A Specification of a Spatial Query Language over GML

J.E Córcoles
Secc. Tecnología de la Información.
Universidad de Castilla-La Mancha
Campus Universitario
s/n.02071.Albacete. Spain
+34967599200 ext 2412
corcoles@idr-ab.uclm.es

P. González
Departamento de Informática
Universidad de Castilla-La Mancha
Campus Universitario
s/n.02071.Albacete. Spain
+34967599200 ext 2457
pgonzalez@info-ab.uclm.es

ABSTRACT

OGC (OpenGIS Consortium) is contributing with a XML specification to the representation of geographic information (called GML 2.0 - Geographical Markup Language) [13]. GML allows the exchange of geographic information in the Web. The models based on XML benefit the interoperability, and thus GML allows the exchange of geographic information on the Web. However, there is another important advantage to the models based on XML (GML, ARCHEOGIS [4]): it may be queried.

In this paper, a query language GML is shown. The data model and the algebra underlying the query language are an extension of [2] to support spatial features. The query language has a familiar *select-from-where* syntax and is based on SQL (Structured Query Language). It includes a set of spatial operators (disjoint, touches, etc.), and includes traditional operators (=,>, <,...) for non-spatial information.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: Languages – *Data manipulation languages, Query languages.*

General Terms

Management, Design, Languages.

Keywords

Interoperability, XML, GML, query language.

1. INTRODUCTION

The Geographical Markup Language (GML) is an XML encoding for the transport and storage of geographic information, including both the spatial and non-spatial properties of geographic features [13]. The mechanisms and syntax that GML uses to encode geographic information in XML are defined in the specification of OpenGIS [12].

The fact that GML is an XML encoding is an important feature for two reasons: the first is that there are a great number of tools that allow you to manage XML files and schemas using a graphical user interface. The second is that there are very many query language proposals for XML files [10]. The latter is the most important advantage since there are many models that have been based on XML (ARCHEOGIS [4], GML,etc.) owing to its facility for being queried.

To date, the query languages over XML are of general use. However, it is necessary to enrich this query language with spatial operators if we wish to apply it over spatial data encoded with GML [5]. Otherwise, these query languages will be used to query alphanumeric features of an XML document and not, for example, the topological relationship between two spatial regions.

In order to understand the main innovative features in this work, the state of the art in query languages over XML is shown [5]. In this paper, we have studied and compared the three most important query languages over XML. For the comparison to show a common pattern, the definition of a group of features that all query languages over XML must support is necessary. These features have been obtained from Bonifati and Ceri [3], and Quass [14]. In addition, the possibility of a language supporting spatial operators is defined. This feature is important because it is supported only by this query language. The languages studied are XQL (XML Query Language) [15], XML-QL [6] and Lorel [1] (XQuery is not included in this comparison because the work is still in progress at the present time [16]). XQL is a notation for selecting the elements and text of XML documents. XQL can be considered a natural extension to the XSL pattern syntax [17]. It is designed with the goal of being syntactically very simple and compact, with reduced expressive power [3]. XML-QL extends SQL with an explicit *construct* clause for building the document resulting from the query and uses the element patterns to match data in a XML document. XML-QL can express queries as well as transformations for integrating XML data from different sources. LOREL was originally designed for querying semi-structured data and has now been extended to XML data. It is a user-friendly language in the SQL\OQL style, including a string mechanism for type coercion and permitting very powerful path expressions, which is extremely useful when the structure of a document is not known in advance.

The comparison is shown in the following table.

Table 1: Comparison between languages over XML

	XQL	XML-QL	LOREL
Clean Semantics	YES	YES	YES
Path Expressions	YES	YES	YES
Ability to return an XML document	YES	YES	YES
Ability to query a return XML tags	NO	YES	YES

and attributes			
Intelligence type coercion	Partial	NO	YES
Handles unexpected data	YES	YES	YES
Allows queries when the DTD is not fully known	YES	YES	YES
Returns unnamed attributes	YES	YES	-
Preserves Order	Partial	YES	YES
Allows Spatial operators	NO	NO	NO

This study is used below to compare these languages with our query language.

