spatial objects. According to the condition, it partitions the input set, similarly to a SQL "group-by." This expression cannot be denoted with relational symbols as second order cannot be expressed explicitly. Below is an example of the merger operation using our reference schemas (Query 1).

**Note regarding the following queries.**

1. From now on, we omit the type of attributes when it is not essential for the 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

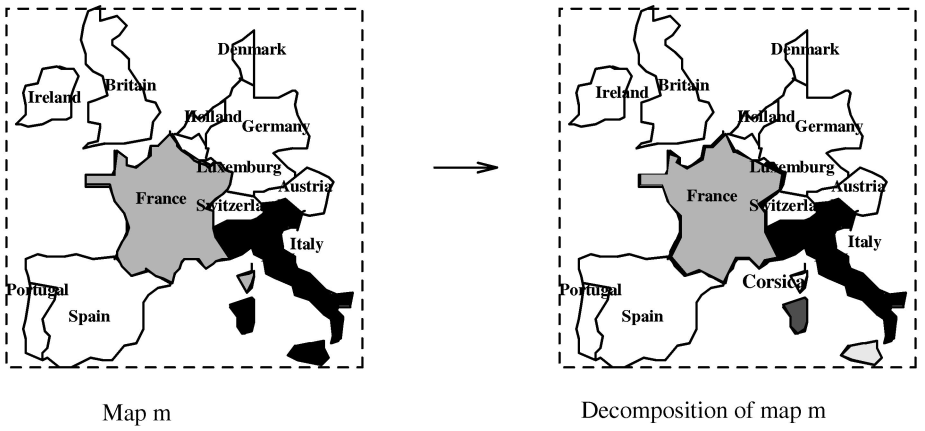

Fig. 8. Map decomposition.

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 languages are identical 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).

a. Initial Schema

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

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

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

c. Project on `MSL` and `Geometry` and `nest` on `Geometry`: `States:"`

```
{County': [MSL, G: {Geometry}]}
```

d. Apply `union-set` to set G (G′ ← `union-set(G)`): `States"`: {`County"`:[`MSL, G′`]}.

In [4], [6], [18], a somewhat 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. Using such an operation, a combination of alphanumeric attributes (such as a sum of populations in Fig. 7) can be considered. New values are assigned to (alphanumeric) attributes as described in our queries.

7. **Map decomposition.** It creates independent geographic objects from their nonconnected geometric components. Its signature is `map`→ `map`. As in the case of merger, it cannot be expressed by a relational expression. Fig. 8 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 made of two nonconnected polygons, the larger 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 1) the nonconnected parts of a geographic object have to be isolated and 2) 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 that 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 described below. In the following expression,

$e.S$ is not mandatory here as it behaves as any other attribute. Nevertheless, given its importance in geographic applications, mentioning it better illustrates our discourse.:

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

In other words, the new attribute value is a function ($g$) that is a combination of: 1) the value of an alphanumeric attribute $A_i$, 2) the value of the spatial attribute $S$, and 3) possibly a function to be applied to the attributes of all the elements of a map, such as average (here $f(m.A_j)$). Such functions are not easily expressible in database languages because of their second order nature, as we will see in next section.

### *2.4.2 Operations Based on Pure 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, 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 (geocoding function). 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) and map it onto the network. Then, the query execution consists of calling the `shortest path` algorithm on the network (external function), with the two end-points.

It is not our goal to elaborate on graphs here. The reader will find in [50] an approach for embedding graphs in databases, together with relevant algorithms. What we want to show through this example is the existence of two independent features: 1) the possibility of defining and calling elaborate functions on the geometry (such as `shortest path`) and 2) the possible heterogeneity of the result of such queries, which can be, for example, a map (a subnetwork 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 from the description of each segment. Therefore, one must be able to specify such an input, which is feasible in our model by applying the composition of functions. As another example, take the allocate function which is a common function on networks in GIS. 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 Expressing Aggregation and Disaggregation

