Analysis of Different Approaches for Storing GML Documents

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

ABSTRACT
The fact that GML is an XML encoding allows it to be queried. In order to query a GML document we have designed a query language over GML/XML enriched with spatial operators. This query language has an underlying data model and algebra that supplies the semantics of the query language. In order to use this query language, it is necessary to find an implementation that allows us to exploit all its features, storing GML documents efficiently. The general aim of this paper is to study the behaviour of different alternatives over XML documents (alphanumeric data) applied to store and query GML documents (alphanumeric and spatial data). The alternatives selected use relational schemas to store GML documents because they use a complete set of data management services (including concurrency control, crash recovery, scalability, etc) and benefit from the highly optimised relational query processor. Three approaches have been used: LegoDB, a structure-mapping approach, and two simple model-mapping approaches, Monet over Relational database and XParent. We focus on the effectiveness of storage models in terms of query processing. A performance study is conducted using three data sets and the experimental results are given.

Categories and Subject Descriptors
H.3.4 [**Information Storage and Retrieval**]: Systems and Software – *performance evaluation (efficiency and effectiveness).*

General Terms
Performance, Design, Languages.

Keywords
GML, Spatial Queries.

1. INTRODUCTION
The Geographical Markup Language (GML) is an XML encoding for the transport and storage of spatial/geographic information, including both spatial features and non-spatial features [1]. The mechanisms and syntax that GML uses to encode spatial information in XML are defined in the specification of OpenGIS [2]. OGC manages consensus processes that result in interoperability between diverse geoprocessing systems. Therefore, GML proposes a standard format to represent the spatial information with XML.

The fact that GML is an XML encoding allows it to be queried. In order to query a GML document we have developed a query language over GML/XML enriched with spatial operators [3]. This query language has an underlying data model and algebra that supplies the semantics of the query language [4]. However, to use this query language, it is necessary to find an implementation that allows us to exploit all its features.

In order to implement our query language, efficient storage of GML documents is necessary. The general aim of this paper is to study different alternatives to store and query GML documents. In order to do this, we have used well-known approaches to the querying of XML documents (with only alphanumeric data) to apply to GML documents. In this paper we show that this is not a trivial problem because, due to the resources required to query and store spatial elements, all approaches that obtain good results with alphanumeric operators do not obtain good results when combined with spatial operators. On the other hand, not all alternatives are valid if they are applied to GML.

Many approaches to store and retrieve XML documents have been carried out in previous studies. On the one hand, there are several database management systems to store XML documents which have been developed, like [5]. On the other, there are several approaches based on the Relational model or Object-Oriented model. When XML documents are stored in off-the-shelf database management systems, the problem of storage model design for storing XML data becomes a database schema design problem. In [6], the authors categorise such database schemas into two categories*: structure-mapping approach* and *model-mapping approach.* In the former the design of the database schema is based on the understanding of DTD (Document Type Descriptor) or XML Schema that describes the

structure of XML documents. Examples include [7][8][9][10][11]. In the latter a fixed database schema is used to store any XML documents without the assistance of XML Schema (DTD), such as [12][6][13][14][15]. Again, commercial database proposes particular alternatives to store XML documents, such as [16][17][18].

In this paper, we tend to use approaches based on relational databases to store GML documents. In this way, we can use a complete set of data management services (including concurrency control, crash recovery, scalability, etc) and benefit from the highly optimised relational query processor. In addition, RDBMS allows us to store spatial objects according to the possibilities offered in [19]. The disadvantage of our approach is that the relational databases have been built to support structured data and the requirements of processing XML data (semi-structured) are vastly different from the requirements for the processing of traditional data.

The paper is organised as follows: Section 2 briefly introduces three existing approaches with a brief discussion; some results of our performance studies are presented in Section 3 and 4; and Section 5 covers conclusions and future projects.

