2010 International Conference on Computer and Communication Technologies in Agriculture Engineering

# Querying GML Documents: An XQuery based Approach

Fubao Zhu
School of Computer and Communication
Zhengzhou University of Light Industry
Zhengzhou, 450002, China
fbzhu@zzuli.edu.cn

Hui Chen, Jihong Guan
Dept. of Computer Science and Technology
Tongji University
Shanghai, 201804, China
{chenhui, jhguan}@tongji.edu.cn

***Abstract*—GML is an XML encoding for representation and exchange of geographic information. With the popularity of GML, more and more spatial data exists in GML format. It raises the problem of how to query the data in GML. Since GML is based on XML, query technologies for XML can be employed to GML. However, the fact that GML differs from XML in many aspects makes these technologies not so applicable for GML. The standard query language XQuery, which is recommended by W3C, only applies to GML non-spatial queries. To query GML documents, we propose a GML query processor by extending XQuery in datatypes and spatial operators. The processor can deal with non-spatial and spatial queries, and output the query results in GML format.**

***Keywords-GML; XML; XQuery; GML Query Processor; GQL***

## I. INTRODUCTION

The Geography Markup Language (GML) is an XML encoding in compliance with ISO 19118 for the transport and storage of geographic information modeled according to the conceptual modeling framework used in the ISO 19100 series and including both the spatial and non-spatial properties of geographic features [1]. As more and more spatial/ geographic 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 import 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. Current specifications or approaches for handling GML documents either require implementation from scratch or lack corresponding data model, algebra, and formal semantics. Even approaches working well with XML, such as Lorel [2], XQL [3], XML-QL [4] and Quilt [5] etc., could not guarantee good results when applied to GML documents that contain both alphanumeric and spatial data. Spatial querying approaches like SpatilSQL [6], GeoSQL [7] are proposed for relational database system, thus not suitable for GML documents.

XQuery [8] is a powerful XML query language recommended by W3C, and is becoming the de facto standard. However, it does not support spatial query and analysis. Research shows that XQuery is more flexible and extensible than any other XML query language [9], so it is reasonable to extend XQuery for querying GML documents.

In order to support GML query, we design a language specification called GQL [10] by extending the XQuery, and a GML query engine is implemented in this paper to verify the proposed specification.

The rest of the paper is organized as follows. In section 2, the related work of GML querying and processing is depicted. The architecture of our GML query engine is described in section 3. In section 4, the experiments are given according to the proposed approach. Finally, the conclusions are given in section 5.

## II. RELATED WORK

The features of GML documents lead to new challenges of querying and integrating those documents. Up to date, there have been some efforts to establish query language for GML documents.

Shekhar[11] described 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 by GML. They gave computational cost models for single scan and multi-scan query-processing strategies by using SAX and DOM parsers. However, they didn't develop any query language for GML documents.

Corcoles[12] proposed 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 [13] to support spatial features. But the proposed language does not conform to standard XML query languages, and it requires implementation from scratch.

Vatsavai[14] described a spatial query language GML-QL that is derived from XQuery. He roughly classified spatial queries into unary and binary types, and defined various spatial functions and operations to support spatial query and analysis. Unary and binary spatial functions can occur in FOR, LET, WHERE clauses and topological operations can be applied in a WHERE clause. Depending on the requirement, these functions can take either scalar, or collections (sets, lists, and bags) as arguments. The possible return values are BOOLEAN, scalar, or collections. Except for some query examples, Vatsavai did not provide detailed information about GML-QL.

Boucelma [15] also proposed a query language for GML called GQuery based on XQuery. However, GQuery has only limited extension to XQuery, and its main goal seems to support spatial information integration, instead of a general-purpose query language.

To query GML documents, we propose a new query language named GQL (an abbreviation of GML Query Language) by extending the XQuery language. We extend the XQuery in five aspects: data model, algebra, functions and operations, and formal semantics. The details of the proposed approach are given in the following sections.