Although *aggregation* and *disaggregation* are meant as database features, they deserve a particular attention 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 $\mathtt{f}$ such as $\mathtt{sum}, \mathtt{union}, \ldots$ is to be applied to attribute $B$. The signature of $\mathtt{f}$ is $\{\beta\} \rightarrow \beta'$. To express that correctly, a first step consists of gathering the values of $B$ in a set $C$ as follows: $\{[A : \alpha, C : \{B : \beta\}]\}$ and then to apply function $\mathtt{f}$ to the created set ($D \leftarrow \mathtt{f}(C)$), 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 $\mathtt{f}$ is to be applied is given (e.g., "$\mathtt{Select}$ Name, $\mathtt{sum}(B)$ $\mathtt{from} \ldots \mathtt{where} \ldots$"). In the case of *disaggregation*, a function $\mathtt{g} : \{\beta\} \rightarrow \beta'$ 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 restructuring 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 nonexplicit 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: 1) compute the total of population of all its counties (function sum) and 2) perform the geometric union over the regions of all its counties (function union on the spatial attribute). The schema evolution is the following:

a. Initial schema (we give up the county names and languages)

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

b. Restructuring (for each state, projection on both the population and the geometry of its counties)

```
States':{State: [StateName,
         P:{Population},G:
        {Geometry}]}
```

c. Treatment: Apply sum to populations ($\mathtt{P}' \leftarrow \mathtt{sum(P)}$) and union to regions ($\mathtt{G}' \leftarrow \mathtt{union(G)}$), 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:

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

The query execution is then the following: 1) apply decompose to the geometry of all counties ("replace," resulting in a set of regions G′ for the geometry and 2) create a new geographic object for each element of G′ ("disaggregation") with a population value computed from the ratio of the respective area.

a. Initial Schema (we focus on the population and geometry of counties)

```
Counties:{County:[Population,
         Geometry]}
```

b. Add attribute G′ by applying decompose to Geometry

```
(G' ← decompose(Geometry)) Counties':
   {County:[Population,
    Geometry,G':{Spatial}]}
```

c. 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' ← (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 size (a real):

```
Counties":{County:[Population,
          Geometry, G':{[P':
          real, Geom:Spatial]}]}
```

d. Project out Population and Geometry and collapse the inner set and tuple: Counties''':{County:[P', Geom]}

## 2.6 Relationships between Maps and Geometric Components

The informal presentation of maps and operations above points out the necessity of having 1) operations on the description (the nonspatial part), as well as 2) 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, our approach also shows intuitively two levels of abstraction, namely, a higher level of abstraction, with collections of geographic objects, and a lower level of abstraction, namely, encapsulated spatial types. The higher level processing relies heavily on operations defined on the lower level. For instance, the merger operation presented above needs the geometric union defined on spatial types. Map overlay needs the intersection operation defined on the geometry of spatial types.

The same underlying model was chosen in [4]. This approach adapts the replace operators (renamed *apply*) to the map context to give the possibility of applying a geometric operation to all geographic objects of a set. It shows that operations on maps can be expressed using this operator and basic geometric operations. For instance, the map overlay operation can be described using apply, the geometric intersection (in the role of the function to be applied), and the Cartesian product. The join of $m_1$ and $m_2$ is expressed as

$$m_1 \bowtie_G m_2 = apply <\neq \emptyset < \bullet >> m_1\ X\ m_2,$$

where $X$ is the Cartesian product of the two maps. Similarly, the fusion ($\uplus$) is expressed as $apply < \oplus > (m)$, where $\oplus$ denotes the geometric union of all the elements of a set. This mechanism turns out to be very useful both at modeling time and at implementation time.

However, having only the notion of map, spatial types, and replace is not enough as operations such as some specific operations on the maps geometry, as illustrated in Sections 2.4.2 and 2.4.3, have also to be considered. Hence, we see a need to combine general operators and user-defined functions. As far as previous approaches are concerned, note that the operations of Section 2.4 were not all treated in [4] as it was not the focus of this approach. In [18], 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 Geographic Features in a DBMS

We now address the problem of defining a logical model that corresponds to the geographic data 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 to handle geographic information:

1. A possibility to consider 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 nonflat database models, the set constructor can be used for describing relationships between objects (as done in Section 2).
3. The possibility of refering to the spatial part of an object, e.g., by a dot notation (for example, `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 1) the implementation of the operators, 2) the representation of geographic data, and 3) 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: 1) ADT or sets of ADTs, 2) sets of objects independent of each other (i.e., object after object), and 3) collections of objects as a whole, i.e., combinations of geographic objects, functions of spatial objects, functions of descriptions, or subpart thereof, etc. In [43], 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 [43] and is oriented toward 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 (nonstandard) 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:[3]
   - *ADT* $\times$ *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.** When executing these operations (e.g., projection and selection) objects (tuples) are considered sequentially and independently 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 [23], which provides the user with the possibility of defining the abstraction of functions and predicates. This enables us 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 "nonlinear" processing.** As illustrated throughout our examples, the input or output of these operators (e.g., `shortest path`) is more complex than a simple set of ADT. 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 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 operator, programming languages need to be considered. From a data model point of view, such an operator is only defined by its signature. In