2. THREE EXISTING APPROACHES

Among all the alternatives to store XML document mentioned above, three approaches have been selected to carry out the study: (1) LegoDB [7] inspired in the solution proposed by Shanmugasundaram et al. [11] (*structure-mapping approach*), (2) Monet [14] and (3) XParent [12] (*model-mapping approach*). These approaches have been selected because they possess five important features:

(i) they obtain good performances over XML documents with alphanumeric data (no spatial data).

(ii) they support or could be updated easily to store spatial information: an efficient storage of spatial information offers us chance to create a spatial index for each set of spatial objects in our document (which is not a trivial solution if we are speaking about storing XML).

(iii) the storage of spatial objects in line with the possibilities offered in [OGC99] must be supported. Due to the characteristics of many of the above-mentioned approaches, this possibility is not allowed for all existing approaches.

(iv) to allow all features of our spatial query language [4]

(v) not to depend on a particular RDBMS to implement this approach.

XRel [6] is not included in this study because although it has the second feature, in comparison with XParent, XRel obtains a lower performance [12]. The *Edges* approach [15] is not included because it stores all values in the same column and so needs to make a type coercion in order to compare value with type difference in the query. For this reason (absence of second feature), the *Edges* approach is not a good solution in our case . The commercial solutions, e.g. Oracle [*16*], DB2 [18] do not satisfy some requirements mentioned above: storing GML documents in LOBs does not allow the spatial information to be indexed correctly (it is necessary to retrieve spatial information efficiently [20]) and, the absence of a definitive standard to define these alternatives makes it impossible to use Object-Relational/Relational storage without depending on a commercial database.

2.1 XParent. 3º Approach

LegoDb and Monet have not been modified to support spatial objects, but XParent needs to be modified. In this section, we show this modification.

XParent [12] is a four table database schema, *LabelPath*, *DataPath*, *Element* and *Data* as follows.

LabelPath (ID; Len, Path) ; DataPath (Pid,Cid) ; Element (PathID, Did, Ordinal) ; Data (PathId, Did, Ordinal, Value)

In the *Data* table all values are stored as text, e.g. XParent converts the values of different types (integer, string, real,…) in values of type string. We have eliminated this limitation storing the spatial objects in different tables. In this way, the simple values continue being stored in the *Data* table, and the complex objects (spatial objects) are stored in other table (the number of tables equals the number of distinct label-paths with spatial objects, like Monet). Thus, we could store different spatial objects (*state boundaryBy, parcel extendof,* …) in different tables with different spatial index. The structure of the new table is equal to the Data table, but the type of the Value column depends on the type of the spatial object that will be stored.

3. A PERFORMANCE STUDY

In this section, in order to assess what is the best approach to query efficiently GML documents, experimental studies have been conducted. Unlike previous studies that emphasise converting XML documents to/from the database schema and translation of queries into SQL queries, in this section, we focus on the effectiveness of storage models in terms of query processing. All the experiments were conducted on an 800Mhz PC with 128Mb RAM, 30Gb hard disk with operative system Windows NT.4.0 Server SP6. The RDBMS used was Oracle 8.1.5 which we selected because it allows the storage of spatial objects and the application of spatial operators over them. Oracle's Spatial object-relational model was used (SDO_GEOMETRY object) [17] for simplicity. However, as mentioned above, the Oracle's relational model or another RDBMS may be used.

We carried out our experiments using our own benchmark and data sets. We did not use other existing benchmarks for XML documents, like [21][22] because these benchmarks are oriented to work with XML documents (only alphanumeric data) and the retrieval and application of spatial objects was not contemplated. In the same way, we used our own data set, because the data sets (DBLP Bibliography, Bosak Shakespeare collection, etc) used in the majority of related works have only alphanumeric data. However, we used the features needed by a benchmark for XML, detailed by [23]. On the other hand, our experiments were aimed at studying the performance of these approaches (LegoDB, Monet and XParent) in the execution of spatial queries and retrieval of spatial and alphanumeric data. Other features, (reconstruction of documents, loading of documents in the model, etc) were not included in this experiment because these problems

have been thoroughly studied in each model, independently of the spatial data.

