*Object-Oriented Modeling for GIS*
*Max J. Egenhofer and Andrew U. Frank*
*URISA Journal 4 (2): 3-19, 1992*

# Object-Oriented Modeling for GIS

Max J. Egenhofer
National Center for Geographic Information and Analysis
Department of Surveying Engineering
University of Maine
Orono, ME 04469, USA

and

Andrew U. Frank
Department of Geo-Information
Technical University, Vienna, Austria

## Abstract

The data model upon which most of today's commercial database management systems are based has shown to be insufficient for geographic information systems (GISs). The recently promoted object-oriented model provides some useful tools for data abstraction and data structuring, which augment the conventional tools and overcomes some deficiencies inherent to the traditional relational model. In particular, the concepts of *complex objects* with *pertinent operations* are more powerful modeling methods than the currently popular structure of relational tables and relational algebra. This survey article presents the concepts of object-oriented modeling applied to geographic data and demonstrates their impact on future GISs.

## 1. Introduction

Database management systems (DBMSs) play a central role in a geographic information system (GIS), because they liberate the GIS designers from building and maintaining a substantial and complex part of a large software system (Frank 1988a). DBMSs have become an accepted tool for storing data in a form that is concurrently accessible for multiple users, prevents loss of data, and provides security for access (Codd 1982). By using a database management system, all the tasks of low-level data management are removed from the programmer's and end-user's responsibility. From a programmer's perspective, data are described by their logical properties, not the physical structure in which they are stored. For instance, a database user may retrieve the object "town of Orono" by providing the name "Orono" as a property of an object of the class "town;" they need no knowledge about the location of the particular record in the file or details about the access method. The DBMS as a subsystem of a GIS can be replaced by another product of the same modeling power, provided the interfaces are compatible.

Computer scientists have studied the design and implementation of DBMSs for many years and several commercial systems are currently available. They are generally based on one of the classical data models—hierarchical (Tsichritzis and Lochovsky 1977), network (CODASYL 1971), or relations (Codd 1982)—or their derivatives. Within the last few years, commercial relational database management systems have gained much popularity, and products such as Oracle and Ingres are widely used. They are built on the relational model, which organizes data as tables or relations (Codd 1970). Columns of the tables are called attributes and all values in an attribute are elements of a common domain that describes the set of all possible values. Rows are referred to as records, tuples, or relation elements (Ullman 1982).

While relational database management systems are suitable and successful for applications dealing with weakly structured data, such as banking accounts and personal files, they fail when they are used for applications of data with a complex structure. GISs, which integrate data from

various resources into a single, homogeneous system, need powerful and flexible data models to serve multiple tasks. These include

- sophisticated treatment of real-world geometry (Frank and Kuhn 1986; Greene and Yao 1986; Herring 1987; Egenhofer *et al*. 1989; Milenkovic 1989);
- representation of the same data at different conceptual levels of resolution and detail (Bruegger 1989; van Oosterom 1990; Buttenfield and McMaster 1991);
- management of history and versions of objects (Sernadas 1980; Snodgras 1987; Langran 1989; Al-Taha and Barrera 1990); and
- combinations of measurements of different resolutions and accuracy (Davis 1986; Buyong *et al*. 1991).

The GIS literature (Morehouse 1990; Frank and Mark 1991) often differentiates between systems that treat objects with distinct identities and raster systems, which store the distribution of properties in space (Tomlin 1990). We restrict our discussion here to GISs that handle objects with a distinct identity.

