Invited paper

OO paradigm meets GIS: a new era in spatial data management

Antonija Mitrović
Computer Science Department, University of Canterbury, Christchurch, New Zealand
tanja@cosc.canterbury.ac.nz
Slobodanka Djordjević - Kajan
Computer Science Department, University of Niš, Niš, Yugoslavia
slobodanka@efnis.elfak.ni.ac.yu

Abstract In this paper we examine the concepts of object-oriented paradigm and characteristics of spatial data management in order to argument their compatibility. Most problems with GIS applications come from inadequacies and low expressiveness of computational models used and the impedance mismatch problem. The OO paradigm is a natural one for highly complex domains such as ones involving spatial entities, because it maintains a direct correspondence between real-world and application objects. The paradigm also supports all phases of software development. We analyze two different approaches to OO development of GIS applications, based on OO and relational DBMSs. The conclusion is reached that as the OO databases has still a number of problems to be solved, the approach based on relational DBMSs provides the necessary effectiveness, ease of use and reusability of existing resources.

1 Introduction

Geographic information systems are computerized systems for managing data about spatially referenced objects. GISs differ from other types of information systems in that they manage huge quantities of data, require complex concepts to describe the geometry of objects and specify complex topological relationships between them. In addition, GIS data are typically used by various groups of users with different views and needs.

Geographic Information Systems (GIS) have been around for over three decades now, and a firm foundation has been established in a form of widely accessible spatial databases, GIS organizations and standards. The basic GIS technology is established, there is a large number of commercially available systems; however, the costs involved in developing a GIS application are still too high for most end–users in terms of hardware/software price, long time needed to get results and uncertainty of the final outcome.

We look at the system development process first in the next section in order to identify the cause of these problems and highlight the inadequacy of the computer models employed as the most important one. Next we consider the object-oriented (OO) paradigm as a way for overcoming the problem. OO paradigm has been a major research topic for a long time now, its main advantage being direct correspondence between real–world and application objects. The third section presents the basic concepts of OO paradigm. We consider illustrate several existing approaches and ...

2 System development process

The process of software development essentially consists of a series of transformations which convert the observations about the real world into the usable software product (figure 1). The first phase of requirement specification and analysis studies the application domain in the real world in order to produce a model of it. The analysis model reflects required functionality of the system to be built. It is further expanded in the design phase, resulting in the model containing details of the chosen computational paradigm as well. In the next phase the design model is expanded into the physical one, containing additional implementational details.

Real Analysis Design Implementation world (conceptual) (logical) (physical) model model model

Figure 1. Software development cycle

The critical factors for the successfulness of the software product are the quality of the chosen representations, the accuracy of the models developed and the complexity of the mappings between them. The latter is also known as the *impedance mismatch problem*: transformation between varying representations. This transformation may in some cases result in loss of information if the resulting model is not as expressive as the starting one; additionally, it may be the case that some information is hidden by different concepts of the final model. Even if we do manage to represent all aspects of one model using the concepts available in the other one, time is needed to restructure the information represented.

The ideal case would be to use the same representation all through the software development cycle. Contemporary software development practice does not follow this recommendation; however, we will see in the next section that OO paradigm supports software development more uniformly than other computational paradigms.

If we take a closer look at the implementational model, we can also see the consequence of the impedance mismatch problem. Figure 2 illustrates various representations an object from the application domain may have in a software system. It has a graphical representation used to communicate with the end-user; the application itself may exploit a very different representation. Finally, in order to store permanently data about the object, the third representation may be used. When the application is run, these representations are mapped to each other as needed, in some cases causing low performance of the software system.

Real Interface Application Database world object object object object

Figure 2. Different representations of an object

GISs as we now today are tools for the development of GIS applications: they are similar to DBMSs. The development of a GIS is a two-phased process: in the first phase the developer of GIS software implements the software going through all the phases in figure 1. In the second phase the end-user (or an organization acting on his/her behalf) develops a GIS application on top of the GIS tool going through same phases again. Such an approach, inspite of obvious advantages compared to the process of GIS application development using a programming language alone, is characterized with the following problems. The GIS tools are generally not extensible and may not match the requirements of an application. Moreover, such tools are usually macro languages which do not support the OO paradigm, which makes the application development very complex.

