BUILDING AN OOGIS PROTOTYPE : EXPERIMENTS WITH GEO2

Laurent RAYNAL, Benoît DAVID, Guylaine SCHORTER
IGN / COGIT
2, Av. Pasteur, 94160 Saint-Mandé (FRANCE)
{raynal,david,schorter}@cogit.ign.fr

ABSTRACT

In 1991, an experiment began at IGN at the COGIT laboratory. Its objective was to examine the potential of object-oriented technology in manipulating and modelling geographical objects. From an experimental point of view, a prototype has been developed on top of a commercial DBMS, namely O2, from O2Technology. The prototype has been called GeO2. This article gives an account of the most prominent experiments that have been carried out at this stage of development. We concentrate on the management of large volume of data and look at processes which could be affected by it: loading, visualizing, accessing to (through a spatial index). A test of portability on C++ storage system is also mentioned. Then results of our experiments allow us to shed new light on object-oriented technology.

INTRODUCTION

IGN-F (the French National Geographic Institute) deals with a large volume of digital geographic information, for example, one data set from the BD Topo® (BDTopo 1992) (a database roughly equivalent to a 1:25,000 scale map), corresponding to an area of 20x28 km², represents as many as 60 Megabytes (in ASCII files). The total surface of France is a thousand times larger than this extract.

Furthermore, geographic data is complex as it combines both geometric components and semantic characteristics. Indeed, in handling geometric primitives (points, lines, polygons) it is necessary to describe geographic notions (reference system, projection system, coordinates), to model primitives and often to manage topological relationships (proximity relationships between primitives). However, geometric data handling generates processes that are algorithmically complex and CPU intensive and it puts high demands on the capabilities of computers. Display data on a map with a legend is a simple example of such a process (display order, definition of the legend according to the scale, numerous points to display, etc...).

In the face of these requirements, it appears that standard relational DBMS are not able to manage geographical information efficiently (Frank 1984). Object-Oriented DBMS seem more suitable for geographic data management than relational DBMS. Indeed, OO DBMS is needed most for managing the geometric primitives. However, in order to develop a GIS from an OO DBMS some reliable modules such as spatial indexes, computational geometry operators or topological structures are also necessary. Adding these modules should not require an unacceptable increase in the cost (memory and computation time) of data management. Otherwise, one could not speak of a geographical DBMS as a geographical extension of a DBMS.
Adding new types and associated operations, referencing objects, are here the basic capabilities a user needs in a DBMS or a GIS. An experiment began in 1991 at IGN at the COGIT laboratory to examine the potential of object-oriented technology in manipulating geographical objects. This aim was attained through the implementation of a prototype using a commercial DBMS, namely O2 (O2 1991), from O2Technology. This prototype, called GeO2 (David 1993a), is the core toolbox for a GIS, that means GeO2 is merely composed of the necessary functions of a GIS and is not designed to meet specific application requirements.

This article will describe the most prominent experiments that have been carried out at this stage of development. In a nutshell, in each experiment, we will briefly focus on activities and performances obtained (section 2). Finally, these experiments will form the basis for estimating the capabilities offered by OO data model, OO languages and OO DBMS (section 3).

**EXPERIMENTS : PRACTICAL USE AND PERFORMANCES**

Four topics are reviewed: a study on spatial indexes, a data visualization module, a generic data loading module and a portage in a C++/Versant environment. The geographic data model of GeO2 has already been presented in (David 1993a).

**Spatial Indexes**

When voluminous geographical databases are handled, most queries are spatial and accessing, scanning data may become a laborious and unrewarding task for the user. Then, specific mechanisms to get access to data quickly, introducing spatial ordering among data, must be added. A spatial index has this function. Three points are extracted from this experiment: the simplicity of our implementation, which allows easy use and easy extension of the module, the variation of window query results according to data sets, which is illustrated with the R*-Tree, one spatial index designed to fit with spatial objects' distribution and the influence of OODBMS in the R*-Tree construction process.