978-1-4244-6947-5/10/$26.00 ©2010 IEEE

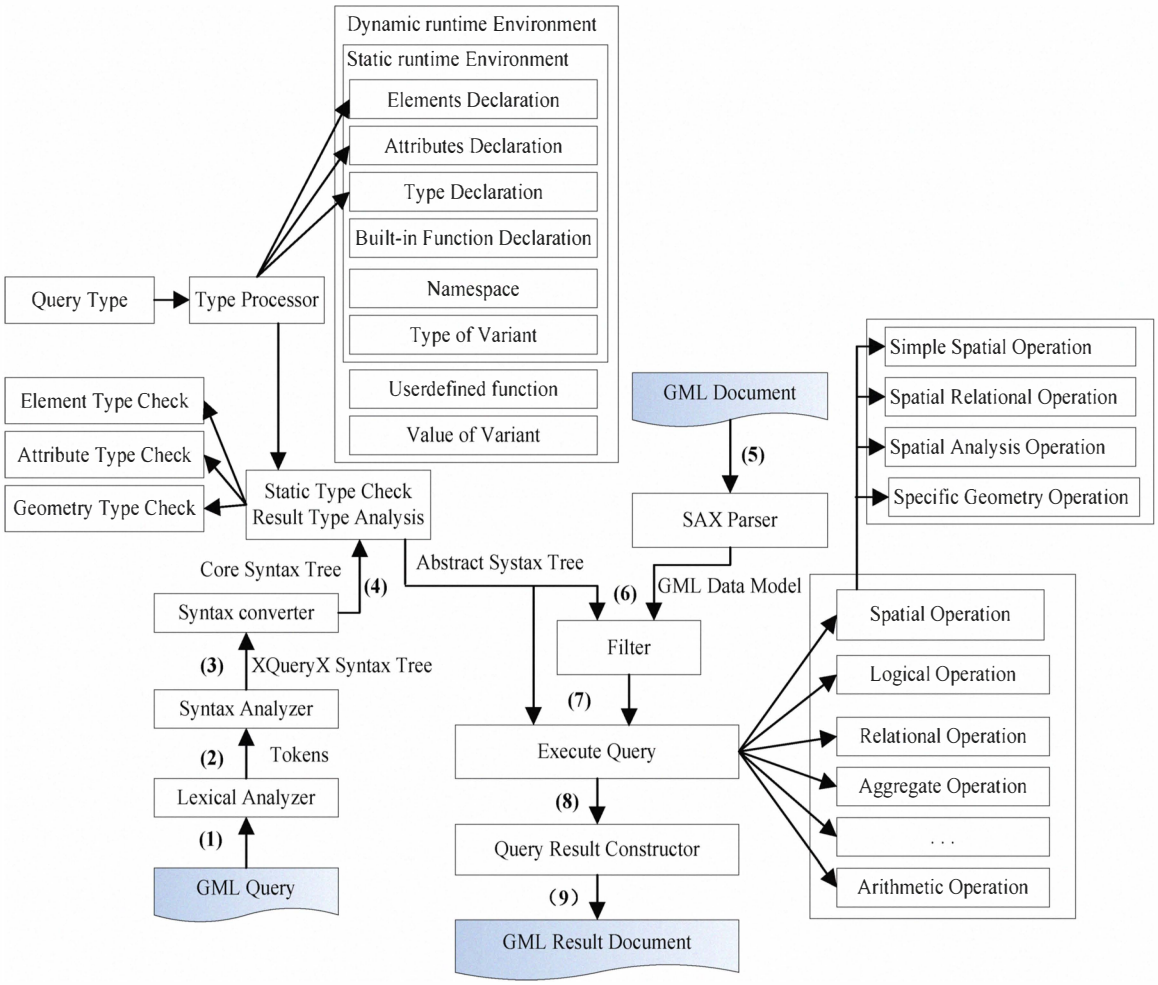


**Figure 1. Architecture of GML processor**

## III. THE TECHNIQUES

### *A. Architecture*

Figure 1 describes the detailed structure of GML query processor. The query process has two phases. One is the static analysis phase, and the other is the dynamic execution phase. The following is the analysis of these two phases. The static analysis phase has four steps:

Tag (1) marks up the lexical analysis procedure. For the input query sentences, the lexical analyzer identifies tokens and their types based on lexical rules.