3 OO concepts

Historically, OO paradigm originates from the Simula 67 programming language, but has gained wide industrial acceptance only since late 1980s. OO paradigm encompasses OO data models and methods for various phases of software development. The main advantage of the paradigm is its ease of understanding; it enables natural representation of real world objects, their mutual relationships and behaviour and is therefore close to end-users. An OO application consists of a set of objects with their own private state, interacting between themselves. OO systems are easy to maintain because they are modular and objects are independent of each other; a change in one object should not affect other objects in the system. OO paradigm eliminates the need for shared data areas, thus reducing system coupling. The paradigm supports reusability: objects are self-contained and may be used in other, sufficiently similar applications. It also supports distributiveness and parallelism.

The greatest difference between OO and earlier data models, like relational (Codd 1970) or semantic ones (Chen 1976, Hammer 1981, Elmasri 1985), is their capability to represent the functional (behavioural) component of entities, not just their structure. OO paradigm is best suited to non–traditional applications, such as CAD, GIS and CASE. The additional requirements introduced in such domains include highly complex objects, special data types for unstructured data such as raster and vector images, non–standard operations and multiple versions of data.

Despite huge interest in the area and numerous OO programming languages, models and methodologies proposed, little agreement has been achieved. Although C++ is the dominant OO programming language nowadays, none of the models and methodologies has gained universal acceptance so far (Maier 1989, Worboys 1990). Moreover, there is no general agreement on what concepts an OO model should support or how these concepts are defined (Worboys 1994). Here we will briefly list the concepts we see as fundamental and attempt to define them as well.

An *object* is an abstraction of an entity in the real world; it reflects the information about the entity and methods for interacting with it. Objects encapsulate complex structures of data with the behavioral component. The structural component of an object is described by means of *attributes*, or its characteristic features. The *behavioural component* of an object is represented as a set of *methods* (operations) that the object performs in appropriate situations. A *class* is a description of a set of objects describable with a uniform set of attributes and methods. A class therefore represents a generalization of a set of objects with common properties and behaviour. Objects are instantiated (generated) from this description.

Object identification enables each object to be uniquely distinguished from all other objects in the database. An object identifier (OID) is generated by the system at the moment when the object is created, independently of the values of its attributes. OID is immutable, that is, stable for the life-time of the object. An OID is dropped only if the object is destroyed; furthermore, it should be used only once in the database in order to be associated with just one real–world object.

Encapsulation is the principle which enables an object to hide its structure and/or behaviour from other objects. Internals of an object are accessible only via its interface, that is the operations known by the system.

Inheritance is a mechanism which allows developing new classes by modifying existing ones. This is the mechanism which facilitates reuse of existing class hierarchies. Inheritance defines generalization and specialization relationships between classes, by developing abstractions or subtypes of classes. Inheritance may be selective, which means that only some parts of the superclass may be inherited. Multiple inheritance means that a class may inherit attributes and methods from multiple superclasses. In this case we have class lattices instead of class hierarchies.

Aggregation is the construct which enables objects of different types to be amalgamated into other objects. This concept facilitates modeling complex objects. Aggregation corresponds to the "a part of" relationship between two objects, where the component objects are also known as embedded objects. A complex object cannot exist without containing at least one aggregated object.

Association enables specifying relationships that exists between various objects in the database. Associations may be expressed explicitly in some OO models, while in others they are represented as reference attributes. In the latter case, the value of a reference attribute is the OID of the associated object. Additionally, some OO models have the construct of *ordered association* which takes into account the order of associated objects.

Operator polymorphism (operator overloading) is the mechanism which enables operator to handle arguments of various types. It ensures that the appropriate version of the operator will be applied on supplied arguments.

Version control is often needed in non–traditional areas, where different "versions" of the same object may be important. For example, in GIS applications it can be the case that the boundary of some spatial object changes in time, so information about the previous state (version) and the new state of the same object is required. Usually, versions are implemented as different objects, which means that they will have different OIDs.