In Sections 2 and 3 an overview of the underlying data model and algebra is shown. This is a very important part that supplies the semantics of our language. The syntax design for this query language is shown in Section 4. The query language describes how to carry out the features enumerated in the Table1. Section 5 shows conclusions and future work.

2. DATA MODEL

The data model dealt with in this paper is an extension of a data model proposed by Beech, Malhotra and Rys [2]. It shows how the components of a XML document and their interrelationships can be represented as a directed graph. This data model has been selected because it can be easily extended. In addition, it is very simple and powerful. Recently, W3C has proposed another data model and algebra, but this proposal is still in progress [9][8].

In order to query XML document, the data model does not need to be able to represent objects of a high abstraction level (for instance Road object, River object, Cadastral Parcel object, etc.). It is only necessary to represent simple types (integer, string, etc.).

However, in order to query geometry types represented by GML, it is necessary to identify high level objects (river, city, park, etc.) over which spatial operators apply, in addition to simple types (low level objects) to query alphanumeric features (number, name, etc.).

The main extension realised to the data model from [2] includes a new type of vertex (called geometry vertex, $V_{geometry}$) and defines its properties. This vertex allows the representation in our data model of geometry types defined in GML. This modification preserves the power of the original data model to query XML documents, because the geometry vertices are not represented if the XML document has no geometry features, i.e. it is not a GML document.

We now show an overview of our data model. All aspects of the data model ([2]) that are not relevant to this work have been omitted. Our data model is a logical model and is silent on how its components should be stored.

A document XML (GML) is represented in our data model by a directed graph G = (V, E, A, R, O). *V* is the set of elements and values vertices in the graph (V= $V_{element}$? $V_{geometry}$? $V_{int,}$? $V_{string,}$ …). *E* represents the set of directed edges that express element containment in a XML document. *A* is the set of directed edges that represent the relationships between elements and values expressed by XML attributes. *R* represents the relationships between elements referenced from other elements via IDREF and IDREFS attributes in the presence of schema information, Xlink, URI. Finally *O* represents the local order between edges of a particular class, *E*, *A* or *R*, which connects a parent element to its children.

2.1 Vertices

The graph contains three categories of vertices (or nodes): vertices that represent data values ($V_{type(v)}$), vertices that represent geometry elements in GML ($V_{geometry}$), and vertices that represent the rest of non-geometry elements of a document ($V_{element}$).

The type of elements represented by $V_{geometry}$ are: coordinate elements (gml:CoordType and gml:CoordinatesType), primitive geometry elements (gml:PointType, gml:LineStringType, gml:LinearRingType, gml:PolygonType, gml:BoxType), and aggregate geometry elements (gml:MultiPointType, gml:MultiLineStringType, gml:MultiPolygonType). Each one of these Types has a direct relationship with the following elements defined in the GML Schemas: coord, coordinates, Point, LinesString, LinearRing, Polygon, Box, MultiPoint, MultiLineString and MultiPolygon.

The type of element vertices $V_{element}$ is *element* and the type of value vertex $V_{type(v)}$ is *decimal, string, …*($V_{integer}$, V_{string}, etc.)

Each XML element is represented as an vertex *v? (*$V_{element}$,$V_{geometry}$*). v* has a unique, logical and immutable abstract identifier. It has to be unique to serve its role as identifying key, it is logical in the sense that the identifier is independent from the physical store, and is immutable to preserve relationships expressed using the identifier. The abstract identifier cannot be directly accessed by the query.

In order to create the data model starting from a XML document, a function *vertex(x)* was specified to transform an XML element or concrete value into a vertex of our data model. Its function extends the original function from [2] to be able to support geometry vertex. The definition of *vertex(x)* as follows:

? If *x* is an XML element whose type is equal to *element*, then this function returns element's abstract identifier and *element* as its type (*vertex(x) ? V ? (get_oid(x), element)* ? $V_{element}$).

? If the type of *x* is one of the geometry types shown above, then this function return the *value(x)* and its type (*vertex(x) ? V ? (value(x), type(x)))* ? $V_{geometry}$). The definition of *value(x)* is shown in Sub-section 2.4. The type of *x* is obtained from the GML schema of the document.

? To the value vertex, this function returns its value and concrete data type *(vertex(x) ? V ? (x, type(x))) ?* $V_{type(v)}$*)*. The type of *x* may be *string*, *integer*, etc.

