

Cooperative Environments for Geographic Databases: a Prescriptive Analysis

Marco A. Casanova¹, Andrea S. Hemerly^{1,2} and Antonio L. Furtado²

¹Centro Científico Rio
IBM Brasil
Av. Presidente Vargas, 824/844
20071 Rio de Janeiro RJ - Brasil

²Departamento de Informática
Pontifícia Universidade Católica do RJ
R. Marquês de São Vicente, 225
22453 Rio de Janeiro RJ - Brasil

Abstract

This work addresses cooperative environments for geographic databases that help users achieve their goals. The nature of geographic data, as well as the requirements for an interface to geographic databases, are first discussed. Then, the components of a rule-based cooperative environment for conventional databases are described. The core of the paper analyzes in detail cooperativeness with respect to geographic data, separately considering query processing and the graphical display of query results.

1. Introduction

When implementing a Geographic Information System (GIS), user training is usually as expensive as software and hardware [?]. Hence, the development of tools that facilitate the use of geographic information systems is a relevant research topic.

We analyze in this paper the construction of cooperative environments that help users access geographic databases. We concentrate on the novel opportunities for cooperativeness that spatial data types and their operators provide, separating the problems of query processing from those related to the graphical display of query results.

Briefly, an environment is cooperative [?, ?] to the extent that it interacts with users in ways that: (1) contribute to the achievement of users' goals; (2) keep the users' understanding of the system in harmony with the definition and contents of the system, avoiding misconceptions and contributing to a fuller and at the same time more efficient usage; and (3) conform to the established integrity and authorization constraints.

We phrase our discussion in terms of the the cooperative environment we proposed in [?]. Briefly, the environment is driven by a set of rules that transform queries and answers. The rules may be specific to the application in question, they may be tuned to

a given class of users, or they may be of general applicability. Orthogonally, the rules may correct a query, complement a query or an answer, give alternatives for a query, provide explanations, solve ambiguities or edit an answer.

We formulate the examples in an SQL-like query language [?, ?, ?, ?], with spatial operators, which means that the queries will have the format **select-from-where**. However, we believe that the cooperative techniques introduced can be easily adapted to other classes of query languages.

The notion of request modification in database management systems was first proposed in [?] to enforce integrity constraints. Cooperative query processing is achieved in [?] by defining a rich conceptual model and a companion inference technique. A similar approach was followed in [?], which also bases query transformation on additional information provided with the schema, under the form of common topics of interest. In another direction, the generalization of failed queries to achieve cooperative behavior was investigated in [?]. A natural language database query system which recognizes users' presuppositions about the application domain is also described in [?]. To the best of our knowledge, there are no published papers that explore geographic data and spatial operators strictly to obtain cooperative behavior.

This paper is organized as follows. In section 2, we discuss the nature of geographic data, and the requirements for an interface to geographic databases. In section 3, we focus on cooperative query processing, while in section 4 we turn to cooperative presentation of geographic data. Finally, in section 5 we have the conclusions and directions for future work. æ

2. Geographic Databases

2.1. The Nature of Geographic Objects

2.1.1 Geographic Data Types

Ooi [?] classifies the attributes of an object stored in a geographic database in three types. The *conventional attributes* describe conventional properties of the object, such as the name and the population of a city. Their type is one of the usual alphanumeric data types, and their treatment follows the well-known technology of record-oriented database management systems.

The *spatial attributes* provide a geo-referenced geometrical description of the object, such as the political boundaries of a city. Their type is one of the special *geo-referenced spatial data types*, that come equipped with *spatial operators*, and that are usually built out of the primitive types 'point', 'line' and 'polygon', with the restriction that all coordinates must be with respect to a geographic coordinate system. In what follows,

we will also refer to these attributes as the *geometries* of the object, and we will frequently omit the adjective ‘geo-referenced’ when referring to their type. We call an object *spatial* if it has at least one spatial attribute.

Note that, to accommodate representations in different scales or distinct levels of detail, a geographic database should support spatial objects with multiple geometries. For example, a city may be represented by a polygon in a 1:10,000 scale, and by a point in a 1:1,000,000 scale, denoting its center.