Three different size of the data set are used, namely D1 (7.1 Mb aprox. 5000 rectangular parcels), D2 (15.4 Mb aprox 10000 rectangular parcels) and D3 (30 Mb aprox. 20000 rectangular parcels) The data size of D2 is double that of D1 and the data size of D3 double that of D2.

For this test, the same tables as shown in Section 2 have been created. For each approach, indexes on relational tables were also properly built to improve query processing as follows:

-In XParent, we created indexes as proposed in [12]. In the new relations for the storage of spatial objects (added in our modification) we have used the same indexes (*B*-tree*) for the attributes *Pathid*, *Ordinal* and *Did* as in the *Data* relations. In the *Value* attributes we have built indexes for the spatial attributes (*Hybrid* [17]).

-In LegoDB the attributes: *TState(state_id), Tcitymember (Citymember_id),TBlock(block_Id), TBlockmember(blockmember _id),TParcel (parcel_id), TArquitet (arquitect_id)* are primary key *TCitymember(parent_state_id), TBlock(parent_Citymember_id), TBlockmember(parent_block_id), TParcel (parent_blockmember _id), TArquitet (Parent_parece_id)* are foreign key. The Spatial Attributes TState(BoundaryBy), TBlock(BoundaryBy,Extendof) and TParcel (extendof) have been also indexed with *Hybrid* indexing.

-In Monet, we created indexes (*B*-tree*) on *source*, *target*, *id* and *value* attributes to speed up the joins for each table. The Spatial attributes (value) have also been indexed with *Hybrid* indexing.

3.1 Comparative Study

In this Section, we study the three approaches mentioned above. In order to analysis implementation alternatives for a query spatial language over XML, we have focused our attention on two important problems:

1. Since we are dealing with an XML query language, a very large number of joins are necessary. It is one of the more important performance problems in the relational implementation of these query languages [24].

2. Storing and querying spatial data require a larger amount of resources than storing and querying alphanumeric data [25].

For these reasons, our work concentrates upon how a RDBMS can retrieve queries, which include these features. First, a set of queries is used to study the behaviour of these approaches, involving only alphanumeric data (these approaches have not been directly compared before). In addition, the user may only wish to query alphanumeric data of a GML document. Secondly, a set of queries with spatial and alphanumeric operators is used. The queries are listed in table 1 (following page):

Q1-Q10 are queries with only alphanumeric operators. Q1-Q3 queries with different length path are shown with 1 attribute in *select* clause and 1 attribute in *where* clause. Q4 and Q5 show queries with 3 attributes (1 in Select and 2 in Where) and 4 attributes (2 in Select and 2 in Where) respectively. In Monet and XParent approaches, the addition of new attributes force us to include new relations (more joins) in the equivalent SQL. In Q6-Q8 joins between attributes have been added. Q6 use 1 joins and 1 condition, Q7 uses only one joins and Q8 uses three join links to *OR* operator. Q9 and Q10 are queries with grouping. Q7 is grouped by 1 attribute and Q8 is grouped by 2 attributes.

The elapsed query times for these queries (with D1data set) are shown in Figure 1. In Table 2, the number of joins necessary in each approach is shown. For LegoDB the number of joins has a limit (5), but in the other approaches, it depends on the number of attributes participating in the query. In any case, the comparison between the joins for each approach is only approximate, as the size of relations joined varies in each approach.

