ATTI DEL
SECONDO CONVEGNO NAZIONALE SU
SISTEMI EVOLUTI PER BASI DI DATI

a cura di
Sonia Bergamaschi
Claudio Sartori
Paolo Tiberio

In collaborazione con:
Consiglio Nazionale delle Ricerche
Centro di Studio per l'Interazione Operatore-Calcolatore
Progetto Finalizzato "Sistemi Informatici e Calcolo Parallelo"

Progetto MURST 40%
"Metodi formali e strumenti per basi di dati evolute"
UNITU.RIM. S.p.A., Rimini

6 - 8 Giugno, 1994
Rimini, Italia
<table>
<thead>
<tr>
<th>Topic</th>
<th>Authors</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASPETTI TEORICI DI BASI DI DATI EVOLUTE</td>
<td>P. Fraternali, D. Montesi, L. Tanca</td>
<td>3</td>
</tr>
<tr>
<td>Active Database Semantics</td>
<td>D. Beneventano, S. Bergamaschi, S. Lodi, C. Sartori</td>
<td>23</td>
</tr>
<tr>
<td>Reasoning with Constraints in Database Models</td>
<td>F. Cesarini, E. Grazzini, G. Soda</td>
<td>39</td>
</tr>
<tr>
<td>BASI DI DATI MULTIMEDIALI</td>
<td>M. Agosti, F. Crivellari, M. Melucci</td>
<td>61</td>
</tr>
<tr>
<td>A Formalization of the EXPLICIT Model Mechanisms for a Development Application Environment</td>
<td>E. Clementini, F. Di Felice</td>
<td>101</td>
</tr>
<tr>
<td>Multidata: uno strumento di modellazione intensionale</td>
<td>F. Benevento, P.D. Monari, L. Pazzi</td>
<td>81</td>
</tr>
<tr>
<td>Topology in an O-O GIS</td>
<td>E. Clementini, F. Di Felice</td>
<td>101</td>
</tr>
<tr>
<td>TECNICHE DI OTTIMIZZAZIONE NELLE BASI DI DATI</td>
<td>P. Arosio, C. Francalanci, B. Pernici</td>
<td>123</td>
</tr>
<tr>
<td>Classificazione gerarchica di schemi concettuali</td>
<td>G. Bellavia, D. Maio, S. Rizzi</td>
<td>141</td>
</tr>
<tr>
<td>A Dynamic Schema for the Storage of Signature Files in Multiple-Disk Systems</td>
<td>P. Ciaccia, G. Melini</td>
<td>161</td>
</tr>
<tr>
<td>BASI DI DATI DEDUTTIVE</td>
<td>F. Buccafurri, S. Greco, E. Spadafora</td>
<td>187</td>
</tr>
<tr>
<td>Implementazione di Interrogazioni Semplici Concatenate</td>
<td>L. Cabibbo, G. Mecca</td>
<td>207</td>
</tr>
</tbody>
</table>
TOPOLOGY IN AN O-O GIS

Eliseo Clementini and Paolino Di Felice

Dipartimento di Ingegneria Elettrica
Università dell’Aquila,
67040 Poggio di Roio, L'Aquila.
[eliseo@dsiaq.ing.univaq.it, pdifelice@vxscaq.aquila.infn.it]

ABSTRACT

Topology is an important kind of knowledge in Geographical Information Systems (GISs). In this paper a small set of user-oriented topological relationships and operators is used as formal background for the implementation of topology on top of the O2 Database Management System (DBMS). Object-oriented (O-O) systems allow to model geometric data and topological relationships thanks to the object identity, the modeling flexibility, the inheritance, and the overriding/overloading features. The representation adopted for the topology is a trade-off between avoiding data redundancy and maintaining database schema independence. Examples of topological queries in O2SQL conclude the paper.

1 INTRODUCTION

A large variety of queries in GISs concerns the spatial relationships among objects. A few examples of this category of queries are:

- Retrieve the names of all the mountains higher than 3500 metres and located inside the boundary of Italy;
- Retrieve all the nations adjacent to the nation whose capital is Berlin;
- Retrieve all the four-star hotels that are nearby the main railway station in Rome.

SQL, the standard query language for relational DBMSs [1], does not support such queries. The limits of SQL to serve as a query language also for geographical applications are clearly pointed out in [11]. A big drawback of SQL, as well as of its dialects (e.g., [10]), is the lack of spatial operators.