3. Only the most common classes are represented here.

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 1) the start and end points, and 2) the network itself, that is, a set E of edges together with their weight and whose schema is `(edge:spatial,weight:real)`. It might return a value such as a set E′ of type `edge:spatial`. The algorithm itself is not relevant here and can be seen as a "black box."

## 3.2 Embedding the Operators in a DBMS

The previous considerations lead us to examine the problem of extensibility in geographic DBMS note that we are focusing on single systems rather than on distributed systems, even though some operators and geometric functions 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 too specific system. In the same way, it is now well-known [51], [18] 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., combining 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 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 nonconnected 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 [31], [52], [33], [53]. This facility has often been used to add a spatial data type within a relational DBMS. ADT enables user-defined operations from Class I, but only from the first subclass "ADT $\times$ 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 paradigm. 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 in query languages such as standard SQL either.

Most of the Class II algebraic operators are part of the relational algebra. In an extended-relational system such as Sabrina-Objet [53], it is possible to use any function or predicate in the operators. Some operators such as *nest* or *unnest* do not fit in the relational context, and we will see later 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 [32] 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 [54] is built on top of an extended relational system [53]. 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 [55] 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,
                        MSL: 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 is the standard end user languages for the moment. We now express the queries of Section 2 in extended-SQL.

**Query 1. Build a "language" map using the main spoken language of each county (*merger*).**

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

Note that this approach does not allow one 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 Java, in which such algorithms are coded (they can also be coded in a more appropriate language such as a functional language called from C, for instance).

**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 1) disaggregate function such as `decompose` (Class I.c) cannot be expressed and 2) the unnest operator is not defined.

## 3.4 Object-Oriented DBMS

In the past 15 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 $O_2$ [51] or ObjectStore [56]. Embedding typical GIS features within a so-called object-oriented GIS was studied as well [57]. For more information regarding the use of object-oriented system to handle geographic data, see [57], [58], [8].

### 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 $O_2$ [59]). 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 [51], [60], [18]. Defining a spatial ADT is done as follows:

```
class spatial  \\ class definition
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. 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 [61] 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, MSL: 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. Moreover, values cannot be shared in the database. It is however possible to define corresponding classes. This would make no difference in the query language and would enable method definition and data sharing. Nevertheless, 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 [61] in order to express the queries of Section 2 (all examples have been tested). $O_2$SQL is a database query language whose syntax is close to SQL standard. It was a basis for defining OQL, the object-oriented SQL-like standard query language proposed in 1994 by the *Object Data Management Group (ODMG)* [62], the OODBMS vendors consortium.

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

```
group x in (select tuple (c.MSL, c.Geometry)
           from s in states, c in s.Counties)
               by (MSL: x.MSL)
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*).**

It 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 aggregates (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 geospatial data in a DBMS environment. We showed that geographic data, or thematic maps, can be decomposed into two levels of abstraction in many database models, such as extended relational, complex objects and object-oriented models. A higher level, the *map level*, is represented by 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 many approaches [3], [4], [63], [6], [15], [18], the geometric level corresponds to the definition of spatial abstract data types (ADTs). Some of this work [3], [15] 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 [4] 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.

Useful mechanisms encountered in many GIS applications are aggregation and 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 seemed necessary to investigate precisely which database features are required for defining powerful geographic models. Our approach consisted of 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 [20] for the following reasons: 1) it is a database data model, 2) it allows the definition of user-defined types and functions, and 3) it allows the *repeated use* of its intrinsic constructors (in contrast to the relational model for instance) which in turn allows one to express the composition of geographic objects *with arbitrary depth*. Besides, through the set of queries we presented, the reader could get an impression of the type of functions that one is likely to consider within a geographic DBMS.