Table 2: Number of joins in each SQL queries

LegoDB (LDB) ; Monet (Mon);Xparent (XPt)

Query	*LDB*	*Mon*	*XPt*	*Query*	*LDB*	*Mon*	*XPt*
Q1	0	1	5	*Q11*	0	2	5
Q2	2	3	7	*Q12*	4	7	9
Q3	5	6	10	*Q13*	5	9	12
Q4	5	6	13	*Q14*	3+1	10+1	21+1
Q5	5	8	16	*Q15*	5	11	19
Q6	3+1	7+1	15+1	*Q16*	5	14	29
Q7	5+1	7+1	13+1	*Q17*	2+1	7+1	10+1
Q8	5+3	10+3	23+3	*Q18*	3+2	10+2	21+2
Q9	5	8	16	*Q19*	5+2	16+2	32+2
Q10	5	7	13				

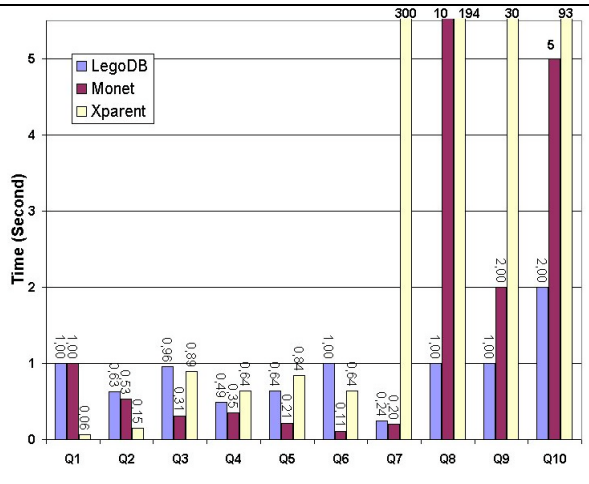


Figure 1: Elapsed query time (Q1-Q9)

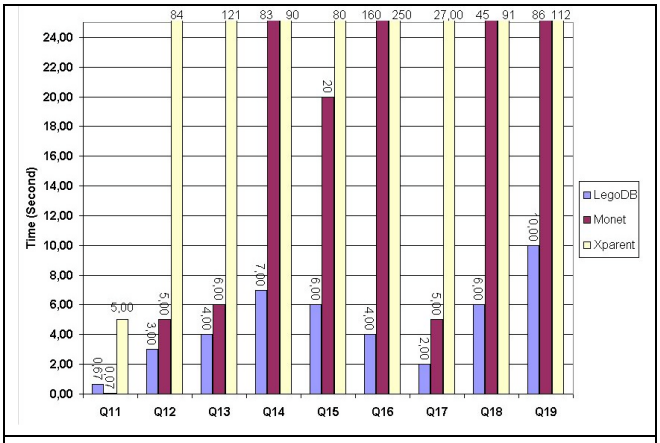


Figure 2: Elapsed query time (Q11-Q19)

For queries that only contain short simple paths, such as Q1 and Q2 XParent obtains the best elapsed times. However, as the complexity of the query increases, the XParent elapsed time also increases (in Q5-Q6 it obtains the worst elapsed time). In general, Monet obtains the best results from Q1-Q6. Although LegoDB uses fewer joins that Monet, the size of relations that is joined is higher in LegoDB. When the queries have joins, Q7-Q8, XParent obtains worse elapsed time (non-viable with respect to the other approaches). In this case, Monet also registers a higher elapsed time than LegoDB. In queries with grouping (Q9-Q10) XParent registers a higher elapsed time and Monet obtains a worse elapsed time than LegoDB. In general, in complex queries, XParent obtains the worst results in our test.

Q11-Q20 are queries with spatial operators, some of which have spatial and alphanumeric operators. These queries have been built varying the number of joins. In this way, we are able to study the elapsed times in relation to the two problems mentioned above. Q11-Q13 queries with different length path and different spatial operators (spatial analysis) are shown. Q14-Q19 are queries with a higher number of joins (Table 2). Operators that support spatial analysis and joins between alphanumeric attributes are included in Q14. Q15-Q16 window query is used (all objects that lie within a window). In Q15 there is one *none equi-join* [17] and in Q16 there are three *none equi-joins*. In Q17-Q19 spatial joins are used. In addition these are combined with joins between alphanumeric attributes (Q18) and Analysis operators and *none equi-joins* in (Q19). Figure 2 shows the query elapsed times. LegoDB and Monet outperform the modification of XParent for all queries. In addition these elapsed times are non- viable with respect to the other approaches. On the other hand, LegoDB outperforms Monet for all queries except Q11 (simple path). In Q14, LegoDB significantly outperforms Monet (up to 11 times faster). Queries with a high number of joins with window queries (Q15,Q16) and spatial joins (Q18,Q19) obtain worse results in Monet than in LegoDB.

