The Design of GODOT: An Object-Oriented Geographic Information System [*][†]

Oliver Günther
Institut für Wirtschaftsinformatik
Humboldt-Universität zu Berlin
Spandauer Str. 1
10178 Berlin, Germany
guenther@faw.uni-ulm.de

Wolf-Fritz Riekert
FAW Ulm
Postfach 2060
89010 Ulm, Germany
riekert@faw.uni-ulm.de

Abstract

This paper describes the design of an object-oriented geographic information system called GODOT (Geographic Data Management With Object-Oriented Techniques). GODOT has a four-layer architecture, consisting of (i) a commercial object-oriented database system; (ii) an extensible kernel with classes and methods for representing complex spatial and non-spatial data objects; (iii) a collection of base components for query processing, graphics, database administration, and data exchange; and (iv) several user interface modules. The conceptual basis for the system is a data model with three categories of objects: Thematic objects, geometric objects, and graphic objects. The GODOT approach facilitates the management of complex geographic and environmental information and leads to an extensible system architecture.

1 Introduction

Geographic and environmental information systems operate on very complex spatial and non-spatial data structures. Relational databases are of only limited use for modeling this complexity. For that reason, most commercial geographic information systems (GIS) rely on specialized file systems or other proprietary solutions for storing the data.

The idea behind GODOT is to develop a GIS prototype on top of a commercial object-oriented database system (OODB) by adding appropriate classes and methods. With this approach, GODOT differs from most other recent GIS developments:

- GODOT's object-oriented data model allows the representation of highly *complex* geographic information (e.g., in environmental applications) as a network of geographic objects.
- The GODOT data model is *extensible* by user-defined classes and methods.
- GODOT's underlying OODB is a general purpose system which allows for both spatial data and ordinary tabular data to be seamlessly *integrated* within the same environment.

[*]This paper has appeared in the *IEEE Data Engineering Bulletin* 16(3), 1993.

[†]The GODOT project is funded by the Environmental Ministry and the Information and Communication Agency of the State of Baden-Württemberg, Siemens Nixdorf Informationssysteme AG, and Siemens AG.

- GODOT is based on a commercial OODB and therefore participates in *new developments* on the database market. This may concern features such as query language standards, graphical tools, transaction management, and distributed processing.

Earlier related work was mostly based on OODB research prototypes, such as PROBE [3] or O_2 [3]. Several other projects were based on extensible database systems, such as DASDBS [3], or POSTGRES [3]. A more recent effort to use POSTGRES for the management of large amounts of geographical and environmental data is the SEQUOIA 2000 project [3].

Section 2 describes the four-layer architecture of the GODOT system. In Section 3 we present the GODOT data model with its three categories of objects: thematic objects, geometric objects, and graphic objects. Section 4 gives a brief overview of the current state of the implementation.

2 System Architecture

The GODOT system has a four-layer architecture (Fig. 1), consisting of:

1. A commercial OODB
2. An extensible kernel with classes and methods for the representation and management of complex spatial and non-spatial data objects
3. A collection of base components for query processing, graphics, database administration, and data exchange
4. Several user interface modules, including a C/C++ program interface, a UNIX command interface and a graphical user interface based on the X Window System and OSF/Motif

In the sequel we discuss those four layers in turn.

2.1 OODB and Kernel

The GODOT kernel is implemented directly on top of the commercial OODB ObjectStore [3]. ObjectStore is a fully object-oriented database system in the sense of the OODB Manifesto [3]. This covers in particular the basic database functionalities, including transaction management and indexing. Some of these functionalities have been adapted and extended for a more efficient management of spatial data.

The GODOT kernel contains the definition of classes and methods that are crucial for the representation and management of geographic information. In particular, the implementation of the GODOT data model (see Section 3) is located here. Furthermore, the spatial clustering and indexing components will be located in the kernel. We are currently in the process of evaluating a number of index structures for this purpose. As all the other GODOT components, the kernel is implemented in the object-oriented programming language C++.

The GODOT data model can be extended by additional classes and methods. This facilitates the customization of the system to serve any particular application.

2.2 Base Components

The *query component* contains several GIS-specific language elements to enhance the query language of ObjectStore. It interacts directly with the interface modules described in Section 2.3. ObjectStore allows the call of a user-defined method within a query; this feature has been used to extend the query language with spatial and topological predicates.