A module for spatial indexes, built on top of O2, is implemented through a very simple schema so that testing, adding or changing a spatial index can be done very easily in the process of manipulating real geographic data. One generic (or virtual) class called SpatialIndex has been introduced. The following four actions are allowed on it:

- insert() : to insert one object in the index (to "index" an object),
- delete() : to remove one object from the index,
- locate() : to make a point query (objects located on the query point are reported),
- search() : to make a window query (objects intersecting the query window are reported).

Three spatial indexes (i.e. a R*-Tree (Beckmann 90), a Quadtree (Samet 90) and a Point-Quadtree (Samet 90)) have been implemented as three sub-classes of the class SpatialIndex. Each function (insert, delete, locate, search) was then defined for each
index. Then tests have been made on several kinds of data sets*. However, because of the variety of data sets, spatial indexes can't be compared. Results fluctuate even for the R*-Tree which is designed to fit with spatial objects' distribution. Gradual queries of 10 objects, 100 objects, 500 objects and 1000 objects have been executed and show this diversity of results (Table 1).

### Table 1 : Time for querying objects in GeO² using an R*-Tree

<table>
<thead>
<tr>
<th>Number of objects</th>
<th>BDCarto ADM 1 tile</th>
<th>BDCarto CTA 1 tile</th>
<th>BDCarto CTA 4 tiles</th>
<th>BDCarto OCS 4 tiles</th>
<th>BDTopo 1/36 tile</th>
<th>BDTopo 1 tile</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>13 in 2 sec</td>
<td>10 in 1 sec</td>
<td>9 in 0 sec</td>
<td>7 in 1 sec</td>
<td>9 in 3 sec</td>
<td>17 in 13 sec</td>
</tr>
<tr>
<td>Average number of object</td>
<td>6 obj/sec</td>
<td>10 obj/sec</td>
<td>-----</td>
<td>7 obj/sec</td>
<td>3 obj/sec</td>
<td>1,3 obj/sec</td>
</tr>
<tr>
<td>100</td>
<td>100 in 2 sec</td>
<td>161 in 2 sec</td>
<td>92 in 2 sec</td>
<td>162 in 2 sec</td>
<td>136 in 5 sec</td>
<td>178 in 16 sec</td>
</tr>
<tr>
<td>Average number of object</td>
<td>50 obj/sec</td>
<td>80 obj/sec</td>
<td>48 obj/sec</td>
<td>81 obj/sec</td>
<td>27 obj/sec</td>
<td>11 obj/sec</td>
</tr>
<tr>
<td>500</td>
<td>-----</td>
<td>402 in 4 sec</td>
<td>633 in 5 sec</td>
<td>404 in 7 sec</td>
<td>553 in 10 sec</td>
<td>737 in 31 sec</td>
</tr>
<tr>
<td>Average number of object</td>
<td>-----</td>
<td>100 obj/sec</td>
<td>126 obj/sec</td>
<td>58 obj/sec</td>
<td>55 obj/sec</td>
<td>24 obj/sec</td>
</tr>
<tr>
<td>1000</td>
<td>-----</td>
<td>-----</td>
<td>1384 in 11 sec</td>
<td>1113 in 15 sec</td>
<td>1187 in 22 sec</td>
<td>1522 in 61 sec</td>
</tr>
<tr>
<td>Average number of object</td>
<td>-----</td>
<td>-----</td>
<td>125 obj/sec</td>
<td>74 obj/sec</td>
<td>53 obj/sec</td>
<td>25 obj/sec</td>
</tr>
</tbody>
</table>

Still working with the R*-Tree, one problem raised concerning its construction time. To reduce it, two issues were followed:
- The first issue concerns tuning of O². By changing the size of the client buffer (O2BUFFSIZE) and the size of the server buffer (O2SVBUFFSIZE) an improvement of about 50% was noted (1h30 instead of 2h30). Nevertheless, this construction process can definitely not be neglected yet.
- Another issue was brought up by using a temporary Unix file outside of the OO DBMS. This file contains all the index nodes while all spatial objects remain into O². However, at the end of the construction, each index node is copied into the OO DBMS. A considerable improvement was obtained (only 0h30 instead of 2h30 for construction). So, the temporary Unix file seems a very good alternative to speed up the R*-Tree construction process.

