GDMS: An abstraction layer to enhance Spatial Data Infrastructures usability

Erwan Bocher[1,3], Thomas Leduc[2,3], Guillaume Moreau[1,3], Fernando González Cortés[3]

(1) Ecole Centrale de Nantes, (2) CNRS CERMA, (3) IRSTV
IRSTV / ENSA Nantes – rue Massenet, BP 81931, 44319 Nantes cedex 3, France
{erwan.bocher,guillaume.moreau}@ec-nantes.fr, thomas.leduc@cerma.archi.fr,
fergonco@gmail.com

Abstract. The practical exploitation of SDI (Spatial Data Infrastructures) raises number of issues as far as it grows. Among them is the heterogeneity of data sources and thus the difficulty for GIS users not to depend on the data source format and of course to learn different systems. This a major flaw with respect to reuse and data sharing. The purpose of our work is to propose a new semantic layer derived from the SQL language that is independent of the underlying data source. This layer, called GDMS (Generic Data source Management System) can first be seen as an abstraction layer between data sources and the SDI tools. We will also show how this layer extends both SQL and spatial semantics and improves the exploitation of the SDI, by providing feedback both in terms of work and data reuse. A simple example mixing heterogeneous data sources will be presented.

INTRODUCTION

Recently, attention has been focused on the development of Spatial Data Infrastructures at different levels (local, national) and for various policy areas and priority of actions such as social, economic, and environmental issues (Clinton, 1994, INSPIRE 2007). All of them share the same goal: maximize user's access to spatial data, minimize the redundancy of efforts and investments (Nebert, 2004). To implement it in European states, the INSPIRE directive defines a set of rules and specifications and some basic components (figure 1).

Theoretically, data is stored on distributed repositories (Database Management Systems that are extended with the capability of manipulating georeferenced data in most cases). A set of tools is coming in between the repositories and the user applications such as GIS applications or thin web clients. These tools provide the user with an access to a set of services like catalog, geoprocessing, dispatcher that permit to view, query or plan advanced analysis. The interoperability of the system is guaranteed by several standards for data exchange and processing (GML, WFS, WMS, WPS) (Smits, 2002).

In practice, the development and exploitation of an SDI is complex and faces many difficulties. Unlike the web client side of an SDI, the server side and desktop clients still deal with the complexity due to the multiple data sources types (WFS, spatial databases, local files...) of typical SDI scenarios. Additionally, this complexity leads to an increasing difficulty to reuse work between the users of the SDI. Finally, there are human factors (new practices) (Rajabifard 2004, de Oliveira, 1996) that reduce SDI acceptance by the users.

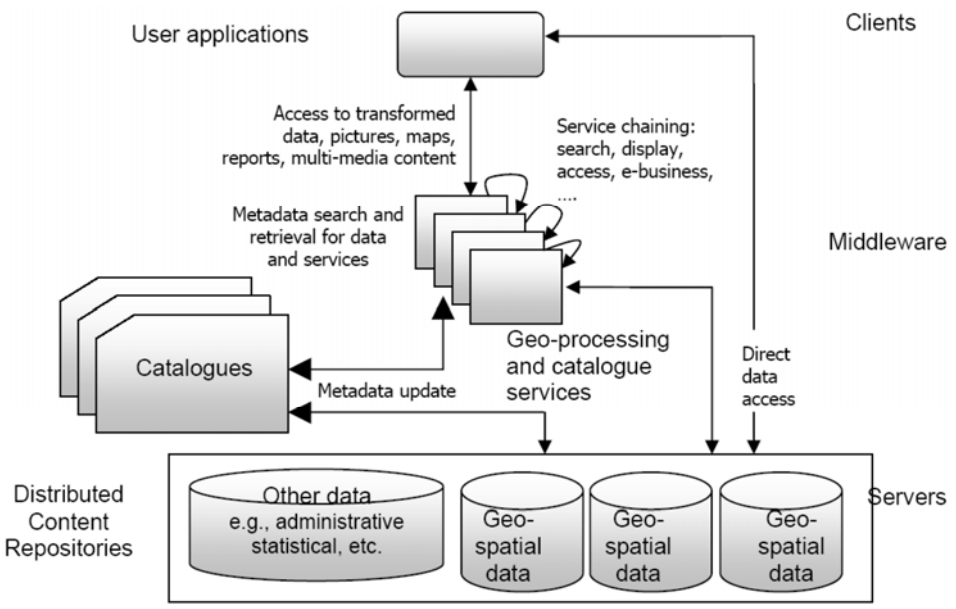

Figure 1: Architecture reference model for the INSPIRE directive (INSPIRE, 2007)

This paper is focused on improving the exploitation of SDI through a semantic layer between data sources and desktop applications. This layer will allow the users to work independently of data source types in the SDI and will let them reuse the semantic functionalities of some other users through a SQL derived language. The goal of this language is to be simple and powerful so that the users can manage the SDI independently of the repositories complexity.