The first step towards the design of an interactive GIS query language requires a sound formalization of spatial relationships among geographic objects. So far there is a good but still incomplete theory of topological relationships, that is the subset of spatial relationships characterized by the property of being preserved under topological transformations, such as translation, rotation, and scaling [18]. Recent results on topological relationships are reported in [6, 7, 12, 13, 14, 15, 16, 17, 20].

Aim of the present paper is, firstly, to report about the implementation of the theory behind the so-called Calculus-based Method (CBM) [6], hence, to give examples of topological queries.

The CBM allows to model the meaningful topological situations between two geographic objects represented as simple geometric elements (namely, points, lines, and areas) in terms of five relationships and three boundary operators. Remarkable properties of the CBM are:

- completeness, the five relationships makes a full covering of all the possible topological situations,
- exclusiveness, it can not be the case that two different relationships hold between the same two objects, and
- expressiveness, the CBM is able to distinguish among finer topological configurations than an entire category of previous methods all based on the point-set theory [4], namely: the four intersection method [12], the dimension extended method [6], and the nine intersection method [13].

As software platform we adopted the O2 DBMS [19]. The suitability of O2 for implementing GIS applications has been already assessed (e.g., [8, 21]). The advantages of using the O-O DBMS technology are, among others, object identity, complex object modeling, inheritance/overriding. Object identity allows the sharing of data among different spatial objects and therefore it is a mechanism to avoid redundancy. Geometric data can be easily modeled in terms of the constructors offered by the O-O DBMS, e.g., a polygon can be modeled as a set of lines. Eventually, inheritance and overriding allow to apply the same general topological operators to spatial objects of different types.

Implementing topological relationships one can basically choose between two methods to achieve this. The first one is based on using a topological data structure, as, for instance, described in [9]. Most queries now result in relatively simple look-up actions of the referred features in the data structure. The more difficult part of this approach is creating and maintaining a consistent topological data structure. The major drawback of this approach is that the implementation of the topological relationships depends on the data structure, i.e. the database schema. If one decides to change certain names in the schema, the implementation will not work any more.

The second approach to implementing topological relationships is based on using the geometric attributes of the independent features, e.g., see the implementation discussed in [5]. In this way the implementation does not depend on a certain database schema, but only uses the geometric data types (point, line, and area) for evaluating the relationships.

Drawbacks of this approach are: if two areas share a boundary, then it has to be stored
twice; moreover, one has to deal with the finite precision of the computer, that is, one has to allow a certain tolerance in the determination of intersections.

Implementing the CBM, we adopted a hybrid approach that uses both methods for representing topology depending on the kind of geometric objects we are dealing with. It is up to the application designer to decide what kind of information should be put on the same "layer" (congruent objects) or on different layers. In the first case, we avoid redundancy, while in the second case we allow it. Such mechanisms are transparent to the users that take advantage of the self-contained set of topological relationships and operators being part of the CBM, all implemented as O2C methods.

The paper is organized as follows. Section 2 is a brief introduction of the O2 data model and its associated query language; Section 3 gives a classification of geographic data as in [22]. Section 4 describes the way we modeled geometric data and how we deal with the problem of redundancy. Section 5 illustrates the formal background used for the definition of spatial relationships and gives the general strategy used for their implementation. Section 6 presents two topological queries; finally Section 7 makes some concluding remarks.

2 THE O2 DATA MODEL AND QUERY LANGUAGE

Aim of this section is to mention those features of the O2 DBMS useful for the purposes of the present paper. Further information can be found in [19].

In O-O data models the information is organized as objects. O2 provides the user with the possibility of defining not only objects but also complex values. A type is associated with every class, describing the structure of its instances. Types are constructed recursively using atomic types (i.e., integer, real, char, string, boolean, and bits) and applying to them the set, list, and tuple constructors.

A class describes the structure and the behavior of a set of objects. The structural part of a class is its type and the behavioral part is a set of methods. In O2, methods can be public and private, being the default private. Two examples of (private) methods, written by using the O2C programming language, are given in Section 5.3. In O2, objects and values can be named. Every named object/value is persistent. Inheritance allows the programmer to define classes in an incremental way by refining already existing ones. O2 supports an inheritance mechanism based on subtyping. In O2 all classes inherit from the predefined class Object.

The schema is a collection of names and definitions of types and classes related by inheritance links and/or composition links. A base is a collection of data (objects and values) whose structure and behavior conform to the definitions in a schema.

Once a database has been created and populated, it is possible to querying it through O2SQL, that is an ad hoc query language whose syntax is styled on standard SQL.

