GeoInformatica 8:2, 107–141, 2004
© 2004 Kluwer Academic Publishers. Manufactured in The Netherlands.

A Filter Flow Visual Querying Language and Interface for Spatial Databases

A.J. MORRIS,[1] A.I. ABDELMOTY,[2] B.A. EL-GERESY,[1] AND C.B. JONES[2]
[1] *School of Computing, University of Glamorgan*
E-mail: {ajmorri2, bageresy}@glam.ac.uk
[2] *Department of Computer Science, Cardiff University*
E-mail: {a.i.abdelmoty, c.b.jones}@cs.cf.ac.uk

Received May 12, 2003; Revised September 5, 2003; Accepted September 23, 2003

Abstract

In this paper a visual approach to querying in spatial databases is presented. A filter flow methodology is used to consistently express different types of queries in these systems. Filters are used to represent operations on the database and pictorial icons are used throughout the language for filters, operators and spatial relations. Different granularities of the relations are presented in a hierarchical fashion for spatial constraints. The language framework and functions are described and examples are used to demonstrate its capabilities in representing different levels of queries, including spatial joins and composite spatial joins. Here, the primary focus is on the query language itself but an overview of the implemented interface of the language is also provided.

Keywords: geographical information systems, spatial databases, spatial querying, visual query languages

1. Introduction

Large spatial databases, such as geographic information systems (GIS) and medical databases, are distinguished by their need to represent and allow for the manipulation of large numbers of spatial objects and relations. The spatial concepts in these systems are naturally visual, and it would therefore seem appropriate to adopt a visual approach to their development. GIS are a prime example of spatial systems and have a substantial number of application areas, including environmental, transportation and utility mapping. Their increasing popularity in recent years highlights the need to design more effective user interfaces for those systems with a high level of usability. Users of current GIS are not expected to be experts in the geographic domain and are possibly even casual users of database systems. Query languages enable users to interact with the GIS databases. There is, therefore, a need to design and develop alternative spatial database query languages.

In this research, a visual approach is adopted to the design and development of a query language for GIS, that addresses the identified issues and requirements for such languages. A filter flow methodology is utilized making it easier to use and learn than textual alternatives. Example queries are given throughout the paper to demonstrate the language.

The paper is structured as follows. Section 2 is a literature survey of spatial query language proposals and approaches. Section 3 lists some general requirements and problems identified for spatial query languages/interfaces. In Section 4 the language is defined and a detailed description of its framework and functions is given in Section 5. Some query examples are described in Section 6 and the language implementation is overviewed in Section 7. Finally, a summary of the main features of the language and concluding remarks are given in Section 8.

2. Literature survey

Query languages to GIS are either textual, non textual, or a combination of both. Currently, many database products support and have extensions to SQL, a standard command language that allows users to query a database.

2.1. Textual query languages

There have been two main approaches at standardizing the SQL framework in order to store and manage spatial data. They are SQL3 and OpenGIS SQL. SQL3 has facilities for the use of abstract data types (ADTs) in its specification [25]. The OpenGIS consortium (OpenGIS inc. [40]) also proposed a specification for incorporating ‘‘geo-spatial ADTs’’ into SQL92 [46]. Many of the features used in this specification were proposed earlier in the SpatialSQL specification [17]. There are some slight differences between SQL3 and OpenGIS SQL and these tend to be in the syntax of the languages [25]. Prior to the above specifications there were a number of propositions to spatially extend SQL, including [17], [25], [26], [43].

Spatial extensions to SQL inherit the same problems in textual query languages as traditional databases. Typing commands can be tiring and error prone [18], with a difficult syntax that is tedious to use and learn [19]. Users could spend ‘‘more time thinking about command tools to use than thinking about the task at hand’’ [18]. Kaushik and Rundensteiner [32] also state that spatial relationships are often thought of in terms of images that depict the spatial positions, but in the SQL style languages these would need to be translated into a non-spatial language.

The ultimate goal of textual querying is to be expressed in a natural language. This is not currently achievable. Mark and Gould [35] suggest that natural language would probably become a more prominent way for users to interact with a GIS. Aufare and Trepied [2] explain that though a natural language approach would appear to be better for the GIS user, query expressions can be ‘‘verbose and difficult’’, and there are also problems with unsolved ambiguities.

2.2. *Non-textual spatial querying languages*

A number of approaches for visual representation of query languages have been proposed. These make use of visual components such as forms, diagrams, icons and images [2].

The query-by-example (QBE) model [54] has been explored in several works. In its original format, querying is carried out using ''2-D skeleton tables,'' and users write examples of results that they require from the database in the grids. Yen and Scamell [52] reported that users were more satisfied and performed better using QBE than SQL. There have been a number of spatial extensions to the QBE model, QPE [9], PICQUERY [27] and PQBE [41] are examples of such extensions. Form-based extensions often do not release the user from having to perform complicated operations in expressing the queries. Complex queries usually need to be typed into a condition box that is similar to the WHERE clause of an SQL statement, and this can cause problems [52]. QBE style languages have poor representations of data model concepts and are therefore less suitable for spatial databases that have a number of visual entities [3].

''Visual languages are today being widely used as a means for reproducing the user's mental model of the data-manipulated content''[45]. A great deal of work is already being carried out to devise such languages for traditional and object-oriented databases in order to address problems in usability. Iconic, diagrammatic, graph-based and multi-modal approaches are noted. In the spatial domain these approaches have also been proposed along with sketch-based interfaces.

Lee and Chin [33] proposed an iconic language where icons are used to represent objects and processes. Queries are expressed by building up iconic sentences in the interface. Difficulties with this approach arise from the fact that objects in a query expression need to be explicitly specified along with their associated class and attributes, which renders the language cumbersome for the casual user. Also, only a limited number of spatial relations can be used in queries [19]. Another icon-based language has been designed by Sebillo et al. [45]. It is called the metaphor GIS query language (MGISQL). In this approach, users drag icons of themes known as ''geometaphors'' onto a ''sensitive'' (drawing) board. Geometaphors are then spatially arranged, and this specifies the spatial operator between them. MGISQL is useful for topological, directional and metric relations, but there is a lack of support for querying the non-spatial parts of the database in a GIS.

Jungert [29] proposed a query language named GRAQULA. Simple queries are carried out by the user selecting objects on the map, and executing functions between the objects. This language is mainly textual, and the graphical aspect of the language refers to the selection of the querying items. Another graph-based language is given in Traynor and Williams [48] and Traynor [49]. Users place panels together in a graphical diagram. Each of the panels can represent categories or themes. In this language, users are not released from having to use textual commands, and the ''And'' construct used has a layout that is opposite of logical and flowchart convention. There is also no ''Or'' function specified in the language and no descriptions of any spatial relationships that one would expect in a spatial query language.

Kaushik and Rundensteiner [31] proposed a direct manipulation approach named

SVIQUEL (spatial visual query and exploration language). Users can specify topological and directional spatial relations between region objects using filters called ''S-sliders''. This approach is limited in that it only allows users to query between two region objects. As it is, the interface is simple to use but there can be ambiguities and different interpretations for a query by the system and end-users [21]. In Kaushik and Rundensteiner [32] the system was extended to allow non-spatial data to be queried in a separate window from the spatial queries. Non-spatial queries are made using sliders which meant that only simple queries could be applied. Direct manipulation interfaces alone are intuitive for simple queries on a smaller and possibly fixed number of data sets, but with data intensive systems such as GIS they could easily become very complex.

Most sketch-based languages adopt a query-by-example approach where users sketch an example of the results that they would like displayed. In spatial databases these tend to be sketches of spatial configurations [19]. Sketch! [36] was one of the first sketch-based languages for spatial databases. Queries are constructed using a ''syntax directed editor''. The non-spatial parts of the database are queried using diagrams that are similar to the E-R model. Haarslev and Wessel [24] argue that although formal semantics are used in Sketch!, there are no mathematical foundations to the spatial relations. Cigales [7] is another example of a sketch-based query language for GIS. Users express queries by selecting features and operations and the system then induces drawings of queries. The main drawback of Cigales comes from the ''multiple interpretations and multiple visual representations of queries'', and also a lack of logical and negation operators [42]. LVIS [42] was defined as an extension of the Cigales language. The extension defined some new operators, attempted to resolve interpretation ambiguities and was integrated into a ''customisable visual environment''. However, the attempt at resolving ambiguities was limited [21].

In spatial-query-by-sketch [5], [19], users build up queries by drawing spatial configurations on a touch sensitive screen. Users can reduce the similarity ranking to lower the accuracy threshold for the result by relaxing spatial relations. This enables exact and similar matches to be browsed through. A pictorial query language (PQL) is proposed in Ferri et al. [20] for geographic features in an object-oriented environment. Users formulate queries by placing together configurations using ''symbolic features''. In Ferri et al. [21], a ''syntactic and semantic correctness'' method for feature configurations in the language is defined in order to reduce the ''multiple interpretations'' of the queries. In general sketch- and drawing-based approaches are suitable for expressing similarity-based queries to spatial databases and can become complex to use in a general context when composite queries are built. Users need to know what they require from the spatial database in these approaches and they do not benefit users who might only want to scan through the data sets [32]. Also, they either assume that users are able to sketch a query and express spatial relationships in a drawing or rely on different modalities for offering the user guidance in developing a sketch. Exact queries can be generally ambiguous due to the several possible interpretations of the visual configurations [42].

VISCO [24], [51] is a multi-modal query language ''for defining approximate spatial constellations of objects''. The basic language elements of VISCO use a metaphor that is based on the semantics of everyday physical objects. It uses a combination of an iconic library, command line operators as well as constructing queries by drawing in the query pane. A drawback of this language is that it can only query the spatial data in a GIS, and only simple thematic descriptors such as ''Lake'' or ''City'' can be typed onto the objects.