2.2 Edges

Let us now turn to the edges structure (E) of the data model. This structure respects the original design from [2]: Every edge *e* ? E is a directed relationship (name ? T_{name}(namespaces[2]), parent ? ($V_{element}$, $V_{geometry}$), child ? V) from an element *parent* vertex to a *child* vertex with the *name*. The edge from parent elements to value vertices have the special name ~data. Edge from parent elements to comment and processing instruction have the names

~comment and ~PI, respectively. Every edge *e* ? A is a directed relationship (name ? T_{name}, parent ? ($V_{element}$, $V_{geometry}$), child ? $V_{type(v)}$) from an element *parent* vertex to a value *child* vertex with the name *name*. Every edge *e* ? *R* is a directed relationship (*parent* ? *V element* , *refegdes* ? *P*(*E* ? *A*), *child* ? *V*) from an element *parent* vertex to an element *child* vertex, which is indicated by an IDREF or IDREFS attribute in the presence of schema information, an XLink, or URI value. *refedges* denotes the set of edges that form the basis of the reference. An IDREFS attribute or a multivalued XLink is mapped into multiple *R* edges, one for every element referred to.

2.3 Order

The order of the data model respects the original design from [2]. *O* defines an order between edges if and only if they share the same parent and they are all of the same class i.e. all *E, A* or *R*. Formally, *O* ? {(*e* ? *E*? *A*? *R, succ* ? *E*? *A*? *R*) | *parent*(*e*) = *parent*(*succ*) ? (*e* ? *E* ? *succ* ? *E* ? *e* ? *A* ? *succ* ? *A* ? *e* ? *R* ? *succ* ? *R*)} where *succ* denotes the successor of *e* in the order. In the case of reference edges, the order among individual references of multi-valued references is defined according to the rules of the reference mechanism. In addition, the order among the *refedges* determines the order among the different references. This implies that there is only a total order among references that have the same type of *refedges* and the order among all reference edges is partial. The predecessor edge in the order can be determined by *pred*(*x*) ? *E*? *A*? *R*? *e*(*o*) | ?*o* ? *O*:*succ*(*o*) = *x*.

2.4 *Value(x)* Operation

Once the structure of vertices and edges of our data model is known, the operation *value* is redefined. This operation allows to obtain the value of a vertex over which an operator of the algebra is applied. Redefinition of this operation is necessary, because in the original data model ([2]) the *value* operation is only applied over *element vertices* and *value vertices*. The *value operation* over *element vertex* returns the unique identifier of the vertex, and if applied over *value vertex* returns the value of this vertex.

The value of a geometry element can be defined depending on the type of element. This function returns a list, which in turn have other lists. Each coordinate is made up of one list with two lists (value of X and Y). The definition of value function is developed using algebra operators which are defined in [2].

```
Value x =
        if type(x) = coord then
        [value(child(? [E,X](x)))] ? value(child(? [E,Y](x)))]
        elseif type(x) = coordinates then
        [Value(x)]
        elseif type(x) = Point then
        [value(*[child(? [E,Coord | Coordinates](x))])]
        elseif type(x) = LineString then
        [value(*[child(? [E,Coord | Coordinates](x))])]
        elseif type(x) = LinearRing then
        [value(*[child(? [E,Coord | Coordinates](x))])]
        elseif type(x) = Box then
        [value(*[child(? [E,Coord | Coordinates](x))])]
        elseif type(x) = Polygon then
        [value(*[child(? [E,LinearRing](x))])]
        elseif type(x) = Polygon then
        [value(*[child(? [E,LinearRing](x))])]
        elseif type(x) = MultiPolygon then
        [value(*[child(? [E,Polygon](x))])]
        elseif type(x) = MultiLineString then
        [value(*[child(? [E,LineString](x))])]
        elseif type(x) = MultiPoint then
        [value(*[child(? [E,Point](x))])]
```

2.5 Overview of the Operations over the Geometry Vertex

The following table gives an overview of the properties of the geometry vertex. These properties are the same as the properties of an *element vertex* in [2]; the behaviour of the *value* and *type*, however, has been redefined, as shown above.