The *pictorial attributes* give a pictorial description of the object, such as an aerial photograph of a city. They are used mostly for geographic analysis and help create spatial attributes. Their type is one of the special *geo-referenced raster data types*, that are usually built out of the primitive type ‘raster’ (or ‘pixmap’), with the restriction that all pixels are related to a coordinate in a given geographic coordinate system. Since their type is derived from the type ‘raster’, their usage is very limited in query languages.

Finally, we observe that, in geographic databases, information is often organized in *information layers* or *thematic planes* [?], which aggregate objects of a given geographic area, at a certain point in time, according to some theme. For instance, an information layer may refer to the land occupation, another to the utility lines and a third one to the political boundaries of a given area.

2.1.2. Spatial Operators for Geographic Databases

In general, spatial operators [?, ?, ?, ?] are functions whose domain or range involves a spatial data domain. They may be classified as:

1. *Topological operators*, which are invariant under topological transformations, such as translation, change of scale, and rotation. For instance, we have the operators `in`, `intersect` and `adjacent`, that return a Boolean value and define relationships between objects. We may also have binary operators, such as `union` and `intersection`, that return spatial objects. For each topological operator, we may in general define a dual one, such as `intersect/disjoint_from`.
2. *Metric operators*, which explore the existence of some metric, as distances and angle directions. Examples are the operators of `length` and `perimeter`.
3. *Linguistic operators*, which generally come from natural language and do not have precise semantics. Examples are the operators `near` and `between`.

Most query languages also offer *spatial constant constructors* [?, ?], which construct simple spatial objects. For example, the operator call `pt(x0,y0)` returns a point with coordinates x_0 and y_0 , the operator call `line(P,Q)` returns a line segment whose end-points are P and Q.

For a more complete list of the operators commonly found in spatial query languages, we refer the reader to [?]. Finally, we observe that the spatial query languages differ from each other not only by the set of spatial operators they offer, but also by the semantics they assign to the operators. Also, some query languages are extensible, in the sense that they allow the definition of new operators, appropriate for the application in question.

2.2. User Interfaces for Geographic Databases

A user interface for geographic databases should have at least the following functions [?]. First, it should allow the visualization of object geometries, which is the most natural way of analyzing spatial objects, especially their topological relationships (neighborhood, inclusion,...). In this context, one can have operations such as rotation, translation, zooming, panning and shifting. The interface should also let the user formulate new queries with the help of the graphical information present on the screen. In particular, it should allow the selection of objects on the screen by pointing to them, through the use of pointing devices (mouse, touch screen, etc...), and the delimitation of subareas (windows). In the latter case, the interface should support operations such as clipping and windowing.

Moreover, to create a friendly and dynamic interaction with the user, the interface should keep track of the queries and operations the user performed, and provide facilities for combining the results of several queries, as well as for interpreting the results shown on the screen. More precisely, in addition to an overlay operation, the interface should provide operations to clear, add, intersect, delete, and highlight query results. Furthermore, it should offer commands to define different display formats (color, pattern, intensity, symbols, associated text, etc...) for the different object classes, either locally to a session, or as defaults of object classes. Also, it should offer a *legend* to help the user examine the screen contents.

This short list by no means exhausts the functional requirements of user interfaces for geographic databases. In what follows, we briefly comment on two more points, namely, the definition of a visual space and the choice of a scale to exhibit the query results.

A *visual space*, or *visual context*, is a set of geometries or pictorial data, which the user did not explicitly request, but which are necessary to interpret the spatial position of the geometries retrieved by a query. To illustrate, suppose that the user submits a query asking for the location of the city of Orono [?]. The system would be less informative, though correct, if it showed on the screen just a point representing the city geometry. In this case, a reasonable visual space would be the political boundaries of the state of Maine, USA, where Orono is located. We discuss in detail the problem of creating visual spaces in section 4.1.

As for the choice of a scale, some authors [?] argue that, instead of working with

variable scales that optimize the allocation of geometries to windows, the interface should fix a reasonable set of scales, depending on the application, and select the one that suits best the current query. This facilitates the interpretation of query results by a GIS specialist, who usually works by analogy.