3. General requirements and identified problems

It is important that GIS provide easy and efficient access to the functions that can be used to manipulate spatial data [34]. Users of query interfaces need to enjoy using, feel comfortable with, and ''easily communicate their requests'' to the system [8]. In this section, a number of requirements are identified for the development of spatial query languages. Some of these issues can be addressed at the language design level, while others need to be addressed at the implementation level of the query interface. Issues arising due to the spatial nature of the database include the following,

3.1. Spatial database issues

1. *Representation of spatial objects:* Geographic objects have associated spatial representations to define their shape and size. Objects may be associated with more than one spatial representation in the database to handle different map scales or different application needs. Spatial representations of objects determine and limit the types of spatial relationships that they may be involved in. Explicit representation of the geometric type(s) of geographic features is needed to allow the user to express appropriate constraints over their locations.
2. *Spatial operations and joins:* It is difficult for a non-expert user to realize all the possible spatial operations that may be applied on a geographic object or the possible spatial relationships that may be applied on a geographic object or the possible spatial relationships that may be computed over sets of geographic objects. The semantics of the operations and relationships are implicit in their names. Those names may not have unique meanings for all users and are dependent on their implementation in the specific system in use. For example, an overlap relationship between two regions may be generalized to encompass the inside relationship in one implementation or may be specific to only mean partial coverage in another as shown in figure 1. In this research a visual, qualitative, representation of spatial operations and relationships is proposed to facilitate their direct recognition and correct use. Also, different granularities of spatial relationships need to be explicitly defined to express different levels of coarse and detailed spatial constraints. Bonhomme et al. [6] note that a number of spatial operators are not provided in query languages, such as directional operators. These are also proposed in this research.
3. *Composite spatial constraints:* Multiple spatial constraints are used in query

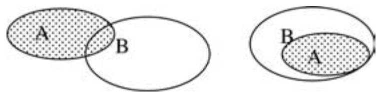


Figure 1. Types of overlap relationship between two spatial regions.

expressions. Again, the semantics of the composite relation may be vague, especially when combined using binary logical operators of AND and OR. Means of visualizing composite spatial relations would therefore be useful. For example, ''Object1 is north-of Object2 and close to it but outside a buffer of 10 m from Object3''.

4. *Integration of spatial and non-spatial data queries:* The handling of spatial and non-spatial data is non-uniform in querying languages, and they should be manipulated together to ''avoid the tiredness of a cognitive overload'' [6].
5. *Non-spatial operators:* With regards to spatial querying, some of the traditional database operators are not specified in the languages [6]. This can be particularly true in the sketch-based systems where everything in the drawings have to ''co-exist'', and separate sketches would be needed to express a disjunction (OR). Negation (NOT) operators are also not expressed in these types of language [16]. A good specification of spatial and traditional operators would therefore be useful in the new language.
6. *Schema visualization:* GIS are often schema poor; for instance, the majority of the object classes stored are spatial with no explicit relationships defined among them. Spatial relationships are usually derived by queries. Devising suitable visualization techniques for the schema will help users in relating locations of the data sets and their scales of representation.

3.2. General database issues

What follows are issues arising in general non-spatial databases but which would apply to spatial databases as well, and in particular the querying systems available for commercial GIS. In a study of three query languages, Greenblatt and Waxman [22] noted that a common error was the application of numeric operators to non-numeric fields. They suggested that this could be solved using more detailed column names, such as ''roadname'' instead of ''road''. In a graphical user interface another solution might be to only allow users to select items from menus or specify them using fixed type edit boxes.

In a review of previous experiments (for example, Greene et al. [23] and Michard [37]), Young and Shneiderman [53] note that a common error in textual query languages is that of using the logical AND operator as opposed to the logical OR operator when translating from English sentences. They also note that users found it difficult to specify precedence in query expressions due to ''parenthesis complexity'' [39]. Syntactical errors are also a problem in query languages [23], [38], [44], [50], [53]. In their evaluations, Young and Shneiderman [53] chose not to rate queries with syntax errors as incorrect, and Reisner

[44] categorized these types of errors as ''minor'' and placed them under the banner ''essentially correct'' queries.

Katzeff [30] found that users of query languages often divided queries into stages. They built complex queries by adding to an initial simpler query (sub-query). Visualizing results of sub-queries while building a query would therefore be useful. Murray [39] suggests that a query history mechanism be provided.

4. Language definition

The approach to the new language design and implementation in this research is visual and can be classed as multi-paradigmatic. It uses a combination of graphical data-flow, icons and direct manipulation. In the approach, query diagrams are constructed using filters, represented by icons, between data input and output elements. The flow of data is constrained by the filters in the diagrams. The approach is an extension and a considerable modification of an example of a filter flow metaphor proposed by Young and Shneiderman [53].

In Young and Shneiderman [53] a fixed interface with only one database table was used, and users could apply filters to restrain the data from the attributes. The approach used a metaphor of a stream of water passing through filters. As the channel of water (data) passed through the filters (attribute menus) it reduced in volume, and this was indicated by the thickness of the line. The binary logic operators of AND and OR were represented by divisions in the channels of the water stream. Figure 2 shows an example of the filter flow language. The system does not require the use of parentheses and the graphical diagrams

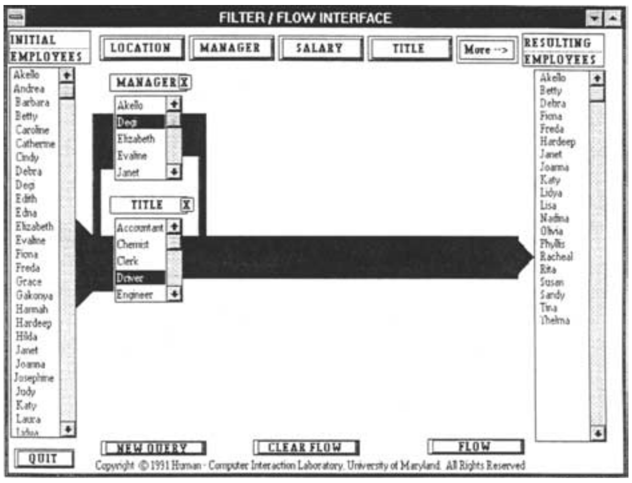


Figure 2. Graphical filter flow [53].

remove the confusion regarding precedence of operations. In a comparative experiment with SQL, Young and Shneiderman found that subjects preferred the filter flow approach, performed better and found it easier to use. The functionality of the system was very simplistic and it was only used with one table. There were no join operations or comprehensive means for handling query results.

Another significant influence on the research in this paper is the work of Murray [39]. He extended Young and Shneiderman's [53] approach to devise a flow-based language for querying object-oriented databases, in traditional business applications. The new approach in this thesis is designed for querying a spatial database. The data flow metaphor designed in this approach is simpler and more logical than that used in the above approaches. Non-spatial and spatial filters are represented visually as opposed to textually with icons that indicate the type of constraint used. Pictorial representations are also designed for expressing spatial relations and spatial objects. Operators are produced in the language for the manipulation of geometry in the display. A new metaphor is designed for the consistent visualization of both non-spatial and spatial joins and composite spatial joins can also be expressed in the language. This research focuses on deriving a query language for the vector data model, a model supported in most popular GIS [34], but the language could also be adapted for use with the raster data model. In what follows, the query language constructs are described in detail.

5. Language framework and functions

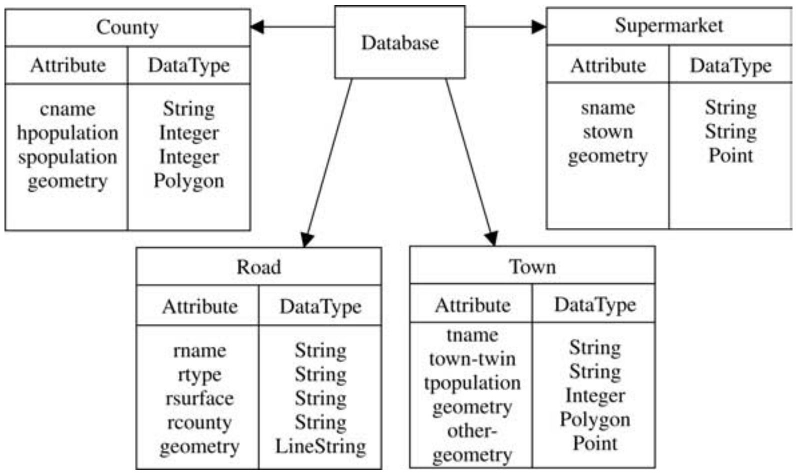
In this section the framework and functions designed for the new language are described.

5.1. Example database

Throughout this paper, the attributes and classes/tables of the example database of figure 3 are used. There are four classes, County, Town, Road and Supermarket. The data types of the attributes are also given in the figure alongside their names. At this point, the way in which the spatial database schema is depicted is not the primary concern. This is considered further in Section 7.

5.2. Object class representation

One of the general requirements noted in Section 3 was the representation of geographic objects that have associated spatial representations to define their shape and size. Objects may be associated with more than one spatial representation in the database to handle different map scales or different application needs. Spatial representations of objects also



County	
Attribute	DataType
cname	String
hpopulation	Integer
spopulation	Integer
geometry	Polygon

Supermarket	
Attribute	DataType
sname	String
stown	String
geometry	Point

Road	
Attribute	DataType
rname	String
rtype	String
rsurface	String
rcounty	String
geometry	LineString

Town	
Attribute	DataType
tname	String
town-twin	String
tpopulation	Integer
geometry	Polygon
other-geometry	Point

Figure 3. Example database.

determine and limit the types of spatial relationships that they may be involved in. There is therefore a need to explicitly represent the geometric types of geographic features to allow the user to express appropriate constraints over their locations.

In the language, object classes are depicted using a rectangular box containing the name of the class and an icon representing its spatial data type (see figure 4). Icons enable users to instantly recognize concepts and make the process of learning them easier, ''because people often remember visual concepts far better than verbal data'' [28]. The spatial data types that could be used include point, linestring, polygon, or any other composite spatial data type defined in the database. Icons offer the user initial knowledge of the spatial representation associated with the feature. A thick edge on the icon box is used if the object has more than one spatial representation in the database (see figure 4 for the town object). For example, the town object from the schema could be represented by a polygon to depict its actual shape and by a point for manipulation on smaller scale maps.