Table 2: Overview of the properties of the geometry vertex

Type	Properties	Description
Geometry vertex	Value	List of lists
($V_{geometry}$)	Type	Geometry Type
	Parent	Geometry or element vertex
	Referredby	List of geometry or element vertices: traversing back over references edge.
	Childelements	List of E edges
	Attributes	Set of attribute edges (A)
	Referents	List of reference edges (R)

2.6 Example of the Data Model

To illustrate our data model, an example is given (figure 1 and figure 2). This example shows the data model for a portion of a GML document, called *cambridge.xml* defined in [13]. It represents part of the definition of a city model that includes River Objects, Road Objects, boundary box of the city, name, etc. In the example only a Road object is defined. In order to avoid confusion, the data types and values of the Geometry vertices have not been included.

```
<cityMember> <Road>
 <gml:name>M11</gml:name>
 <linearGeometry>
  <gml:LineString ID=”23">
  <gml:coord>
   <gml:X>0</gml:X><gml:Y>5.0</gml:Y>
  </gml:coord>
  <gml:coord>
   <gml:X>20.6</gml:X><gml:Y>10.7</gml:Y>
  </gml:coord>
  <gml:coord>
   <gml:X>80.5</gml:X><gml:Y>60.9</gml:Y>
  </gml:coord>
  </gml:LineString>
 </linearGeometry>
 <classification>motorway</classification>
 <number>11</number>
</Road> </cityMember>
```

Figure 1: GML Document

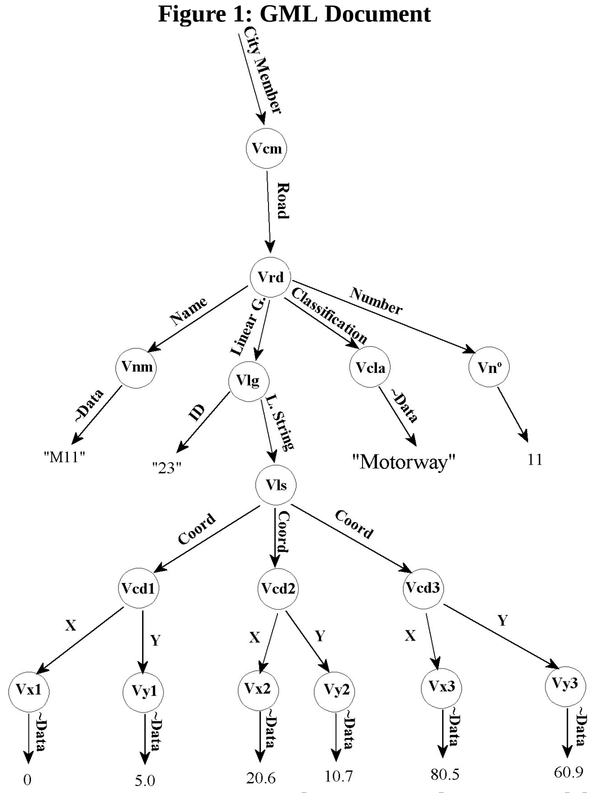

Figure 2: Mapping a GML document to the Data model

The node *Vls* is of type *LinearString* (information obtained on the document schema). *Vcd1*, *Vcd2* y *Vcd3* is of type Coord. According to the definition of value function, the values of *Vls*, *Vcd1*, *Vcd2* y *Vcd3* are:

Value(Vcd1) = [[0],[5.0]]
Value(Vcd2) = [[20.6][10.7]]
Value(Vcd3) = [[80.5] [60.9]]
Value(Vld) = [[[0],[5.0]] [[20.6][10.7]] [[80.5] [60.9]]]

3. XML ALGEBRA

Now that the data model has been defined, we will define the algebra applied over this data model. This algebra should provide capabilities for selecting documents or components of documents that meet given. The algebra should also support the composition of XML documents from selected documents and their components. [2] proposes an algebra with these features that is minimal enough to provide an abstraction of the basic functionality. However, in order to complete this algebra it is necessary to define a set of spatial relationship predicates to be applied over the geometry vertex.

The original algebra from [2] has the following operators: navigation (?), Kleene star (*), map, selection (with existential and universal quantification), Joins, distinct, sort, unorder, and operations to results construction. All of these operators can be used in our data model explained above.