Tag (2) notes syntax analysis procedure. The syntax analyzer transforms the XQuery token stream into XQueryX productions to produce XQueryX syntax tree. XQueryX is the XML representation format of XQuery. Comparing to XQuery, XQueryX is easier to parse and better to represent the underlying structure of query sentences. Directly mapping the abstract syntax of XQuery into XML productions will produce XQueryX expressions.

Tag (3) records syntax conversion procedure. The syntax converter is used to convert the XQueryX syntax tree into core syntax tree.

Tag (4) is the procedure of static type analysis. This procedure checks whether the query expressions are type-safe and determines their static types. The static type analysis is composed of type processing, static type checking and query result analyzing. The type processor transforms the type description into type environment for type checking and analyzing. The static type environment is used for mapping type information. If the static analysis succeeds, it will return the type-determined syntax tree of every expression.

The dynamic execution phase is the procedure of executing queries on GML documents. The following is the description of dynamic execution phase.

Tag (5) marks up document parsing procedure. The DOM parser parses the input GML document and constructs the corresponding DOM tree. Tag (6) is the procedure of filtering. The filter cuts the DOM tree based on the query syntax tree. The filter only preserves paths involved in the query sentences and outputs the pruned DOM tree.

Tag (7) is the module of query execution. The inputs of this module are preprocessed XQueryX syntax tree and pruned DOM tree. Query executor searches on the DOM tree and computes on the XQueryX syntax tree to get the query results. All the operators needed in the computation are existed in the form of inner functions. They are combined with the specific functions during the compiling time. We can get the query results by directly calling those functions. The spatial operators are also added in the form

of inner functions to handle the spatial query. Tag (8) and (9) mark up the process of result constructing. The constructor outputs the result in the form of GML document.

The GML queries can be divided into three types according to Corcoles:

Queries only concern non-spatial attributes querying. This kind of queries is called non-spatial query.

Queries only concern spatial attributes and spatial relations querying. This kind is called spatial query.

Queries mix non-spatial and spatial querying. This kind is called mixed query.

XQuery can only accomplish non-spatial queries of GML documents. The GML query processor extends the datatypes of XQuery, and adds spatial operators to implement spatial queries and mixed queries.

### *B. Spatial Extension*

The GML query processor is base on XQuery, so it can directly query non-spatial information. To query spatial information in the GML document, the query processor adds the spatial datatypes and spatial operators we defined above.

The spatial datatypes we defined are: Geometry, Coord, Coordinates, Point, LineString, LinearRing, Polygon, Box, GeometryCollection, MultiPoint, MultiLineString, and MultiPolygon. All these datatypes consist with the datatypes defined in the OGC Simple Features Specification (SFS). The spatial operations we defined are: simple geometry operation, spatial relation operation, spatial analysis operation and specific geometry operation.

JTS [16] is the abbreviation of Java Topology Suite. It is an API of spatial predicates and functions. The GML query processor calls the spatial functions in JTS to accomplish the computing on document tree when it encounters spatial query.

### *C. Format conversion*

The query processor calls the spatial function in JTS to handle spatial query. However, the parsed GML documents exist in the form of document trees. It is impossible to deliver the document trees to those spatial functions. The prototype of GQUERY converts the format of GML data to handle the spatial operations. Our GML query processor also uses this method. Figure 2 shows the format conversion procedure.

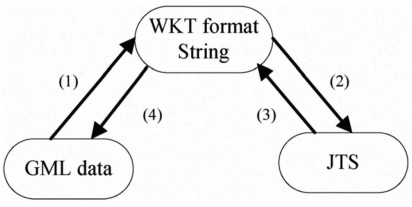


Figure 2. Format conversion

When calling the spatial functions in JTS, the parser converts the data in the GML document tree to the format of string recognized in JTS. This kind of format is called WKT format (Well Known Text Format). For example, a LineString is represented by LINESTRING (0 50, 70 60, 100, 50) in WKT format. The processor then calls the spatial functions in JTS to do the spatial query.

```
GMLtoWKTF(xmlNode) //convert GMLdata to WKT format string
Input: XMLNode xmlnode, Output: String wktf
1.  get the local name str of xmlnode;
2.  if str equals Point or LineString or LinearRing then // simple geometry type node
3.    append the value of getWKTF(xmlnode) to wktf;
4.  end if
5.  else //complex geometry type node
6.    append the upper case of str to wktf;
7.    get child node list of xmlnode;
8.    for each node childNode in child node list
9.      append the value of getWKTF(childNode) to wktf;
10. end else
11. return wktf
12.
13. Function getWKTF(XMLNode node)  //get WKT format string of simple node
14. String str;
15. append the upper case of local name of node to str;
16. get child node childNode of node;
17. if childNode's local name equals Coord then
18.   append all the cooord value of childNode to str;
19. end if
20. else
21.   append all the coordinates value of childNode to str;
22. end else
23. return str;
```