The object classes are used as inputs to the queries. A basic query skeleton is shown in figure 5. It consists of a data input and data output elements and a filter in between. Every input spatial object will have a related result object that can be displayed. The filters

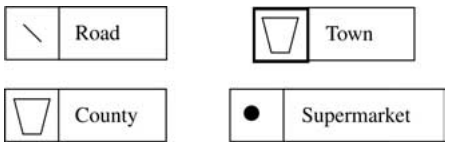


Figure 4. Visual representations of spatial objects

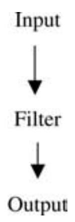


Figure 5. Basic query diagram.

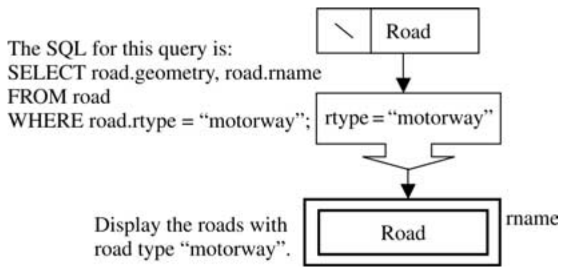
act on the instances flowing from the input element to restrict the data according to the defined criteria.

5.3. *Query results*

The results of the query (output element in figure 5) are depicted as shown in figure 6, by a double-edged rectangular box, with the class name written within. Any particular attributes that have been selected to appear in the results are listed on the right hand side, and if none of them are selected then they all pass through to the results. In the case of figure 6, the road names are selected. By default, the result of the query is displayed if the object has a spatial representation, thus eliminating the need to list the geometry attribute found in the spatial SQL SELECT clause.

The results box can be checked at any time during query formulation and its content displayed as a map and/or by listing the results in a table. A natural language English expression of the query producing the result box is also available for examination through the result box as shown in figure 6. The result box shown in figure 7 contains the results of a query where the road and county classes have been joined. More than one geometric type has been produced from the query. In this case roads and counties that satisfy the join condition will be displayed on the result map. A result box containing a join table is indicated with a ⊞ symbol on the top left hand side of the box.

One of the general requirements identified in Section 3 is the visualization of the results of sub-queries during the process of query formulation. This is achieved in the new



The SQL for this query is:
SELECT road.geometry, road.rname
FROM road
WHERE road.rtype = "motorway";

Figure 6. Result box in query.

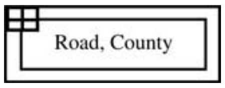


Figure 7. Join table result box.

language, where users can view the results in the result boxes and then build more complex queries by adding to an initial simpler query.

5.4. Filters and operators

This section describes the filter element of the query skeleton shown in figure 5. Filters or constraints in a query are made on the non-spatial (aspatial) properties of a feature as well as on the spatial properties and location of a feature. The use of non-spatial operators in a query language for spatial databases was noted as a requirement in Section 3. Another requirement was the integration of spatial and non-spatial data queries. In the language, both spatial and non-spatial data are manipulated together.

All the spatial operators proposed in the OpenGIS [40] SQL specifications, summarized in tables A.1, A.2, A.3 and A.4 (in the appendix), are provided in the new approach. The concept of filters between the input and output elements is first described, and then the spatial operators that can be applied to the results are explained.

5.4.1. Filters

i. *Non-spatial/spatial filters*
Figure 8 demonstrates a non-spatial filter depicted by an A (for attributes) symbol. It represents constraints over the stored attributes. The condition is written beside the icon as shown in the figure. Figure 9 demonstrates a spatial filter depicted by the co-ordinate symbol. It represents constraints that can be computed on the geometric properties of the object. Three possible operators are defined in the language: length, area and boundary. The length operator computes the length of a linestring, and the area and boundary operators are used with polygon geometry. In the OpenGIS

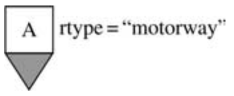


Figure 8. Non-spatial filter.

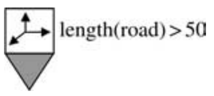


Figure 9. Spatial filter.

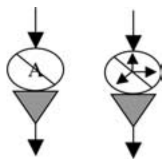


Figure 10. Negated constraints.

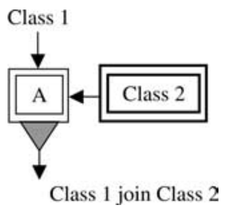


Figure 11. Non-spatial join.

specification [40], the boundary operator returns a geometry. Here, the boundary operator computes the size of the circumference of the polygon object. Other operators could be defined in a similar fashion. For example, an envelope operator that returns a minimum bounding rectangle of a geometry could be used; Envelope (town). Composite operators could then be written; for example, Area(Envelope(town)) $>$ 15 selects all objects where the area of the envelope is greater than 15 miles squared. Negated constraints (NOT) are depicted by the filters shown in figure 10.

ii. *Join filters*

Two kinds of join operators are possible in spatial databases namely, non-spatial joins and spatial joins. Both types are represented coherently in the language. Figure 11 demonstrates a non-spatial join filter. It is also depicted by an A symbol that represents a join between stored attributes of two classes. A result box is associated with every joined class and linked to the join filter. In the figure, class 1 flows to the filter and is joined by class 2 using a common attribute of both.

Spatial joins are expressions of spatial relationships between spatial objects in the database. The visual representation of spatial relationships was noted in Section 3 as a requirement in a spatial database querying language. Spatial relationships may be classified as topological, directional or proximal.

Topological spatial joins. Figure 12(a) demonstrates the general filter for the spatial join. On specifying the type of spatial relationship the icon in the filter is replaced. In figure 12(b) a topological relationship has been selected and the icon demonstrates the relationship between linestring and polygon geometries of classes 1 and 2. A choice of possible topological relations are available depending on the spatial data types of the

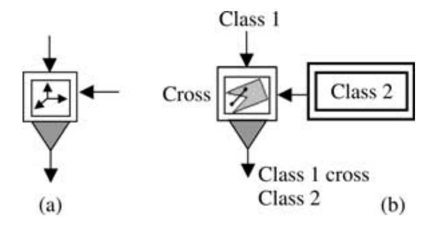


Figure 12. (a) Spatial join filter, (b) topological spatial join.

objects joined. These are specified in the OpenGIS [40] SQL specification and are now explained.

Topological relations
The visual representation of spatial relations was noted as a requirement for a spatial database querying language in Section 3. This section explains the topological relations and different levels of granularity of relations in the new language. These are intended to allow users to express different levels of granularity of spatial constraints.

Defined relationships
The OpenGIS [40] SQL specification used the calculus based method [10]–[12] to describe topological relations. In the method, a set of relations were named and defined for the DE + 9IM model, which was a combination of the ''Dimension extended method'' [10] and the ''9-intersection method'' [15] for describing spatial relationships. Five general relationships were defined in Clementini and DiFelice [11]: Touch, In, Cross, Overlap and Disjoint that could express all of the spatial configurations of the DE + 9IM model (table 1).

Topological icons
In the spatial join filter shown in figure 12, it was explained that the icon in the filter was replaced with an icon depicting the relation that is used. From an understanding of the studies on spatial relations described above, different topological icons have been designed for the new language. They depict relationships between simple geometries and are shown in figures 13–18.

Table 1. Spatial relations [11].

Relation	*Description*
Touch	The Touch relationship applies to area/area, line/line, line/area, point/area, point/line situations.
In	The In relationship applies to every situation.
Cross	The Cross relationship only applies to line/line and line/area situations.
Overlap	The Overlap relationship applies to area/area and line/line situations.
Disjoint	The Disjoint relationship applies to all situations.

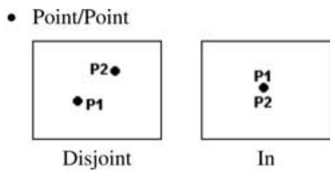


Figure 13. Point/point spatial relationships.

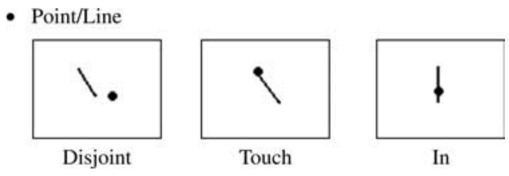


Figure 14. Point/line spatial relationships.

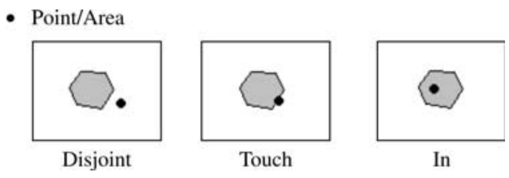


Figure 15. Point/area spatial relationships.

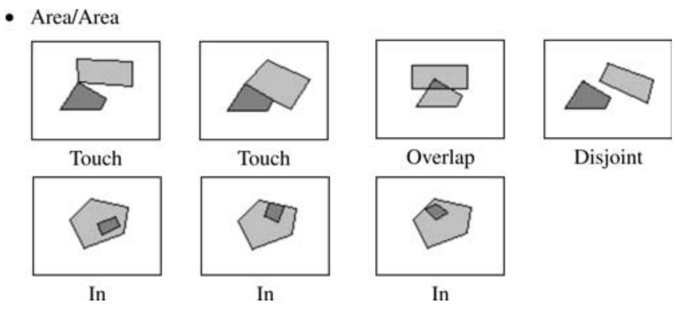


Figure 16. Area/area spatial relationships.

Note that there are two icons for the Touch relation and three icons for the In relation. These indicate the different levels of granularity in the relations and are defined in the language so that users can express detailed spatial constraints in their queries. Different icons for different levels of relation are also defined between Line/Area and Line/Line geometries.

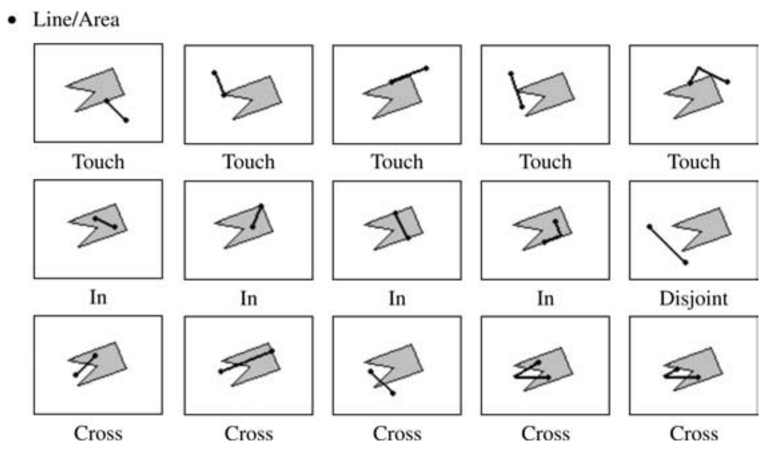


Figure 17. Line/area spatial relationships.