In addition to these operators, the main contribution to this algebra is the definition of spatial relationship predicates. The set of predicates is obtained from OpenGIS specification [11]. This is based on the Dimensionally Extended Nine-Intersection Model [7]. The six predicates are named Disjoint, Touches, Crosses, Within and Overlaps. The definition of these predicates [11] is given in the Table 3. The term P is used to refer to 0 dimensional geometries (Points and MultiPoints), L is used to refer to one-dimensional geometries (LineStrings and MultiLineStrings) and A is used to refer to two-dimensional geometries (Polygons and MultiPolygons).

Table 3: Definition of predicates

Domain	**Operator**
Topologically closed geometries	Disjoint(a,b) ? a ? b = ?
a and *b* applies to the A/A, L/L, L/A, P/A and P/L groups of relationships.	Touches(a,b) ? (I(a)? I(b) = ?) ? (a ? b) ??
a and *b* applies to the P/L, P/A, L/L and L/A.	Crosses(a,b) ? (dim(I(a) ? I(b)) < max(dim(I(a)), dim(I(b)))) ? (a ? b ?a) ? (a ? b ?b)
a and *b* applies to the A/A, P/A,L/A,L/L	Within(a,b) ? (a ? b = a) ? (I(a) ? I(b) ? ?)
a and *b* applies to the A/A, P/A,L/A,L/L	Contains(a,b) ? Within(b,a)
a and *b* applies to the A/A, P/A,L/A,L/L	Intersects(a,b) ? not.Disjoint(a,b)

In addition, other operators that support spatial analysis have been included: distance, buffer, convexhull, intersection, union, difference and symdifference. The domain of these operators is detailed in [11]. These operators are applied over geometry vertex.

An example of a query composed with this algebra is now shown. To select all Road names that have a distance less that 30, we could write the following expressions:

A:= ?[E,Road](*[child(?[E,#](x))] (x:root))

B:=? [distance(value(child(?[E,LinerGeometry|LineString])))<30](c:A)

C:= child(?[E,Name](child(B)))

A contains all Road edges in the document. The operator * reaches all vertices that are reachable from the root vertex (or vertices), and the follow operator then follows all existing Road edges. *B* contains all Road edges that have at least one LineString or LinearGeometry with the requested distance less than 30. *C* is the final result: the requested Road names.

4. QUERY LANGUAGE

The definition of queries with algebraic expressions is complex and describes the process inefficiently. For this reason, a query language should be defined to make a more user-friendly interface, and to allow several techniques to be applied in order to achieve a more efficient implementation. In this Section, the main features of a query language are shown. This language is still being developed at present, and therefore in this paper our purpose is simply to show its syntax and semantics and not its implementation features.

The syntax chosen for this language is based on SQL. The *Select-From-Where* statement is widespread in query languages since it allows for rapid learning of the language. Note that in this language no effort is made to give the definition of these features of SQL. Only important features for running simple and powerful queries are developed. On the other hand, many features of the query languages upon XML are not considered in traditional query languages (SQL, OQL, etc.), although they are necessary in this kind of language (i.e. path expressions). The syntax of our language is very similar to the syntax defined in Lorel [1]

The characteristics of the language are shown using the features that any language over XML must have [3][14]. We can compare this language with the other languages studied in Section 1. It is assumed that queries are performed over the document cambridge.xml defined in [13]. The characteristics of our language are:

Clean Semantics: All languages based on *select-from-where* statements allow powerful and well-structured queries to be written.

Path Expressions: The dot notation is used to navigate in the data model.

Ability to return an XML document: For compatibility with other tools, the result set of a query is another XML document. The structure of the new document should respect the GML schemas if it has geometry types. For example, if the user wants to obtain the Roads (<LinearGeometry>) of "Motorway" classification (<classification>) and number (<number>) equal to 11 then he or she should make the following query

```
Select C.Road.LinearGeometry
From [http://www.uclm.es//prove.xml].CityModel C
Where C.Road.classification Like 'Motorway' and C.Road.number = 11
```