We then studied the embedding of both the spatial ADTs and operations on maps in a database environment We proposed three classes of operators for manipulating geographic data: 1) user-defined functions on ADT (e.g., geometric union), 2) generic database algebraic operators with tuple-by-tuple processing such as selection and projection (which we called "linear processing"), and 3) specific algebraic operators with "nonlinear" processing (e.g., shortest path or allocate). We discussed the distribution 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 supports, namely, a relational model extended to ADT and an object-oriented one. In both cases, we studied 1) data representation (ADT and object composition) and 2) query formulation, i.e., adequacy for defining and using the three classes of operators. As far as our underlying reference data model is concerned, its neutral aspect together with its power for representing and manipulating complex geographic objects was very useful in our approach. We could define all the geographic features needed, and once the mapping of this model onto database model was done we could see the drawbacks and advantages of the two approaches. The example 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 *ad hoc* operations, queries based on aggregation (e.g., queries 1 and 2) are easily expressible in extended SQL (use of `group by`). In contrast, 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 [55]) although such a solution may lead to interfaces difficult to use. Finally, $O_2$SQL allows us to embed both set handling and calls to specific operators in a unified way.

## Acknowledgments

The authors would like to thank Michel Scholl for early discussions on this topic. Agnès Voisard would like to thank the International Computer Science Institute at Berkeley, California, where part of this work was carried out.

## References

[1] R.H. Güting, "An Introduction to Spatial Database Systems," *The Very Large Data Base J.*, vol. 3, no. 4, pp. 357–399, 1994.

[2] P. Rigaux, M. Scholl, and A. Voisard, *Spatial Databases—With Application to GIS,* San Francisco: Morgan Kaufman, May 2001.

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

[4] M.O. Scholl and A. Voisard, "Thematic Map Modeling," *Design and Implementation of Large Spatial Databases (SSD '89),* A. Buchman, O. Günther, T.R. Smith, and Y.-F. Wang, eds., 1989.

[5] S.-J. Jiang, H. Kitagawa, N. Ohbo, I. Susuki, and Y. Fujiwara, "Abstract Data Types in Graphics Databases," *Proc. IFIP Conf. Visual Database Systems,* T.L. Kunii, ed., 1989.

[6] M. Gargano, E. Nardelli, and M. Talamo, "Abstract Data Types for the Logical Modeling of Complex Data," *Information Systems,* vol. 16, no. 5, 1991.

[7] M.J. Egenhofer and A.U. Frank, "Object-Oriented Modeling in GIS: Inheritance and Propagation," *Proc. Auto-Carto 9,* pp. 588–598, 1989.

[8] M. Worboys, "Object-Oriented Approaches to Geo-Referenced Information," *Int'l J. Geographical Information Systems (IJGIS),* vol. 8, no. 4, 1994.

[9] T. Hadzilacos and N. Tryfona, "An Extended Entity-Relationship Model for Geographic Applications," *SIGMOD Record (ACM Special Interest Group on Management of Data),* vol. 26, no. 3, 1997.

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

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

[12] P. Rigaux and M.O. Scholl, "Multi-Scale Partitions: Application to Spatial and Statistical Databases," *Advances in Spatial Databases (SSD'95),* M.J. Egenhofer and J.R. Herrings, eds., pp. 170–184 1995.

[13] V. Delis, T. Hadzilacos, and N. Tryfona, "An Introduction To Layer Algebra," *Advances in GIS Research (SDH '94),* Waugh and Healey, eds., 1994.

[14] T. Hadzilacos and N. Tryfona, "Logical Data Modelling for Geographic Applications," *Int'l J. Geographical Information Systems (IJGIS),* vol. 10, no. 2, pp. 179–203, 1996.

[15] B. David, "Modeling, Representing, and Managing Geographic Information: An Extended Relational Approach," PhD thesis, Univ. of Paris VI, 1991. (In French.)

[16] M.O. Scholl, A. Voisard, J.-P. Peloux, L. Raynal, and P. Rigaux, *Systèmes de Gestion de Bases de Données Géographiques. Spécificités.* Paris, France: Int'l Thomson Publishing, Paris, 1996. (In French.)

[17] R.H. Güting and M. Schneider, "Realms: A Foundation for Spatial Data Types in Database Systems," *Advances in Spatial Databases (SSD '93),* D. Abel and B.C. Ooi, eds., 1993.