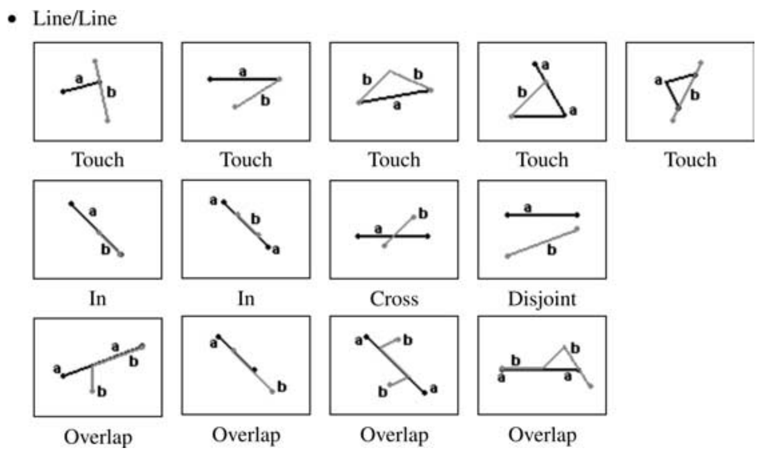


Figure 18. Line/line spatial relationships.

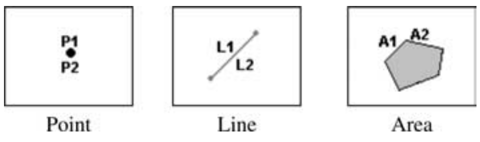


Figure 19. Spatially equal geometry.

The OpenGIS [40] SQL specification also defines the equals relation for geometries that are ‘‘spatially equal’’. Figure 19 illustrates the icons that have been designed in the new language for this relation.

The Contains and Intersect relations were defined in OpenGIS [40] for user convenience. Contains has the same representations as the In (Within) relations and Intersect has opposite representations from the Disjoint relations.

Directional spatial joins. In figures 20 (a), (b), (c) and (d) the general filter has been replaced by directional spatial join filters. In 20(e), the icon depicts the North of relationship. It finds the geometries of class 1 that are North of the geometries of class 2. The South, East and West relations are also available.

Proximal spatial joins. In figure 21 an icon for a proximal relationship has replaced the general filter of figure 12(a). Qualitative proximal relationships, such as Near and Far are vague unless they clearly have a pre-defined range of measures. Hence, using proximal relationships requires an indication of the measure of proximity required, for example

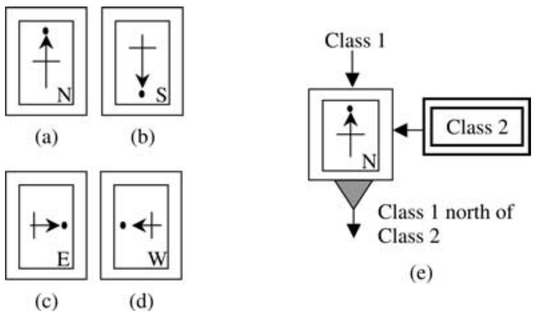


Figure 20. Directional spatial joins.

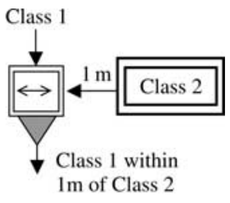


Figure 21. Proximal spatial join.

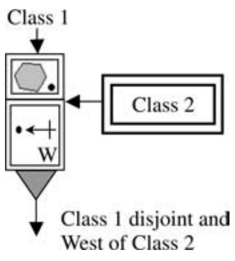


Figure 22. Composite join.

within a distance of *x* miles. In figure 21, the filter finds the geometries of class 1 that are within a distance of 1 mile of class 2. The distance operator was defined in the OpenGIS specification and is summarized in Chapter 2.

Composite joins. Multiple spatial joins may be expressed in the language by combining join filters. The use of composite spatial constraints was noted as a requirement in Section 3. In figure 22, the filter finds the geometries of class 1 that are disjoint and West of class 2.

5.4.2. Geometric operators. There are a number of operators defined in the OpenGIS SQL specification that can result in geometry to be displayed on a map or data that can be written in a table of results. These operators are defined in the basic functions, spatial analysis, point, linestring and polygon tables in the Appendix (tables A.1, A.2, A.3 and A.4). Some of the new language visual representations for the operators defined in the tables have already been described in Section 5.4.1. This section will describe the new languages visual representations of the remaining operators.

Display operators. All of the operators defined in table 2 can result in the selection and display of geometry on a map when applied. In the flow diagrams, they are placed beneath the result box as shown in figure 23.

Tabular operators. All of the operators defined in table 3 can result in data that is written to a table of results. In the flow diagrams, the operators are placed on the right or left of the result box as shown in figure 24. The double arrow indicates the flow of geometry into the operator and back into the result box. Some of the operators are very specialized and might not be utilized by normal users of GIS. They are, however, defined in the language and

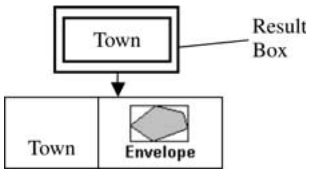
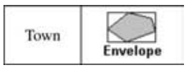
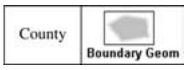
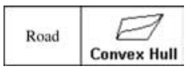
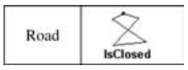
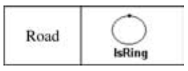
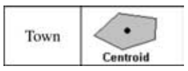
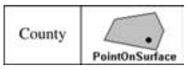
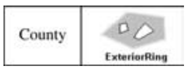
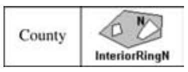
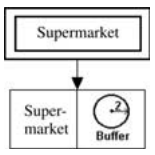


Figure 23. Envelope operator.

Table 2. Result display operators.

Operator	*Example Description*
	This operator will result in the display of the minimum bounding rectangle of the town geometry.
	This shows the boundary geometry operator that in this case will display the boundary of a county.
	This demonstrates the convex hull operator that in this case will display the convex hull of the road geometry.
	This demonstrates the IsClosed operator for linestring geometry. It assesses whether a curve is closed (start point = end point). In this case it is applied to the road class and any roads that meet the constraint are displayed.
	This demonstrates the IsRing operator for linestring geometry. It assesses whether a curve is closed (start point = end point) and simple (it does not pass through the same point more than once). In this case it is applied to the road class and any roads that meet the constraint are displayed.
	This demonstrates the Centroid operator for polygon geometry. It finds a point that is the mean centre value of the polygon. In this case, it finds the Centroid of the towns in the town class and displays the points.
	This demonstrates the PointOnSurface operator for polygon geometry. It finds a point that is guaranteed to be on the surface (the surface is the input and the operation returns a point). In this case, it finds and displays a point on the counties of the county class.
	This demonstrates the ExteriorRing operator for polygon geometry. It finds the exterior ring of the polygon. In this case, it finds and displays the exterior rings of counties in the county class.
	This demonstrates the InteriorRingN operator for polygon geometry. It finds the *N*th interior ring of the polygon. The *N* is an integer defined by the user. In this case, the *N*th interior ring of the county geometry is displayed.
	This demonstrates the Buffer operator. It finds the geometries that are less than or equal to the distance of another geometry. In this case, the supermarkets that are within a buffer of 2 miles of the supermarkets in the result box are displayed. (if ASDA supermarkets have been selected, then the operator will display all supermarkets within 2 miles of the ASDAs)

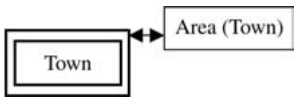


Figure 24. Area tabular operator.

Table 3. Result table operators.

Operator	*Description*
Length (Road)	Length operator for linestring geometry: It has also been described in the spatial filter section of 5.4.1. In this case, it lists the lengths of the road geometries.
Area (Town)	Area operator for polygon geometry: It has also been described in the spatial filter section of 5.4.1. In this case, it lists the areas of the town geometries.
Distance (Town, Supermarket)	Distance operator: It has also been described in the spatial join section (in 5.4.1). In this case, it lists the shortest distances between the geometries of the town and supermarket classes.
IsEmpty (Road)	IsEmpty operator: In this case, the operator will indicate if the set of road geometry is empty or not.
IsSimple (Road)	IsSimple operator: In this case, the results will indicate those roads that are simple (the linestring geometry does not pass through the same point more than once).
(a) *X* (Supermarket) (b) *Y* (Supermarket)	*X* and *Y* operators for linestring geometry: If the operator in (a) is used, it will list the *x* co-ordinate values for the supermarket geometries, and if (b) is used it will list the y co-ordinate values of the supermarket geometries.
(a) StartPoint (Road) (b) EndPoint (Road)	StartPoint and EndPoint operators for linestring geometry: If the operator in (a) is used, then the start point (*x* and *y*) of the roads in the road class will be listed, and if (b) is used then the end point of the roads in the road class will be listed.
NumPoints (Road)	NumPoints operator for linestring geometry: It finds the number of points in a linestring. In this case, it will find the number of points in the road geometries.
PointN (Road, 3)	PointN operator for linestring geometry: It will find a point (*x* and *y*) that the user specifies (*N*: integer) in a linestring. In this case, it finds the third point of the road geometries.
NumInteriorRing (Country)	NumInteriorRing operator for polygon geometry: It finds the number of interior rings in the polygon. In this case, it uses the county geometries.
AsBinary (Road)	Export operator: It is used to convert the geometry into either a textual or binary representation. For textual representations, the AsText( ) operator should be used, and for binary representations the AsBinary( ) operator should be used. In this case, the road geometries are converted into a binary representation.
SpatialReference (Town)	SpatialReference operator: In this case, it finds the spatial reference IDs for the town geometries.