As a conclusion, Monet with alphanumeric operators obtains better elapsed times than LegoDB i.e., in order to obtain a good performance querying spatial XML documents, it is preferable to increase the size of relations (LegoDB), and to reduce the number of joins between relations (Monet). A high number of joins and spatial operators require many resources to obtain a good performance. However, when only alphanumeric data are queried (with no joins and with no *group by* clause), the best option is to reduce the size of relations and increase the joins. In the modification of XParent a good performance is obtained only when simple alphanumeric queries are run.

4. SCALABILITY TEST

Due to the bad results obtained by the modification of XParent, in this section we only investigate the scalability of LegoDB in comparison with Monet using D1, D2 and D3 data set detailed above. The queries used in the study are a selection that gather the most important features mentioned above: Q5, Q8, Q10, Q12, Q13, Q17, Q18, Q19.

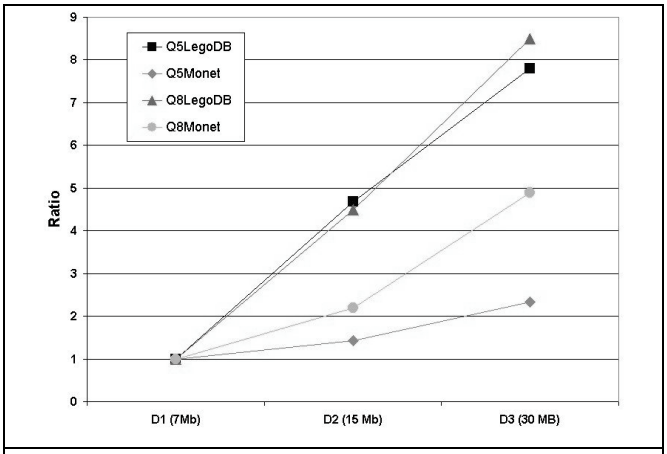
Figure 3 shows the elapsed time ratios for LegoDb and Monet grouped by items of two, where the x-axis shows the data size for D1 (7.1 Mb), D2 (15.4 Mb) and D3 (30 Mb). The elapsed query times ratios are defined as follows. Assuming the elapsed time of a query using D1 is treated as scale t_a, and the elapsed time of the same query is t_b, using either D2 or D3, the elapsed time ratio is t_b/t_a.