[18] R.H. Güting and M. Schneider, "Realm-Based Spatial Data Types: The ROSE Algebra," *The Very Large Data Base J.,* vol. 4, no. 2, pp. 243–286, Apr. 1995.

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

[20] S. Abiteboul and C. Beeri, "On The Power of Languages for the Manipulation of Complex Objects," *The Very Large Data Base J.* vol. 4, no. 4, pp. 727–794, 1995.

[21] P.P. Chen, "The Entity Relationship Model—Toward an Unified View of Data," *ACM Trans. Database Systems,* vol. 1, no. 1, p. 9, Mar. 1976, reprinted in M. Stonebraker, *Readings in Database Systems.* San Mateo, Cailf.: Morgan Kaufmann, 1988.

[22] S. Abiteboul, R. Hull, "IFO: A Formal Semantic Database Model," *ACM Trans. Database Systems,* vol. 12, no. 4, pp. 525–565, Dec. 1987.

[23] F. Bancilhon, T. Briggs, S. Khoshafian, and P. Valduriez, "FAD, A Powerful and Simple Database Language," *Proc. Conf. Very Large Data Bases (VLDB),* 1987.

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

[25] S. Shekhar, M. Coyle, B. Goyal, D.-R. Liu, and S. Sarkar, "Data Models in Geographic Information Systems," *Comm. ACM,* vol. 40, no. 4, pp. 103–111, 1997.

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

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

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

[29] L. Cardelli and P. Wegner, "On Understanding Types, Data Abstractions, and Polymorphism," *ACM Computing Survey,* vol. 17, no. 4, pp. 471–522, 1985.

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

[31] M. Stonebraker, B. Rubenstein, and A. Guttman, "Application of Abstract Data Types and Abstract Indices to CAD Databases," *Proc. Ann. Meeting Database Week,* pp. 107–113, 1983.

[32] S. Osborne and T. Heaven, "The Design of a Relational Database System with Abstract Data Types for Domains," *ACM. Trans. Software Eng.,* vol. 11, no. 3, pp. 357–373, 1986.

[33] M. Stonebraker, "Inclusion of New Types in Relational Data Base Systems," *Proc. Int'l Conf. Data Eng.,* pp. 262–269, 1986.

[34] B. David and A. Voisard, "A Unified Approach to Geographic Data Modeling," Technical Report 9,316, Univ. of Munich (LMU), 1993, available at http://www.inf.fu-berlin.de/voisard/pub.html.

[35] M.J. Egenhofer, A.U. Frank, and J.P. Jackson, "A Topological Data Model for Spatial Databases," *Design and Implementation of Large Spatial Databases (SSD '89),* A. Buchman, O. Günther, T.R. Smith, and Y.-F. Wang, eds., pp. 167–192 1989.

[36] M.J. Egenhofer and J.R. Herring, "A Mathematical Framework for the Definition of Topological Relationships," *Proc. Fourth Int'l Symp. Spatial Data Handling,* K. Brassel and H. Kishimoto, eds., vol. 2, pp. 803–813, 1990.

[37] M.J. Egenhofer and R.D. Franzosa, "Point-Set Topological Spatial Relations," *Int'l J. Geographical Information Systems (IJGIS),* vol. 5, no. 2, pp. 161–174, 1991.

[38] E. Puppo and G. Dettori, "Towards a Formal Model for Multiresolution Spatial Maps," *Advances in Spatial Databases (SSD '95),* M.J. Egenhofer and J.R. Herrings, eds., 1995.

[39] M. Erwig and M. Schneider, "Vague Regions," *Advances in Spatial Databases (SSD '97),* M.O. Scholl and A. Voisard, eds., pp. 298–320, 1995.

[40] Open GIS Consortium, *The OpenGIS Abstract Specification Model (Version 2)}.* 1997, available at URL: http://www.ogis.org.

[41] R.B. Tilove, "Set Membership Classification: A Unified Approach to Geometric Intersection Problems," *IEEE Trans. Computers,* vol. 29, no. 10, Oct. 1980.

[42] B.-C. Ooi, *Efficient Query Processing in a Geographic Information System.* Lecture Notes in Computer Science, no. 471. Berlin: Springer-Verlag, 1990.

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

[44] L. de Floriani, P. Marzano, and E. Puppo, "Spatial Queries and Data Models," *Spatial Information Theory (COSIT '93),* pp. 113–138, 1993.