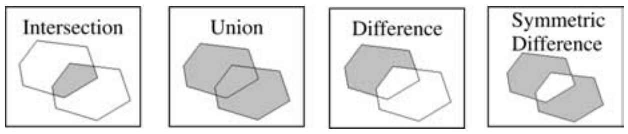


Figure 25. Geometric set operation filters.

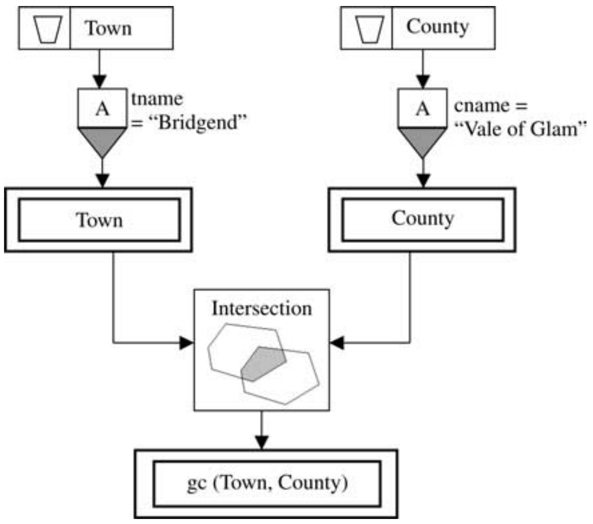


Figure 26. Example using intersection geometric set operator.

may be used for spatial analysis by an expert user or to form the basis for composing more meaningful queries to the system.

Geometric set operators. These functions are similar to the set operators of traditional SQL, but here they execute set-theory and constructive geometry operations on geometry values [40]. The filters that have been described for the new language are shown in figure 25. They are Intersection, Union, Difference and Symmetric Difference.

An example of a geometric set operation is shown in figure 26. The result boxes from each query are shown flowing into the sides of the intersection filter. Following this, a new result box is made that will contain the geometry of a polygon made from both classes. This is indicated in the result box using gc.

5.5. *Dataflow*

One of the general issues noted in Section 3 was the common errors that are often made in using logical AND and OR operators when translating from an English sentence. In the query language, the arrangement of the filters in the flow diagrams denote the logical

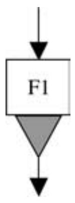


Figure 27. Single filter.

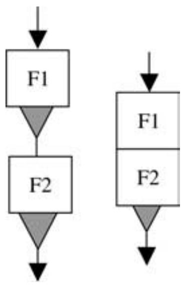


Figure 28. Filters joined by AND.

operators of AND and OR. This eliminates the need for using textual operators. Parentheses are also not needed with this approach due to the fact that the filter arrangement determines the precedence between the constraints of a query. This was also noted as an issue that needed to be addressed in Section 3. This section will describe the different data flow layouts that can be used in the language.

A single filter is shown in figure 27. The data flows through only when the filters criteria is satisfied. In figure 28, two filters are shown in series, representing filters joined by an AND. The data flows through only when the criteria in both are satisfied. The logical operation between the filters is: F1 AND F2. A concise representation of the filters is shown on the right of figure 28. In figure 29, a parallel arrangement of filters is used to indicate that the data flows through when the criteria in either or both filters are satisfied. This represents filters that are joined by an OR. The logical operation between the filters is:

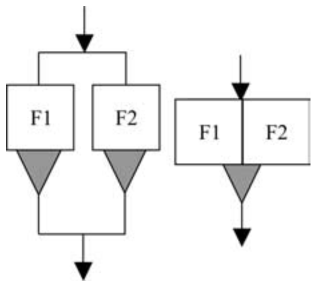


Figure 29. Filters joined by OR.

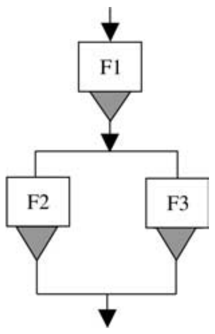


Figure 30. Filter configuration: (F1 AND (F2 OR F3)).

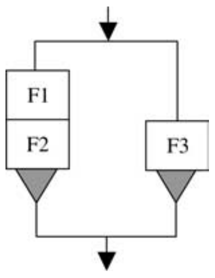


Figure 31. Filter configuration: ((F1 AND F2) OR F3).

F1 OR F2. Again, a concise representation of the filters is shown on the right of figure 29.

Any number of filters may be joined together by the representations of logical operators as shown in figures 30, 31 and 32. In figure 30 the logical operations between the filters are: (F1 AND (F2 OR F3)). For the filters in figure 31 the logical operations are: ((F1 AND F2) OR F3). Finally, in figure 32 the logical operations between the filters are: (F1 AND (F2 OR F3 OR (F4 AND F5)) AND F6).

6. Query examples

In this section, a number of example queries are demonstrated in the filter flow language. The SpatialSQL expressions are also given along with the queries.

6.1. Foundational queries

The example in figure 33 demonstrates a basic query that uses a non-spatial filter. In this case, it finds and displays the counties with a sheep population greater than 150,000, and

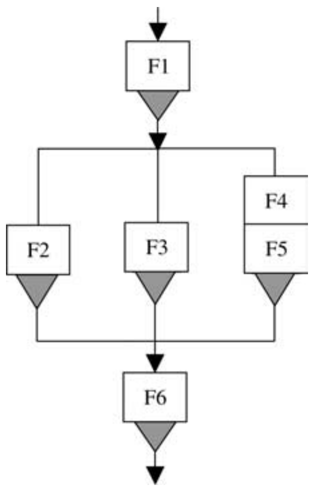
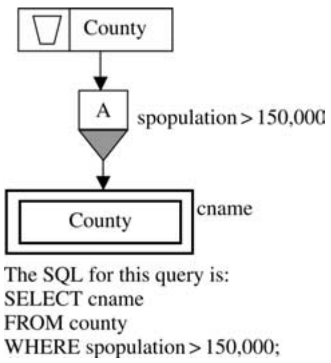


Figure 32. Filter configuration: (F1 AND (F2 OR F3 OR (F4 AND F5)) AND F6).

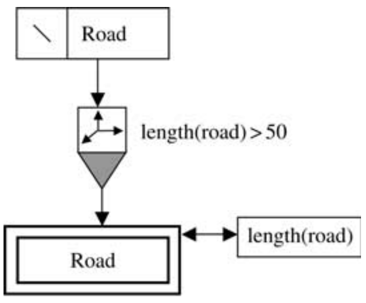
selects their names. Other comparison operators could be applied such as equals or less than. Also, spatial (unary) operators may be used to filter the results, for example area, length, and perimeter/boundary. An example of using a spatial filter is shown in figure 34. This query allows only roads with a length greater than 50 miles to pass through to the results. The lengths of the roads are also selected.

In figure 35, three filters are used and are joined together using the dataflow representations of the Boolean logic operators. One of the filters in the query is a negated filter. In this query, towns that either have a population greater than 20,000 with a twin town that is not named ‘‘Esslingen’’, or towns with an area greater than 15 miles are selected. The envelope operator is also used in the query to place a minimum bounding rectangle around the geometry of towns in the result box.



```
The SQL for this query is:
SELECT cname
FROM county
WHERE spopulation > 150,000;
```

Figure 33. A non-spatial filter in a simple query.



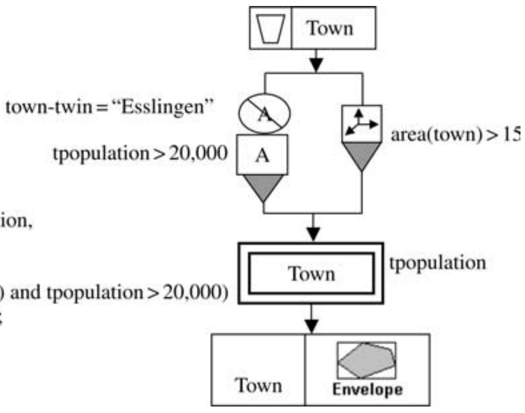
The SQL for this query is:
```
SELECT length(road.geometry)
FROM road
WHERE length(road.geometry) > 50;
```

Figure 34. A spatial filter in a simple query.

6.2. *Joins*

Two kinds of join operations are possible in spatial databases namely, non-spatial joins and spatial joins. Both types are represented coherently in the language. Spatial joins are expressions of spatial relationships between spatial objects in the database. Examples of spatial join queries are: Display all the motorway objects crossing Mid Glamorgan, and Display all the towns north of Cardiff within South Glamorgan. An example of a non-spatial join is given in figure 36. It selects all of the towns that have an ASDA or Tesco supermarket. The names of the town and supermarkets are also selected.

Note that the result box from the join operation has been modified to reflect the contents of the join table. In figure 37, a spatial join query is given. The query finds all motorway roads that cross counties with a human population greater than 50,000. A symbol of the spatial relationship sought is used to replace the general ''co-ordinate'' symbol in the



The SQL for this query is:
```
SELECT town.geometry, town.tpopulation,
        Envelope(town.geometry)
FROM town
WHERE (not (town-twin = "Esslingen") and tpopulation > 20,000)
        or (area(town.geometry) > 15);
```

Figure 35. A query that uses and, or and not.