Figure 3. Format conversion algorithm

Figure 3 shows the format conversion algorithm. For the length of this paper, here we only describe the conversion algorithm of GML data to WKT format string. The conversion of WKT format string to GML data is an inverse process. Line 1-2 checks whether the input node is simple geometry type node or not, that is, the type is Point, LineString or LineRing. The judgment criterion is whether this kind of geometry type comprises coordinate values directly. If it is a simple geometry type node, we call the getWKTF function to get the WKT format coordinate values. Line 5-10 handles the complex geometry type node, that is, the type is Polygon, Box, MultiPolygon, MultiLineString, MultiPolygon or GeometryCollection. For the complex geometry type node, we call the getWKTF function on every sub node of this node to get the WKT format string.

## IV. PERFORMANCE EVALUATION

### *A. Platform*

All the experiments are run on a Window XP machine with an AMD 4000+ 2.10G Processor and 1.78GB main memory. The GML query processor is implemented in Java by using Eclipse and JDK 1.6.

We use query processor itself to measure the query time. The consumed memory, which is composed of query consumed memory and Java virtual machine used memory, is measured by Process Explorer [17].

### *B. Data sources*

All the GML documents used in the experiments are ArcView3.2 and MapInfo6.0 instance documents. These documents vary from 50KB to 1500KB and contain all kinds of spatial information. We use GQL defined above to construct the GML query documents.

### C. *Experimental results and analysis*

We use GQL to construct three kinds of queries, which are non-spatial query, spatial query and mixed query. The most common query expression in GQL is FLWR (For-Let-Where-Return) expression. The query documents we constructed only contain FLWR expressions. We use For/Let to combine the same document node for these three kinds of queries. The only difference is the sentences of Where and Return. The non-spatial query is restricted on non-spatial attributes and returns only non-spatial attribute information. The spatial query only contains spatial operators and returns only spatial attribute and relation information. The mixed query combines the above two kinds of queries. It is worth mentioning that the query documents we design include all the spatial operators above. For all these three kinds of queries, the query processor can return the right results.

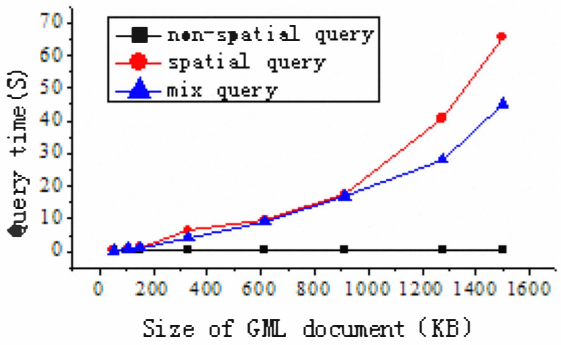


Figure 4. Query time elapsed

Figure 4 shows the variation of query time as a function of the GML document size and Figure 5 shows the consumed memory for each query. It need a lot of time and memory to do coordinate computing when do spatial query, so the time and memory consumed by non-spatial query are high below of spatial query and mixed query. The mixed query consumes a little less time and memory than spatial query since the non-spatial attributes can filter part of candidate results.