O2SQL allows to querying both data and methods [2]. The advantages induced by such a facility will be evident in Section 6. The O2SQL interpreter may be invoked either interactively or under program control via the system supplied function o2query. Two examples of interactive O2SQL queries are given in Section 6.

3 GEOGRAPHIC DATA TYPES

The geographic entities in a GIS are based on two different types of data: spatial data and thematic data. In turn, spatial data have two components: geometric and topological data. In this paper these terms are defined as in [22]:

Geometric data cover aspects concerning location (coordinates) and shape of geographic objects, so allowing the modeling of the basic map features, namely, point features, line features, and area features.

Topological data describe the relationships between the geometric data. Adjacency and inclusion are examples of topological relationships. Topological data are not always stored explicitly, because in principle they can be derived from geometric data.

Thematic data are alphanumeric data related to geographic entities; e.g., the name and capacity of a lake. Thematic data may be any kind of data that can be found in traditional databases, e.g., strings, integers, and reals.

Section 4 deals with the modeling of geometric data in O2, while Section 5 concerns the modeling of topological data. Thematic data are not treated in the paper, because their modeling is straightforward.

```
Class Point
  type tuple (x: real, y: real)
  method init(x: real, y: real), ...
end;

Class Line
  type tuple (shape: list (Point))
  method init(shape: list (Point)), ...
end;

Class Polyline
  type tuple (shape: list (Line))
  method init(someLines: set (Line)), ...
end;

Class Polygon
  type tuple (shape: list (Line))
  method init(someLines: set (Line)), ...
end;
```

Fig.1. The basic classes.
4 MODELING OF GEOMETRIC DATA

A way of modeling geometric data in O2 consists in defining four basic classes: Point, Line, Polyline, and Polygon (Fig. 1). Note the presence in the class Polyline (Polygon) of a tuple with a single attribute. This may appear strange at first sight, but it is the ordinary way in O2 to use Polyline (Polygon) as the superclass of geographic classes like, for instance, River and Road (Lake and Country).

4.1 AN EXAMPLE

Each country is organized in terms of a certain number of administrative units; for instance, Italy has three levels of administrative units: municipalities, provinces, and regions. The class AdUnit is an incomplete description of one of these three levels.

```java
Class AdUnit inherit Polygon
rename attribute shape from class Polygon as boundary
```

The geometric attribute boundary is a polygon representing the boundary of an administrative unit (the inherited attribute shape has been renamed for readability purposes).

To enter instances of the class AdUnit into the database they must be persistent. In O2 objects of a given class are not automatically persistent. A way to achieve persistence consists in giving a specific name to objects. One strategy is to assign a name to each administrative unit; but the more practical way is to create and maintain a single named set value to hold the entire extension of the class AdUnit. Following the second strategy, we need to add to the O2 schema the following statement:

```java
name AdUnits: set(AdUnit);
```

The following piece of O2C code implements the strategy for entering instances of the class AdUnit in the O2 base:

```java
aPolygon = new Polygon(aSetOfLines);
/* it builds a polygon and checks for redundancy */

anAdUnit = new AdUnit(aName,aPolygon);
/* it creates a new administrative unit */

AdUnits += set(anAdUnit);
/* it makes anAdUnit persistent */
```

4.2 CONTROLLING REDUNDANCY

A big problem with the geometric data is redundancy. By controlling redundancy we can:

(i) keep the data consistent (storing only once the common boundary between two adjacent polygons ensures that they do not overlap because of, for instance, numerical roundings);

(ii) reduce the size of the database (this is significant, because GISs usually deal with very large data sets).

The following generic example shows where redundancy comes from.

The simple map above is made up of two polygons (plg1 and plg2), defined in terms of four lines (ln1, ln2, ln3, and ln4), that is:

- plg1=[ln1, ln2], plg2=[ln2, ln3, ln4], where:
  - ln1={(1,2,3), ln2=(3,1), ln3=(3,4), ln4=(1,4)}

From the definition of plg1 and plg2, we can see that ln2 appears twice because plg1 and plg2 are neighbors. This implies that all the points in ln2 (i.e., 3 and 1) are stored twice.

The situation above is undesirable. Our wish is to store each line once and share it several times. In principle, the sharing of objects is not a problem when the O-O technology is used. In fact, objects can be shared by referring to them through their oid. The method init of the class Polygon tests if a line is already in the base. If so, instead of storing redundant data, the method stores the link to the original object.