As stated earlier, OO paradigm specifies methodologies for various phases of the software development cycle. There is abundance of OO methodologies based on different OO models and notations, which are in essence semantically equivalent. They include analysis methodologies (Coad 1990, Shlaer 1988) design methodologies (Coad 1991) and integrated ones (Rumbaugh 1991, Booch 1994).

An OO DBMS must support the OO concepts and also provide persistent storage of objects. There are commercially available OO DBMS like ONTOS (1991), Gemstone, ObjectStore and O_2. However, they have not been widely used in large-scale applications and there are still open problems to be solved. Query optimization is very difficult in OO databases due to complex structures of objects and large number of operations supported. The problem with indexing comes from the information hiding principle used in OO paradigm; the state of an object is not inspectable directly. Furthermore, shareability of users views is hard to support, since the

4 Spatial data management

Spatial entities (also referred to as spatial objects or features) are natural, man–made or abstract objects of interest. These objects are described by geometrical (position and shape of the feature), topological (relations to other features) and attribute (all other non–spatial, usually numerical and textual) data.

Spatial data representation and management have always been the primary concern in GIS research. Let us briefly turn to the history of spatial data handling. At the time when first GISs appeared, the task of displaying graphical data was very demanding due to limited hardware resources. Therefore, early GISs were usually based on systems that supported visualization of spatial data, such as CAD systems. Low–level data structures used in CAD systems, like arrays, linked lists implemented as arrays and dynamic linked lists (Nagy) were used to store data and traditional file handling techniques were employed in systems for managing data. This approach provided facilities for displaying and editing geographic data. However, the nature of geographic data differs significantly from CAD data, mainly because of its volume and importance of topological links. This discrepancy resulted in the development of new data structures for GIS, such as various types of trees (R, Quad or KD trees) (Samet 1990) for fast spatial retrieval on large data sets. A number of retrievals in GISs is not possible or is computationally too intensive if only geometric data are available. Examples of this include analyses of networks (power, phone or gas supply networks), polygon overlapping and similar problems. Therefore, such analyses were supported by numerous topological data structures, like POLYVRT, DIME or TIGER (Peuquet 1990). These structures encompass both geometrical and topological data, sometimes accompanied with attribute data as well. Topological data contain data about neighbouring objects (e.g. blocks of buildings on the left and on the right side of a street).

The commercial success of data base management systems and their wide usage reflected in the area of GIS also. The first GISs based on relational databases stored only attribute (non–spatial) data in the database, mostly because of limited performances of contemporary computer systems. Spatial data were still kept in property data models; these two parts had links to each other, but were supported by separate data managers. Such a hybrid architecture was commercially successful (mostly due to successfulness of ARC/INFO (ESRI 1991) and is still dominant at the market today. Despite of the success of hybrid GISs, the separation of the underlying data in two parts introduced significant problems, specially concerning the lack of support for ensuring data security, integrity control, multiple user access and concurrency management for spatial component of the database.

A natural solution was to integrate spatial and non-spatial components under the control of a RDBMS. Initial efforts to implement spatial database on pure relational model (van Roessel 1987) showed that such an approach, although theoretically feasible, is unsatisfactory due to low performances. As pure relational data model is not suitable for spatial data, information concerning one object is spread over many relations due to the normalization performed upon the relations, and many join operations have to be performed in order to recreate complex spatial objects. The relational data model alone cannot provide appropriate indexing and retrieval operations for spatial data.

There are two subsequent research directions in using databases for storing of both attribute and spatial data. The first one is the application of OODBMSs, resulting from the general acceptance of C++ as the major implementation tool for GIS. This approach is further discussed in the next section. The other approach is the extension of the relational data model and modification of RDBMSs so that spatial data can be stored while retaining the advantages of the relational data model. This approach is based on abstract data types and relaxing the constraints of the relational data model (normal forms). Research in this area started in 1980s (Abel 1989, Guptil and Stonebraker 1992, Mitrović 1993), and commercial GIS tools based on the extended relational model are available (Smallworld, ESRO SDE, Oracle Multidimension).

