

Query Processing in a Geographic Mediation System

Mehdi Essid, Omar Boucelma, François-Marie Colonna and Yassine Lassoued
LSIS-CNRS, University of Aix-Marseille
Avenue Escadrille Normandie-Niemen
F-13397 Marseille Cedex 20
< First > . < Last > @cmi.univ-mrs.fr

ABSTRACT

Despite various interoperability recommendations, heterogeneity of Geographic Information Systems (GIS) is still an issue. This led to an increasing need for a data integration system that allows transparent and uniform access to spatial data disseminated over a network. In this paper, we describe the internals of query processing in the VirGIS mediation system. Recall that a data mediation system provides users with a uniform access to a multitude of (local/remote) data sources, without duplicating the data. The user poses his query against a virtual (global) schema, the query is in turn rewritten into queries sent to the local sources. VirGIS complies with Open GIS Consortium recommendations in using the Geography Markup Language (GML) for the encoding and the transport of geographic information, and the Web Feature Server (WFS) interfaces to perform communications with clients and data sources.

Categories and Subject Descriptors

H.2.8 [DATABASE MANAGEMENT]: Database Applications — *Spatial databases and GIS*

General Terms

Algorithms, Languages

Keywords

Data Integration, Query Rewriting, GML, WFS

1. INTRODUCTION

The proliferation of spatial data on the Internet is beginning to allow a much larger audience to share data currently available in various Geographic Information Systems (GIS). As spatial data increases in importance, many public and private organizations need to disseminate and have access to the latest data at a minimum (right) cost and as fast as possible. In order to move to a real Web-based spatial

data system, we need to provide flexible and powerful GIS data integration solutions. Indeed, GIS are highly heterogeneous: not only they differ by their data representation, but they also offer radically different query languages. The two main problems resulting from the data integration are the data modeling (how to integrate different source schemas) and their querying (how to answer correctly to the queries posed on the global schema).

The mediation systems provide to the user a uniform interface of the different data sources via a common model. The issues of the schema integration are the sources heterogeneity, global schema modeling and definition, the definition and the management of the mapping rules that express the correspondence between the global schema and the data source ones, the source semantics and the management of the coherence and the evolution of the schema.

The global schema definition, that provides an uniform view of the different resources, can be done using two different approaches. The *Global As View* approach (GAV), consists in defining the global schema as a set of views over local schemas, while the *Local As View* one (LAV) consists in defining the local sources as a set of views made on the global schema.

Even if the conception of the schema in the global approach is easy, the addition of a new source can force us to redesign the global schema, and so to reconsider all the sources. This drawback represents a serious limitation of this approach when the number of integrated sources is high. At the opposite, addition of sources in LAV approach is easy since it consists in defining a view of the global schema.

To compensate the insufficiency of the two approaches, other approaches, such as *Global Local As View* (GLAV) [8] or *Both As View* (BAV) [5], have been proposed. The GLAV approach combines the expressive power of both LAV and GAV, allowing flexible schema definitions independent of the particular details of the sources. The BAV approach is based on the use of reversible schema transformation sequences.

The fundamental question, when attempting to interoperate several data sources, is twofold: on the one end, the identification of different objects having a semantic link, and coming from different data sources, on the other end, the resolution of structural differences (schematic differences) between objects having a semantic link.

The schematic conflict between data may arise when equivalent concepts are represented differently in local data sources. Those conflicts can be associated to the names of the concepts, their data types (for example a building can be a polygon in a data source and a point in another), the unit

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

GIS'04, November 12–13, 2004, Washington, DC, USA.
Copyright 2004 ACM 1-58113-979-9/04/0011 ...\$5.00.

(perimeter can be expressed in meters in a data source and in kilometers in another one), the attributes (some attributes may be absent in some data sources). Another kind of schematic conflict is the difference in the representation of an attribute. For example, one can represent an address with a single attribute and another can represent it with a tuple (Number, Street, City, postal code).

The semantic conflict concerns the differences of meaning, of interpretation or of use of the same data. Therefore, schematic conflicts are significant only if there is a semantic similarity between objects.

In this paper, we describe the query processing internals of VirGIS [4], a mediation platform whose main characteristics are as follows:

- It uses the OpenGIS standards: Geography Markup Language (GML) [16] for the encoding and the transport of geographic information, and the Web Feature Server interfaces (WFS) [14] to perform communications with clients and data sources,
- It uses GQuery [6], a geographic XQuery-based.
- It includes a WFS interface which provides a WFS enabled mediation system. This provides WFS enabled clients to access a set of data-sources in a transparent fashion,
- It uses a specific join engine to compute the join operations alternatively to join engines being part of existing XQuery implementations.