After presenting the aims of the work we will describe the architecture of the solution and its internal model. Finally we will show the semantic layer in action in a theoretical use case of classifying watersheds by its runoff risk

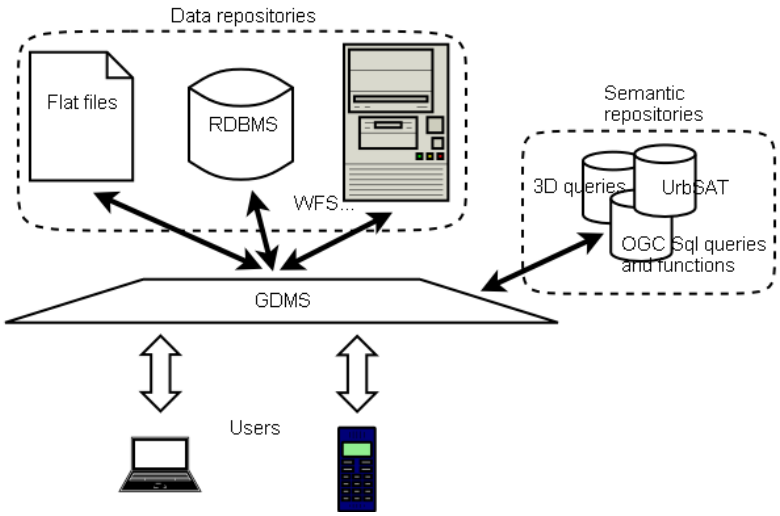

Figure 2: GDMS ability to access Data and Semantic repositories

RELATED WORK

The questions of spatial query language, process layer and spatial semantic repositories have been thoroughly studied over the last years. As reviewed by (Lopes de Oliveira, 1996), the need for spatial query formalism has been clearly identified by and answered in (Egenhofer, 1988a,b), (Güting, 1988), (Goh, 1989), with GEOQL (Ooi, 1990), with DML (Calcinelli, 1991), with SQL (Svensson, 1991), GPL (Egenhofer, 1991), Spatial SQL (Egenhofer, 1994), geoPOM (Nittel, 1997), GeoSQL (Wang, 2000). And there are already several efficient and industrial implementations that spatially enable Relational DataBases Management Systems (such as the PostGIS, a geographic extension for PostgreSQL).

Spatial and alphanumerical data may be located on volatile memory or persistent storage devices. As for databases, common geographic software applications require, in order to be considered usable, implementing a minimum set of features: ability to *Create* new entries, as well as to *Read*, *Update* and *Delete* existing entries. The Structured Query Language (SQL) is a world famous declarative programming language dedicated to CRUD data from Relational DataBase Management Systems (RDBMS). Beyond this, it is a data-oriented programming and query language that consists of three more command subsets:

- A standard Data Definition Language (DDL) with CREATE, DROP, ALTER common command,
- A standard Data Control Language (DCL) with GRANT, REVOKE common commands,
- A Transaction Control Language (TCL).

The SQL language has already been spatially enabled and extended by the Open Geospatial Consortium in the OpenGIS ”Simple Features Specification for SQL“ (Herring 2006a,b). Indeed, this specification defines a spatial object model and fundamental geometric functions, including:

- Spatial predicates (based on the DE-9IM model),

- Overlay functions (intersection, difference, union, symmetric difference),
- Buffer,
- Convex hull,
- Area and distance functions.

It proposes a standard SQL schema that supports storage, retrieval, query and update of simple geospatial feature collections but adapted to RDBMS (Herring 2006a,b). For example, the following query returns a new geometry defined by buffering a distance *d* around geometry object, where *d* is in the distance units for the Spatial Reference of geometry.

```
SELECT Buffer(geometry, d) FROM roads;
```

In this case *roads* is a table stored in a Relational DBMS and geometry is a column that contains the value encoded in Well Known Binary format (Herring 2006a).

GDMS (Anguix, 2005) is related with a similar layer used in the gvSIG project to manage the alphanumeric access. This layer is called *gdbms* and can only used for alphanumeric purposes. The work presented here is a general refactoring that mainly adds spatial functionalities and a more powerful SQL processor to GDBMS.