5 OO GIS

Let us first restate the fundamental requirement for GIS applications. A GIS should facilitate abstract representations of real world objects which are understandable and easy to use.

No commercial database management system directly supports spatial data management nowadays (Kim et al. 1993). The additional requirements for a spatial data model include single data storage for spatial and attribute data, good spatial capabilities, persistence and storage in layers (van Oosterom 1993).

If we reconsider the characteristics of OO paradigm, it can be seen that it is a suitable one for the development of GIS applications. There are two approaches to coupling OO applications with databases (Lee 1994). The *direct object storage* uses the same OO model for the application and the database. The *indirect base relation storage* couples an OO application with the relational database. The advantages of the former are obvious: as the same data model is used, the impedance mismatch problem is avoided. The disadvantages stem from the problems with OO DBMS discussed earlier.

Choi (1992) presents a GIS kernel built on top of an OO DBMS, by augmenting it with an additional layer containing specific geographic classes and operations and the second one performing query processing. Milne (1993) reports of the similar endeavour built on top of ONTOS and using the US Spatial Data Transfer Standard (SDTS 1992) as a model for basic spatial data classes. The authors extend the class library in C++ provided in ONTOS with spatial classes. Other extensions include a GUI and additional geographic modules. They use IFO (Abiteboul 1987) as the underlying OO data model.

If a RDBMS is used at the implementational level instead, then the problem of impedance mismatch becomes significant. On the other hand, RDBMSs are widely used and hardware and software platforms needed are already available. The problems of sharable concurrent access to data are solved, and the growth of the system is supported. These are the reason in favour of the indirect base relation storage approach for practical applications, especially taking into account the estimate that the current market for OODBMS is less than 1% of the total database market (Kim 1993). The relational data model is the prevalent data model today and it will be the dominant one for at least the next decade.

The indirect base relation storage approach can be realized from two perspectives: *object-centered* and *relation-centered* (Lee 1994). The difference between the two comes for the primary source of data. In the object-centered perspective, relation schemas are generated from class descriptions. In the latter one, it is assumed that relation schemas already exist, and class descriptions are derived from them.

When an OO application is built on top of an existing relational database, the programmer specifies the class descriptions, and the mappings between the classes and relations, and class attributes and relation attributes also. In order to instantiate objects from relations, a programmer must write the definition of the view which contains the necessary data. In such a way, there are three levels in the architecture:

1. a set of base relations (relational database);
2. an intermediate level consisting of view–objects generators and decomposers;
3. objects themselves.

View–object generators extract data from the database and assemble the data into objects. Essentially, there are as many view–object generators as there are classes in the application. A view–object generator contains the query needed to collect data from the database, information about relations that hold relevant data and class definitions so as to be able to create appropriate objects. As pointed out by Wiederhold, the query used to instantiate an object from corresponding relations can be quite incomprehensible for users (programmers). The reason of this comes from the normalization process applied to the relations, so that information about one spatial object is spread over several relations in general. If the object is a complex one, this problem gets even bigger, because a complex object contains a number of primitive objects.

Object-centered perspective gives more freedom to the programmer. GinisNT is one realization of this approach (Mitrović 1994, 1995). GinisNT is an OO tool for development of GIS applications. We use an OO model at the conceptual level and the relational data model at the implementation level. The existence of a RDBMS is completely transparent to the user by existence of intermediary components in the system which perform mappings between the two automatically. The application developer is freed from worrying about data storage details and can concentrate on the application which appears to him/her to be completely object–oriented.

In order to elevate the burden of the interaction from the end–user, GinisNT uses metadata repository to find necessary data about organization of the database. The existence of the metadata repository ensures that the end–user does not have to remember how data are actually stored and how to access it. The classes in the metadata model correspond to the concepts supported by the GinisNT object–oriented data model.