Our work on spatial indexes showed us that a major improvement can be expected from OO DBMS. It concerns the building time of the index. Indeed, it is mandatory to produce the indexation of objects in less or just as much time as for the indexation of objects with a temporary file. This process affects also the querying of objects through

---

* Data sets belong to two geographical databases produced by IGN : the BDTopo®, already mentioned and the BDCarto®. The BDCarto is a database equivalent to a 1:100,000 scale map and contains four main themes : ADM (administrative boundaries), CTA (transportation networks), HYA (hydrography) and OCS (land cover). One tile corresponds to an area of 20x28 km². 1/36 of a tile corresponds to the BDTopo acquisition unit.
the index. Partly because there is no clustering inside O2, the search process is still rather slow. We hope that these two limiting factors will be by-passed with the implementation of a spatial index inside the kernel of the DBMS, and of a clustering procedure for objects according to the index structure, which would map out each index node with each physical page of the OO DBMS.

The final result of our experiment is the simplicity of our DBMS environment to test indexes (very simple class hierarchy), easy extensibility (by the addition of SpatialIndex subclasses) and easy implementation on O2 with O2C (later experiments were made by Pr. Scholl's team (Peloux 1994) at CNAM using GeO2 too).

**Data Visualization**

A GIS without a visualization aid for geographic data is not a GIS. But in 1991, O2 did not provide vector graphical interface and we decided to develop our own user interface for vector data on top of Xlib functions. We wrote it in O2C, by the means of O2 classes.

The first benchmark consists in the display of objects on the screen because the display time is crucial in GIS. The key to produce a rather quick interface is to look at the time of polyline display. Indeed, while many vertices can belong to a polyline, it is important to retrieve them very quickly.

Two kinds of storage means for polylines have been compared:
- the first considers a polyline as a list of point objects (using O2 list constructor) and the display time is quite long;
- the second solution considers a polyline as an array of points, ensuring contiguity of storage and no identifier for points. This kind of constructor is not provided by O2. So, this storage means has been implemented as a C structure (an array of points) and is inserted inside O2 as a bits chain. This storage means clearly outperforms the first solution as it accelerates display time by a factor 5.

Our best results with the second solution (shown in table 2) reveal that displaying one hundred polyline needs approximately from 0,6 to 0,9 seconds. This value is confirmed even if there are many points per arc (see fifth column BDTopo contour lines tile). The conclusion of this experiment is polyline modelling. The polyline must be seen as an array of points, which means a contiguous storage of points dynamically extensible and no particular identifier for the points.

**Table 2 : Display time for geographic data**

<table>
<thead>
<tr>
<th></th>
<th>BDCarto ADM 1 tile</th>
<th>BDCarto CTA 1 tile</th>
<th>BDCarto CTA 4 tiles</th>
<th>BDCarto OCS 4 tiles</th>
<th>BDTopo contour lines 1 tile</th>
<th>BDTopo 1 tile</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of objects</strong></td>
<td>106 lines</td>
<td>688 lines</td>
<td>4173 lines</td>
<td>6023 lines</td>
<td>7082 lines</td>
<td>115 162 lines</td>
</tr>
<tr>
<td><strong>Number of points</strong></td>
<td>2 765 points</td>
<td>11 665 points</td>
<td>38 514 points</td>
<td>82 042 points</td>
<td>415 636 points</td>
<td>804 788 points</td>
</tr>
<tr>
<td><strong>Display time</strong></td>
<td>1 sec</td>
<td>5 sec</td>
<td>25 sec</td>
<td>47 sec</td>
<td>46 sec</td>
<td>15 min 38 sec</td>
</tr>
<tr>
<td><strong>Redraw time</strong></td>
<td>&lt; 1 sec</td>
<td>3 sec</td>
<td>18 sec</td>
<td>29 sec</td>
<td>31 sec</td>
<td>12 min 50 sec</td>
</tr>
<tr>
<td><strong>Average display time for 100 lines</strong></td>
<td>0,94</td>
<td>0,7</td>
<td>0,6</td>
<td>0,78</td>
<td>0,65</td>
<td>0,81</td>
</tr>
</tbody>
</table>
Generic data loading
A geographic database is designed to be used, exchanged or sold among producers and users. Loading geographic data sets is then the first task requirement in any GIS. Since each geographic data set has a data schema in accordance with an exchange data model, loading a geographic data set consists in the translation of the data schema into another one and in the loading of data. This task appears to be painstaking and critical since many exchange data models are available and the management of geographic data consumes a lot of computer resources.