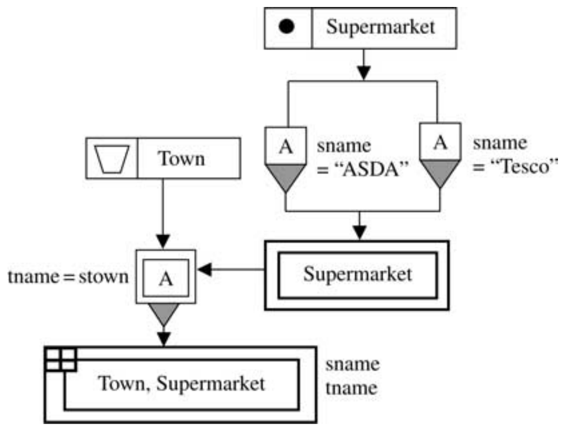
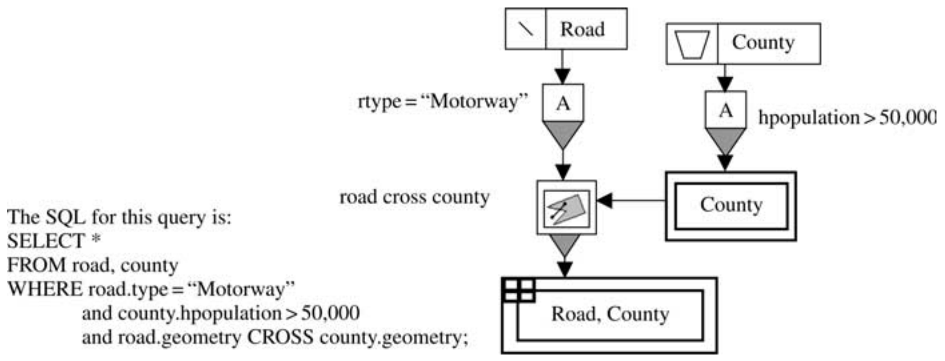


Figure 36. Example query of a non-spatial join.

spatial join filter. A choice of possible spatial joins is available depending on the spatial data types of the objects joined. These are explained in Section 5.4.1. In figure 37, all the possible relationships between line (for roads) and polygons (for counties) will be available.

6.3. *Composite joins*

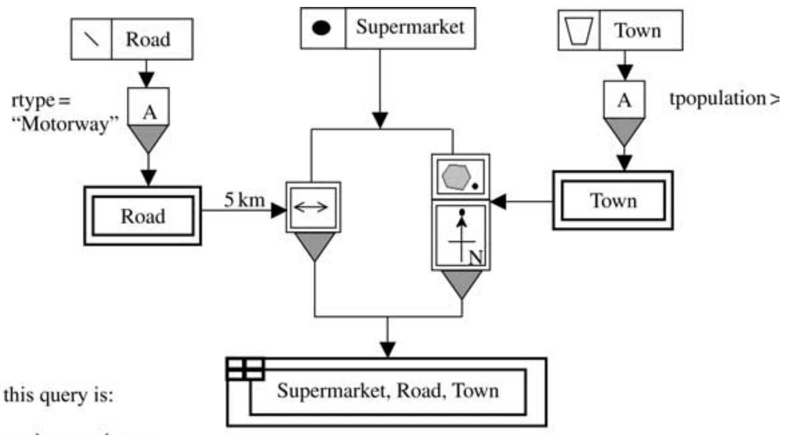
Multiple spatial joins may be expressed similarly either with the same object type, for example to find the supermarkets outside and north of towns, or with more than one object type, for example to find the supermarkets that are either outside and north of towns that have a population greater than 10,000 or are within a buffer of 5 km from motorways as shown in figure 38.



The SQL for this query is:

```
SELECT *
FROM road, county
WHERE road.type = "Motorway"
          and county.hpopulation > 50,000
          and road.geometry CROSS county.geometry;
```

Figure 37. Example query of a spatial join. Specific relationship icon replaces general spatial join to indicate the cross relationship.



```
The SQL for this query is:
SELECT *
FROM supermarket, road, town
WHERE (road.type = "Motorway" and distance(supermarket.geometry, road.geometry) < 5) or
        (town.tpopulation > 10000 and direction(supermarket.geometry, town.geometry) = North and
        supermarket.geometry DISJOINT town.geometry);
```

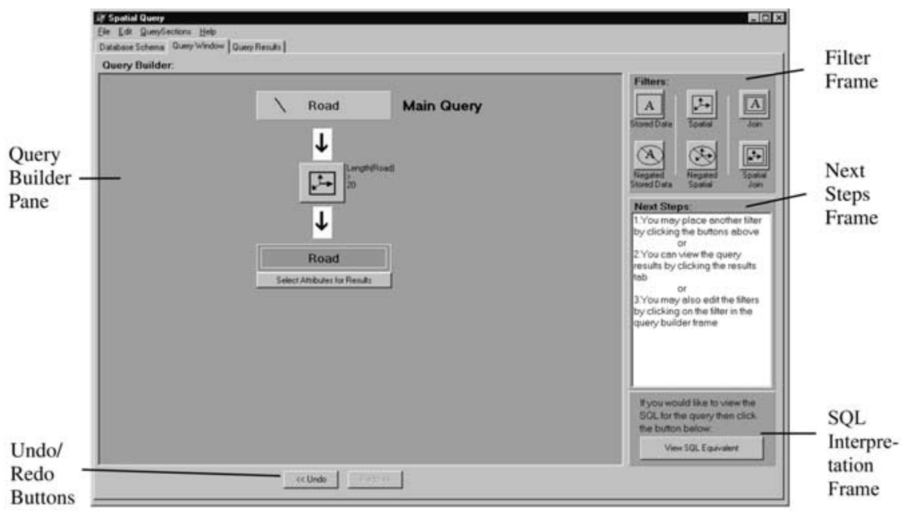
Figure 38. Composite query. Find the supermarkets that are within a buffer of 5 km of a motorway or are outside and north of a town whose population is greater than 10,000.

7. Implementation

This section will briefly explain the language interface. Full details of the language implementation are a subject of a different report. In implementing the language, a number of issues identified in Section 3 are addressed including: error prevention, guided querying, feedback, query history and dynamic result visualization.

The main part of the interface is the query formulation window where users build up the queries. A result window is also implemented to present the results of the queries in the form of maps and tables. Multiple windows are placed into pages in a tabbed notebook format. The query formulation window is accessed by clicking the Query Window tab on the interface notebook. A window with four main frames is then displayed. The frames are: a query builder pane, the filter options, message guiding to the following steps and an SQL interpretation of the formed query. Error recovery is facilitated by undo and redo buttons shown at the bottom of the page. Figure 39 shows an example of this window with pointers to the above items.

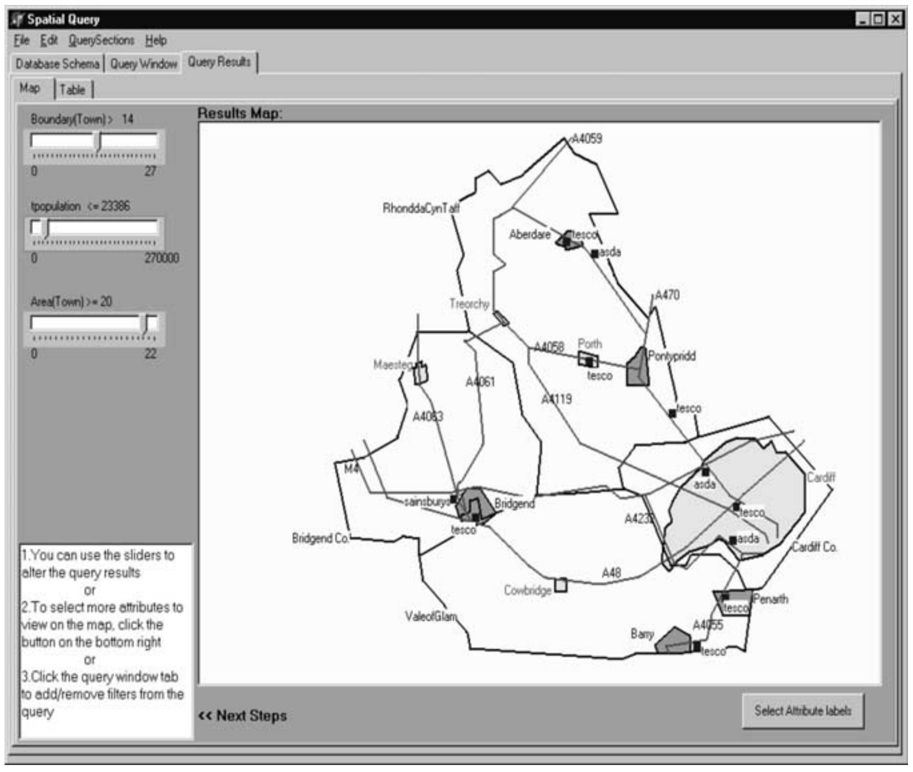
Users build up filters in a consistent guided manner, and direct manipulation is used for the prevention of errors. The system decides all of the possible filters and choices that may be utilized by the user at the various stages of query formulation. Six filters are available to the users. These can be placed into the filter flow diagrams on the query frame. The filters available are non-spatial (stored data), spatial, negated non-spatial, negated spatial, and spatial and non-spatial joins. Join filters are distinguished by a double border around their icon, and three types of spatial join are available in the interface that are based on the direction, proximity or connectivity between the objects geometries.



1.You may place another filter by clicking the buttons above
or
2.You can view the query results by clicking the results tab
or
3.You may also edit the filters by clicking on the filter in the query builder frame

If you would like to view the SQL for the query then click the button below:

Figure 39. Query formulation page.



1.You can use the sliders to alter the query results
or
2.To select more attributes to view on the map, click the button on the bottom right
or
3.Click the query window tab to add/remove filters from the query

Figure 40. Results map.