Table 1: Studied queries

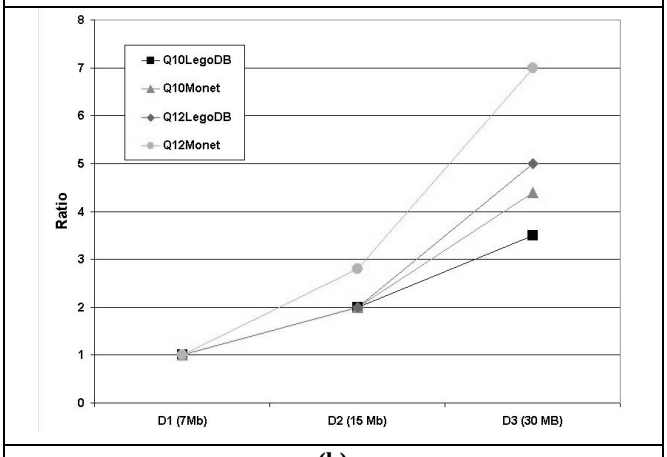
Q1: select the *Parcel(description)* where *Parcel(nºFlats)* is greater that 20.
Q2: select the *Blocks(name)* where *Parcels(nºFlats)* are less than 10.
Q3: Obtain *State(Name)* where parcels are built by "Sanchez Lopez".
Q4: Obtain the *State(Name)* where *Architect(name)* is equal to "Oliver Gómez" or *Parcels(nºFlats)* are less than 12.
Q5: select the *Block(name)* and *State(Name)* where *Architect(address)* is equal to "c/ Rosario" or *Parcel(name)* is greater than 1001.
Q6: Obtain *Block (name)* and *Architect(name)* where *Parcel(number)* is equal to *Parcels(nºFlats)* and *Parcels(nºFlats)* equal to 13.
Q7: Obtain *State(name)* where *Architect(name)* is equal to *Block(name)*.
Q8: Obtain *State(description)* where *Block(name)* is equal to *Architect(name)* or *Architect(name)* is equal to *State(Name)* or *Parcels(nºFlats)* are equal to *Parcel(number)*.
Q9: Count all parcels where *Architect(name)* is equal to "Calvo Gómez" and *Parcels(nºFlats)* are equal to 10 and grouping by *State(name)* and *Block(name)*.
Q10: Sum the *Parcels(nºFlats)* grouping by States.
Q11: Select the *State(BoundaryBy)* where *Area* of the *State(BoundaryBy)* is greater that 50000 meters.
Q12: Obtain the Area of *State(BoundaryBy)* where *Perimeter* of the *Parcels(Extentof)* is greater that 300 metres.
Q13: Return the *State(BoundaryBy)* buffered 20 meters and Area of the *Block(extentof)* where Area of *Parcel(Extentof)* is greater than 300 metres.
Q14: Select the *Block(Name), Architect(name)* and *Block(Extentof)* where *Parcel(number)* is equal to *Pacels(nºFlats)* and *Parcel(nºFlats)* is greater than 9 and Area of the *Parcel(Extentof)* is greater than 300.
Q15: Select *State(name)* and Area of the *State(BoundaryBy)* where *Block(name)* equal to Architect(Name) or *Parcel(Extentof)* is contained in a window.
Q16: Obtain the *State(Description)* and Area of the *State(BoundaryBy)* where *Block(name)* is different from *Architect(name)* and *Block(name)* is distinct from *Parcel(name)* and *Parcel(number)* is distinct from *Parcel(nºFlats)* and *Parcel(extentof)* is contained in a window.
Q17: Return the *Parcel(Number)* with *Parcel(BoundaryBy)* intersect with *Parcel(Extendof)*.
Q18: Select the *Block(number), Architect(name)* and *Block(Extentof)* with *Parcel(Number)* equal to *Parcel(nºFlats)* and *Parcel(nºFlats)* equal to 13 and the *Parcel(Extentof)* intersect with its *Block(Extentof)*.
Q19: Select *State(Description)* and Area of the *State(BoundaryBy)* where *Block(Nomber)* is distinct from *Architect(name)* and *Block(name)* is distinct from *Parcel(name)* and *Parcel(number)* is distinct from *Parcel(nºFlats)* and the *Parcel(Extentof)* intersect with its block(Extentof) and the *Block(Extentof)* is contained in the *Parcel(Boundary)*.

The good elapsed times of Monet in simple alphanumeric queries are corroborated with a large data set (D3). An example is Q5 and Q8 in Figure 3 (a), but in queries with grouping, LegoDB obtains the best results (Q10 in Figure 3 (b)). In spatial queries, with a large data set (D3), Monet has better scalability than LegoDB in complex queries (Q13, Q17, Q18, Q19), but LegoDB obtains the best ratio in the simple query Q12. Note that the difference is very significant in Q17 and Q19. In the medium data set (D2) both approaches are more similar. Even here, Q17 and Q18 obtains a better ratio in LegoDB than in Monet. This fact is related to the number of joins and the size of the table joined. Both parameters determine the best performance with large data sets.