So, the generic data loader developed for GeO2, is a useful module that automatically reads and translates the schema of geographic data into corresponding O2 classes, and ensures the loading of data, decomposing, if necessary, the loading of geographic data into several transactions. Several stages are necessary to achieve such a purpose (see figure 3).

1. The schema of the geographic data set is translated into our own schema definition file. This is our pivot data model to which translation rules must be specified for each exchange data model.
2. Then this file is parsed to constitute our data dictionary. The latter has been stored as O2 objects for easy development reason.
3. By using a temporary file the schema of the data set can be generated following O2 syntax and compiled into O2.*
4. Geographic data is effectively loaded (geographic objects, geometric primitives).

Two topics are now tackled: data schema translation and data loading.

Figure 3 : Synthesis of the loading of a data set

The decomposition of data schema translation in three stages ensures us a kind of re-usability for each stage. The first stage only depends on the exchange data model definition. So, if another exchange data model is chosen, only the first part will be modified. Similarly, the second stage depends on our data model while the third stage depends on DBMS. Then, each factor is clearly distinguished and evolution is easier.

* This generation part has been developed before O2 proposes the meta-schema function.
Nevertheless, the fourth task cannot be generic at all and will be modified whatever the evolution.

For the fourth stage, performances have been examined on the BD Topo tile which includes 95 000 nodes, 115 000 lines (with 733 000 vertices), and 37 000 areas. Loading such a volume of data (and building spatial indexes, constructing topology) is consuming computer resources and requires approximately 25 hours as for current commercial GIS. From a DBMS point of view, long transactions would be here convenient. However, this loading can not be done with O2 in a one-transaction process due to a limitation in memory size. No garbage collector, which could act as a background process, can't be made during the loading phase.

The solution we choose, consists in splitting the loading of geographic data (phase 4) into several DBMS sessions, launching O2 and quitting O2 respectively at the beginning and at the end of each step. For example, for loading geometric primitives, one step corresponds, in terms of nodes, to 5000 nodes each, in terms of lines, to 2500 lines and in terms of faces, to 5000 faces. But these values are completely configuration-dependent (for the determination of these thresholds see (David 93b)).

So, two conclusions emerge:
First, regarding time for loading, we have seen the great influence of topology building and spatial index construction. Indeed, they are the main cause of time consumption during loading and it is imperative that they should be optimized. Spatial index construction has already been tackled in the study of spatial indexes. But in topology building, we can only question the optimization of our algorithm.

The last point concerns the generic loading process that translates one schema in one model (the exchange data model) into one schema in O2 and that needs access to the meta-schema in order to create classes on the fly depending on what the user wills. This function is very useful and it must be offered by object-oriented languages.

Portage of GeO2 on C++/Versant
To test the portability of the prototype on a C++ persistent storage system, the portage of GeO2 in C++ using the OO DBMS Versant was decided in 1993. The whole application had to be re-coded in C++. We focused at the beginning on automatic translation between O2C and C++. But, after a pre-study, this task turns out to be too heavy to be done automatically. So, a "manual" rewriting of GeO2 has begun while keeping in mind the idea of an automatic translator (Cuzon 93).

