GQL: Extending XQuery to Query GML Documents

GUAN Jihong  ZHU Fubao  ZHOU Jiaogen  NIU Liping

ABSTRACT GML is becoming the de facto standard for electronic data exchange among the applications of Web and distributed geographic information systems. However, the conventional query languages (e.g. SQL and its extended versions) are not suitable for direct querying and updating of GML documents. Even the effective approaches working well with XML could not guarantee good results when applied to GML documents. Although XQuery is a powerful standard query language for XML, it is not proposed for querying spatial features, which constitute the most important components in GML documents. We propose GQL, a query language specification to support spatial queries over GML documents by extending XQuery. The data model, algebra, and formal semantics as well as various spatial functions and operations of GQL are presented in detail.

KEYWORDS XML; GML; spatial feature; query language; XQuery

CLC NUMBER P208

Introduction

The geography markup language (GML) \cite{1} is an XML encoding in compliance with ISO 19118 for the transport and storage of geographic information model according to the conceptual modeling framework used in the ISO 19100 series and including both the spatial and non-spatial properties of geographic features. As more and more geographic/spatial data on the Web is being represented and stored in GML format, there is an urgent need of query language for GML documents. One of the important features of GML is its flexibility in representing different types of spatial/geographic information from diverse sources. To exploit this flexibility, a GML query language must provide features for retrieving and interpreting information from these diverse sources. Even approaches working well with XML, such as Lorel\cite{2}, XQL\cite{3}, XML-QL\cite{4} and Quilt\cite{5} etc., could not guarantee good results when applied to GML documents that contain both alphanumeric and spatial data. Spatial querying approaches like SpatialSQL\cite{6}, GeoSQL\cite{7} are proposed for relational database system, thus not suitable for GML documents.

Reference \cite{8} describes the interoperable WMS(Web map server) compliant map server to integrate GML using URLs as the form of query, and visualize the query results on client-side in GML. The authors give computational cost models for single scan and multi-scan query-processing strategies by use of SAX and DOM parsers. However, they have not developed any query language for GML documents. Reference \cite{9} proposes a specification of spatial query language over GML based on SQL syntax. The data model and the algebra underlying the query language are an extension of Beech\cite{10} to support spatial features, but the proposed language does not conform to standard XML query languages, and it requires implementation from scratch. Vatsavai\cite{11} describes a spatial query language GML-QL derived from XQuery\cite{12}. He roughly classifies spatial queries into unary and binary types,

Received on December 12, 2005.
Funded by the Youth Chengguang Project of Science and Technology of Wuhan City of China(No. 20045004071-161).

GUAN Jihong, professor, Ph. D. School of Computer, Wuhan University, 129 Luoyu Road, Wuhan 430079, China.
E-mail: jhguan@whu.edu.cn
and defines various spatial functions and operations to support spatial query and analysis. Except for some query examples, he does not provide detailed information about GML-QL. Reference [13] also proposes a query language called GQuery based on XQuery. However, GQuery has only limited extension to XQuery, and it aims to support spatial information integration, instead of a general-purpose query language.

To query GML documents, we propose a new query language named GQL (GML query language) by extending XQuery language in this paper. We extend XQuery in four aspects: data model, algebra, functions, and formal semantics. We define new data types, accessors and nodes related to spatial query and analysis in data model, new predicates and operations in algebra, various functions to realize spatial computation and formal semantics to enhance the GQL specification by defining the meaning of GQL expressions.

1 Data model

Like XML, GML documents are also tree-structured. GML documents can be represented by a directed graph $G = (N, E, r)$, where $N$ is a set of tree nodes in $G$; $E$ is a one-to-many relation of edges from parent nodes to children nodes, indicating the inter-relationships between parent elements and their children in GML documents; and $r$ is the distinguished root node of the tree in the graph, pointing out the entry when the tree is handled. We extend the XQuery data model with new data types, accessors and nodes related to spatial query and analysis in data model, new predicates and operations in algebra, various functions to realize spatial computation and formal semantics to enhance the GQL specification by defining the meaning of GQL expressions.