Motivations

Frequently, the heterogeneity of source types makes difficult the reuse of algorithms that are tightly coupled with the specified file format, database vendor, etc. With an intermediate layer between the user and the information source, the work developed by the former will not be coupled with the specificities of each format but with the intermediate layer itself, letting the work to be reused in a much more wide set of scenarios and of course simplify the learning curve for new developers.

One of the problems that such a layer arises is that it has to be able to access any potential source type . As the current number of source types is huge (ESRI Shapefile, spatial postgis, oracle spatial, WFS...) and can grow indefinitely, the user has to be able to extend the access capabilities of the layer. By introducing the concept of *driver*, GDMS intends to potentially fit any situation, even if the types of sources are not well known or currently the layer does not have the driver to access it.

The development of an abstract layer to manipulate spatial data in SQL is not a recent innovation. Several vendors like Esri with ArcSDE or MapInfo with SpatialWare proposed such a middleware. However, these systems have some limitations, due to:

- Interoperability across data repositories and spatial extensions of database technology
- Interoperability about SQL semantics. MapInfo used the User DefinedTypes (UDT) framework to build SpatialWare while ArcSDE used Abstract DataTypes (ADT). Practically, it has an impact on SQL syntax. For example, with ADTs to compute the area of a geometry the user writes "select geometry.area() from mylayer" while with UDTs he write "select area(geometry) from mylayer".
- Complexity of the their installation (need server like SQL Server 2000).

With GDMS, the idea is to postpone these problems by developing a high flexible and portable tool to build SQL queries. Currently, GDMS is dedicated for GIS clients but in the future it will be also a software component for SDI to process spatial data according to Web Processing Service.

GDMS provides an API that lets the user operate independently of source type. However, this API is not friendly enough for an end user and reduces the acceptance of the solution by final users. With the purpose of simplifying both access and manipulation of data sources, GDMS provides a

SQL processor that lets the execution of the common Data Manipulation Language statements against any source mapped by a driver. To avoid introducing a new grammar, GDMS fully preserves the SQL-92 grammar and adds to this standard geometric concepts and spatial functions as in OGC simple features SQL specification. As an analogy to spatial SQL for RDBMS, GDMS provides an extended SQL query language on heterogeneous data types.

It is the main purpose of GDMS to improve data creation and sharing. As in an SDI the consumption of data is as important as the production and sharing of data, the SQL processor in GDMS allows data feedback as if they were new data sources (materialized views). This means that the result of SQL queries can easily be integrated into the SDI as a *new* data source. Those data will be ready to be used by further SQL statements as any other existing data source (figure 3).

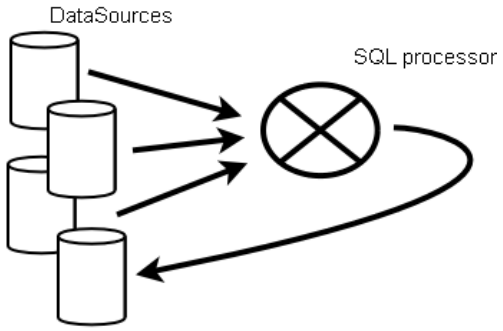

Figure 3: Data sources feedback after processing: implementation of materialized views

Finally, to improve further code reuse, GDMS introduces the concept of semantic repository. GDMS allows an extension to the semantics of the SQL language in terms of functions and custom queries. Functions and custom queries are artifacts that contain the implementation of some operations on the data and can be reused just by referencing its name into an SQL statement. This way, some user can implement a buffer operation and other can reuse it just by calling *buffer* in an SQL query: *select buffer(the_geom, 20) from mydata*. The collection of all this artifacts is the semantic repository. The purpose of GDMS is to maintain such a repository and encourage the feedback of new artifacts from the user to make it a growing knowledge base (figure 4). The semantic repository is more than a library of functions that can be extended by new user-defined add-ons; we aim to go a step further with this concept. Indeed, the goal is to provide a high level of spatial semantics:

- *checking*: a « just in time » validation process of input data type, range... made by a dedicated spatial SQL pre-processor
- *sharing*: Future releases will embed network-groupware capacity because user needs also to share spatial process and not only geographic data
- *interoperability*: The GDMS core is fully OGC compliant, so it is possible to copy and paste spatial SQL scripts from/into PostgreSQL/PostGIS.

At last, our semantic repository provides not only spatial functionalities but also description and (sometimes) useful use-cases for each of them.