Three levels of translation have been distinguished:
The first level is a translation out of context. It consists in substitutions of keyword blocks into other blocks (class declaration...). This is the easiest part.
The second level is much more context-dependent. Hence some capabilities offered by Versant/C++ have been chosen explicitly and this slightly modifies the meaning of the code. For example, object and object reference can be distinguished in C++ in class definition, in function parameter whereas object reference alone can be used in O2C.

And finally the third level reveals conflicts that have been detected only lately and that have prevented us from finishing and validating the portage.
Conflicts. The first conflict concerns inheritance and its two approaches; either overloading (in C++), or sub-typing (in O2C). As it is illustrated in figure 4, this mismatch implies that the code was adapted and some supplementary casts were made.

Figure 4: From O2C to C++

<table>
<thead>
<tr>
<th>in O2C</th>
<th>in C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>class SimpPoly</td>
<td>class SimpPoly</td>
</tr>
<tr>
<td>...</td>
<td>{... virtual SimpPoly&amp; geom();</td>
</tr>
<tr>
<td>method geom : SimpPoly,</td>
<td>...</td>
</tr>
<tr>
<td>... end class</td>
<td>}</td>
</tr>
<tr>
<td>class CompPoly inherit SimpPoly</td>
<td>class CompPoly: virtual public SimpPoly</td>
</tr>
<tr>
<td>...</td>
<td>{... CompPoly&amp; geom();</td>
</tr>
<tr>
<td>method geom : CompPoly,</td>
<td>...</td>
</tr>
<tr>
<td>... end class</td>
<td>}</td>
</tr>
</tbody>
</table>

But the most important conflict that considerably affects the developer’s productivity concerns persistence. Here the greatest difference in object-oriented development has been detected.

In fact the problem occurs each time one decides to create an object. With persistence by reachability systems, the programmer does not have to care about the temporary or persistent character of the object. The system will garbage this object if it is no longer referenced.

On the contrary, with an explicit persistence system, the programmer must decide if this object will be persistent or temporary. This requires a preliminary knowledge of which are the temporary objects, which are the persistent ones. Because the transformation of temporary objects into persistent objects causes a duplication, there is less efficiency and the heap can be saturated (we have come up against this problem). So the right decision is, once again, left to the user's common sense through the adoption of conventions (i.e., always distinguish two kinds of methods: the persistent one and the temporary one).

After this experiment, we can establish the critical aspect of the portage that reveals all the weak points in the design of the prototype. In fact, every unsettled situation, every breakpoint quite often corresponds to the situation of an implicit action in another context (as in object and object-reference differentiation).

Finally, due to the conflicts, we have given up the comparison of OODBMS performance but only felt the great difference in the development with an explicit persistence system.

ON OBJECT-ORIENTED TECHNOLOGY

Thus, on the basis of the experiments described above, several questions about OO model, OO languages and OO DBMS are raised. This segmentation into three packages results in a re-organization and discussion of each conclusion in our experiments.
Object-Oriented Model

One delicate point is the object/value dichotomy found in O2. And the reasons for choosing an object rather than a value are explained rather scantly. From our experience, two indicators can serve as a guide to choice-making. They are directly related to the use of the data, as it is shown now:

• Must this data be shared between several other objects?
  An object reference is mandatory in case of sharability. Otherwise a value is sufficient.
• Could we attach any behaviour to the data? Is it dependent on the user's choices?
  In this case, we rely upon the classical encapsulation concept that is the association of data structure and data behaviour into one class. Once again, for active data, it will be better to choose an object rather than a value.

The second extension is the inheritance mechanism. Inheritance is a powerful abstraction for sharing similarities among classes while preserving their differences. However, its meaning can be ambiguous.

First, it can be interpreted as a generalization construct. It facilitates then modelling by structuring classes, capturing the similar external behaviour of different classes. A virtual super-class is introduced and can act as a representative of its sub-classes in other modules. For example, it is the SpatialIndex class in the module of spatial indexes. This is a conceptual approach similar to the sub-typing approach in programming languages.