1.1 Data types

XQuery 1.0 and XPath 2.0 Data Model define 25 basic and derived types, but these data types are not sufficient for describing spatial data defined in GML. We define 12 additional types, including gml:Geometry, gml:Coord, gml:Coordinates, gml:Point, gml:LineString, gml:LinearRing, gml:Polygon, gml:Box, gml:GeometryCollection, gml:MultiPoint, gml:MultiLineString and gml:MultiPolygon to support GML spatial data representation.

Fig. 1 is the hierarchy of data types of GQL. Here, gml:Geometry is abstract datatype, indicating the geometric property of geographical objects, and it is the basic type of all the primitive geometric types defined below; gml:Coord defines the coordinates of geographic objects in one, two or three dimensions, which can be formally described as $(X, Y, Z)$; gml:Coordinates is a sequence of gml:Coord data; gml:Point is derived from gml:Coord; gml:Box is a sequence of two Coords or a Coordinates containing only two coordinate tuples, and serves as the location of a concrete geographical object; gml:LineString consists of at least two Coords or a Coordinates containing no less than two coordinate tuples to describe a line string; gml:LinearRing is a closed line string and composed of at least four coordinates to depict a linear ring; gml:Polygon is derived from gml:LinearRing composed of at most one exterior boundary and zero or more interior boundary, and serves as the area of a geographical object.

The hierarchy in Fig. 1 is based on XQuery data model, which is extended with the geometry node to represent the spatial features defined in GML specification and gml:Geometry datatype and its subtypes to describe the geometry properties of GML instances. An item is either a node or geometry or an atomic value.

1.2 Accessors

The XQuery data model defines a family of accessor functions in order that the processors are able to operate on instances. These are not functions in the literal sense, and they are not available for users or applications to call directly. Rather they are descriptions of the information that an implementation of the data model must expose to applications. We define additional accessors for geometry nodes as follows:

dm: geometry-type ($n$ as node ( ) ) as xs: string; this accessor returns a string identifying
the geometry type of the input node. The returned result should be one of the following, “Point”, “LineString”, “LinearRing”, “Polygon”, “Box”, “MultiPoint”, “MultiLineString”, “MultiPolygon”, etc.

dm:srs-name($n as node()) as xs:string.
this accessor returns a string identifying the spatial reference system of the input node, and usually the result is a URI reference.

dm:node-type($n as node()) as xs:string.
this accessor returns a string identifying the kind of the input node. The returned result should be one of the following: “geometry”, “attribute”, “comment”, “document”, “element”, “namespace”, “processing instruction”, or “text”.

dm:dimension($n as node()) as xs:integer.
this accessor returns the number of dimensions of the input node. This definition is restricted to geometric objects of two-dimension.

1.3 Nodes

XQuery defines seven kinds of nodes to describe the data model of XML documents: document, element, attribute, text, namespace, processing instruction, and comment nodes. A•
document is a tree with the document node as the root of the tree, and a document fragment is a tree with an element at its root. To describe the geographic information in GML documents, we define geometry node. A geometry node is a spatial element node, describing spatial features, such as location, shape, area, spatial reference system (SRS), etc.

Geometry nodes encapsulate GML geometric information of features. They have the following properties: base-uri (possibly empty), node-name, parent, type-name, children, attribute (possibly empty), namespaces (possibly empty), string-value, typed-value, srs-name (possibly empty), and geometry-type. Geometry nodes must satisfy the following constraints.

1) The children must consist exclusively of geometry, element, comment, and text nodes if it is not empty. Attribute, namespace, and document nodes can never appear as children.

2) The attribute nodes of a geometry must have distinct xs:QNames.

3) If a node N is among the children of a geometry G, then the parent of N must be G.

4) Exclusive of attribute and namespace nodes, if a node N has a parent geometry G, then N must be among the children of G. Attribute and namespace nodes have a parent node, but they do not appear among the children of their parent node.

5) If an attribute node A has a parent geometry G, then A must be among the attributes of G. If a namespace node N has a parent geometry G, then N must be among the namespaces of G.

6) If dm: type-name of an geometry node is gml:Geometry, then dm: type-name of all its descendants must also be gml:Geometry or its derived type.