We refer the reader to [?] for a discussion about the questions of visual context, scale and legend. We also note that, in PSQL [?], the definition of a data type also includes the specification of default background. æ

3. Cooperative Query Formulation

In [?], we proposed a rule-based cooperative environment for conventional databases, and analyzed strategies to obtain rules to drive the environment, restricting ourselves to conventional properties of the conceptual schema. In this section, we briefly review these concepts and extend the analysis to take into account the peculiarities of querying geographic databases. In section 4, we carry the analysis further and focus on the cooperative presentation of geographic data. We express the ideas through informal examples, classified according to the role cooperativeness plays.

3.1. A Rule-Based Cooperative Environment

To achieve cooperative behavior, we introduce a set of rules to guide the query processor and results presentation. We identify *domain-independent rules*, which pertain to the data model adopted, as opposed to *domain-dependent rules*, which are specific to a given application or user class. A full description of the rule language can be found in [?].

The specification of a *rule-based cooperative environment* for a database application then consists of a set of domain-independent rules, which are proper to the environment, and a *database application context*. The database application context contains: the database conceptual schema, the external schemas and the application integrity constraints; an application domain model; one or more user models; and, for each user currently accessing the database, a session log registering queries from and responses to the user. We assume that the data model adopted supports the notions of specialization and aggregation.

There are two general classes of rules: the *request modification rules (RM-rules)*, which indicate the transformations queries may suffer, and the *template rules*, which define how to present query results. Orthogonally, we classify the RM-rules according to the execution phase where they are used. Thus, we have *pre-rules*, which are invoked before the execution of a query, *s-post-rules*, which follow the successful execution of a request, and *f-post-rules*, which follow execution in case of failure.

The request modification algorithm is formally described in [?]. Briefly, the algorithm successively applies all possible pre-rules to the query, generating a new one. Then, it executes the modified query against the database. After a successful execution, it can perform new actions as the result of the cumulative processing of s-post-rules. In case of a failed execution, the algorithm may apply f-post-rules, one at time, generating alternative queries, until one succeeds. If all alternatives fail, it also cumulatively tries f-post-rules to compensate for the failure. Finally, the algorithm uses template rules to edit query results.

To illustrate the use of the rule-based cooperative environment, we informally specify a sample geographic database, **Sample**, as well as some RM- and templates rules. For simplicity, in this section (and throughout the paper), the examples use an SQL-like language, extended with some spatial operators.

The **Sample** database schema has a class **city**, with attributes **name**, **population** and **geometry**, classes **school** and **health_institution**, with attributes **name**, **street**, **number** and **geometry**, and classes **district**, **river** and **lake**, with attributes **name** and **geometry**. It also introduces the classes **hospital** and **clinic** as specializations of **health_institution**, which means that they inherit its attributes. In addition, the class **hospital** has the attribute **number_of_beds**.

3.2. Correcting a Query

The automatic correction of ill-formulated queries perhaps provides the most natural examples of cooperative query processing. Our first example then illustrates the use of pre-rules for correcting the misuse of spatial operators.

Example 1 :

Suppose that the underlying DBMS offers a polymorphic version of the operator **intersect** that accepts as parameters polygons and lines, but not points, and a polymorphic version of the operator **in** that accepts as parameters polygons, lines and points. Consider the query ‘Retrieve all roads that cross city C’:

```
select road.name
from road, city
where city.name = 'C' and
       road.geometry intersect city.geometry
```

This formulation of the query presupposes that the geometries of cities and roads are polygons and lines, respectively. However, assume that the conceptual schema of the database in fact defines that the geometry of a city is a point, which implies that

the formulation of the query is mistaken. The system can detect this problem by a simple type checking, since the operator `intersect` does not accept a point as argument, by assumption. It may then correct the problem by invoking a domain-independent pre-rule that replaces an expression of the form `P.geometry intersect Q.geometry` by an expression of the form `Q.geometry in P.geometry`. The new query then becomes:

```
select road.name
from road, city
where city.name = 'C' and
      city.geometry in road.geometry
```

Note that the net effect of the pre-rule is to extend the semantics of `intersect` to cover other types of geometries, without actually modifying the underlying DBMS. But, more importantly, this approach permits defining elaborated extensions that can be made context dependent, or even user dependent, simply by creating more sophisticated rules.

3.3. Resolving ambiguities

In this section we analyse how a cooperative approach may help in the treatment of queries with ambiguous semantics, especially those formulated with the help of linguistic operators whose semantics depends on context information. However, we do not intend to discuss here all possible semantics for linguistic operators.