[45] L. de Floriani and E. Puppo, "Geometric Structures and Algorithms for GIS," *Handbook of Computational Geometry,* J.R. Sack and J. Urrutia, eds., 1998, also available as DISI-TR-9708.

[46] D.J. Peuquet, "A Conceptual Framework and Comparison of Spatial Data Models," *Cartographica,* vol. 21, no. 4, pp. 66–113, 1984.

[47] A. Kemper and M. Wallrath, "An Analysis of Geometric Modeling in Database Systems," *ACM Computing Surveys,* vol. 1987, no. 1, pp. 48–91, 1987.

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

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

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

[51] M.O. Scholl and A. Voisard, "Object-Oriented Database Systems for Geographic Applications: An Experiment with O2," *Building an Object-Oriented Database System: The Story of O2,* pp. 585-618, 1992.

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

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

[54] T. Larue, D. Pastre, and Y. Viémont, "Strong Integration of Spatial Domains and Operators in a Relational Database Systems," *Advances in Spatial Databases (SSD '93),* D. Abel and B.C. Ooi, eds., pp. 53–72, 1993.

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

[56] O. Günther and W.-F. Riekert, "The Design of GODOT: An Object-Oriented Geographic Information System," *IEEE Data Eng. Bull.,* vol. 16, no. 3, p. 4, Sept. 1993.

[57] M.J. Egenhofer and A.U. Frank, "Object-Oriented Modeling for GIS," *J. Urban and Regional Information Systems Assoc.,* vol. 4, no. 2, 1992.

[58] O. Günther and J. Lamberts, "Object-Oriented Techniques for the Management of Geographical and Environmental Data," *The Computer J.,* vol. 37, no. 1, pp. 16–25, 1993.

[59] *Building an Object-Oriented Database System: The Story of O2.* F. Bancilhon, C. Delobel, P. Kanellakis, eds., San Francisco: Morgan Kaufmann, 1992.

[60] B. David, L. Raynal, G. Schorter, and V. Mansart, "GeO2: Why Objects in a Geographical DBMS?" *Advances in Spatial Databases (SSD '93),* D. Abel and B.C. Ooi, eds., pp. 264–276, 1993.

[61] O2 Technology, "The O2 User Manual, version 4.3," technical report, 1993

[62] Object Data Management Group (ODMG), URL: http://www.odmg.org, 2000.

[63] M. Gargano and E. Nardelli, "A Logical Data Model for Integrated Geographical Databases," *Proc. First Int'l Conf. Systems Integration,* pp. 473–481, Apr. 1990.

**Agnès Voisard** received the PhD degree in computer science from the University of Paris at Orsay and INRIA (French National Computer Science Research Institute) in 1992. In 1992/93, she was an INRIA Postdoctoral Fellow at the University of Munich. Since 1993, she has been an assistant professor of computer science at the Free University of Berlin, where she received her "Habilitation" in 1999. Her areas of expertise include geographic and environmental information systems, object-oriented databases, data modeling, user interfaces, and documentation management in cooperative work. She has participated in several program committees, is an editorial board member of *GeoInformatica*, and is a Steering committee member of the I-20 initiative on GIS interoperability, US National Center of Geographic Information and Analysis (NCGIA). In 1997, she was General Chair of the Fifth International Symposium on Spatial Databases (SSD'97). Dr. Voisand is coauthoring a book entitled *Spatial Databases— Applications to GIS* with Philippe Rigaux and Michel Scholl to be published by Morgan Kaufmann in 2001.

**Benoit David** received the engineer diploma from the Ecole Nationale des Sciences Géographiques (Saint-Mandè, France) in 1984 and the PhD degree in computer science from the University of Paris VI in 1991. He then joined the "Institut Geographique National" (IGN), Saint-Mandè, France, the French National Geographic Institute. In 1991/94, he was a researcher at the COGIT Laboratory of IGN and, in 1994/97, he was leading a team on definition and evaluation of quality of IGN geographic databases. Between 1997 and 2000, he was coordinating the information system of the French Ministry of Territory Planning and Environment. He is now with the French Ministry of Transportation. His areas of expertise include geographic and environmental information systems, extended relational system, object-oriented databases, data modeling and quality evaluation. He has participated in several standardization committees, including CEN and ISO.