This paper focusses on query processing in the VirGIS system. Since we manipulate geographic data, we are always (at least in our case) brought to handle features which can be mapped to the global schema with a one-to-one schema transformation under some constraints. These transformations are expressed in the form of mapping rules using a mapping language which is quickly presented in this paper. The mapping rules are used by a rewriting algorithm which computes the user’s query execution plan. The main idea of this algorithm is to compute, first, the different features which participate in the user’s query (in other terms, the set of subgoals). A global execution plan is then computed for each feature(or subgoal). The final execution plan is a reconstitution of the user query from the different global execution plans.

The paper is organized as follows. Section 2 describes a concrete integration example. In Section 3 we describe how mappings are done between the global schema and the local ones. The VirGIS architecture and the rewriting process are described in Section 4. Finally we conclude in Section 6.

2. INTEGRATION EXAMPLE

Data used in this example are drawn from a concrete geographic data application being developed as part of the REV!GIS European project [1]. For sake of simplicity, we are considering two data sources only, denoted CITS (Centre d’Information Topographique de Sherbrook), and BDTQ (Bases de Données Topographique du Quebec) [2], both of them covering regions around the city of Sherbrook in Canada.

As in most integration systems, a virtual global schema built from different local schemas is presented to the user. Both local and global schemas are detailed in the next sections.

2.1 Global and Local Sources

Figure 1 illustrates the UML simplified version of BDTQ and CITS schemas. We are using UML instead of XSD (XML Schema Description) for readability reasons.

CITS schema consists of two classes that cover the bridge and road *features* at a scale of 1/50 000. Class ROADL represents linear roads, whereas class BRIDGEL represents linear bridges.

BDTQ schema consists of two classes covering the transportation network at the scale 1/20 000. Class VCOMM_P and VCOMM_L respectively describes points and linear objects belonging to the transportation network. For example, roads, railways and bridges are instances of the BDTQ source.

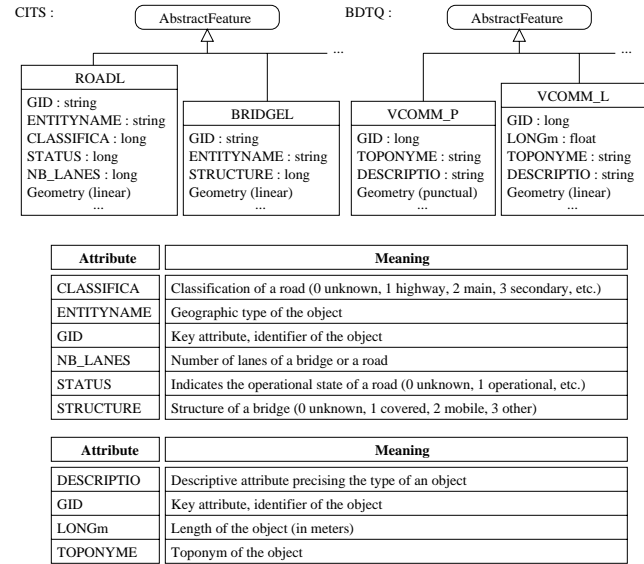


Figure 1: CITS and BDTQ Source Schemas

As for data sources, we are using a fragment of the global schema, which is represented in figure 2. Class Road (resp. Bridge) represents linear roads (resp. bridges) of any geometry feature.