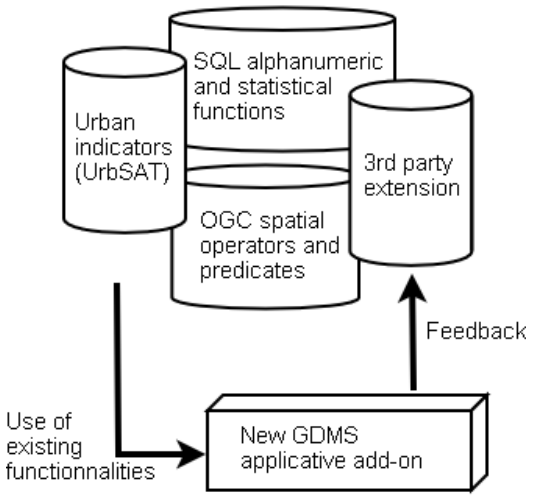

Figure 4: Feedback into the semantic repositories

IMPLEMENTATION DETAILS

Architecture

GDMS has a layered architecture (figure 5) where the upper layer contains the SQL interface and the semantic repository and the lower one is the driver layer. It contains details that are specific to each the source types. In case of the ESRI Shapefile driver, it contains implementation details to open files, decode the information and transform the contents into the GDMS internal model. PostgreSQL drivers will open the connection and will transform the information of the table into the internal model. This layer also provides extensibility to GDMS in terms of data source supported types. It is possible to create a driver to support potentially any type of data source.

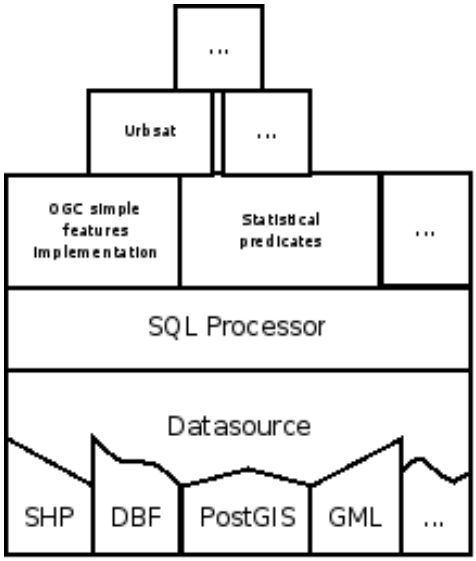

Figure 5: GDMS layered architecture

In spite of its extensibility, the driver layer is absolutely heterogeneous and building a SQL processor on top of such a layer would be very complicated. To solve this problem we place the *Datasource* layer on top of the drivers one. The main responsibility of this layer is the creation of a common API to manage all source types.

The SQL processor is built on top of this *Datasource* layer so it uses the common interface it provides to access any of the available data source types. Consequently it is possible to mix different types of data sources in the same SQL statement. For example it is possible to join a *shapefile* with a *CSV* file and a PostgreSQL table in one single SQL query. To reference the sources in a SQL statement it is necessary to map a name to each involved data source before. Here we have associated a Shapefile with 'shape', a PostgreSQL table with 'postgres' and a coma separated values file with 'csv'. Note that the associated name is the last argument of each "select register..." statement. Sources are referenced with those names in the 'from' clause of the last statement.

```
SELECT REGISTER ('postgresql','localhost', '5432',
'mygisdb','postgres','postgres','watershed', 'postgres');

SELECT REGISTER ('/tmp/waternetwork.shp','shape');

SELECT REGISTER ('/tmp/codes.csv','csv');

SELECT s.the_geom FROM shape s, postgres p, csv c WHERE s.id = p.id
AND p.id = c.id AND c.code = 3 AND p.code = 4;
```

On top of the stack there is the semantic repository that provides a functional package ready to be reused in new queries by the SQL processor.

Data Model

We have seen that the *Datasource* layer deals with the heterogeneity of the drivers and provides a common interface to the upper layers. We will now focus on the data model that is used to implement all features. It consists of one main entity called *DataSource*. A *DataSource* is an abstraction that

allows the management of one unique data source. This means that it is necessary to have as many *DataSource* instances as the number of sources to be accessed.

This abstraction has a typical tabular structure that uses rows to store each of the elements of the data source. This structure is close to the JDBC standard, where each row consists of a set of fields with a common structure for all rows in the *DataSource*. It is possible to see a relationship between this model and the *GeoAPI* model (currently being standardized in ISO19123) where *FeatureCollection* and *Feature* respectively match *DataSource* and its rows. Finally the field values of each row match the attribute values of a Feature.