In the current implementation, redundancy of the geometric data is prevented with respect to objects that satisfy either condition (a) or (b):

(a) they are homogeneous, i.e., the objects are instances of the same class,

(b) they are congruent, i.e., the objects are instances of classes whose geometric attributes correspond to geographic entities that are not independent each other, in the sense that they have to satisfy a common geometric constraint. It is the case of various administrative units; another example of congruent objects are rivers and lakes. The congruence between objects is basically a choice in the design of a GIS application.
Fig. 2. A hierarchy of geographic classes.

To clarify this point, below we discuss three examples by referring to the hierarchy of classes of Fig. 2.

**Example 1**

The assumptions:
- \( b_1 \) is the boundary of an administrative unit (e.g., a municipality)
- \( b_2 \) is the boundary of a lake
- \( l_{n1} \) and \( l_{n2} \) are lines with the same end point values (Fig. 3).

![Fig. 3. Two non-congruent objects.]

**Example 2**

The lines \( l_{n1} \) and \( l_{n2} \) are part of the boundaries of two distinct instances that are not homogeneous nor congruent, therefore \( l_{n1} \) and \( l_{n2} \) have not to be considered as a repetition of data. This approach allows to keep the boundary of a political area (i.e., an administrative unit) conceptually and practically distinct from the boundary of a physical area (i.e., a lake). That is useful because we know that changes in the shape of one of the two boundaries do not necessarily involve changes in the other.

**Example 2**

The assumptions:
- \( b_1 \) and \( b_2 \) are the boundaries of homogeneous/congruent instances.
- \( l_{n1} \) and \( l_{n2} \) are lines with the same end point values (Fig. 4).

According to the assumptions, it is sufficient to store \( l_{n1} \) (\( l_{n2} \)). Obviously, when entering the data concerning \( b_2 \) (\( b_1 \)) instead of \( l_{n2} \) (\( l_{n1} \)) we have to store a pointer to such an object. If, for instance, \( b_1 \) and \( b_2 \) represent the boundaries of two bordering countries, it is correct that if the shape of \( b_1 \) changes then the shape of \( b_2 \) also changes. This ensures that the topological relationships between the two countries does not change if their boundary does.

**Example 3**

The assumptions:
- \( b_1 \) is the boundary of an administrative unit (e.g., a municipality)
- \( b_2 \) is the shape of a road
- \( l_{n1} \) and \( l_{n2} \) are lines with the same end point values (Fig. 5).
5 MODELING OF TOPOLOGICAL DATA

Topological data describe the relationships between the geometric data. Topological knowledge can be easily modeled in terms of binary topological relationships. Egenhofer [12] originally described a method for classifying binary topological relationships between area features. The classification is based on the intersections of the boundaries and interiors of the two features. Later, Egenhofer extended the classification process also to point and line features, resulting in 6 major groups of binary relationships: area/area, line/area, point/area, line/line, point/line, and point/point [13].

This classification method suffers of two serious drawbacks:
- too many names of relationships to be remembered;
- it is impossible to distinguish between certain cases, which are usually regarded as different by users. For example, two areas that have one point in common, and two areas that have a complete line in common, do both fall under the same "touch" relationship.

In [6] a classification method that overcomes the limits above is proposed: the CBM. According to the CBM, all possible topological relationships are grouped together into only five topological relationships named: touch, in, cross, overlap, and disjoint. These relationships are overloaded concepts in the sense that they may be used for point, line, and area type of features. More detailed distinctions among topological situations are made possible by combining the previous five topological relationships with three topological operators able to extract boundaries from area and line features.

5.1 THE CALCULUS-BASED METHOD

In this section, we recall the definitions of binary topological relationships and topological operators as in [6]. To this purpose, it is necessary to introduce firstly some notations.

The symbols P, L, and A are used for point, line, and area features, while the symbol \( \lambda \) is used to represent one of the three feature types. The notation \(<\lambda_1, r, \lambda_2>\) means that the features \( \lambda_1 \) and \( \lambda_2 \) are involved in the binary topological relationship \( r \). The function \( \text{dim} \) returns the dimension of a point-set, which can be undefined, 0, 1, or 2, respectively.

The boundary of a feature \( \lambda \) is denoted by \( \partial \lambda \). It is defined for each of the feature types as follows:
- \( \partial P \): the boundary of a point feature is empty;
- \( \partial L \): the boundary of a line is the set of the two separate end-points;
- \( \partial A \): the boundary of an area is a circular line consisting of the area's limit points.