2 Algebra

In the database world, it is a tradition to translate a language into algebra. Reference [14] proposes algebra for XML Query, which has the following operations; projection, selection, quantification, join, order, sorting, aggregation, and functions for well-formed documents. All of these operations and functions can be used for our data model. However, they are not sufficient for querying GML. In order to support spatial query and analysis over GML documents, we define a set of spatial relationship predicates being applied to the geometry nodes and a family of spatial operators and functions. Certainly, these predicates, operators and functions have been proposed in traditional spatial query languages, what we do here is to transplant them into our GML query language GQL.

2.1 Predicates

Referring to OpenGIS simple features specification for SQL[15], we first define a set of spatial predicates based on the dimensionally extended nine-intersection model (DE-9IM), which has the following form

\[ \Gamma_1(A, B) = \begin{bmatrix} A^\circ \cap B^\circ & A^\circ \cap \partial B & A^\circ \cap B^\circ \\ \partial A \cap B^\circ & \partial A \cap \partial B & \partial A \cap B^\circ \\ A^- \cap \partial B & A^- \cap B^\circ & A^- \cap B^\circ \end{bmatrix} \]

where \( A^\circ, \partial A \) and \( A^- \) represent the interior, boundary and exterior of \( A \) respectively. For simplicity of representation, given a geometry object \( a \), we denote \( I(a), B(a) \) and \( E(a) \) the interior, boundary and exterior of \( a \) respectively. The intersection of any two from \( I(a), B(a) \) and \( E(a) \) can result in a set of geometries, \( x \), of mixed dimension. Let \( \text{dim}(x) \) return the maximum dimension (0, 1, 2) of the geometries in \( x \), with -1 corresponding to \( \text{dim}(\emptyset) \).

The seven predicates we define for GQL are; Disjoint, Touches, Crosses, Within, Overlaps, Contains and Intersects. Let \( P \) refer to 0 dimensional geometries (Points and MultiPoints), \( L \) refer to one-dimensional geometries (LineStrings and MultiLineStrings) and \( A \) refer to two-dimensional geometries (Polygons and MultiPolygons), the definitions of these predicates are given in Table 1.

2.2 Operations

Spatial operations can be categorized into unary and binary spatial operations. Unary spatial operations are functions that operate with a spatial
Table 1 Spatial predicates in GQL

| Predicate     | Definition                                                                 | Application domain                  |
|---------------|---------------------------------------------------------------------------|-------------------------------------|
| Disjoint      | \( a \cap b = \emptyset \) \land (a \cap b) \neq \emptyset               | Any two topologically closed geometries |
| Touches       | \( \text{Touch}(a, b) \equiv \dim(I(a) \cap I(b)) = \emptyset \land (a \cap b) \neq \emptyset \) | A/A, L/L, L/A, P/A and P/L situations |
| Crosses       | \( \text{Cross}(a, b) \equiv \dim(I(a)) \cap \dim(I(b)) < \max(\dim(I(a)), \dim(I(b))) \land (a \cap b) \neq \emptyset \) | P/L, P/A, L/L and L/A situations     |
| Within        | \( \text{Within}(a, b) \equiv (a \subseteq b) \land (a \cap b) \neq \emptyset \) | A/A, P/A, L/A, L/L situations       |
| Overlaps      | \( \text{Overlaps}(a, b) \equiv (\dim(I(a)) \cap \dim(I(b))) = \emptyset \land (a \cap b) \neq \emptyset \) | A/A, L/L and P/P situations         |
| Contains      | \( \text{Contains}(a, b) \equiv \text{Within}(b, a) \)                    | A/A, L/L and P/P situations         |
| Intersects    | \( \text{Intersects}(a, b) \equiv \text{Disjoint}(a, b) \)                | A/A, L/L and P/P situations         |

Table 2 to Table 5 show basic functions on geometry, functions for testing spatial relations between geometric objects, functions that support spatial analysis, and functions on Point, LineString, LinearRing and Polygon. Unary and binary spatial functions can occur in FOR, LET, WHERE clauses and topological operations can be applied in WHERE clause. Depending on concrete requirements, these functions can take either scalar, or collections as arguments, and the possible return values are boolean, scalar, or collections.