All the information about field types is stored into a *Metadata* object that contains all field names, field types and restrictions for the types, such as length for string values, and coordinate reference system for spatial types. The model in this point allows a wide range of types for the fields that are compliant with the JDBC standard for alphanumeric types or with the OpenGIS Simple Features Specification for spatial types. Also note that there is no restriction in field types for a *DataSource*, it can have zero or more fields of any type so it is as possible to have several spatial fields as to have just alphanumerical.

The major components of the GDMS data model are shown in figure 6.

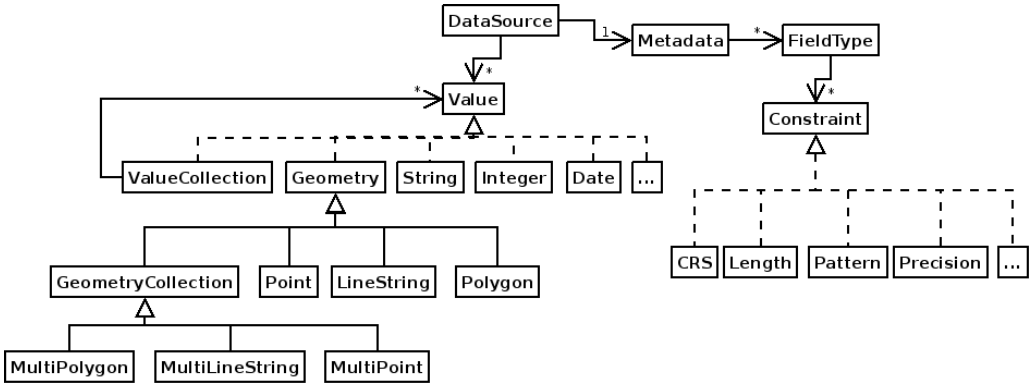

Figure 6: GDMS data model

EXAMPLE CASE STUDY

The result of our development is the explained semantic layer. It's an open source project that can be easily embedded in third party applications. Since the usage of the GDMS layer would be difficult without a user-friendly interface, GDMS is part of a broader GIS and SDI project named OrbisGIS (Leduc, 2007). As the rich client application of our SDI, OrbisGIS allows executing one or more queries and permits the presentation of the geometry of the data source and the visualization of issues that are related to spatial objects. For non-spatial features, the user can display the result into tables. In this section we will demonstrate how various SQL expressions can be applied including heterogeneous data source and different kinds of processing: spatial and non-spatial. We suppose a theoretical example where the aim is to classify watersheds comparing to runoff risk. The watersheds are classified on the basis of one arbitrary criterion: The percentage area of cereals being less than 20 meters away from a river network.

Data and processing schema

Three vector spatial data sources are considered: *watershed*, *waternetwork* and *landcover2000*. They are described in table 1.

Name	Geometry	Description	Data repository
watershed	polygon	A set of 3 watersheds used to compute density.	PostGIS table
waternetwork	linestring	Ditches and rivers in Sterenn watershed.	Esri Shapefile
landcover2000	polygon	Land cover classification in 2000.	Esri Shapefile

Table 1: Input data sources

The example is divided in the 6 steps illustrated in figure 7.