Return the next document:

```
<linearGeometry>
  <gml:LineString
srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
<gml:coord><gml:X>0</gml:X><gml:Y>5.0</gml:Y>
</gml:coord>
<gml:coord><gml:X>20.6</gml:X><gml:Y>10.7</gml:Y>
</gml:coord>
<gml:coord><gml:X>80.5</gml:X><gml:Y>60.9</gml:Y>
</gml:coord>
</gml:LineString>
</linearGeometry>
```

Ability to query and return XML tags and attributes: This language allows some attributes and tags to be partially defined in the query. The symbol '%' is used as a wildcard in the partial definitions. For example: find all Road names with number equal to 11. Since the user does not know the exact name of the tag *number*, the query may be

```
Select R.name
From [http://www.uclm.es//prove.xml].Road R
Where R.(num%) = 11
```

Intelligence type coercion: This language supports the automatic coercion between simple types. However, the coercion is not defined for geometry types.

Handles unexpected data: To define this feature, the operators 'Like' and '=' are handled in the same way as an existential quantifier. Thus, if a tag has several values it then allows the first to be selected to carry out the condition.

Ability to allow queries when the schema is not fully known: In order to make it unnecessary for the user to know the schema of a document, this language shows the optional tags and attributes in brackets each one of them being separated by the symbol '|'. This symbol '|' means "optimality between different tags or attributes. In this way, the user indicates that he or she wants to find values stored in elements with a different syntax but with the same semantics. For example, to return all cities of the two documents (prove1.xml ,prove2.xml) which have roads with name or description equal to 'M11', the following query may be used:

```
Select C.name
From [http://www.uclm.es//prove.xml,
http://www.uclm.es//prove2.xml].CityModel as C,
Where C.road.(name | description) Like 'M11'
```

Returns unnamed attributes: The language allows us to establish in the *Select* clause whether it returns only tags or tags with all these children. To achieve this the symbol '^' is used. This symbol means that a determined tag or attribute is not included in the result set. For example, if the user wants to return all elements of roads (<*road*>), the following query should be written:

```
Select C.(^Road)
From [http://www.uclm.es//prove.xml].CityModel as C
```

Preserves Order: The queries respect the order in which each piece of data is stored.

Allows Geometry operators: This feature is the main difference with other query languages. There are two types of operators: methods for testing Spatial Relations and methods that support Spatial Analysis. Table 4 shows the syntax and returned types of these operators. The algebra operators are combined to obtain these high-level abstraction operators. Those operators returning Boolean types may be combined with the *not* operator.

Tabla4: Spatial operators supported by the query language

Syntax	Return	Description
Geometry1 equals Geometry2	Boolean	if Geometry1 is 'spatially equal' to Geometry2.
Geometry1 disjoint/intersects/ touches/crosses/within/contains/overlap ListGeometries	Boolean	If Geometry1 is disjoint/intersects, touches/crosses/within/contains/overlaps with some Geometry of ListGeometries (1 or more items).
Geometry1 disjoint_all/ intersects_all/ touches_all/ crosses_all/within_all/ contains_all/ overlap_all ListGeometries	Boolean	If Geometry1 is disjoint/ intersects/touches/crosses/within/contains/overlaps with all Geometry of ListGeometries (1 or more items).
Distance (Geometry1,Geometry2)	Double	Returns the shortest distance between any two points in the two geometries.
Buffer (Geometry1, double)	Geometry	Returns a geometry that represents all points whose distance from Geometry1 is less than or equal to distance (double).

ConvexHull (Geometry1)	Geometry	Returns a geometry that represents the convex hull of Geometry1.
Intersection(Geometry1, Geometry2)	Geometry	Returns a geometry that represents the point set intersection of Geometry1 with Geometry2.
Union (Geometry1, Geometry2)	Geometry	Returns a geometry that represents the point set union of Geometry1 with Geometry2
Difference (Geometry1, Geometry2)	Geometry	Returns a geometry that represents the point set difference of Geometry1 with Geometry2
SymDifference (Geometry1, Geometry2)	Geometry	Returns a geometry that represents the point set symmetric difference of Geometry1 with Geometry2

To resume, the next query contains most of features defined. It obtains all names of Roads with classification such as "motorway", number or "num" greater than zero and intersect with all rivers with names such as "cam%".

```
Select C.%.Road.name.
From [http://www.uclm.es//prove.xml].CityModel C
Where C.%.clasification. Like 'Motorway' and C.%.[number|num]>0
and
C.CityMember.Road.%.LineString intersect_all (
select CM.%.River.LineString
from [http://www.uclm.es//prove.xml].CityModel CM
where CM.CityMember.River.name like (cam%)
```

Note that nested select is possible in the query language.