The following example deals with an imprecise metric operator, `near`, considered fundamental for the definition of spatial relationships. When a user formulates a spatial query, the notion of ‘near’ is naturally influenced by the context of the query [?]. For example, cities in a state map may appear near if they are less than one hundred kilometers apart, depending on the scale, whereas streets may appear near in a city map only if they are less than one kilometer apart. Robinson [?] pointed out that, besides depending on the context of the query, the notion of near may also depend on the user.

Thus, for the purposes of our discussion, we consider the following definition for `near`:

$$(P \text{ near } Q) \equiv (\text{dist}(P,Q) < n * \text{length_unit})$$

where `length_unit` is a context dependent scalar parameter, and `n` is a scalar parameter, that can be either unique for the application and thus defined in domain model, or specific for each class of users and therefore defined in the user models. This translational definition of `near` is easily captured by a pre-rule, as the following example illustrates.

Example 2 :

Consider the query ‘Retrieve all hospitals near school S1’:

```

select hospital.name
from hospital, school
where school.name = 'S1' and
      hospital.geometry near school.geometry

```

To process this query, the system will first apply the pre-rule that defines *near*, which results in the final query:

```

select hospital.name
from hospital, school
where school.name = 'S1' and
      dist(hospital.geometry,school.geometry) < n*length_unit

```

We next analyze the imprecise topological operator ‘between’. For simplicity, we assume in what follows that all objects involved are points.

When $m > 2$, we may establish a precise semantics for *between* by defining that:

$$(P \text{ between } (O_1, \dots, O_m)) \equiv (P \text{ in polygon}(O_1, \dots, O_m)) \quad (m \geq 2)$$

that is, *P between* (O_1, \dots, O_m) is ‘true’ iff the object *P* lies in the interior of the polygon created by connecting O_1, \dots, O_m , in this order (we ignore here the problem of non-convex polygons).

When $m = 2$, this definition would have to be adjusted to: *P between* (O_1, O_2) is ‘true’ iff *P* lies in the line connecting O_1 and O_2 . However, a better approach would be to define the semantics of *between* for $m = 2$ as follows:

$$(P \text{ between } (O_1, O_2)) \equiv (P \text{ in ellipsis}(O_1, O_2, n*length_unit)) \quad (m = 2)$$

where the parameters of *ellipsis* indicate that one of the axis is the line from O_1 to O_2 and that the other axis is n length units long. Indeed, this definition tries to capture the imprecise idea that a point between two others may not be exactly aligned with them.

We now turn to the spatial order operators *before/after*, whose semantics we define with the help of the concept of *observation point*. For example, a school may be before or after a hospital with respect to a given observation point. The system’s interface may represent an observation point by a special symbol and provide specific commands to move it around the user’s visual space. It may also be interesting to define multiple named points of observation in the same visual space. A possible treatment for *before* and *after* would be to reduce them to *between* as follows:

$$\begin{aligned} (P \text{ before } R \text{ from } O) &\equiv (P \text{ between } (O, R)) \\ (P \text{ after } R \text{ from } O) &\equiv (R \text{ between } (O, P)) \end{aligned}$$

where O is the observation point.

Just like *before/after*, we may provide semantics for the operators *left/right* and *in_front_of/behind*, except that they also admit variants that do not depend on the concept of observation point, if we restrict ourselves to classes of spatial objects with a clear concept of ‘side’. For example, ‘in front of a church’ makes sense without an observer, which is not necessarily the case for a lake.

Finally, most of the operators introduced above admit variants defined with respect to a given line. For example, the variant of *between*, for $m = 2$, will capture queries such as ‘Retrieve all hospitals between school S1 and school S2 via avenue A1’. A possible syntax for such operator would be *P between (O₁,O₂) via O₃*, where O_1 and O_2 are points on line O_3 . This expression would assume the value ‘true’ iff P lies in the segment of O_3 determined by the points O_1 and O_2 . The syntax and semantics of variants of other operators can be similarly defined. Naturally, all alternative semantics for linguistic operators can be captured by pre-rules.