The interior of a feature \( \lambda \) is denoted by \( \lambda^0 \). It is defined as \( \lambda^0 = \lambda - \partial \lambda \). Note that the interior of a point is equal to the feature itself. We call \( f \) (from) and \( t \) (to), respectively, the end-points of a line feature.

\[ \text{Def.1.} \quad <\lambda_1, \text{touch}, \lambda_2> \iff (\lambda_1^0 \cap \lambda_2^0 = \emptyset) \land (\lambda_1 \cap \lambda_2 \neq \emptyset) \]

\[ \text{Def.2.} \quad <\lambda_1, \text{in}, \lambda_2> \iff (\lambda_1 \cap \lambda_2 = \lambda_1) \land (\lambda_1^0 \cap \lambda_2^0 = \emptyset) \]

\[ \text{Def.3.} \quad <\lambda_1, \text{cross}, \lambda_2> \iff (\dim(\lambda_1^0 \cap \lambda_2^0) = \max(\dim(\lambda_1^0), \dim(\lambda_2^0)) - 1) \land (\lambda_1 \cap \lambda_2 \neq \lambda_1) \land (\lambda_1 \cap \lambda_2 \neq \lambda_2) \]

\[ \text{Def.4.} \quad <\lambda_1, \text{overlap}, \lambda_2> \iff (\dim(\lambda_1^0) = \dim(\lambda_2^0) = \dim(\lambda_1^0 \cap \lambda_2^0)) \land (\lambda_1 \cap \lambda_2 \neq \lambda_1) \land (\lambda_1 \cap \lambda_2 \neq \lambda_2) \]

\[ \text{Def.5.} \quad <\lambda_1, \text{disjoint}, \lambda_2> \iff \lambda_1 \cap \lambda_2 = \emptyset \]

\[ \text{Def.6.} \quad \text{The boundary operator } b \text{ for an area } A \text{ returns the circular line } \partial A. \]

\[ \text{Def.7.} \quad \text{The boundary operators } f, t \text{ for a line } L \text{ return the two separate points belonging to the set } \partial L. \]

5.2 QUALITATIVE DESCRIPTION

The relationships of the previous section correspond to the following five groups of topological situations.

- **Touching of features**: Two features touch each other, if the only thing they have in common is contained in the union of their boundaries (Fig. 6a).
- **Containment of features**: One feature is in another one if the former is completely contained into the latter (Fig. 6b).
- **Crossing of features**: Two lines cross each other if they meet on an internal point (note that it could not be a touch because in that case the intersection is only on the boundaries). Similarly, a line crosses an area if the line is partly inside the area and partly outside (Fig. 6c).
- **Overlapping of features**: Two features overlap each other if the result of their intersection is a third feature of the same dimension, but different from both of them. It
comes out from the definition that this relationship can apply only to homogeneous cases (Fig. 6d).

Disjunction of features: Two features are disjoint if their intersection is void (Fig. 6e).

Other examples can be found in [3].

5.3 THE IMPLEMENTATION OF THE CBM

The five relationships: touch, in, cross, overlap, and disjoint and the three operators: f, t, and b were all implemented as O2C methods.

The class GeometricElement (Fig. 7) plays a strategic role with regard to the implementation of the topology.

Implementation of the operators

The first step was to define three body-empty methods: f, t, and b, inside the class GeometricElement. They are inherited by all the subclasses of GeometricElement.

The second step was to override methods f and r inside the classes Line and Polyline and method b inside the class Polygon. Fig. 8 shows the implementation of the method f in the class Polyline.

The implementation of operators f, t, and b was made possible by the overriding/overloading feature of O-O systems.

Implementation of the relationships

The objective was to implement each relationship as a single method able to manage the following six general situations: area/area, area/line, area/point, line/line, line/point, and point/point. Therefore the name of the method implementing a relationship is an overloaded identifier that can be used to define topological conditions in the where clause of the O2SQL query language (see next section).

Let r be the name of one of the five topological relationships of the CBM. It was implemented by adding the method:

```plaintext
method r (anObj: GeometricElement): boolean;
```
in the specification of the class GeometricElement (Fig. 7). In this way the method \( r \) is inherited by all the subclasses of GeometricElement, hence the message \( r \) can be sent to instances of any of these classes.