Table 2 Basic functions on geometry

| Syntax                        | Description                                                                 |
|-------------------------------|-----------------------------------------------------------------------------|
| fn:dimension( $ g $ as gml:Geometry) as xs:integer | Returns the dimension of the geometry, which is less than or equal to the dimension of the coordinate space. |
| fn:geometryType( $ g $ as gml:Geometry) as xs:string | Returns the name of the instantiable subtype of gml:Geometry of which this instance is a member. |
| fn:srs-name( $ g $ as gml:Geometry) as xs:string | Returns the spatial reference system name for the given geometry. |
| fn:envelope( $ g $ as gml:Geometry) as gml:Geometry | Returns the rectangle bounding $ g $ as a Polygon, defined by the corner points of the bounding box. |
| fn:Boundary( $ g $ as gml:Geometry) as gml:Geometry | Returns the closure of the combinatorial boundary of the given geometry. |

Table 3 Functions for testing spatial relations between two geometric objects

| Syntax                        | Description                                                                 |
|-------------------------------|-----------------------------------------------------------------------------|
| op:equals( $ g_1 $ as gml:Geometry, $ g_2 $ as gml:Geometry) as xs:boolean | Returns value 1 for true, 0 for false. Returns 1 if geometry $ g_1 $ is spatially equal to geometry $ g_2 $. |
| op:disjoint( $ g_1 $ as gml:Geometry, $ g_2 $ as gml:Geometry) as xs:boolean | Returns value 1 for true, 0 for false. Returns 1 if geometry $ g_1 $ is spatially disjoint from geometry $ g_2 $. |
| fn:intersects( $ g_1 $ as gml:Geometry, $ g_2 $ as gml:Geometry) as xs:boolean | Returns value 1 for true, 0 for false. Returns 1 if geometry $ g_1 $ is spatially intersects from geometry $ g_2 $. |
| op:touches( $ g_1 $ as gml:Geometry, $ g_2 $ as gml:Geometry) as xs:boolean | Returns value 1 for true, 0 for false. Returns 1 if geometry $ g_1 $ is spatially touches from geometry $ g_2 $. |
| op:crosses( $ g_1 $ as gml:Geometry, $ g_2 $ as gml:Geometry) as xs:boolean | Returns value 1 for true, 0 for false. Returns 1 if geometry $ g_1 $ is spatially crosses from geometry $ g_2 $. |
| op:within( $ g_1 $ as gml:Geometry, $ g_2 $ as gml:Geometry) as xs:boolean | Returns value 1 for true, 0 for false. Returns 1 if geometry $ g_1 $ is spatially within from geometry $ g_2 $. |
| op:contains( $ g_1 $ as gml:Geometry, $ g_2 $ as gml:Geometry) as xs:boolean | Returns value 1 for true, 0 for false. Returns 1 if geometry $ g_1 $ is spatially contains from geometry $ g_2 $. |
| op:overlaps( $ g_1 $ as gml:Geometry, $ g_2 $ as gml:Geometry) as xs:boolean | Returns value 1 for true, 0 for false. Returns 1 if geometry $ g_1 $ is spatially overlaps from geometry $ g_2 $. |
| op:relate( $ g_1 $ as gml:Geometry, $ g_2 $ as gml:Geometry, $ patternMatrix $ as xs:string) as xs:boolean | Returns value 1 for true, 0 for false. Returns 1 if geometry $ g_1 $ is spatially related to geometry $ g_2 $, by testing for intersections between the interior, boundary and exterior of the two geometries as specified by the values in the patternMatrix based on DE-9IM. |
Table 4  Functions for spatial analysis

| Syntax | Description |
|--------|-------------|
| fn:distance( $ gl as gml:Geometry, $ g2 as gml:Geometry) as xs:double | Returns the shortest distance between any two points in the two geometries. |
| fn:Buffer( $ g as gml:Geometry, $ distance as xs:double) as gml:Geometry | Returns a geometry that represents all points whose distance from geometry $ g$ is less than or equal to distance. |
| fn:convexHull( $ g as gml:Geometry) as gml:Geometry | Returns a geometry that represents the convex hull of the given Geometry. |
| fn:intersection( $ gl as gml:Geometry, $ g2 as gml:Geometry) as gml:Geometry | Returns a geometry that represents the point set intersection of geometry $ gl$ and geometry $ g2$. |
| op:Union( $ g as gml:Geometry, $distance as xs:double) as gml:Geometry | Returns a geometry that represents the point set union of geometry $ g1$ with geometry $ g2$. |
| op:Difference( $ g as gml:Geometry, $ distance as xs:double) as gml:Geometry | Returns a geometry that represents the point set difference of geometry $ gl$ with geometry $ g2$. |
| op:SymDifference( $ g as gml:Geometry, $ distance as xs:double) as gml:Geometry | Returns a geometry that represents the point set symmetric difference of geometry $ g1$ with geometry $ g2$. |