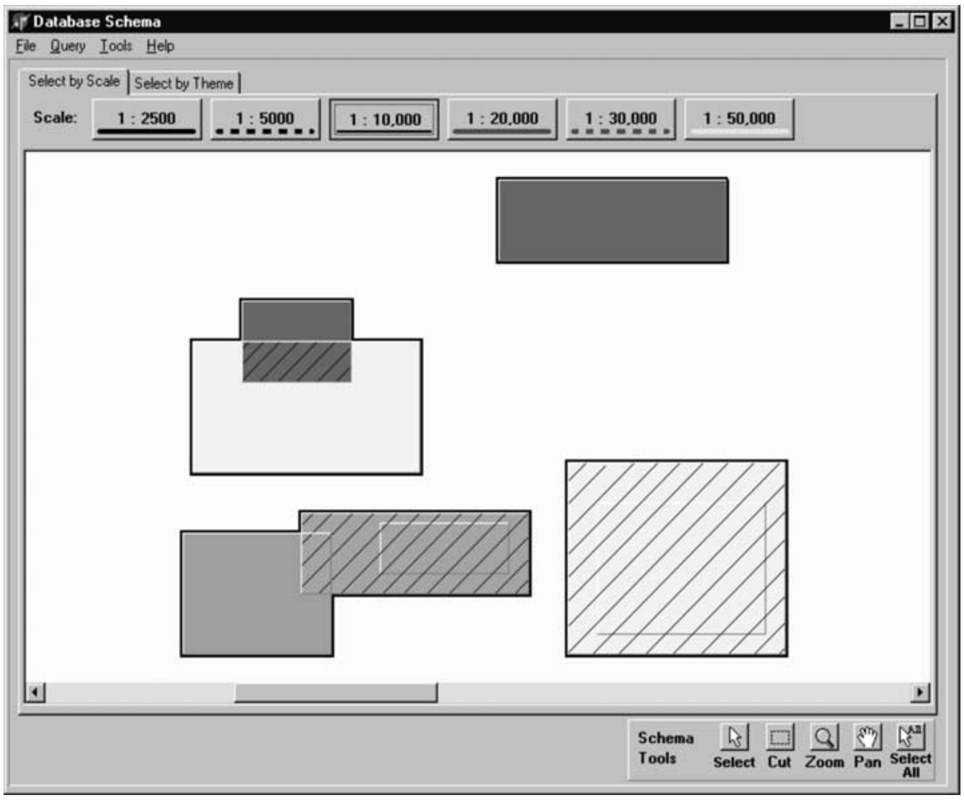


Figure 41. Database schema window using Select by Scale.

When a query is complete in the formulation window, queries are translated and processed by clicking the Query Results tab. The results are then displayed on a map, as shown in figure 40. The results can also be viewed in tabular form. The sliders shown on the left of the window (figure 40) allow users to dynamically adjust the queries and instantly view the results in the map or table.

The schema window shown in figure 41 has only been partially implemented and is currently under construction. The new schema allows users to select the database tables by themes (roads, towns . . .) or source data scale. The prototype implementation differs slightly from what is shown in the figure. In the figure, a minimum-bounding rectangle (MBR) is used to illustrate the spatial extent of the different map layers. Upon restricting the scale of viewing, only those data sets or themes with that source data scale are shown. A flashing boundary line indicates the existence of more than one source data scale for the theme viewed. Also the MBR are color coded to distinguish the different types of map themes. When two map themes overlap, the boxes or parts of the boxes are shaded to indicate the existence of more than one map theme in the location. The approach allows users to have an overview of the database contents, and enables them to make efficient decisions on data sets for querying. In the complete implementation of the schema, a link will be provided to an Input Data panel in the query formulation window. Users could then drag an icon of the selected data set into the input box of the flow diagram.

8. Conclusion

In this paper, a visual approach to querying spatial databases was explained. Examples from the GIS domain have been used throughout to demonstrate the expressiveness of the language. The design of the language itself tried to address several of the requirements and problems associated with query languages to GIS. The following is a summary of the design aspects.

- Icons are used to represent the geographic features with explicit indication of their underlying spatial representation, and thus offering the user a direct indication to the data type being manipulated.
- Visual representations of spatial operators and relationships are designed so that the user could realize all of the possible operations that were available. Composite spatial relations were also included.
- A data flow metaphor is used consistently with different types of query filters namely, non-spatial and spatial filters as well as negated filters and spatial and non-spatial joins.
- The consistent use of the metaphor is intended to simplify the learning process for the user and should make the process of expressing and reading queries easier.
- The query history is retained, and complex queries can be built by adding to initial simpler queries.
- Sub-queries and complex queries are built consistently.
- Parenthesis complexity and common errors in the use of Boolean logical operators (AND and OR) have been addressed by utilizing a series and parallel flow metaphor that allows users to understand order of precedence and recognize the difference between AND and OR.

The approach is aimed at casual and non-expert users, or at expert domain users who are not familiar with query languages to databases. The implementation of the language aims to cater for different levels of user expertise. One of the intentions of the filter flow language was to design a comprehensive language that provides much of the functionality of a spatially extended version of SQL. In the interface, visual queries are parsed and translated to spatially extended SQL queries.

The proposed language design takes into account all of the specifications for a standard

SQL that supports ‘‘geo-spatial feature collections’’ [40]. It can therefore be considered complete as a spatial query language.

Appendix

This section describes the set of operators that have been specified by the OpenGIS consortium [40] for relations between geometries and operations on sets of geometry. The OpenGIS [40] have specified a number of operators for the geometry class. These are summarised in table A.1, compiled by Shekar et al. [46] and Clementini and DiFelice [13].

A spatial relationship that is not defined in the OpenGIS specification [40] but has been defined in a number of papers [1], [12], [14], [19] is a directional relation. It describes the location of an object with respect to the location of another object (North of, East of. . .).

In the OpenGIS specification there are also methods defined for operations on the particular geometry types. These are shown in tables A2, A3 and A4. Note, some of the operators have an (O) in their descriptions. This indicates that the operators result in information on an object level rather that a geometric level.

Table A.1. OpenGIS [40] geometry operators [13], [46].

Basic functions	SpatialReference()	Returns the Reference System of the geometry
	Envelope()	The minimum bounding rectangle of the geometry
	Export()	Convert the geometry into a different representation
	IsEmpty()	Tests if the geometry is a empty set or not
	IsSimple()	Returns True if the geometry is simple
	Boundary()	Returns the boundary of the geometry
Topological operators	Equal	Tests if the geometries are spatially equal
	Disjoint	Tests if the geometries are disjoint
	Intersect	Tests if the geometries intersect
	Touch	Tests if the geometries touch each other
	Cross	Tests if the geometries cross each other
	Within	Tests if the given geometry is within another given geometry
	Contains	Tests if the given geometry contains another given geometry
	Overlap	Tests if the geometry overlaps another geometry
Spatial analysis	Distance	Returns the shortest distance between two geometries
	Buffer	Returns a geometry that represents all points whose distance from the given geometry is less than or equal to the specified distance
	ConvexHull	Returns the convex hull of the geometry
	Intersection	Returns the intersection of two geometries
	Union	Returns the union of two geometries
	Difference	Returns the difference of two geometries
	SymDiff	Returns the symmetric difference of two geometries

Table A.2. Point geometry operators [40].

For Point geometry, the following operators are defined:	
X( )	The *x* co-ordinate value for this Point (O)
Y( )	The *y* co-ordinate value for this Point (O)

Table A.3. LineString geometry operators [40].

For LineString geometry, the following operators are defined:	
Length( )	The length of this Curve in its associated spatial reference
StartPoint( )	The start point of this Curve (O)
EndPoint( )	The end point of this Curve (O)
IsClosed( )	Returns 1 (TRUE) if this Curve is closed (StartPoint ( ) = EndPoint ( ))
IsRing( )	Returns 1 (TRUE) if this Curve is closed (StartPoint ( ) = EndPoint ( )) and this Curve is simple (does not pass through the same point more than once)
NumPoints( )	The number of points in this LineString (O)
PointN( )	Returns the specified point *N* in this LineString (O)

Table A.4. Polygon geometry operators [40].

For Polygon geometry, the following operators are defined:	
Area( )	The area of this Surface
Centroid( )	The mathematical centroid for this Surface as a Point.
PointOnSurface( )	A point guaranteed to be on this Surface (O)
ExteriorRing( )	Returns the exterior ring of this Polygon
NumInteriorRing( )	Returns the number of interior rings in this Polygon
InteriorRingN( )	Returns the Nth interior ring for this Polygon as a LineString

References

1. A.I. Abdelmoty and B.A. El-Geresy. An Intersection-based Formalism for Representing Orientation Relations in a Geographic Database in Second ACM Workshop On Advances In Geographic Information Systems. ACM Press: Gaithersburg, MD, USA, 44–51, 1994.
2. M.-A. Aufaure-Portier and C. Trépied. ‘‘What approach for searching spatial information?’’ *Journal of Visual Languages and Computing*, Vol. 12(4):351–373, 2001.
3. M.-A. Aufaure-Portier and C. Trépied. ‘‘A survey of query languages for geographic information systems,’’ in *Interfaces to Databases (IDS-3)*, Edinburgh, U.K. Springer, 1996.
4. F. Benzi, D. Maio, and S. Rizzi. ‘‘VISIONARY: A viewpoint-based visual language for querying relational databases,’’ *Journal of Visual Languages and Computing*, Vol. 10(2):117–145, 1999.
5. A. Blaser and M. Egenhofer. ‘‘A visual tool for querying geographic databases,’’ in AVI 2000. ACM Press: Palermo, Italy, 211–216, 2000.
6. C. Bonhomme, C. Trépied, M.-A. Aufaure-Portier, and R. Laurini. ‘‘A visual language for querying spatio-temporal databases,’’ in *7th ACM International Symposium on Advances in Geographic Information Systems*, ACM Press, 34–39, 1999.
7. D. Calcinelli and M. Mainguenaud. ‘‘Cigales. A visual query language for a geographical information system: The user interface,’’ *Journal of Visual Languages and Computing*, Vol. 5(2):113–132, 1994.