3.4. Providing alternatives to a query

In addition to the strategies adopted for conventional databases, in the context of geographic databases, one may guide the generation of alternatives to failed queries by defining rules that relax search conditions based on spatial operators. However, since the cost of executing spatial queries may be high, the definition of such rules must be very judicious. In fact, it might be a sound strategy to force the system to validate the alternative queries before their execution, if their cost is considered high.

Example 3 :

Consider the query ‘Retrieve the name of the cities that contain lakes’:

```
select city.name
from city, lake
where lake.geometry in city.geometry
```

Suppose that the query fails. The system may exhibit cooperative behavior by modifying the query to retrieve instead all cities that share lakes with other areas. In general, we may define a generic f-post-rule that replaces a condition of the form P in Q by the disjunction $(P \text{ intersect } Q)$ or $(P \text{ adjacent } Q)$. The previous query then becomes:

```
select city.name
```

```
from city, lake
where lake.geometry intersect city.geometry or
lake.geometry adjacent city.geometry
```

3.5. Providing explanations

The integrity constraints of a conceptual schema, as well as the semantic constructs of the data model adopted, are sources of rules that provide useful failure explanations to the user. In the context of geographic databases, one may additionally explore topological properties [?] to generate explanation rules, as we exemplify below.

Example 4 :

Consider the following topological consistency rule, which is domain independent:

```
if    x.geometry in y.geometry and
      not (z.geometry intersect y.geometry)
then not (z.geometry intersect x.geometry)
```

Suppose that the conceptual schema has an integrity constraint saying that no road crosses a city:

$$(\forall r \in \text{road}) (\forall c \in \text{city}) (\neg (r.\text{geometry intersect } c.\text{geometry}))$$

Consider now the query ‘Which districts of city C does road R cross?’:

```
select district.name
from road, district, city
where road.name = 'R' and
      city.name = 'C' and
      district.geometry in city.geometry and
      road.geometry intersect district.geometry
```

When processing this query, a sophisticated system will use the topological consistency rule and the integrity constraint to inform the user that the query must fail a priori because, for any district D of C, since D lies in C and road R does not cross city C, then road R cannot cross D.

3.6. Complementing Queries

We close this section by addressing the problem of complementing queries, in the sense of retrieving more information than the user explicitly asked for. The basic strategy is again to provide rules that modify queries, perhaps by adding new attributes to their goals. However, such rules tend to depend on the application in question and are difficult to define. The following example illustrates these observations:

Example 5 :

Consider the query ‘Show all schools in district D’:

```
select school.geometry
from school, district
where district.name = 'D' and
      school.geometry in district.geometry
```

and suppose that there is a pre-rule saying that whenever the user requests the schools in an area, the system must also return the streets of that area. The application of this rule results in the new query - ‘Show all schools and all streets in district D’ - formulated as follows:

```
select school.geometry, street.geometry
from school, district, street
where district.name = 'D' and
      school.geometry in district.geometry and
      street.geometry in district.geometry
```

The interface should, in general, distinguish any geometry the user did not explicitly request from those that he did. In this example, the interface might use faded colors for the streets and bright colors for the schools.

æ

4. Cooperative Presentation of Geographic Data

4.1. Creating Visual Contexts for Exhibiting Query Results

As discussed in section 2, the design of an interface for geographic databases must carefully determine a context for the presentation of the query results. To help address

this question, we define the concept of a *visual space* as consisting of: a *background*, which is a collection of geo-referenced geometries or pictorial data; a *length unit*, which is a real number; an *observation point*, which is a point.

The concepts of length unit and observation point were already discussed in section 3.3, while the role of the concept of background is to help users interpret the spatial position of the geometries retrieved by their queries.

We assume in what follows that the interface will exhibit the geometries of the query results superposed with those of the background. Hence, the geometries of the background must be distinguished from those explicitly requested, and the background must not be dense to avoid shifting the focus of attention from the query results. However, instead of always superposing the query results onto a background, the interface may also exhibit the query results and the background on separate windows, or present a selection menu to the user and let him decide which background he wants, if any.

We assume that each geographic database includes a collection of predefined visual spaces, some of which may be specific to a given class of users. We allow visual spaces to be constructed out of virtual objects, defined over the other objects stored in the database. For example, in the above visual space, we may define the background as a (database) view over the streets of the city map, which implies that any new major avenue added to the city map will be included in the background automatically.