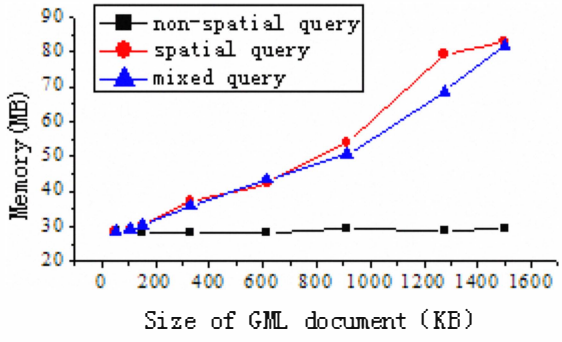


Figure 5. Memory consumption

## V. CONCLUSION

In this chapter, GQL is proposed as a GML query language by extending the XQuery. We extended the data model, algebra and semantics of XQuery to support spatial analysis and query processing on GML document. Future work will focus on two aspects: on one hand, we will do further extensions to GQL to support topological and temporal operations; and on the other hand, we will try to completely implement this language.


## ACKNOWLEDGMENT

This work was supported by the Doctoral Research Fund of Zhengzhou University of Light Industry (No. 2008BSJJ012) and the Open Research Program of Key Lab of Earth Exploration & Information Techniques of Ministry of China (No. 2008DTKF008).


## REFERENCES


[1] C. Portele (Ed.). OpenGIS® Geography Markup Language (GML) Encoding Standard. Version: 3.2.1, http://www. opengeospatial. org/standards/gml, 2007.

[2] S. Abiteboul, D. Quass, J. Mchugh, and et al. The Lorel Query Language for Semistructured Data. International Journal on Digital Libraries, 1(1): 68-88, 1997.

[3] L. J. Robie, J. Lapp, and D. Schach. XML Query Language (XQL). Proceedings of Query Languages Workshop. Boston, Massachussets, USA, Dec. 3-4, 1998.

[4] A. Deutsch, M. Fernandez, D. Florescu, and et al. XML-QL: A Query Language for XML. Proceedings of Query Languages Workshop. Boston, Massachussets, USA, Dec. 3-4, 1998.

[5] D. Chamberlin, J. Robie, and D. Florescu. Quilt: An XML Query Language for Heterogeneous Data Sources. Proceedings of ACM SIGMOD Workshop on The Web and Databases (pp. 1-25). Dallas, Texas, USA, May 18-19, 2000.

[6] M. J. Egenhofer. Spatial SQL: A Query and Presentation Language. IEEE Transactions on Knowledge and Data Engineering, 6(1): 86-95.

[7] F. Wang, J. C. Sha, H. W. Chen, and et al. GeoSQL: A Spatial Query Language of Object-oriented GIS. Proceedings of International Workshop on Computer Science and Information Technologies, 2000.

[8] W3C (2005). XQuery 1.0: An XML Query Language. http://www.w3.org/TR/2005/WD-xquery-20050211/.

[9] A. Bonifati and S. Ceri. Comparative Analysis of Five XML Query Languages. ACM SIGMOS Record, vol. 29 (pp. 68-79), 2000.

[10] J. Guan, F. Zhu, J. Zhou and et al. GQL: Extending XQuery to Query GML Documents. Journal of Geo-Spatial Information Science, 9(2): 118-126, 2006.

[11] S. Shekhar, R. R. Vatsavai, N. Sahay, and et al. WMS and GML based Interoperable Web Mapping System. Proceedings of the 9th ACM International Symposium on Advances in Geographic Information Systems, ACM Press, 2001.

[12] J. E. Corcoles, and P. Gonzalez. A Specification of a Spatial Query Language over GML. Proceedings of the 9th ACM International Symposium on Advances in Geographic Information Systems. ACM Press, 2001.

[13] D. Beech, A. Malhotra, and M.Rys. A formal data model and algebra for XML. University of California, Berkeley, CS 298-13, Digital Library Seminar, 2000.

[14] R. R. Vatsavai. GML-QL: A Spatial Query Language Specification for GML. http://www. cobblestoneconcepts.com/ucgis2summer 2002/vatsavai/vatsavai.htm, 2002.

[15] O. Boucelma and F. M. Colonna. GQuery: a Query Language for GML. Proceedings of 24th Urban Data Management Symposium, Chioggia-Venice, Italy, October 27-29, 2004.

[16] JTS. Java Topology Suite. http://www.vividsolutions.com/JTS/ JTSHome.htm, 2004.

[17] PE. Process Explorer. http://technet.microsoft.com/zh-cn/ sysinternals/default(en-us).aspx, 2007.