1.Accessing data,

2. Filtering data,

3. Identify the parcels that are located less than 20 meters away from a river feature,

4. Extract the parcels that are intersected with watershed,

5. Aggregate all parcels by watershed to compute the total area,

6. Compute density.

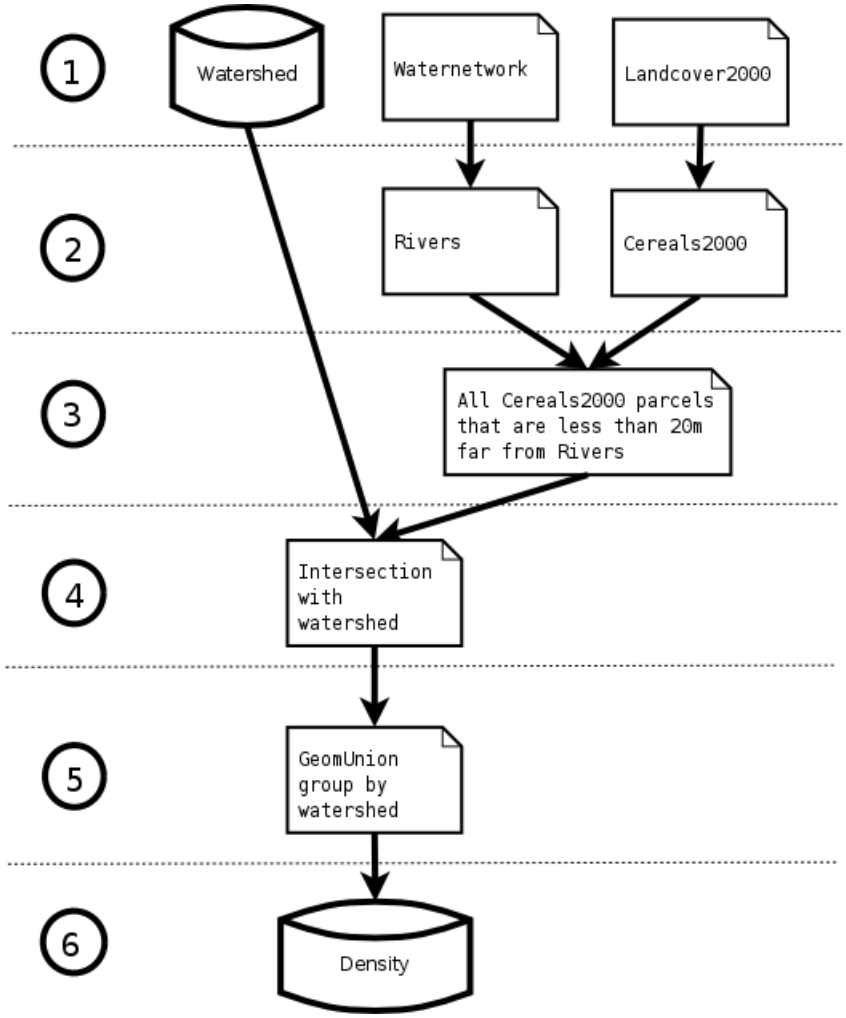

Figure 7: Data processing sequence

Application with GDMS

First, to start processing, the user must associate a name with all the input data sources.

```
SELECT REGISTER ('postgresql','localhost', '5432',
'mygisDbName','postgres','postgres','watershed', 'watershed');

SELECT REGISTER ('/temp/waternetwork.shp','waternetwork');

SELECT REGISTER ('/temp/landcover2000.shp','landcover2000');
```

The data are loaded in the OrbisGIS graphical environment and represented with specific colors and symbols (see Appendix for graphical results).

The first step of the methodology (step 2 in figure) involves restricting the data area. The user identifies and displays only the parcels where landcover type is equal to 'cereals' and only the waternetwork where the type is 'rivers'.

```
SELECT REGISTER ('/temp/rivers.shp','rivers');
SELECT REGISTER ('/temp/cereals2000.shp','cereals2000');
CREATE TABLE rivers AS SELECT * FROM waternetwork where
type_axe='rivers';
CREATE TABLE cereals AS SELECT * FROM landcover2000 where
type='cereals';
```

The second step uses two spatial functions (Buffer and Intersects) to select the parcels that are less than 20 meters of river features.

```
SELECT REGISTER ('/temp/cerealsLess20.shp','cerealsLess20');
CREATE TABLE cerealsLess20 AS SELECT a.the_geom FROM cereals AS a,
rivers AS b WHERE Intersects(Buffer(b.the_geom, 20), a.the_geom);
```

The third step calculates the spatial intersection between *cerealsLess20Intersects* and the watersheds.

```
SELECT REGISTER ('/temp/cerealsLess20Intersects.shp','cerealsLess20Intersects');
CREATE TABLE cerealsLess20Intersects AS SELECT Intersection(a.the_geom, b.the_geom)
AS the_geom, b.gid AS gid FROM cerealsLess20 AS a, watershed AS b
WHERE Intersects(a.the_geom, b.the_geom);
```

Before computing the area we will group all the parcels that are in the same watershed keeping their spatial information with an aggregated function called *GeomUnion* that will compute the union of all geometries.

```
SELECT REGISTER ('/temp/cerealsUnion.shp','cerealsUnion');
CREATE TABLE cerealsUnion as SELECT GeomUnion(the_geom) as the_geom,
gid FROM cerealsLess20Intersects
WHERE the_geom IS NOT NULL GROUP BY gid;
```

Finally, with the aggregated data the user can compute the percentage area of cereals which is less than 20 meters of a river network.

```
SELECT REGISTER ('/temp/density.shp','density');
CREATE TABLE density as SELECT ((area(a.the_geom) /
area(b.the_geom))*100) As density, b.the_geom as the_geom, gid FROM
cerealsUnion AS a, watershed AS b WHERE a.gid = b.gid;
```

The final result is a list of three watersheds with their corresponding cereals density that the user can display with a thematic map analysis.