The current visual space may be determined in various ways: (1) the domain model or the current user model provides a visual space as default; (2) the user explicitly selects one of the visual spaces stored in the database, or he may construct a new one; and (3) the system automatically selects one of the visual spaces stored in the database, or it constructs a new one. Naturally, the visual space may change during a session, or the user may even decide that he does not want visual spaces.

The interface may follow several strategies to automatically select a visual space from those predefined. For example, it may choose the visual space whose background contains all geometries of the query results, or one whose background covers the geographic area of all the geometries of the query results. If these criteria fail, the interface may try to partition the query results by geographic area before re-applying them. On the other hand, if more than one visual space meets these criteria, the interface may resort to heuristics to guide the choice, such as to select the visual space whose background has the largest scale.

The above suggestions for the automatic selection of visual spaces are sources of rules that capture cooperative behavior, as the next example illustrates.

Example 6 :

Consider the query ‘Show all schools in district D’:

```
select school.geometry
from school, district
where district.name = 'D' and
      school.geometry in district.geometry
```

Assume that the database in question contains a family of visual spaces, one for each district, whose backgrounds contain the perimeter and the major avenues of the districts.

The strict result of the above query naturally contains just the geometries of the schools in district D, which is not very informative. The system will then exhibit these geometries along with the visual space for district D, which is clearly suitable for the query, since its qualification indicates that all retrieved geometries belong to the geographic area of district D.

Finally, we may compare the concepts of visual space and query complementation discussed in section 3.6, since both offer to the user additional information not originally requested. On one hand, visual spaces are relevant only when visualization of query results is in discussion, whereas query complementation applies even to conventional queries. On the other hand, the selection of a visual space may depend on factors outside a single query, whereas query complementation considers each individual query by itself.

4.2. Other Examples

Depending on its conventional attributes and on the current scale, a spatial object may have distinct forms of presentation, which differ from each other by the color, the filling pattern and the associated symbol and text, defined according to international cartographic conventions, conventions adopted by the institution, or even depending on the user's preferences. A geographic database system must then have mechanisms to guide the presentation of spatial objects. Naturally, template rules can be used for this purpose, as well as for resolving ambiguities or conflicts that sometimes arise, such as presentation conflicts in the superposition of spatial objects.

Also, we observe that, instead of resolving ambiguities and conflicts automatically, the system may just detect their occurrence and ask the user for guidance. This alternative must be used with care to avoid long and tedious interactions with the user.

As already argued in section 2.2, the interface must provide functions to help the user distinguish which objects populate his visual space. The system may also include template rules that automatically include a short legend in the visual space.

Moreover, a query may retrieve certain objects that are not shown in the visual space because, for example, they were discarded for the scale in question. In this case, the system may use template rules to generate explanations upon request.

Finally, the session log can be explored in several ways to create domain-independent rules that improve the cooperativeness of the system, for instance, in conjunction with a rule that tries to determine the user's focus of interest. For example, suppose that, during a session, the user repeatedly submits queries involving schools. The system may infer that his primary interest is therefore on schools and, from now on, highlight all schools that show up in the visual space.

æ

6. Conclusions and Directions for Future Research

Geographic databases generally hold a large volume of data, characterized by a rich variety of types and complex relationships. Moreover, users express queries with the help of both conventional and spatial operators. Section 2 briefly reviewed some basic concepts of geographic databases.

In this paper, we explored the new opportunities, not present in the context of conventional databases, that the spatial data types and their operators provide for cooperative query processing. We analyzed, in sections 3 and 4, the use of cooperativeness for query modification and for the presentation of query results. We showed, among other features, that a cooperative environment may adequately capture the semantics of spatial linguistic operators, as well as provide visual contexts for exhibiting query results.

The prototype of the rule-based cooperative environment described in section 3.1 is completely operational. As future work, we intend to re-target the prototype to a GIS system, called SPRING [?], and experiment with some of the rules described in sections 3 and 4.

Acknowledgments

This work is part of a joint research program carried out by the Rio Scientific Center of IBM Brazil and the Brazilian Space Research Institute (INPE). We wish to thank Gilberto Camara, Ubirajara Freitas, Ricardo Cartaxo and Diogenes Alves for many hours of helpful discussions about geographic databases. æ