The major problem encountered implementing the method \( r \) was how to find out the actual topological situation among the six possible. This is essential for the activation of the appropriate \( \text{O2C block} \) inside the body of method \( r \). This problem was solved by adopting the \text{type of} system supplied method. \text{type of} is an internal \( \text{O2} \) method that returns an identifier indicating the actual type of the data encapsulated by the receiver object.

```plaintext
method body Touch(anObj: GeometricElement): boolean
in class GeometricElement

c2 Polygon  aPolygon;
c2 Polyline  aPolyline;
c2 Point  aPoint;
c2 boolean  result;

if (self->type_of == aPolygon->type_of) &&
   (anObj->type_of == aPolygon->type_of) result = self->TouchPgPg(anObj);
if (self->type_of == aPolygon->type_of) &&
   (anObj->type_of == aPolyline->type_of) result = self->TouchPgPl(anObj);
if (self->type_of == aPolyline->type_of) &&
   (anObj->type_of == aPolygon->type_of) result = anObj->TouchPgPn(anObj);
if (self->type_of == aPolyline->type_of) &&
   (anObj->type_of == aPoint->type_of) result = self->TouchPlPn(anObj);
if (self->type_of == aPoint->type_of) &&
   (anObj->type_of == aPolygon->type_of) result = anObj->TouchPntPn(self);
if (self->type_of == aPoint->type_of) &&
   (anObj->type_of == aPolyline->type_of) result = self->TouchPlPnt(anObj);
if (self->type_of == aPoint->type_of) &&
   (anObj->type_of == aPoint->type_of) result = self->TouchPntPnt(anObj);

return result;
```

Fig. 9 shows the code of the method \( \text{Touch} \) able to recognize the type of the receiver of the message \( \text{Touch} \), as well as the type of the second object involved in the relationship. After this initial step, the method starts the real computation by activating the routine able to assess the specific topological situation.

### 6 TOPOLOGICAL QUERIES

In this section two examples of topological queries are presented. The queries are posed against a geographical database concerning European countries. For each country, information about the longest rivers and the largest cities are stored in the database. Fig. 10 shows the properties of the corresponding \( \text{O2} \) classes: Country, River, and City.

```plaintext
Class Country inherit A_Unit
  public type
tuple (capital: string,
       flag: MyImage,
       language: string,
       description: ImportedText)

name Countries: set (Country);
name Rivers: set (River);
name Cities: set (City);
name Roads: set (Road);
name Railways: set (Railway);
```

Fig. 10. Definition of classes and names.
Query 1: Select all Italian cities with more than 500,000 inhabitants.

The O2SQL query is the following:

```sql
select city
from city in Cities,
  country in Countries
where country -> name = "Italia" and
  city -> population > 500000 and
  city -> in (country)
```

In the previous query, "city -> in (country)") has the following meaning: in is the name of the method implementing the relationship is of the CBM. The method in has country as argument. The message in (country) is sent to an instance of the class City and it checks if the city is inside the boundary of the country. Cities and Countries are two sets containing, respectively, the instances of the classes City and Country stored in the database. Technically speaking, Cities and Countries are both O2 named objects and their definition is part of the database schema (Fig. 10).

Thanks to the possibility of referring to methods in the where clause of O2SQL queries, O2SQL becomes an "open" query language, therefore it is easy to define topological queries without the need of extending it. The screen dump in Fig. 11 is the answer to the Query 1. By clicking on one of the cities that are part of the answer, the full list of its properties appears. The presentation is managed by the O2LOOK graphic user interface [19].

The following query uses another topological method (touch) to find the countries sharing (part of) their boundary with Italy.

Query 2: Select all the countries bordering on Italy.

The O2SQL query is the following:

```sql
select country2
from country1 in Countries,
  country2 in Countries
where country1.name = "Italia" and
  country2 -> Touch (country1)
```

Fig. 12 displays the answer.

7 CONCLUSIONS

In this paper, we summarized an experience of use of the O2 DBMS as a software platform for the representation of geographic data types. We saw that the O2 data model is particularly suitable for the representation of these categories of non-traditional data. The modeling flexibility, the inheritance, and the overriding/overloading are the O-O features majorly useful for the representation of geographic data types. Furthermore, the methods are essential for the implementation of redundancy checks on geographic data and topological relationships and operators.

The spatial data structure we defined is a trade-off between a topological data structure (i.e., topological information is stored aside geometric data) and an integrated data structure (i.e., each object is stored independently from the others). In this way, redundancy can be avoided in single layer maps, still maintaining object independency. Using this data structure, we implemented methods for assessing topological relationships and operators of the CBM, which is a model for topological information featuring high expressiveness and ease of use in a spatial query language.

Acknowledgments - Work supported by the Italian National Research Council (C.N.R.) under the project "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo"; grant n. 93.01595.PF.69.

REFERENCES


Fig. 12. The output of Query 2.