Extending SQL semantic is one of GDMS main purpose and ability. Indeed, as a matter of fact, one of the first use cases was to use it to produce morphological indicators. Among around ten indicators, compacity (Miguet, 2007) as the ratio between the exchange areas of the envelope and the floor area - has been implemented as a GDMS function. Compacity can be used to analyze landscape structuring with respect to current agricultural landcover changes.

CONCLUSION AND FUTURE WORKS

With the implementation of GDMS the exploitation of the SDI is improved in terms of usability due to the fact that a simple universal SQL interface is provided to the user to potentially manage all the sources in the SDI. This interface is as close as possible to the SQL-92 standard and to the "Simple Features Specification for SQL" (Herring 2006a,b) to guarantee a smooth learning curve.

In spite of its simplicity from the user point of view, the SQL interface provided by GDMS allows the extension of the semantics of the SQL language. These extensions can be developed and easily

reused by the users of the SDI. The set of all these extensions is called the semantic repository and it will improve the cooperation and the reuse of work between the users of the SDI. Additionally, we have seen that the SQL processor is able to execute queries to create data sources of different types. This will let the user of the SDI to enrich the set of sources it has providing a feedback in terms of data through the common SQL interface.

As a further step in terms of simplicity for the user, we have planned to embed the GeoProcess tool. It is a graphic process builder that will let the user to create his own complex data processing sequence. The processes managed by this tool are composed of data sources and SQL scripts. GDMS has the ability to process both of them (execute script against data sources), so it is only necessary to link the inputs and outputs of the scripts between them. The input of the SQL scripts are the names of all data sources involved in the queries while the output results are the names of the produced data sources. As shown in figure 12, labeled rectangles correspond to data sources, while labeled circles correspond to the scripts. The directed edges that connect the nodes symbolize source dataflows. As an example, we provide the process diagram for the previous use case specified in figure 7, and represented in figure 8.

This user interface enhancement will not only simplify the definition and the implementation of complex process, but will also let the possibility to export this whole data processing sequence to the semantic repository. Thus, all this complex data processing sequence will be stored ("serialize") in order to be re-used as a "black box" in next processes ("unserialize"). This sequence of serialization and unserialization mechanism in a local or remote semantic repository, will be a new step towards reuse, workflow and even geoinformatic groupware. This will be step towards both data and methods sharing which is the ultimate goal of our scientific SDI.

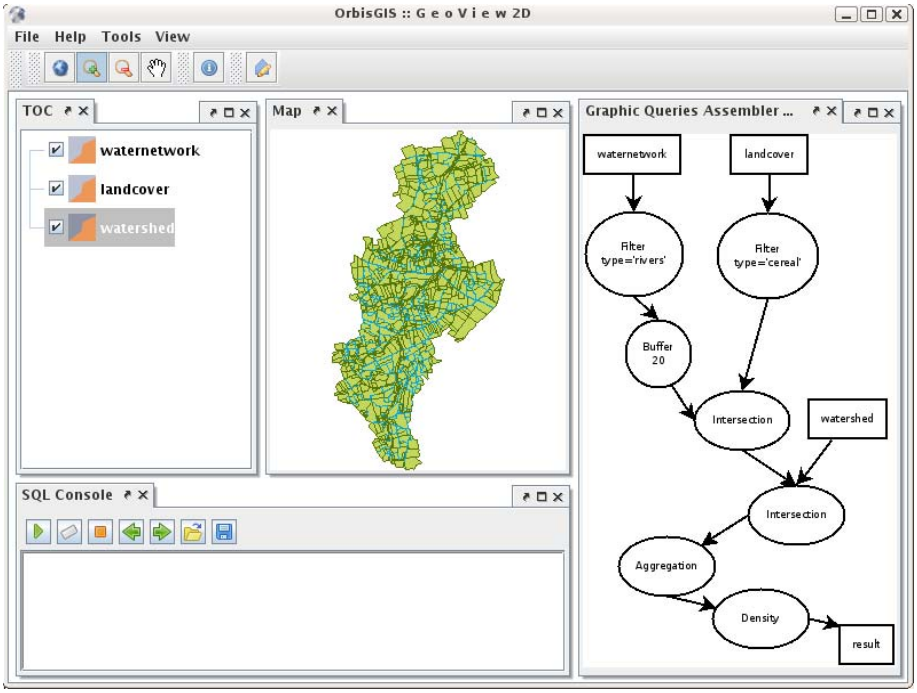

Figure 8: Use case translation in the graphic query builder context

AVAILABILITY

GDMS alpha version is available for public download at http://sourcesup.cru.fr/projects/orbisgis/

REFERENCES

Anguix A., Carrión G. gvSIG: Open Source Solutions in spatial technologies. GIS Planet Estoril, Portugal, 2005.