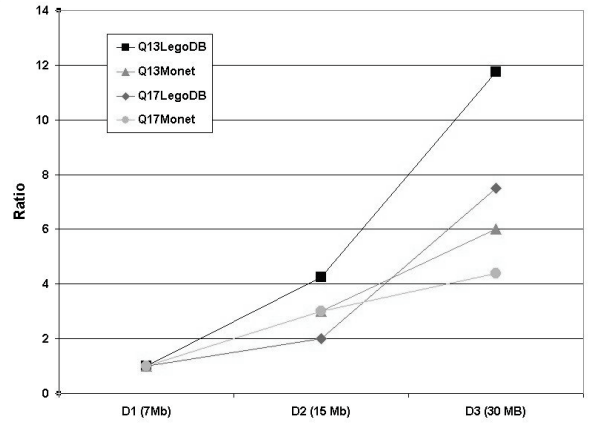
As a conclusion, several joins between small tables (Monet) obtain better scalability than few joins with large tables (LegoDB) when a very large data set is used. Even so, in our experiments, LegoDB also obtains better elapsed time in spatial queries using D3 (this comparative is not included for reason of space).



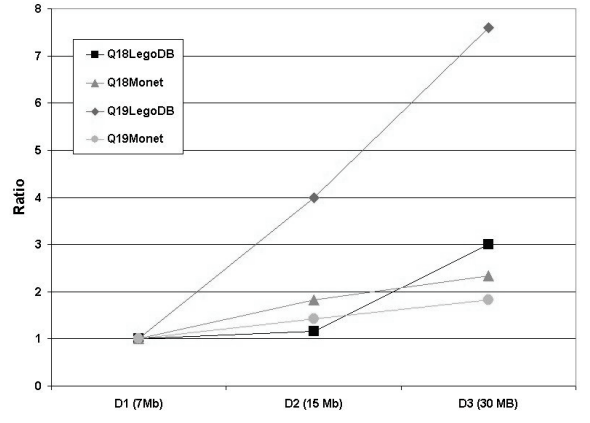
(a)



(b)



(c)



(d)

Figure 3: Scalability Test: LegoDB vs Monet

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have studied three storage models for the implementation of a query language over GML. These alternatives are well-known storage models for XML documents over the RDBMS. We have used alternatives based on RDBMS because they have efficient storage and retrieval techniques and can make use of various indexing mechanisms to evaluate a query. In order to store complex objects (spatial objects) a modification of XParent has been carried out. Modification of the models Monet and LegoDB was not necessary

The experiments show that the inclusion of spatial operators influences the performance of these approaches. The performance obtained with only alphanumeric operators varies greatly when we include spatial operators. We can therefore infer that the modification of XParent is not a good solution for spatial queries. Monet has a good scalability in comparison with LegoDb, but the elapsed times are a good deal worse than LegoDB including GML documents with 30 Mb. In addition, the number of joins in Monet depends on the length of the path and the number of paths involved in the query. This limitation makes Monet a good solution to query spatial data, in shallow documents and in queries with few attributes involved (two features not guaranteed in GML). LegoDB considerably reduces the number of joins and obtains a limit regardless of the attributes involved in the query. According to the results obtained, the best approach for the storage and retrieval of GML documents with our query language is LegoDB. In addition, the solution of LegoDb applied in this study is the simplest to implement of all these offered by LegoDB. We could use the cost-based XML-to-relational mapping engine (LegoDB [7]) to obtain the best distribution of relations for each application. On the other hand, we think the fact that LegoDB is a *structure-mapping approach* makes this solution more *natural* to store GML, because the XMLSchema (DTD) of a GML document has to be known.

Future work foresees, the study of different mapping models offered by LegoDB, to obtain a common pattern --if such exists-- for most applications of GML documents. We will also study the general size of the data set in which Monet and LegoDB converge (section 4.2). At the same time, we intend to look at the development of a Web environment to allow spatial distributed queries on the Web.