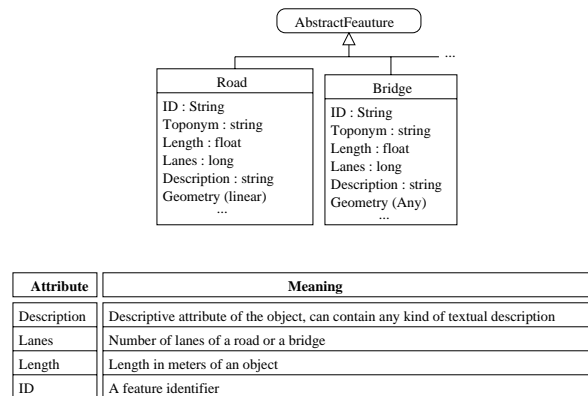


Figure 2: Global Schema

3. SCHEMA MAPPING

3.1 Dealing with Geographic Heterogeneity

One of the main issues faced when integrating geographic data is ontological heterogeneity. As a matter of fact, each geographic data source is an abstraction of the real world according to a particular point of view and purpose. A data source may represent regular topographic maps (IGN, Ordnance Survey, etc.) or a statistical survey (national Census), or even a satellite image (meteo or land-cover oriented). For each dataset, the underlying motivation leads to a particular representation of geographic objects. As a result, depending on the source point of view, its scale, its producer and its final user, the data source has its own way of classifying geographic objects. For example, CITS (as well as the global schema) distinguishes bridges and roads, whereas BDTQ groups all transportation network objects in two classes according to their geometries. Often different classifications have different levels of granularity, and this situation leads to partial correspondences between objects. We can refer, for instance, to the correspondence between class ROADL from CITS and class VCOMM_L from BDTQ. Note that the latter uses the DESCRIPTIO attribute to distinguish real object functions, i.e., to make a more precise classification of the objects. Actually the values of this attribute are among a set of terms such as : "Railway", "Street", "Highway", "Bridge", "Unpaved street", "Cross-street", "Road under construction", "...".

As mentioned above, class Road has a partial match with BDTQ class VCOMM_L. This match is valid under a certain restriction, defined on the DESCRIPTIO attribute of VCOMM_L. Consequently, we are not in a classical mapping situation dealing with simple 1-to-1 mappings. Indeed, we need to express partial correspondences between classes using restrictions, as illustrated below :

$$\text{Road} \xrightarrow{\sigma_{vr}} \text{VCOMM_L}$$

which means that Road corresponds to VCOMM_L under the restriction σ_{vr} , where σ_{vr} is defined by the constraint :

$$\text{DESCRIPTIO} \in \{\text{"Street"}, \text{"Highway"}, \text{etc.}\}.$$

3.2 More on Mappings

Data integration is a process by which several local schemas are integrated to form a single virtual (global) schema. As seen in Section 1, data integration approaches are known either as GAV, LAV, GLAV, or BAV to cite a few. Whatever approach adopted, we need to provide a framework for schema transformations, i.e., a language for schema mappings.

In the current VirGIS version, we are using a LAV like approach, with simple *mapping rules* that allow the specification of one-to-one schema transformations under some constraints: aggregations and one-to-many mappings are not considered.

A mapping is composed of a right term, a left term and a restriction (in the case of a class mapping). The left term is a global schema construct, while the right one consists of one or more paths of the source schema construct (possibly an empty path). Such a mapping means that the global schema element of the left term corresponds to the local schema path(s) under the given restriction. A *mapping rule* is identified to a part of the global schema where each element is expressed as depending on one or more paths of

the local schemas using a mapping. For example, rule R_1 in Figure 3 describes the mapping of global class Road in CITS source. Similarly, rule R_2 illustrates the mapping of class Road in BDTQ source.

As illustrated in this figure, for each local source, we describe the mapping of each class of the global schema. We are using <targetTag> (resp. <sourceTag>) conditions tags to denote local schema constructs (rep. global ones); this syntax is inspired from XQueryX [18]. We can see clearly that class Road corresponds to VCOMM_L under the restriction that attributes DESCRIPTIO must match "Street" or "Unpaved street". In the same way, ID global attribute corresponds to gid local attribute, Description to DESCRIPTIO, Toponym to TOPONYME etc. Let us note also that, for a given local source, not all global attributes have their counterpart (correspondent) in the local schema. For example, in the BDTQ mapping, attribute Lanes does not have a correspondent.

4. QUERY REWRITING

Here, we consider the problem of answering conjunctive queries using a set of views defined as explained above (by a one-to-one mapping rules). Previous work has considered several algorithms for this purpose. We cite for example the *inverse rule* algorithm [7], the *bucket* algorithm [12], *Mini-Con* [15] or the algorithm used in the *Styx* [3] prototype, the one that we adapted in our system.

In the sequel, we first give an overview of the VirGIS integration system and the query language used. Then we give more details on the query processing.

4.1 System Overview

Figure 4 illustrates the current system architecture, that complies to the approach described in this paper. VirGIS is in charge of processing (geographic) user queries which are expressed either in GQuery [6], or in XML/KVP (Keyword-Value-Pair) according to the OpenGIS WFS recommendation. Sources are published on the VirGIS portal via WFS servers. An *admin interface* allows the addition, modification or deletion of sources located by their WFS addresses. Local schemas and capabilities are extracted using WFS queries (*GetCapabilities* and *DescribeFeatureType*). The *global schema* is specified as an *XML Schema Description* [17].