Table 5  Functions on Point, LineString, LinearRing and Polygon

| Syntax | Description |
|--------|-------------|
| fn:x( $ p as gml:Point) as xs:double | Returns the $ x$-coordinate value for the given point. |
| fn:y( $ p as gml:Point) as xs:double | Returns the $ y$-coordinate value for the given Point. |
| fn:startPoint( $ c as gml:LineString) as gml:Point | Returns the start point of the given linestring. |
| fn:endPoint( $ c as gml:LineString) as gml:Point | Returns the end point of the given linestring. |
| fn:isClosed( $ c as gml:LineString) as xs:boolean | Returns true if this linestring is closed (startPoint( ) = endPoint( )). |
| fn:isRing( $ c as gml:LineString) as xs:boolean | Returns true if the given linestring is closed and this linestring is simple (does not pass through the same point more than once). |
| fn:numPoints( $ ls as gml:LineString) as xs:integer | Returns the number of points in the given linestring. |
| fn:pointN( $ p as gml:Polygon) as gml:Point | Returns the specified point $ n$ in the given linestring. |
| fn:area( $ p as gml:Polygon) as xs:double | Returns the area of the given polygon in its associated spatial reference system. |
| fn:centroid( $ p as gml:Polygon) as gml:Point | Returns the mathematical centroid for the given polygon as a Point. |
| fn:ExteriorRing( $ p as gml:Polygon) as gml:LineString | Returns the exterior ring of the given polygon. |
| fn:numInteriorRing( $ p as gml:Polygon) as xs:integer | Returns the number of interior rings in the given polygon. |
| fn:InteriorRingN( $ p as gml:Polygon, $ n as xs:integer) as gml:LineString | Returns the $ n$th interior ring for the given polygon as a linestring. |

3  Formal semantics

Formal semantics is used to enhance the GQL specification by defining the meaning of GQL expressions, ensure that no corner case is left out, and provide a reference for implementation. Besides inheriting the formal semantics of XQuery, GQL revises some of the existed components to support spatial analysis and query processing and has some additional components.

3.1  Geometry kind test

We modify SequenceType syntax definition of KindTest defined in XQuery formal semantics, and add new definitions of GeometryTest, GeometryNameOrWildcard, GeometryName, GeometryType to build the semantics of geometry node in GQL, and give the corresponding static type analysis and dynamic evaluation. The following EBNF is the definition of KindTest defined in GQL:

KindTest ::= DocumentTest | ElementTest | GeometryTest | AttributeTest | SchemaElementTest | SchemaAttributeTest | PITest | CommentTest | TextTest | AnyKindTest

GeometryTest ::= <"geometry" "(" > (GeometryNameOrWildcard ( "," TypeName "?" )? ) )

GeometryNameOrWildcard ::= GeometryName | "*"

GeometryName ::= QName

GeometryType ::= "geometry" GeometryName? TypeSpecifier?

Kind test is an alternative form of a node test, which can select nodes according to their kind, name, and type annotation. When a kind test is used in a node test, only those nodes on the designated axis that qualify the kind test are selected.

3.2  Static type analysis and dynamic evaluation

Query processing consists of two phases: static
The static analysis phase depends on the query expression and the static context. Its purpose is to detect errors (e.g., syntax errors or type errors) at compile time, rather than at run-time. The dynamic evaluation phase, sometimes also called "execution", is the phase during which the query expression is evaluated. We use inference rules in Reference [16] to represent these two phases.