6. REFERENCES

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

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

[3] J. Córcoles and P. González. A spatial query language over XML documents. Fifth IASTED International Conference on Software Engineering and Applications (SEA). pp.1-6. 2001

[4] J. Córcoles and P. González. A Specification of a Spatial Query Language over GML. ACM-GIS 2001. 9th ACM International Symposium on Advances in Geographic Information Systems. 2001

[5] J. McHugh, S. Abiteboul, R. Goldman, D. Quass and J. Widom. Lore: A database management system for semistructured data. SIGMOD Record, 26(3) pp. 54-66. 1997

[6] M. Yoshikawa and T. Amagasa. XRel: A path-based approach to storage and retrieval of XML documents using relational databases. ACM Transactions on Internet Technology, 1(1). 2001

[7] P. Bohannon, J. Freire, P. Roy and J. Simeon. From XML Schema to Relations: A Cost-Based Approach to XML Storage. 18th International Conference on Data Engineering (ICDE2002). 2002.

[8] G. Kappel, E. Kapsammer, S. Raush.Schott and W. Retschitzegger. X-ray – towards integrating xml and relational database systems. In proccedings of International Conference on Conceptual Modeling. 2000.

[9] D. Lee and W. Chu. Contraints-preserving transformation from xml document type definition to relational schema. In proceeding of the International Conference on Conceptual Modeling, pp 323-338. 2000

[10] M. Klettke and H. Meyer. Managing XML documents in

Object-Relational databases. Workshop on the Web and Databases (WebDB). 2000.

[11] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J.Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In Proc. of the Int'l. Conf. On Very Large Data Bases, pages 3002-314. 1999.

[12] H. Jiang, H. Lu, W. Wang and J. Xu Yu. Path Materialization Revisted: An efficient Storage Model for XML Data. 2nd Austrlian Institute of Computer ethics Confrence (AICE2000). Canberra. Australia. 2002.

[13] C. Kanne and G. Moerkotte. Efficient storage of XML data. In proceedings of the international conference on Data engineering. 2000.

[14] A. R. Schmidt, M. L. Kersten, M. A. Windhouwer, and F. Waas. Efficient Relational Storage and Retrieval of XML Documents. Workshop on the Web and Databases (WebDB). 2000.

[15] D. Florescu and D. Kossmann. Storing and Querying XML Data Using an RDBMS. Data Engineering Bulletin, 22(3), 1999.

[16] S. Banerjee, V. Krishnamurthy, M.. Krishnaprasad and R. Murthy. Oracle8i — The XML Enabled Data Management System. In: IEEE ICDE. 2000.

[17] Oracle9i Database Documentation. http://otn.oracle.com:80/docs/products/oracle9i/content.html. 2002.

[18] J. M. Cheng, J. Xu, "XML and DB2". In: IEEE ICDE..2000.

[19] Open GIS Consortium, Inc. OpenGIS: Simple Features Specification For SQL Revision 1.1 OpenGIS 99-049 Release. 1999.

[20] H. Samet. Applications of spatial data structures. Computer Graphics, Image processing and GIS. Addison – Wesley. 1990.

[21] U. Nambiar, Z. Lacroix, S. Bressan, M. Li Lee and Y. Li. XML Benchmarks put to the test. 3rd International Conference on Information Integration and Web-based Applications and Services (IIWAS). 2001.

[22] A.R. Schmdit, F. Waas, M. Kerste, D. Florescu, I. Manolescu, M. Carey and R. Busse. The XML Benchmark Project. Technical Reports INS-R0103. 2001.

[23] A.R. Schmdit, F. Waas, M. Kerste, D. Florescu, M. Carey, I. Manolescu and R. Busse. Why and How to Benchmark XML Databases. SIGMOD Record. nº3 vol. 30. 2001.

[24] S. Abiteboul, P. Buneman and D. Suciu. Data on the Web. From Relations to Semistructured Data and XML. Morgan Kaufmann Publishers. 2000.

[25] P. Rigaux, M. Scholl and A. Voisard. Spatial Databases with Application to GIS. Morgan Kaufmann Publishers. 2002.