The (global) user query is rewritten into sub-queries expressed in terms of source schemas. These sub-queries constitute the local execution plan. First of all, the *decomposition* module decomposes the user query into a set of elementary sub-queries expressed in terms of the global schema. Each such (elementary) sub-query aims to extract information about a single feature. The decomposition module computes a global execution plan expressed as a join over elementary sub-queries.

Next, sub-queries of the global execution plan, are processed by the *Rewriting* module in order to be expressed in terms of source schemas. This is performed by this module in using the mapping rules; global to local bindings are also computed at this stage. Hence, we obtain a local execution plan for each elementary sub-query of the global execution plan. Each local execution plan is a combination of queries over source schemas using joins, unions and other (possibly spatial) operations. The *Mapping/Reformulation* module reformulates these queries such that the results conforms to the global schema. This results in a final execution

<pre> <catalog> <application>Roads around Sharebrooke</application> <annotation>maps sources CITS-21e05 and BDTQ-21ee201. </annotation> <mapping sourcePath="http://194.167.251.191:8080/geoserver/GetFeature"> <sourceName>CITS-21e05</sourceName> <annotation>Source CITS-21e05 is at scale 1:50000</annotation> <rule label="R1" > <classMapp> <targetTag>Road</targetTag> <sourceTag>ROADL</sourceTag> <attributeMapp> <targetTag>ID</targetTag> <sourceTag>GID</sourceTag> <attributeMapp> <targetTag>Description</targetTag> <sourceTag>ENTITYNAME</sourceTag> <attributeMapp> <targetTag>Lanes</targetTag> <sourceTag>NB_LANES</sourceTag> <attributeMapp> <targetTag>Geometry</targetTag> <sourceTag>Geometry</sourceTag> </attributeMapp> </classMapp> </rule> </mapping> <mapping sourcePath="http://194.167.251.191:8080/geoserver/GetFeature"> <sourceName>BDTQ-21ee201</sourceName> <annotation>Source BDTQ-21ee201 is at scale 1:20000</annotation> <rule label="R2" > <classMapp> <targetTag>Road</targetTag> <sourceTag>VCOMM_L</sourceTag> </pre>	<pre> <condition> <OR> <GeneralComp> = <sourceTag>DESCRPTIO</sourceTag> <Literal>Street</Literal> </GeneralComp> <GeneralComp> = <sourceTag>DESCRPTIO</sourceTag> <Literal>unpaved street</Literal> </GeneralComp> </OR> </condition> <attributeMapp> <targetTag>ID</targetTag> <sourceTag>gid</sourceTag> <attributeMapp> <targetTag>Description</targetTag> <sourceTag>DESCRPTIO</sourceTag> <attributeMapp> <targetTag>Toponym</targetTag> <sourceTag>TOPONYME</sourceTag> <attributeMapp> <targetTag>Length</targetTag> <sourceTag>LONGM</sourceTag> <attributeMapp> <targetTag>Geometry</targetTag> <sourceTag>Geometry</sourceTag> </attributeMapp> </classMapp> </rule> </mapping> </catalog> </pre>
--	---

Figure 3: Mapping rules R_1 (ROADL, Road) and R_2 (VCOMM_L, Road)

plan which may contain three kinds of queries:

1. HTTP queries are WFS queries in charge of extracting a feature from a local source. These are processed by the WFS Query Engine.
2. GQuery queries are (mainly) used to express spatial operations, and are processed by the GQuery engine.
3. JOIN queries play a central role in the execution plan. Because we are handling a large amount of data and, because existing XQuery implementations do not supply efficient join modules, we decided to distinguish this kind of queries to execute them with a specific algorithm based on the Sort-Merge algorithm. The WFS join engine module is in charge of JOIN queries processing.

The final execution plan is then processed by the *Execution* module which in charge of dispatching the queries to the appropriate engine. This module uses a cache repository to save temporary result of the different queries. When the execution is finished, the *Recomposition* module eliminates the duplicated answers and returns a GML file that can be either processed (if needed) or simply (generally) displayed by a GML viewer.

4.2 Query Language

The capability of a mediator system is bounded by its query language expressiveness. And having some valid query unanswered by the mediator system is frustrating. To avoid this problem, the query language should capture the whole system abilities.

In the VirGIS system we are using GQuery. GQuery is based on XQuery [19] and allows users to perform both attribute and spatial queries against GML documents. Spatial semantics is extracted and interpreted from XML tags without affecting XQuery syntax, i.e. a spatial query is a pure

FLWR (Flower) expression, and looks like a regular XQuery expression. GQuery may be used either as a stand-alone program, or as a query language in the context of an integration system, to query several distributed heterogeneous GIS data sources.