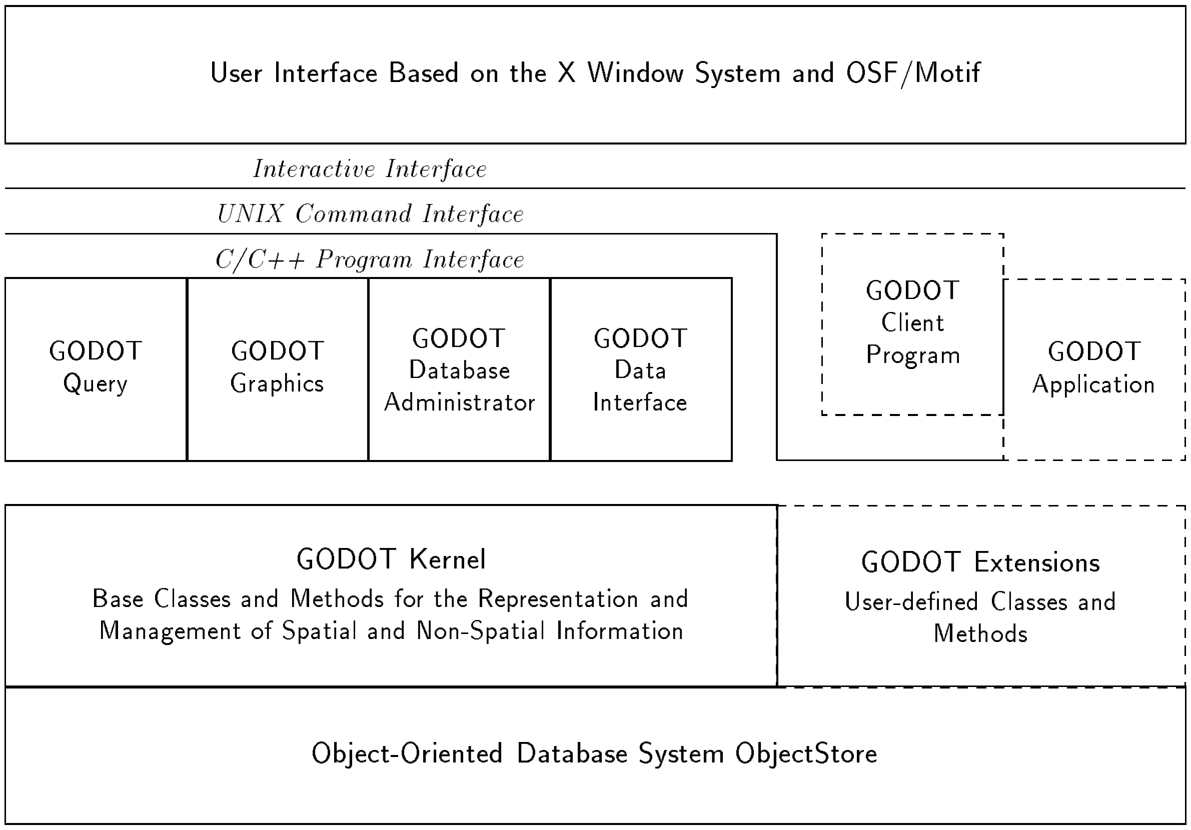
User Interface Based on the X Window System and OSF/Motif
Interactive Interface
UNIX Command Interface
C/C++ Program Interface
GODOT Query
GODOT Graphics
GODOT Database Administrator
GODOT Data Interface
GODOT Client Program
GODOT Application
GODOT Kernel
Base Classes and Methods for the Representation and Management of Spatial and Non-Spatial Information
GODOT Extensions
User-defined Classes and Methods
Object-Oriented Database System ObjectStore

Figure 1: GODOT Architecture

The query component is tightly linked to a *graphics component* that manages the graphical representation of the geographic information. In particular, it is common to use the graphics component to visualize the result of user queries. On the other hand, one can use the graphics component to specify parts of a query by pointing to certain objects on the screen. This includes the ability to update geographic information by manipulating the corresponding graphic objects interactively.

The *database administrator component* provides the usual features for database schema manipulation, user administration, and system support. Once again, we can use some of ObjectStore's functionalities directly, while others need to be adapted to manage spatial data efficiently.

A major issue in GIS is the exchange of geometric data encoded in different formats. GODOT supports the integration of *data interfaces* as base components; these can be activated through any of the interface modules described in the following section. GODOT also has its own external data format, which is a subset of C++, encoded in ASCII. This external data format can be read easily by other systems. The execution of the code leads directly to the generation of the corresponding object classes and instances in the given database.

2.3 User Interfaces

One of GODOT's core functionalities is to be a GIS data server for a diverse and distributed collection of applications. For this purpose, GODOT offers a variety of client-server style interfaces. An *interactive interface* under the X Window System gives high-level graphical access to GODOT, especially for the occasional or non-expert user. A different kind of access is provided by the *UNIX command interface*, where GODOT queries can be formulated by means of specialized commands that form an extension of the UNIX shell. Command procedures can then be implemented as shell scripts. Finally, a *C/C++ program interface* makes the GODOT modules available as a program library. This interface is typically used for more complex GODOT applications; it also allows remote access via remote procedure calls. For the implementation of the C/C++ program interface, the recommendations of the Object Management Group with regard to the Object Request Broker [3] will be taken into account.