3.2.1 Static type analysis
The rules for the geometry kind test are similar to those for the element kind test. If the type of the expression is a subtype of the geometry kind test, then it is guaranteed that during evaluation, the geometry value of expression will always match the geometry kind test, and therefore the type of the entire expression is the type of the input expression. That is,

\[
\begin{align*}
\text{statEnv} & |- \{\text{GeometryTest}\}_{\text{geometrytype}} = \text{GeometryType} \\
\text{statEnv} & |- \text{Type1} <; \text{GeometryType} \\
\text{dynEnv} & |- \text{test GeometryTest with "geometry" of Type1 ; Type1}
\end{align*}
\]

On the contrary, if the type of the geometry kind test is a subtype of the expression, then during evaluation, the geometry value of expression may or may not match the geometry kind test, therefore the type of the entire expression is zero-or-one of the type of the geometry kind test, i.e.,

\[
\begin{align*}
\text{statEnv} & |- \{\text{GeometryTest}\}_{\text{geometrytype}} = \text{GeometryType} \\
\text{statEnv} & |- \text{GeometryType} <; \text{Type1} \\
\text{dynEnv} & |- \text{test GeometryTest with "geometry" of Type1 ; GeometryType?}
\end{align*}
\]

If the types of the expression and geometry kind test are unrelated (i.e., neither type is a subtype of the other), then we must compare the structure of the type of the geometry test with the type of the geometry expression, as the geometry type or test may contain wildcards.

3.2.2 Dynamic evaluation
If no error is detected during the static analysis phase, the processing enters into the second phase, i.e., dynamic evaluation. Semantically, the dynamic evaluation process is:

\[
\begin{align*}
\text{statEnv} & |- \{\text{GeometryTest}\}_{\text{geometrytype}} = \text{GeometryType} \\
\text{statEnv} & |- \text{GeometryValue matches GeometryType} \\
\text{dynEnv} & |- \text{test GeometryTest with "geometry" of GeometryValue} \Rightarrow \text{GeometryValue}
\end{align*}
\]

The geometry test is normalized to the equivalent geometry type by applying the \([\_\_\_\_\_\_]_{\text{geometrytype}}\) normalization rule. If the value matches the type, then the judgment yields the value, otherwise the judgment yields the empty sequence.

\[
\begin{align*}
\text{statEnv} & |- \{\text{GeometryTest}\}_{\text{geometrytype}} = \text{GeometryType} \\
\text{statEnv} & |- \text{not(\text{GeometryValue matches GeometryType})}
\end{align*}
\]

\[
\text{dynEnv} |- \text{test GeometryTest with "geometry" of GeometryValue} \Rightarrow ()
\]

Formal semantics complements the GQL specification, and provides a reference for implementation of GQL query processing engine. Fig. 2 shows GQL query processing architecture. On one hand, the GQL query expression is parsed to an abstract syntax tree; on the other hand, the input document and the optional GML schema are parsed in streaming fashion by using SAX (simple API for XML), then loaded in memory as GML data model instance. Finally, the query is applied to the data model instance to yield an expecting result, or an error.

![GQL Query Processing Architecture](image)

**Fig. 2 Architecture of GQL query processing**

4 Example
Fig. 3 is a GML fragment used to show how to represent GML document by the extended data model, and this fragment is extracted from the "cambridge.xml", which conforms to "city.xsd" defined in Reference [17].

Fig. 4 shows the tree representation of the GML fragment shown in Fig. 3. The values E1, E2, etc. represent element nodes, the values G1, G2, etc. represent geometry nodes, the value A1 represents attribute node, and the values T1, T2, etc. represent text nodes. Each node has its accessor, for example, node G1 can be described as follows:
\[
\text{dm; srs-name (G_2) = http://www.opengis.net/srs.epsg.xml#4326;}
\]

\[
\begin{align*}
&\text{dm; node-kind (G_2) = "geometry";} \\
&\text{dm; node-name (G_2) = "LineString";} \\
&\text{dm; geometry-type (G_2) = "LineStringType";} \\
&\text{dm; dimension (G_2) = 1;}
\end{align*}
\]

Fig. 3 A GML fragment cityMember

\[
\begin{align*}
&\text{dm; parent (G_2) = ([G_1]);} \\
&\text{dm; children (G_2) = ([G_3],[G_4],[G_5]);} \\
&\text{dm; attributes (G_2) = ([A_1]).}
\end{align*}
\]

Fig. 4 Mapping GML fragment cityMember to GQL data model

Fig. 5 shows a query that is to retrieve the roads whose length is not less than 10 km of in the city. In this example, the Length() function is used to implement spatial evaluation and analysis.