4.3 Obtaining a Global Execution Plan

The Global execution plan is the result decomposing the user query into a set of elementary sub-queries expressed in terms of the global schema. Actually, data sources are queried through a WFS interface. Although providing an easy access to any geographic repository, WFS is unable to handle complex queries and does not provide support for data integration. Thus execution plans are a combination (join, union and operation) of elementary queries, each of them being posed against a single feature.

As an example, let us consider query of Figure 5 which consists in extracting all 2-lane roads that cross bridges and having a length up to 1000 meters.

As we can see, this query involves two features (bridge and road) and three conditions.

The condition ($cross(x/geometry,y/geometry)$) is not a simple one because it contains two attributes of two different features. Consequently, Q_0 is split into three sub-queries as illustrated in figure 6. The first sub-query consists in extracting all 2-lanes roads having length up to 1000. The second one consists in extracting all the bridges and the last one consists in joining the first query and the second one with the condition $cross(x/geometry,y/geometry) = true$.

Figure 7 illustrates the decomposition algorithm. The functions used in this algorithm have the following meaning:

- ExtractSimpleQueries: extracts simple queries as well as their simple conditions (queries consisting in extracting features) from the user query.

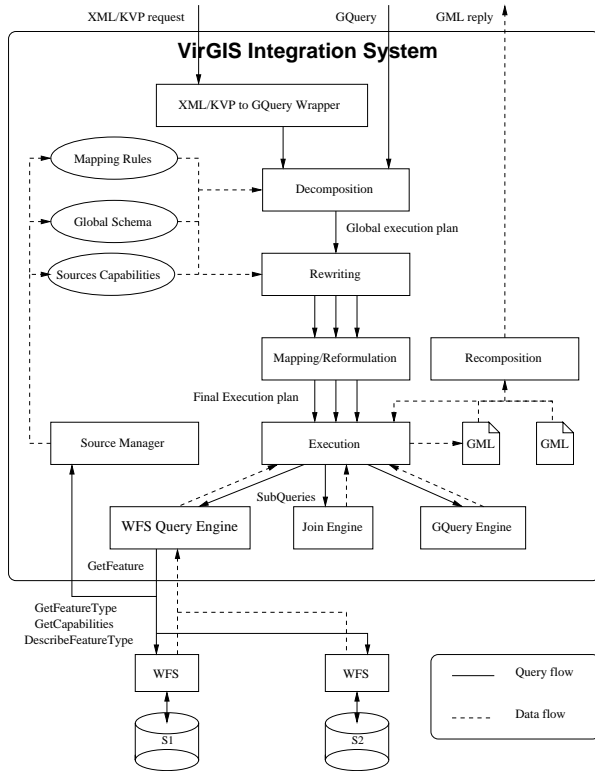


Figure 4: The VirGIS Architecture

for \$x in document(bridge), \$y in document(road)
 where cross(\$x/geometry, \$y/geometry) = true
 and \$x/length > 1000 and \$x/lanes = 2
 return \$x

Figure 5: Q_0 : User Query Example

- RemoveSimpleConditions: removes all the simple conditions from the user query because they are already treated in the simple queries.
- ReplaceFeatureByQueryID: replaces the features by the id of their corresponding simple query.

The execution plan is represented as an XML document, composed from query nodes. Each query has an id, a priority attribute which indicates its execution order and other properties that we will see in the next section. For example, queries `query0` and `query1` are executed first, followed by `query2`, etc.

4.4 Obtaining a Final Execution Plan

Assuming that the global execution plan contains $N + 1$ subqueries, we rewrite each of the first N subqueries in terms of the local schemas, the last one being used to compute the final result. To this purpose, we adapted the approach described in [3]. For each query, we start in computing its *binding*. A binding of a basic query is the set of mappings that support (totally or partially) the feature of the query.

```
<Queries>
<Query id="query0" priority="0">
  for $x in document(road)
  where $x/length > 1000 and $x/lanes = 2
  return $x
</Query>
<Query id="query1" priority="0">
  for $x in document(bridge)
  return $x
</Query>
<Query id="query2" priority="1">
  for $x in document(query0),
  $y in document(query1)
  where cross($x/geometry,$y/geometry) = true
  return $x
</Query>
</Queries>
```

Figure 6: Global Execution Plan for Q_0

Input : User Query Q_0
 Output : Global Execution Plan
 Algorithm :

```
ExecutionPlan EP
SimpleQueries SQ = ExtractSimpleQueries(Q0)
QueryCpt = 0
For each Q in SQ Do
  Q.setId("query"+ QueryCpt)
  QueryCpt++
  EP.AddQuery(Q)
End For
RemoveSimpleConditions(Q0)
ReplaceFeatureByQueryID(Q0)
EP.AddQuery(Q0)
```

Figure 7: The Decomposition Algorithm

When a source supports all the attributes used in the query (totally) we say that this query has a *full-binding* with this source. The binding of the query allows query rewriting into a set of sub-queries expressed in terms of the local schemas.