8. T. Catarci, M.F. Costabile, S. Levialdi, and C. Batini. ''Visual query systems for databases: A survey,'' *Journal of Visual Languages and Computing*, Vol. 8(2):215–260, 1997.
9. N.S. Chang and K.S. Fu. ''Query-by-pictorial example,'' *IEEE Transactions on Software Engineering*, Vol. 6(6): 519–24, 1980.
10. E. Clementini, P. Di-Felice, and P.V. Oosterom. ''A small set of formal topological relationships suitable for end-user interaction,'' in *Advances in Spatial Databases—Third International Symposium, SSD'93*, Singapore: Springer, 277–295, 1993.
11. E. Clementini and P. Di-Felice. ''A comparison of methods for representing topological relationships,'' *Information Sciences*, Vol. 3:149–178, 1995.
12. E. Clementini and P. Di-Felice. ''A model for representing topological relationships between complex geometric features in spatial databases,'' *Information Sciences*, Vol. 90(1–4):121–136, 1996.
13. E. Clementini and P. Di-Felice. ''Spatial operators,'' *ACM SIGMOD Record*, Vol. 29(3):31–38, 2000.
14. M.J. Egenhofer. ''Extending SQL for graphical display,'' *Cartography and Geographic Information Systems*, Vol. 18(4):230–245, 1991.
15. M. Egenhofer and J. Herring. ''Categorizing binary topological relations between regions, lines, and points in geographic databases,'' *Technical Report*, Department of Surveying Engineering: University of Maine, 1991.
16. M.J. Egenhofer and J.R. Herring. ''Querying a geographical information system,'' in D. Medyckyj-Scott and H.M. Hearnshaw (Eds), *Human Factors in Geographical Information Systems*, Belhaven Press: London, 124–135, 1993.
17. M.J. Egenhofer. ''Spatial SQL: A query and presentation language,'' *IEEE Transactions on Knowledge and Data Engineering*, Vol. 6(1):86–95, 1994.
18. M. Egenhofer and T. Bruns. ''Visual map algebra: A direct-manipulation user interface for GIS,'' in *Third IFIP 2.6 Working Conference on Visual Database Systems*, Chapman and Hall: Lausanne, Switzerland, 235–253, 1995.
19. M. Egenhofer. ''Query processing in spatial query by sketch,'' *Journal of Visual Languages and Computing*, Vol. 8(4):403–424, 1997.
20. F. Ferri, F. Massari, and M. Rafanelli. ''A pictorial query language for geographic features in an object-oriented environment,'' *Journal of Visual Languages and Computing*, Vol. 10(6):641–671, 1999.
21. F. Ferri, E. Pourabbas, and M. Rafanelli. ''The syntactic and semantic correctness of pictorial configurations to query geographic databases by PQL,'' in *17th ACM Annual Symposium on Applied Computing (ACM SAC 2002)*, Madrid, Spain, 432–437, 2002.
22. D. Greenblatt and J. Waxman. ''A study of three database query languages,'' in B. Shneiderman (Ed.), *International Conference on Databases: Improving Usability and Responsiveness*, Academic Press: Haifa, Israel, 77–97, 1978.
23. S.L. Greene, S.J. Devlin, P. Cannata, and L.M. Gomez. ''No IFs, ANDs, or ORs: A study of database querying,'' *International Journal of Man-Machine Studies*, Vol. 32(3):303–326, 1990.
24. V. Haarslev and M. Wessel. ''Querying GIS with animated spatial sketches,'' in *IEEE Symposium on Visual Languages*, IEEE Comp. Soc.: Capri, Italy, 201–208, 1997.
25. B. Huang and H. Lin. ''Design of a query language for accessing spatial analysis in the web environment,'' *GeoInformatica*, Vol. 3(2):165–183, 1999.
26. K. Ingram and W. Phillips. ''Geographic information processing using a SQL based query language,'' in *8th International Symposium on Computer Assisted Cartography. AUTO-CARTO 8*, Baltimore, 326–335, 1987.
27. T. Joseph and A.F. Cardenas. ''PICQUERY: A high level query language for pictorial database management,'' *IEEE Transactions on Software Engineering*, Vol. 14(5):630–638, 1988.
28. Y. Jun, I. Kim, and R. Myung. ''Icon-Design guidelines based on the Korean user's mental model,'' in *7th Annual International Conference on Industrial Engineering*, Busan, Korea, 176–179, 2002. http://avi.im.isu.edu.tw/proceedings/ijie2002/pdf/b103_5.pdf
29. E. Jungert. ''Graqula-A visual information-flow query language for a geographical information system.'' *Journal of Visual Languages and Computing*, Vol. 4(4):383–401, 1993.
30. C. Katzeff. ''Dealing with a database query language in a new situation,'' *International Journal of Man-Machine Studies*, Vol. 25(1):1–17, 1986.

31. S. Kaushik and E.A. Rundensteiner. ''SVIQUEL: A spatial visual query and exploration language,'' in *9th Int. Conf. on Database and Expert Systems Applications (DEXA'98)*, Vienna, Austria: Springer, 290–299, 1998.
32. S. Kaushik and E.A. Rundensteiner. ''SEE: A spatial exploration environment based on a direct-manipulation paradigm,'' *IEEE Transactions on Knowledge and Data Engineering*, Vol. 13(4):654–670, 2001.
33. Y.C. Lee and F.L. Chin, ''An iconic query language for topological relationships in GIS,'' *International Journal of GIS*, Vol. 9(1):25–46, 1995.
34. H. Lin and B. Huang, ''SQL/SDA: A query language for supporting spatial data analysis and its web-based implementation,'' *IEEE Transactions on Knowledge and Data Engineering*, Vol. 13(4):671–682, 2001.
35. D. Mark and M. Gould, ''Interaction with geographic information: A commentary,'' *Photogrammetric Engineering and Remote Sensing*, Vol. 57(11):1427–30, 1991.
36. B. Meyer. ''Beyond icons: Towards new metaphors for visual query languages for spatial information systems,'' in *International Workshop on Interfaces to Database Systems (IDS 92)*, Glasgow, U.K. Springer, 113–135, 1993.
37. A. Michard. ''Graphical presentation of boolean expressions in a database query language: design notes and an ergonomic evaluation,'' *Behaviour and Information Technology*, Vol. 1(3):279–288, 1982.
38. N. Murray, N. Paton, and C. Goble. ''Kaleidoquery: A visual query language for object databases,'' in Advanced Visual Interfaces, ACM Press: L'Aquila, Italy, 247–257, 1998.
39. N. Murray. ''A visual query language for object databases and its implementation in a 3D environment,'' Ph.D. Dissertation, Department of Computer Science, University of Manchester, 1999.
40. OpenGIS, ''Simple features specification for SQL. Revision 1.1,'' Open GIS Consortium Inc.. http://www.opengis.org/techno/specs/99-049.pdf, 1999.
41. D. Papadias and T.K. Sellis. ''A pictorial query-by-example language,'' *Journal of Visual Languages and Computing*, Vol. 6(1):53–72, 1995.
42. M.A.A. Portier and C. Bonhomme. ''A high level visual language for spatial data management,'' in *Visual '99*, Amsterdam. Springer, 325–332, 1999.
43. S. Ravada and J. Sharma. ''Oracle8i spatial: experiences with extensible database,'' in *SSD' 99*, Hong Kong, China. Springer-Verlag, 355–359, 1999.
44. P. Reisner. ''Use of psychological experimentation as an aid to development of a query language,'' *IEEE Transactions on Software Engineering*, SE-3(3), 218–229, 1977.
45. M. Sebillo, G. Tortora and G. Vitiello, ''The metaphor GIS query language,'' *Journal of Visual Languages and Computing*, Vol. 11(4):439–454, 2000.
46. S. Shekhar, S. Chawla, S. Ravada, A. Fetterer, X. Liu, and C-T. Lu. ''Spatial databases – accomplishments and research needs,'' *IEEE Transactions on Knowledge and Data Engineering*, Vol. 11(1):45–55, 1999.
47. B. Shneiderman. Designing the User Interface. Strategies for Effective Human-Computer Interaction. 3rd ed. Addison-Wesley, Reading, MA, 1998.
48. C. Traynor and M.G. Williams. ''A study of end-user programming for geographic information systems,'' in Seventh workshop on Empirical studies of Programmers. ACM Press: Alexandria, Virginia, United States, 140–156, 1997.
49. C. Traynor. ''Putting power in the hands of end users: a study of programming by demonstration, with an application to geographical information systems,'' in *CHI 98 Conference Summary on Human Factors in Computing Systems*, ACM Press: Los Angeles, California, United States, 68–69, 1998.
50. C. Welty. ''Correcting user errors in SQL,'' *International Journal of Man-Machine Studies*, Vol. 22(4):463–477, 1985.
51. M. Wessel and V. Haarslev. ''VISCO: Bringing visual spatial querying to reality,'' in *IEEE Symposium on Visual Languages*, IEEE Comp. Soc.: Nova Scotia, Canada, 170–177, 1998.
52. M.Y.Y. Yen and R.W. Scamell. ''A human factors experimental comparison of SQL and QBE,'' *IEEE Transactions on Software Engineering*, Vol. 19(4):390–409, 1993.
53. D. Young and B. Shneiderman. ''A graphical filter/flow representation of boolean queries: a prototype implementation and evaluation,'' *Journal of the American Society for Information Science*, Vol. 44(6):327–339, 1993.
54. M.M. Zloof. ''Query-by-example: a database language,'' *IBM Systems Journal*, Vol. 16(4):324–343, 1977.

Andrew Morris has recently received his Ph.D. in Computing from the University of Glamorgan, U.K. He also holds a Bachelors degree in Electrical and Electronic Engineering and a Masters degree in Computer Science from Swansea University, U.K. His research interests include database querying, user interface design and spatial data analysis.

Alia Abdelmoty is a Lecturer in Computer Science in Cardiff University, Wales. She has worked as a Lecturer in Computing and GIS in the University of Glamorgan since 1995, having previously worked as a Research Associate in Computing at Heriot-Watt University, Edinburgh. She holds a M.Sc. and a Ph.D. in Computer Science, from Heriot-Watt and a B.Eng. in Biomedical Engineering from Cairo University, Egypt. Her research interests are in the fields of spatial and spatio-temporal databases and include the development of qualitative representation and reasoning techniques, visual query interfaces and the development of geographic ontologies for supporting web-based geographic information retrieval.

Baher El-Geresy has been working as a senior researcher in Computer Science in the University of Glamorgan since 1996. He holds a M.Sc. and Ph.D. in Mechanical Engineering from Strathclyde University, Scotland, and from Bath University, England. He has worked in the Information Technology industry for a significant period of time where he managed software engineering projects in the Oil field. His interests are in the field of spatio-temporal databases and data mining.

Chris Jones is Professor of Geographical Information Systems in the School of Computer Science at Cardiff University. Previously, he led the GIS Research Group in the School of Computing at the University of Glamorgan and held a University Lectureship in Cambridge, in the Department of Geography, and a fellowship of Fitzwilliam College. Prior to that he was employed on geological computing applications at BP Exploration and at the British Geological Survey. His current research interests include the development of ontology and agent-based geographical information retrieval techniques; multiscale spatial databases; environmental change detection; spatio-temporal data modeling; the computer reconstruction and visualisation of fossils; and automated cartographic design with regard to map generalization and automated text placement.