GinisNT provides support in OO modeling and design of the GIS application. OO model for the application can be developed by selecting some of the existing spatial and non-spatial classes from GinisNT class library, specifying new classes by inheritance from the existing ones or by developing completely new classes. The components of the system map the OO model of the application being developed into the relational schema and create necessary relations. GinisNT also provides run-time support for the applications, by interpreting user's requests, invoking appropriate methods and generating database statements on the basis of information present in the metadata repository. As object creation, instantiation, update and retrieval operations are provided automatically, the user is not aware of the relational database used on the internal level. Further details about this approach can be found in this volume (Mitrović 1996a, 1996b, Stojanović 1996, Stoimenov 1996).

6 Conclusions

Despite significant improvements in the area of GIS, the diffusion of GIS technology still confronts numerous difficulties, like the trade-off between high resource requirements in most commercial GIS tools and low level of available resources. GISs are complex systems the requirements of which differ from the ones posed before traditional information systems. GISs manage huge quantities of data, require complex concepts to describe the geometry of objects and specify complex topological relationships between them.

Object-oriented paradigm is perfectly suited to such requirements. It is natural to represent spatial objects as objects in the application; the paradigm also greatly reduces the problem of impedance mismatch, as it supports the usage of the same model in different phases of software development. GISs request persistence storage of objects; therefore, the ideal solution is to use an OO DBMS to store data. However, there are still unsolved problems with OO databases. They are not widely used in industrial applications.

On the other hand, there is a well established stage for relational DBMSs. As the resources are available, in form of software and existing databases, we feel that the solution is to use the relational database on an internal level and to provide support for OO application development on the conceptual level. All the mappings between the two models are done transparently and the existence of a RDBMS is completely transparent to the user. The OO paradigm supports the scalability and reusability of developed software, which are especially important in resource-demanding areas such as GIS.

Among the research currently being done, work on the SQL 3 standard promises a lot. SQL3 is an integration of the relational and OO data models; it is a complete language for the definition and management of objects (Sulima 1995). It is based on abstract data types, thus promoting interoperability and sharing of data. In addition to the basic standard, a number of additional projects are being pursued. One of them is SQL/MM, an effort to standardize multimedia spatial applications. SQL/MM, besides other things, contains 219 spatial ADT elements, such as coordinates, points, lines, vectors, cellular structures, time, metadata as well as temporal ones. The completion of SQL3 is estimated for 1998 and future DBMSs supporting it will undoubtedly play a significant role in GIS applications.

References

1. Abel, D., *SIRO-DBMS: A database toolkit for GIS*, Int. J. GIS. Vol. 3, pp. 103-115, 1989.

2. Abiteboul, S., Hull, R., *IFO : a formal semantic database model*, ACM Transactions on Database Systems, Vol. 12, pp. 525, 1987.

3. Booch, G. *Object-oriented Analysis and Design with Applications*, Benjamin Cummings, 1994.

4. Chen, P., *The Entity Relationship Model — Toward a Unified View of Data*, ACM Transactions on Database Systems, Vol. 1, No. 3, 1976.

5. Choi, A., Luk, W.S., *Using an OO Database System to Construct a Spatial Database Kernel for GIS Applications*, Computer System Science and Engineering, Vol. 7, pp. 100-121, 1992.

6. Coad, P., Yourdon, E, *Object-oriented Analysis*, Yourdon Press, 1990.

7. Coad, P., Yourdon, E, *Object-oriented Design*, Prentice Hall, 1991.

8. Codd, E.F., *A Relational Model of Data for Large Shared Data Banks*, Communication of the ACM, Vol. 13, 377, 1970.

9. Elmasri, R., Weeldreyer, J., and Hevner, A., *The Category Concept: an Extension to the Entity Relationship model*, Int. Journal on Data and Knowledge Engineering, Vol. 1, No. 1, 1985.

10. ESRI Inc., *Understanding GIS: the ARC/INFO Method*, 1991.

11. Guptil, S., Stonebraker, M., *The Sequoia 2000 Approach to Managing Large Spatial Object Databases*, in Proc 5th Symposium on Spatial Data Handling, D.Cowen (ed), pp. 642-651, 1992.