Let us consider, for instance, a basic query Q_e and its binding $\mathcal{B} = \{\mathcal{M}_i, i \in 1 \dots n\}$ where n is the number of sources that support the feature queried by Q_e and $\mathcal{M}_i, i \in 1 \dots n$, are their respective associated mapping. To extract the maximum of information located in local sources we rewrite Q_e with a given mapping of the binding, then we try to look for the unsupported attributes (if they exist) using the other mappings of the binding. To do this we define the notion of *prefix query* and *suffix query*. Given a mapping M_i of a source S_i , A prefix query Q_p is a sub-query of Q_e that consists in extracting only the attributes supported by S_i . Note that M_i provides a full-binding for query Q_p . The suffix query Q_s , is the sub-query consisting in extracting the remaining attributes (plus the key's attributes).

In considering these definitions, for each mapping $\mathcal{M}_i \in \mathcal{B}$, the query Q_e is processed as follows:

1. compute Q_p and Q_s of the query Q_e and the mapping \mathcal{M}_i .
2. let $\mathcal{B}_s = \{\mathcal{M}_j, \mathcal{M}_j \in \mathcal{B}, j \neq i\}$
3. let $\mathcal{B}_i = \{\mathcal{M}_i\}$
4. the execution plan of query Q_e using the binding \mathcal{B}_i is the join between Q_p and the execution plan of Q_s using the binding \mathcal{B}_s .

Since queries that are responsible for extracting feature information are executed by the WFS servers, they are reformulated, using the mapping rules, into queries that are expressed in terms of local schemas (each feature is replaced by the correspondent feature, each attribute is replaced by the correspondent attribute and the conditions of the rule (cf.3) are added to the set of condition of the query). This reformulation generates a problem because the answer to this query (the answer of the WFS server) is written in terms of the local schema. To alleviate this problem, an additional GQuery query is added to do the inverse transformation. Queries `query03` and `query04` illustrated in Figure 9 are examples of such kind of queries.

Let us now discuss constraints of the prefix and the suffix queries. It is easy to see that the prefix query contains only the subset of conditions over attributes of the prefix query. However, the suffix query contains the subset of conditions over attributes of the suffix query extended by the set of conditions of the prefix query. This extended set is used to refine the prefix query in adding more conditions. This allows us to minimize the number of extracted tuples, hence minimize the execution time. Finally, a condition that is not supported by a data source is added to the GQuery expression which is responsible for the transformation of the result: this is the case where the source contains the attributes on which the condition is made but the capabilities of the source can not support it.

Figure 8 illustrates the rewriting algorithm which, given an elementary query Q_e and its binding \mathcal{B} , computes the execution plan expressed in terms of the local sources .

As we may notice in the rewriting algorithm, \mathcal{B}_s is defined as $\{\mathcal{M}_j \in \mathcal{B}, j > i\}$ and not as $\{\mathcal{M}_j \in \mathcal{B}, j \neq i\}$ as mentioned in the beginning of this section. We made this

```

Input      : Elementary query  $Q_e$ 
            Binding  $\mathcal{B}$ 
Output     : Execution Plan expressed in term
            of local schemas
Algorithm   :  $RW(Q_e, \mathcal{B})$ 

ExecutionPlan  $EP$ 
If (fullbinding( $Q_e, \mathcal{M}_1$ ))
     $tmpEP = Q_e$ 
Else
     $Q_p = \text{getPrefixQuery}(\mathcal{M}_1)$ 
     $Q_s = \text{getSuffixQuery}(\mathcal{M}_1)$ 
     $\mathcal{B}_s = \{\mathcal{M}_j \in \mathcal{B}, j > 1\}$ 
     $tmpEP = (Q_p \bowtie RW(Q_s, \mathcal{B}_s))$ 
End if
 $EP = EP \cup tmpEP$ 

```

Figure 8: Rewriting Algorithm

change in order to eliminate duplicated answers. In fact, the subset of mappings $\{\mathcal{M}_k \in \mathcal{B}, k < i\}$ is already treated in the previous step. Thus, it is useless to recompute it.