3 Data Model

The GODOT data model is partitioned into three categories of objects:

1. *Thematic objects* are used to represent real-world objects. An important subcategory of thematic objects are the *geographic objects* or *geo-objects*. A thematic object is a geo-object if it is geometric in nature, i.e., if it has a spatial extension.

2. *Geometric objects* or *geometries* are used to describe the geometric features of geo-objects.

3. *Graphic objects* are used for the display of thematic objects. *Cartographic objects* are an important subcategory of graphic objects.

The various relationships between these categories are shown in Figure 2, using a simple example. Note that we use strings of type X<Y> as object identifiers, where X denotes the class of the object. There are three thematic objects in that example: Two geo-objects City<Ulm> and City<Neu-Ulm> to represent the twin cities Ulm and Neu-Ulm, and a simple thematic object CoordCommittee<31> to represent the coordination committee of the two cities. The two cities, respectively their corresponding geo-objects, are connected to geometric objects to represent their shapes and to several graphic objects for their cartographic representation.

In the sequel, the three object categories are discussed in turn.

3.1 Thematic Objects and Geo-Objects

In the GODOT data model, geographic and environmental information is represented by so-called *thematic objects*. Thematic objects may be simple or complex, i.e., composed of several other thematic objects. Examples of thematic objects include the coordination committee and the cities in Figure 2, or a species in a natural resource information system.

Thematic objects may have different kinds of attributes to represent a variety of geographic and environmental features:

1. Attributes of an elementary type (e.g., text strings or real numbers)

2. Attributes of a complex type (e.g., embedded classes in C++)

3. Attributes of a referential type (e.g., pointers to other thematic objects)

The most important subset of the thematic objects is formed by the *geo-objects*, which are characterized by an attribute that is a geometric object. A geo-object therefore has a location and a spatial extension. If a geo-object is complex, a number of integrity constraints need to be enforced. For example, the geometry of a complex geo-object has to be the union of its component geometries.