5. CONCLUSIONS

In this paper a new query language over GML has been shown. The main feature of this language is that it includes spatial operators in its specifications. This feature is not included in the most widely-known XML query languages. To carry out the implementation of this language it is necessary to previously define a data model and an algebra that support basic features of query languages XML and spatial features. For this reason, we have extended a data model and algebra [2] to allow the representation of geometry elements and geometry operators over these elements. Owing to this extension, with the language specified, querying GML or XML documents is possible, because the original features from [2] are conserved.

When this algebra and data model were defined, we described the necessary features that the final query language must have. The syntax of the query is similar to SQL. Apart from the geometry features, the remaining features of the query language are obtained from previous works by Bonifati and Quass. The specification of this language is of great importance for the immediate future of GIS

Future work foresees the implementation of this language applying optimisation techniques to spatial operators. Furthermore, a Web environment will be developed to allow spatial distributed queries on the Web. Temporal operators will also be added to the algebra to achieve a spatio-temporal query language.

6. REFERENCES

[1] Abiteboul, S., Quass, S., McHugh, J., Widom, J. and Wiener, J.. The Lorel Query Language for Semistructured Data. International Journal on Digital Libraries, 1, 1, 68-88, 1997

[2] Beech, D., Malhotra. A., Rys, M. A Formal Data Model and Algebra for XML. http://www-db.stanford.edu/dbseminar/Archive/FallY99/malhotra-slides/malhotra.pdf. 1999

[3] Bonifati,A. and Ceri, S., Comparative Analysis of Five XML Query Languages. SIGMOD Record 29, 1, 68-79, 2000

[4] Brovelli, M.A., Maurino. A. ARCHEOGIS: an interoperable model for archaeological data. ISPRS 2000 Conference, 2000

[5] Córcoles, J.,García-Consuegra, J., Peralta, J. and Navarro E. A Spatio-Temporal Query Language for a Data Model based on XML.6th EC-GI & GIS Workshop. Lyon, France. 2000

[6] Deutsch, A., Fernandez, M., Florescu, D., Levy, A. and Suciu, D., XML-QL: A Query Language for XML. Technical Report NOTE-xml-ql-19980819. http://www.w3.org/TR/1998/NOTE-xml-ql-19980819.html. 1998

[7] Egenhofer, M. and Herring J., Categorizing Binary Topological Relations Between Regions, Lines, and Points in Geographic Databases. 4th International Symposium on Spatial Data Handling, Zurich, International Geographical Union, 803-813. 1990

[8] Fankhauser, P., Fernandez, M., Malhotra, A., Rys, M., Siméon, J and Wadler, P. The XML Query Algebra. W3C Working Draft. http://www.w3.org/TR/query-algebra/. 2001

[9] Fernandez, M. and Robie, J. XML Query Data Model. W3C Working Draft. http://www.w3.org/TR/query-datamodel. 2001

[10] Marchiori M. The Query Languages Workshop http://www.w3.org/TandS/QL/QL98/pp.html. 1999

[11] Open GIS Consortium, Inc. OpenGIS Ò Ò Simple Features Specification For SQL Revision 1.1OpenGIS Project Document 99-049 Release. 1999

[12] OpenGis Consortium. Specifications. http://www.opengis.org/techno/specs.htm.1999

[13] OpenGIS. Geography Markup Language (GML) v2.0. Document Number: 01-029. 2001. http://www.opengis.net/gml/01-029/GML2.html

[14] Quass, D. Ten features necessary for an XML query language. In proc. of the Query Language workshop, Cambridge, Mass., 1998.

[15] Robie., J. The design of XQL. http://www.w3.org/style/XSL/Group/1998/09/XQL-design.html. 1998

[16] XQuery: A Query Language for XMLW3C Working Draft 15 February 2001 http://www.w3.org/TR/2001/WD-xquery-20010215. 2001

[17] XSL Specification. W3C recommendation. http://www.w3. org/TR/REC-xml. 1998