But it could also be interpreted as a means to re-using services defined in another class, overriding operations for extension. For example, if we want to re-use the coordinate storage capabilities for topological entities, topological entities will inherit from geometric entities. Through this link, several classes tend to be closely joined and consequently, two modelling constructs tend to be joined. A new modelling hierarchy is created which is much more complex and very difficult to master on a long-term basis. For example, evolution inside the geometric domain is limited because it must fit with an evolution in the topological domain.

On the other hand, this approach has the advantage of being more efficient because it avoids to create two objects, to redefine all operations. Here, design options, for optimization, are opposite to long-term development and evolution. Note that this inheritance meaning is close to the overloading approach in programming languages.

Object-Oriented Languages

The function of a language is to easily translate the notions developed in the model and to create some uniformity through syntax rules and efficient processing. An object-oriented language gives the developer a more efficient way of organizing and testing his/her programs by means of classes (a kind of re-usable software component). And they are successfully used. Nevertheless, the management of persistence is still delicate.

Persistence is the property of data that enables it to exist for an arbitrary period of time: as short or as long as necessary. But distinguishing heap-allocated instances and long-term instances generates a considerable programming overhead (Atkinson 1983), a disturbing existence of two distinct type systems; instead, the type-orthogonal persistence allows each kind of persistence to every data whatever its type. It makes long-term storage completely transparent and is supported by an extension of classical
garbage collection (every data that is no more referenced by a persistent object is
garbaged). Persistence by reachability system belongs to this group. But, currently,
garbage collection is not efficient enough and produces an heavy overhead.

Then the current solution, that represents a backward step in comparison with
orthogonal persistence, is explicit persistence. This means of development tends to be
close to current developments on RDBMS.

The last point is about the high level concepts that have been enumerated all along
the experiments described. Those represents the short-term need for developers: meta-
schema functionality (mandatory when loading), array constructor (for polylines).

**Object-Oriented DBMS**

With OO DBMS, we face the last component that must surround every concept
mentioned above. It is therefore all the more difficult to produce and fortunately, we
found a rather complete product, which met our requirements, on the current market.
So only two subjects are tackled:

As we have already said, the major difficulty in OODBMS was the insertion into
the kernel of the spatial index and of the clustering mechanism (associated to the
spatial index structure). But a more general problem concerns the extensibility of such
a system (Schek 1993) with regard to the user’s needs. What is the solution if we
discover that however great our optimization efforts, topology building still remains a
bottleneck in the loading process for geo-data? And this question can be extended to
each new research subject on geographical information (spatio-temporal database,
multi-scale database). These models are, or will be, more elaborate than the current
object-oriented model but we do not intend to build a completely new system. Perhaps
it is necessary to re-think the DBMS as a module and clearly define what is the outer
interface to this module?

The second point, regarding short-term needs, concerns indicators that are lacking at
present among available products. Much information on time, memory management,
actual buffer size, I/O requests is needed to explain where time is wasted, where to
search, and this kind of aid is not provided by OO DBMS yet.

**CONCLUSION**

It is worth noting that these subjects are not examined in a global context. We kept
aside the subject of distributed system, of interoperable systems etc.... But results of
our experiments allow us to shed new light on object-oriented technology. Indeed,
three keywords can be extracted concerning the adequation of OO DBMS for GIS
activities: performances, extensibility and standardization (portability).

GeO2 is now a platform for further geographical research. Some particular points are
being treated, such as, a study on a multi-scale database (Raynal 1994), on a spatio-
temporal database (Latarget 1994) and on the modelling of geometric accuracy (Vauglin
1994). We know the capabilities of our prototype and some of its limitations.
However, by choosing O2 and O2C, we preserve the qualities of easy development
and easy exploratory experimentation that is provided when working with persistence
by reachability system. This is a conscious choice even if it does not go hand in hand
with present-day industrial processes. But, on the other hand, it is not the prime
objective of a research.
REFERENCES


David, B., Raynal, L. and Schorter, G. 1993b, Evaluation of the OO approach for geographical applications: Deliverable of ESPRIT project AMUSING n°6881.


Samet, H. 1989, The Design and Analysis of Spatial Data Structures, Addison-Wesley Reading, MA.