To compute the final execution plan, we just rewrite each of elementary query of the global execution plan. As an example, let us process the first query of the global execution plan of Q_0 . This consists in extracting all Roads having a length upper than 1000 meters and 2 lanes. The catalog illustrated in figure 3 shows that the first mapping (the CITS one) does not map attributes Toponym and Length. Hence, the query is divided in two sub-queries. The first one (prefix query) extracts from CITS the attributes ID, Description, Lanes and Geometry that satisfy the condition Lanes = 2. The second one (suffix query) extracts from BDTQ the attributes ID (the key), Toponym and Length satisfying the condition Length > 1000. A join is then performed between the results of the prefix and the suffix queries in order to build the final result. Figure 9 illustrates the execution plan of the first query (`query0`) of the global execution plan.

5. RELATED WORK

Integration of distributed and heterogeneous data is still an important research area. Several systems are being built to tackle this problem. Among those are TSIMMIS [9], PICSEL [10], Information Manifold [11], AGORA [13], Styx [3], etc. Some of these systems are using relational technology, others are using XML. Our system uses the XML (GML) technology and propose a geographic query language which allows to integrate geographic data sources.

In addition to the query language, the power of an integration system is based on how schema mapping is performed and how efficient is the rewriting algorithm. Most of the existing data integration systems use views to express the correspondence between the real data source schemas and the integrated one. For example, in Information Manifold, the real data sources are expressed as a set of relational views over the global schema. The query rewriting process

<pre> <Queries> <Query id="query01" priority="0" Type="HTTP" Address="http://194.167.251.191:8080/geoserver/GetFeature"> <GetFeature xmlns:gml="http://www.opengis.net/gml"> <Query typeName="ROADL"> <PropertyName>gid</PropertyName> <PropertyName>ENTITYNAME</PropertyName> <PropertyName>NB_Lanes</PropertyName> <PropertyName>Geometry</PropertyName> <Filter> <PropertyIsEqualTo> <PropertyName>NB_Lanes</PropertyName> <Literal>2</Literal> </PropertyIsEqualTo> </Filter> </Query> </GetFeature> </Query> <Query id="query02" priority="0" Type="HTTP" Address="http://194.167.251.191:8080/geoserver/GetFeature"> <GetFeature xmlns:gml="http://www.opengis.net/gml"> <Query typeName="VCOMM_L"> <PropertyName>gid</PropertyName> <PropertyName>TOPONYME</PropertyName> <PropertyName>LONGM</PropertyName> <Filter> <PropertyIsGreaterThan> <PropertyName>LONGM</PropertyName> <Literal>1000</Literal> </PropertyIsGreaterThan> </Filter> </Query> </GetFeature> </Query> </pre>	<pre> <Query id="query03" priority="1" Type="GQUERY"> for \$x in document(QUERY01)/RAODL return (<featureMember> <Road> <id>{\$x/gid/node()}</id> <Description>{\$x/ENTITYNAME/node()}</Description> <Lanes>{\$x/NB_Lanes/node()}</Lanes> <Geometry>{\$x/Geometry/node()}</Geometry> </Road> </featureMember> </Query> <Query id="query04" priority="1" Type="GQUERY"> for \$x in document(QUERY02)/VCOMM_L return (<featureMember> <Road> <id>{\$x/gid/node()}</id> <Toponym>{\$x/TOPONYME/node()}</Toponym> <Length>{\$x/Longm/node()}</Length> </Road> </featureMember> </Query> <Query ID="query0" Priorite="2" Type="JOIN"> for \$x in document(query03)/FeatureCollection/featureMember/Road , \$y in document(QUERY04)/FeatureCollection/featureMember/Road where \$x/id/node() = \$y/id/node() return (<featureMember> <Road> <id>{\$x/gid/node()}</id> <Description>{\$x/ENTITYNAME/node()}</Description> <Lanes>{\$x/NB_Lanes/node()}</Lanes> <Geometry>{\$x/Geometry/node()}</Geometry> <Toponym>{\$x/TOPONYME/node()}</Toponym> <Length>{\$x/Longm/node()}</Length> </Road> </featureMember> </Query> </Queries> </pre>
---	--

Figure 9: Final Execution Plan

is done by the *Bucket* algorithm which tries to rewrite the query subgoals one by one independently. The *MiniCon* algorithm, an amelioration of the *Bucket* one, uses the input/output common variables between the query subgoals to reduce the bucket size. This allows to reduce the time of rewriting a query and to capture some solutions ignored by the bucket algorithm. The *Styx* algorithm is an ontology oriented algorithm which uses the parent/child dependencies of query variables for query decomposition.