Static type analysis checks an expression that it is type correct, and if so, determines its static type. It can result in a static error if the expression is not type correct. For instance, a comparison between an integer value and a string value is an error that can be detected during static type analysis. If static type analysis succeeds, it results in an abstract syntax tree where the top-level core expression and each sub-expression are annotated with its static type.

In dynamic evaluation phase, the value of an expression is computed. The dynamic semantics is defined only on core expressions. Evaluation proceeds by applying value inference rules to the abstract syntax tree of a core expression, starting with the top-level query expression and conditionally applying inference rules to sub-expressions top down and synthesizing values bottom up. The dynamic semantics guarantees that ev-
5 Conclusions

In this paper, a new GML query language GQL is proposed by extending the XQuery. The data model, algebra, and semantics of XQuery are extended to support spatial analysis and query processing on GML document in GQL. The geometry node, geometry type and its subtypes such as Point type, LineString type, LinearRing type, Polygon type are added. Moreover, the predicates such as Disjoint, Touches, Crosses, Within, Overlaps, Contains, Intersects, and a set of operations on geometry objects, and the formal semantics on geometry to ensure the correctness of query and to provide a reference for implementation are given. An example of GQL is also shown in this paper. The future work will focus on further topological and temporal extensions and completely implementation of GQL.

REFERENCES

[1] OGC (2004) Geography markup language (GML) implementation specification Version 3.1.0[OL]. http://portal.opengis.org/files/?artifact_id=4700
[2] Goldman R, McHugh J, Widom J (1999) From semistructured data to XML: migrating the lore data model and query language [C]. ACM SIGMOD Workshop on the Web and Databases, Philadelphia, PA
[3] Robie L J, Lapp J, Schach D (1998) XML query language (XQL) [C]. Query Languages Workshop, Boston, Massachusetts
[4] Deutsch A, Fernandez M, Florescu D, et al. (1998) XML-QL: a query language for XML[C]. Query Languages Workshop, Boston, Massachusetts
[5] Chamberlin D, Robie J, Florescu D (2000) Quilt, an XML query language for heterogeneous data sources [C]. ACM SIGMOD Workshop on the Web and Databases, Dallas, Texas, USA
[6] Egenhofer M J (1994) Spatial SQL: a query and presentation language([C]. IEEE TKDE, 6(1), 86-95
[7] Wang Feng, Sha Jichang, Chen Huowang, et al. (2000) GeoSQL: a spatial query language of object-oriented GIS[C]. The 2nd International Workshop on Computer Science and Information Technologies, Ufa, Russia
[8] Shekhar S, Vatsavai R R, Sahay N, et al. (2001) WMS and GML based interoperable Web mapping system[C]. The 9th ACM International Symposium on Advances in Geographic Information Systems, Atlanta, GA
[9] Corcoles J E, Gonzalez P (2001) A specification of a spatial query language over GML[C]. The 9th ACM International Symposium on Advances in Geographic Information Systems, Atlanta, GA
[10] Beech D, Malhotra A, Rys M (2000) A formal data model and algebra for XML[OL]. http://elib.cs.berkeley.edu/seminar/2000/20000207.pdf
[11] Vatsavai R R (2002) GML-QL: a spatial query language specification for GML[OL]. http://www.cobblestoneconcepts.com/ucgis2summer2002/vatsavai/vatsavai.htm
[12] W3C (2005) XQuery 1.0: an XML query language [OL]. http://www.w3.org/TR/2005/WD-xquery-20050211
[13] Boucelma O, Colonna F M (2004) GQuery: a query language for GML[C]. The 24th Urban Data Management Symposium, Chioggia-Venice, Italy
[14] Fankhauser P, Fernandez M, Malhotra A, et al. (2000) The XML query algebra[OL]. http://www.w3.org/TR/2000/WD-query-algebra-20001204
[15] OGC (1999) OpenGIS simple features specification for SQL[OL]. http://www.opengis.org/techno/specs/99-049
[16] W3C (2005) XQuery 1.0 and XPath 2.0 formal semantics[OL]. http://www.w3.org/TR/2005/WD-xquery-semantics-20050211
[17] OGC (2001) Geography markup language (GML) 2.0[OL]. http://www.opengis.net/gml/01-0291GML2.html