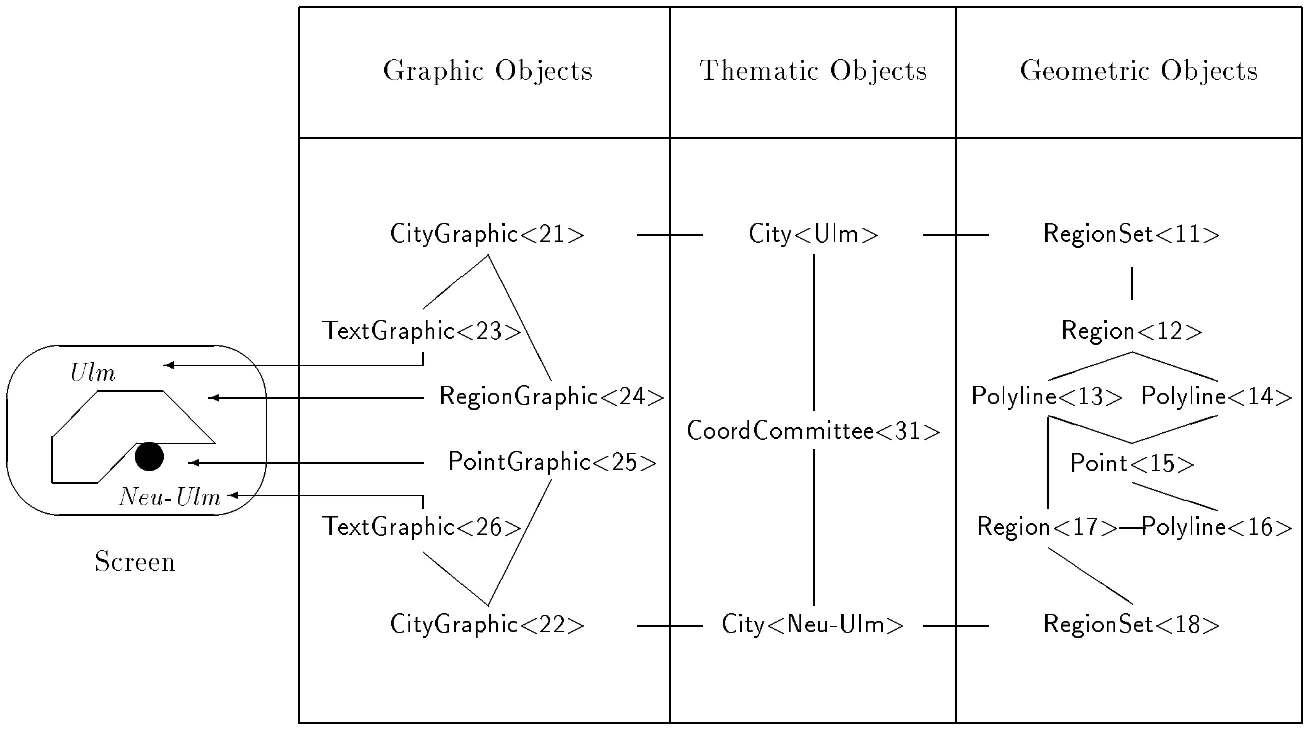

Figure 2: The three categories of the GODOT data model

3.2 Geometric Objects

Associated with each geo-object is a *geometric object*, which is typically composed of elementary geometric objects. These elementary geometric objects form the class *Geometry* with its three subclasses *Region*, *Arc*, and *Point*. An arc may be curved although the current implementation is restricted to (piecewise linear) polylines. Between these classes there exist the usual geometric relationships: A region has several bounding arcs. An arc may in turn belong to any number (including zero) of regions. Similarly, an arc has two endpoints, and any point may be the endpoint of any number of arcs.

Note that a geometric object can either be a singular point, arc, or region, or a collection of such singulars. If these singulars all belong to the class *Region* then the resulting complex object belongs to the class *RegionSet*. The classes *ArcSet* and *PointSet* are defined analogously. However, if the collection of singulars is heterogenous in the sense that it contains singulars of different types, it belongs to the class *GeometrySet*, which is the superclass of *RegionSet*, *ArcSet*, and *PointSet*.

This design guarantees that the result of a boolean set operation on two geometric objects can always be represented by exactly one geometric object. It also enables us to represent the geometries of any geo-object, however oddly shaped, with just one geometric object. Consider, for example, a river whose width varies widely, such that its geometry in the chosen accuracy is partly arc, partly region. In GODOT, the geometry of this river would be modeled by an instance of the class *GeometrySet*. Another example is a country whose area consists of several disconnected regions (e.g., the USA). This area would be represented by an instance of *RegionSet*.

3.3 Graphic Objects

Graphic objects are used for the (interactive or printed) display and the interactive update of thematic objects, in particular of geo-objects. The looks of a graphic object are defined in detail by its attributes, such as color, line width, or text font. Possible graphic representations include business graphics, tables, videos, raster images, or GIS-typical vector graphics. A thematic object can be linked to several graphic representations (e.g., for multiple scale display).

An important subcategory of graphic objects is formed by the *cartographic objects*, which are used for the graphic display of geo-objects. With this design, GODOT implies a clear separation between geographic and cartographic information. Cartographic objects contain methods that determine how the properties of a given geo-object are represented in terms of the graphics available. In particular, questions of scale and cartographic generalization are handled at this level and *not* at the level of thematic objects.

4 Outlook

In this short paper we sketched the basic architecture of the GODOT object-oriented GIS. We have finished the implementation of our data model on top of ObjectStore, and we are currently working on the graphic user interface. The first complete prototype should be functional no later than the end of 1993.

References

[1] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik, The Object-Oriented Database System Manifesto, in *Proc. First Int. Conf. on Deductive and Object-Oriented Databases*, Kyoto, 1989.

[2] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb, The ObjectStore Database System, *Communications of the ACM*, 34(10):50–63, October 1991.

[3] J. Orenstein and F. Manola. PROBE Spatial Data Modeling and Query Processing in an Image Database Application, *IEEE Transactions on Software Engineering*, 14(5):611–629, May 1988.

[4] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, OMG, Framingham, Mass., 1992.

[5] P. Oosterom and T. Vijlbrief, Building a GIS on Top of the Open DBMS POSTGRES, in *Proc. EGIS'91*, Brussels, April 1991.

[6] M. Scholl and A. Voisard, Object-Oriented Database Systems for Geographic Applications: An Experiment With O_2, in: F. Bancilhon, C. Delobel and P. Kanellakis (Eds.), *The O_2 Book*, Morgan Kaufmann, San Mateo, Calif., 1992.

[7] H.-J. Schek and A. Wolf, From Extensible Databases to Interoperability Between Multiple Databases and GIS Applications, in: D. Abel and B. C. Ooi (Eds.), *Advances in Spatial Databases*, Springer, Berlin, 1993.

[8] M. Stonebraker, The Sequoia 2000 Project, in: D. Abel and B. C. Ooi (Eds.), *Advances in Spatial Databases*, Springer, Berlin, 1993.