The VirGIS rewriting algorithm is inspired from the *Styx* one. It uses the description of each data source feature to decompose the query into a prefix query and a suffix one. This allows, in addition to tuples that may be returned by the *Bucket* and *MiniCon*, to return other solutions which ensure a maximal retrieval of feature properties: for example if one source does not map property p_1 of feature f_1 , our algorithm tries to look for another source which maps this property and then returns the join of both sources. In the current version VirGIS, we use only the one-to-one correspondences with conditions to express the mapping between geographic features. This restriction comes from the geographic domain where, generally, data describe features which are pretty much the same, but the difference resides in the feature properties.

6. CONCLUSION

With the proliferation of GIS data and resources over the Internet, there exists a huge demand for robust geospatial information services that allow federation/interoperation of massive repositories of heterogeneous data and metadata. To tackle this problem, we developed an XML (GML) based integration system called VirGIS. The technical choices we made address effective integration needs expressed by the GIS community, and incidently by the database community.

Indeed, since VirGIS is based on XML technology, it can be deployed in any heterogeneous mediation context.

Future research should include an extension of the mapping model to take into account one-to-n mappings and aggregations, query optimization and automatic schema matching.

7. REFERENCES

- [1] REV!GIS: Revision of the Uncertain Geographic Information. www.lsis.org/REVIGIS/FULL.
- [2] Base de données topographiques du Québec (BDTQ), 1:20,000. felix.geog.mcgill.ca/heeslib/bdtq_20000.html, July 2001.
- [3] Bernd Amann, Catriel Beeri, Irini Fundulaki, and Michel Scholl. Querying XML Sources Using an Ontology-Based Mediator. In *On the Move to Meaningful Internet Systems, Confederated International Conferences DOA, CoopIS and ODBASE*, pages 429–448. Springer-Verlag, 2002.
- [4] O. Boucelma, M. Essid, Z. Lacroix, J. Vinel, J-Y. Garinet, and A. Betari. VirGIS : Mediation for Geographical Information Systems. In *Proc. ICDE 2004, Boston*, March 30 - April 2 2004.
- [5] M. Boyd and *al.* AutoMed: A BAV Data Integration System for Heterogeneous Data Sources. In *Advanced Information Systems Engineering 16th International Conference, CAiSE 2004, Riga, Latvia, June 7-11, 2004, Proceedings*. Springer-Verlag, 2004.
- [6] F-M. Colonna and O. Boucelma. Querying GML Data. In *Proc. 1ère Conférence en Sciences et Techniques de l'Information et de la Communication (CoPSTIC'03), Rabat*, 11–13 décembre 2003.

- [7] Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *Proceedings of ACM PODS*, pages 109–116. ACM Press, 1997.
- [8] Marc Friedman, Alon Y. Levy, and Todd D. Millstein. Navigational Plans For Data Integration. In *AAAI/IAAI*, pages 67–73, 1999.
- [9] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaman, Y. Sagir, J. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and Languages. *Journal of Intelligent Information Systems*, 1997.
- [10] Francois Goasdoue, Veronique Lattes, and Marie-Christine Rousset. The Use of CARIN Language and Algorithms for Information Integration: The PICSEL System. *International Journal of Cooperative Information Systems*, 9(4):383–401, 2000.
- [11] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. In C. Knoblock and A. Levy, editors, *Information Gathering from Heterogeneous, Distributed Environments*, Stanford University, Stanford, California, 1995.
- [12] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proceedings of the 22th International Conference on Very Large Data Bases*, pages 251–262. Morgan Kaufmann Publishers Inc., 1996.
- [13] Ioana Manolescu, Daniela Florescu, and Donald Kossmann. Answering XML Queries over Heterogeneous Data Sources. In *Proc. of VLDB*, pages 241–250, 2001.
- [14] OpenGIS. Web Feature Server Implementation Specification, September 19th 2002.
- [15] Rachel Pottinger and Alon Y. Levy. A Scalable Algorithm for Answering Queries Using Views. In *The VLDB Journal*, pages 484–495, 2000.
- [16] Ron Lake Simon Cox, Adrian Cuthbert and Richard Martell. Geography Markup Language (GML) 2.0. www.opengis.net/gml/01-029/GML2.html, February 20th 2001.
- [17] W3C. XML Schema Part 1: Structures. www.w3.org/TR/2001/REC-xmlschema-1-20010502.
- [18] W3C. XML Syntax for XQuery 1.0 (XQueryX). www.w3c.org/TR/2003/WD-xqueryx-20031219/.
- [19] W3C. XQuery 1.0: An XML Query Language.