Calcinelli D., Maingenaud M. The Management of the Ambiguities in a Graphical Query Language for GIS. In Proceedings 2nd Symposium on Spatial Database Systems, 99-118. Springer Verlag Lecture Notes in Computer Science 525, 1991.

Clinton W. Executive Order 12906: Coordinating geographic data acquisition and access: The National Spatial Data Infrastructure, apr. 1994.

Egenhofer M.J., Frank A. U. Designing Object-Oriented Query Languages for GIS: Human Interface Aspects. In Third International Symposium on Spatial Data Handling, 79--97, 1988.

Egenhofer, M.J. and Frank, A., Towards a Spatial Query Language: User Interface Consideration, In Proceedings of the 14th International Conference on VLDB, L.A., California, 124-133, 1988.

Egenhofer M.J. Concepts of spatial objects in GIS user interfaces and query languages. Technical Report Report 90-12, National Center for Geographic Information & Analysis/NCGIA, 1989.

Egenhofer M.J. Spatial SQL: A query and presentation language. IEEE Transactions on Knowledge and etData Engineering, 6(1):86–95, 1994.

Goh P.C. A Graphic Query Language For Cartographic and Land Information Systems. In International Journal of Geographic Information Systems, 3(3): 145-255, 1989.

Güting R. Geo-Relational Algebra : a Model and Query Language for Geometric Database Systems. In EBDT88 – International Conference on Extending Database Technology, Venice, Italy, 506-527, 1998.

Herring. J.R. Opengis® implementation specification for geographic information - simple feature access - part 1: Common architecture, 2006.

Herring J.R. Opengis® implementation specification for geographic information - simple feature access - part 2: Sql option, 2006.

INSPIRE. Directive of the European Parliament and of the Council establishing an Infrastructure for SpatialInformation in the European Community (INSPIRE), http://www.ec-gis.org/inspire/, 2007.

Leduc T., Gonzaléz Cortés F., Bocher E. OrbisGIS : A GIS for scientific simulation. World Free Software Meeting (RMLL), Amiens, France, 2007.

Nebert D. Editor. Developping Spatial Data Infrastructures: the SDI Cookbook. Online book: www.gsdi.org/docs2004/Cookbook/cookbookV2.0.pdf , 2004.

Miguet F, Groleau D. Urban bioclimatic indicators for urban planners with the software tool SOLENE. in : Portugal SB07 Sustainable Construction, materials and practices : challenges of the industry for the new millennium, Lisbon, Portugal, 348-355, 2007.

Nittel S., Muntz R., Mesrobian E. geoPOM: a heterogeneous geoscientific persistent object system. Statistical and Scientific Database Management, 252:263, 1997.

Ooi B.C. Efficient Query Processing in Geographic Information Systems, Lecture Notes in Computer Science 471, Springer-Verlag (1990).

de Oliveira, Juliano L. and Medeiros, Claudia M. B. User Interface Issues in Geographic Information Systems. Technical Report IC96-06, Institute of Computing, Unicamp, 1996.

Rajabifard A. Developing Spatial Data Infrastructures to Facilitate Industrial and Mining Decision-Makings. Geomatics Seminar, United Nations Development Program, 2004.

Smits P.C., Düren U., Østensen O., Murre L., Gould L., Sandgren U., Marinelli M., Murray K., Pross E., Wirthmann A., Salgé F., Konecny M. INSPIRE Architecture and Standards Position Paper. JRC-Institute for Environment and Sustainability, Ispra, 2002.

P. Svensson and Z. Huang. Geo-SAL: A Query Language for Spatial Data Analysis. In 2nd Int. Symp. on Large Spatial Databases, LNCS 525, 119--140, Springer Verlag, 1991.

Wang F. Sha J., Chen H., Yang S. GeoSQL; a Spatial Query Language of Object-Oriented GIS. Proceedings of the 2nd International Workshop on Computer Science and Information Technology, Ufa, Russia, 2000.

APPENDIX: EXAMPLE RESULTS

The following snapshots are the results of the SQL statements of the example:

- up: initial data
- left low: *rivers* and *cereals2000* layer representation
- center low: *cereals2000* parcels close to the rivers
- center right: *cereals2000* parcels close to the rivers that intersect watershed