12. Hammer, M.M., McLeod D.D., *Database description with SDM: a Semantic Database Model*, ACM Trans. on Database Systems Vol.6, No.3, pp.351-386, 1981.

13. Kim, W., Garza, J., Keskin, A., *Spatial Data Management in Database Systems: Research Directions*, in Proceeding of 3rd int. syposium on Advances in Spatial Databases, D. Abel (ed.), SBerlin: pringer-Verlag, 1993.

14. Lee, B.S., Wiederhold, G., *Outer Joins and Filters for Instantiating Objects from Relational Databases through Views*, IEEE Trans, Know. Data Engineering, Vol. 6, No. 1, pp. 108-119, 1994.

15. Maier, D., *Why isn't There an Object–oriented Data Model?*, Proc IFIP 11th World Computer Congress, 1989.

16. Milne, P., Milton, S., and Smith, J., *Geographical Object–oriented Databases — a Case Study*, Int. Journal of GIS, Vol. 7, pp. 39-56, 1993.

17. Mitrović, A.; Mitrović, D.; Djordjević-Kajan, S.; Rančić, D. *A scalable, object-oriented GIS framework*, accepted for ISPRS Workshop on New Developments in GIS, Milan, Italy, March 1996a.

18. Mitrović, A.; Mitrović, D.; Djordjević-Kajan, S. *GinisNT - a platform for object-oriented development of GIS* In: YUGIS'96, Beograd, Yugoslavia, 1996b (in Serbian).

19. Mitrović D. **An extension of RDBMS for application in GIS**, Master's thesis, Computer Science Department, University of Niš, 1993 (in Serbian).

20. Mitrović, D., Djordjević, S., Stoimenov, L. **An open GIS architecture for inexpensive hardware platforms**, Proc. 10th AM/FM Europian conference, Heidelberg, 1994.

21. Mitrović, D.; Mitrović, A.; Djordjević-Kajan, S.; Rančić, D. *A GIS solution for developing countries*, Proc. AURISA-95, Melbourne, pp. 262-271, 1995.

22. ONTOS Inc., ONTOS Developers Guide, 1991.

23. Peuquet, D.J. *A Conceptual Framework and Comparison of Spatial Data Models*, in Introductory Readings in GIS, D. Peuquet, D.F. Marble (eds), London: Taylor & Francis, 1990.

24. Rumbaugh, J., Blaha, M., Premerlani, W. *OO Modeling and Design*, Prentice Hall, 1991.

25. Samet, H. *The design and analysis of Spatial Data Structures*, Reading, MA: Addison-Wesley, 1990.

26. Shlaer, S., Mellor, S.J. *Object-oriented System Analysis: Modelling the World in Data*, Yourdon Press, 1988.

27. *Spatial Data Transfer Standard (SDTS)*, FIPS PUB 173, US Dept of Commerce, 1992.

28. Stoimenov, L.; Mitrović, A.; Stojanović, D. *Coupling OO GIS applications with relational databases*, In: YUGIS'96, Beograd, Yugoslavia, 1996 (in Serbian).

29. Stojanović, D., Petković, M., Stoimenov, L. *Version control in spatial databases*, In: YUGIS'96, Beograd, Yugoslavia, 1996 (in Serbian).

30. Sulima, J. (1995) *Progress Report on the ISO SQL3 Multimedia Spatial Standard and OenGIS Interoperability Specification*, Proc. AURISA'95, pp. 228-235, 1995.

31. van Oosterom, P. *Reactive Data Structures for Geographic Information Systems*, New York: Oxford University Press, 1993.

32. van Roessel, J.W. *Design of a Spatial Data Structure Using the Relational Normal Forms*, Int. J. GIS, Vol. 1, No. 1, pp. 33-50, 1987.

33. Worboys, M.F., Hearnshaw, H.M. and Maguire, D.J., *Object–oriented Data Modelling for Spatial Databases*, Int. J. GIS, Vol. 4, pp. 369-383, 1990.

34. Worboys, M.F., *Object–oriented Approaches to Geo–referenced Information*, Int. J. GIS, Vol. 8, No. 4, pp. 385-399, 1994.