Since the relational data model does not match the natural concepts humans have about spatial data, users must artificially transform their mental models into a restrictive set of non-spatial concepts. For example, it imposes too many restrictions, such as normalization rules (Codd 1972), os that spatial objects must be artificially decomposed into smaller parts (Nyerges 1980; Frank 1988a). This implies a performance degradation of the databases that are based on this model, proportional to the size of the data collection, and impedes the use of traditional database management systems for geographic applications. This becomes particularly visible during interactive graphics sessions when large amounts of spatial and non-spatial data must be accessed to produce an understandable rendering (Egenhofer 1990) and users have to wait unreasonably long for responses to their queries (Frank 1981). Though these deficiencies reveal performance problems, they are clearly based on conceptual issues rather than on hardware limitations. Problems that can be solved by exploiting additional or faster hardware will not be addressed here, since the requirements for these spatial database management systems are not hardware issues (Frank *et al*. 1991). For example, even with much faster access to storage devices, spatial indexing structures and large object buffers will be needed to provide adequate performance for queries for which the results are to be presented as interactive drawings (Smith and Frank 1990). Faster and less expensive CPUs, larger hard disks, and more computer memory are available, but they do not overcome the data structuring limitations in database technology applied to problems in spatial data handling. It is assumed that hardware technology will continue its rapid growth over the next decade, but substantial performance improvements of hard disks are not expected. Disk I/I is recognized as the major performance bottleneck in database management systems, and improvements by clustering and shadowing several disks are only marginal remedies (Stonebraker *et al*. 1988).

Several branches in computer science (artificial intelligence, software engineering, database management systems, human-computer interaction) have recently promoted an *object-oriented* approach:

- Object-oriented data models have been developed to capture more semantics than the relational model (Brodie *et al.* 1984; Peckham and Maryanski 1988).
- Object-oriented user interfaces make systems appear more natural and easier to use (Schmucker 1986).
- Object-oriented database management systems have been investigated to provide the corresponding features for storage and retrieval of complex objects (Zdonik and Maier 1990).

- Object-oriented software engineering techniques and programming languages have been developed to support the implementation of software systems that were designed following an object-oriented approach. They allow for immediate implementations of object concepts rather than simulating them with traditional programming languages (Stroustrup 1986; Meyer 1988).

The goal of this paper is to familiarize the reader with the principles of object-oriented modeling for spatial data handling. The focus will be on conceptual achievements that will help to improve the modeling power of spatial information systems so that the often complex spatial phenomena may be expressed in terms closer to humans' thinking. Attaching the adjective *object-oriented* to a system just because it was implemented in an object-oriented programming language is misleading. Such a detail should not be visible to the user and is thus irrelevant (Herring 1991). Unlike a paper with a similar title (Worboys *et al*. 1990), which stressed the *structural* aspects of a very specific object-oriented model (Abiteboul and Hull 1984), this paper attempts to present a synthesis of concepts common to different object-oriented models, and focuses on *behavioral* aspects.

This paper proceeds as follows: First, an overview of the need for formal models in GIS is given. Then an object-oriented model is introduced that builds upon the four major abstraction methods of classification, generalization, association, and aggregation. The behavior of objects is explained in terms of the concepts of inheritance and propagation to model dependencies among objects in generalization and aggregation hierarchies. Finally, the object-oriented data model is compared with the traditional relational data model.

## 2. Modeling for GIS

Geographic information systems serve as repositories of observations humans make about spatially related objects and their properties. In order to concentrate on the issues, necessary to solve a given task, humans build mental models of real objects and further simply reality using abstraction mechanisms until only the necessary components remain. Such mental models are either informally communicated by using natural language, or in a system based on a formal abstract model of reality.

Experientialists have observed that the base concepts humans use are established early in their lives as abstractions from their bodily experiences (Johnson 1987). This experience is the same for all humans, as they depend primarily on the physiology of the human body, and establishes a foundation on which to base communication and to avoid subjectivism. The *meta model* is the generic description of a situation (Ellis *et al*. 1990) (e.g., "a person owns a building" as opposed to the specific model "Doe Smith owns the building at 56 Park Street"). The abstraction mechanisms available in the data model determine the meta models and specific models which can be used, and thus the expressive power of a GIS (Figure 1). If the abstraction mechanisms are insufficient, the meta model of reality is inadequate and the mapping from the user concept of the object behavior onto the GIS model will be strenuous and difficult to understand. This would make a GIS difficult to use.

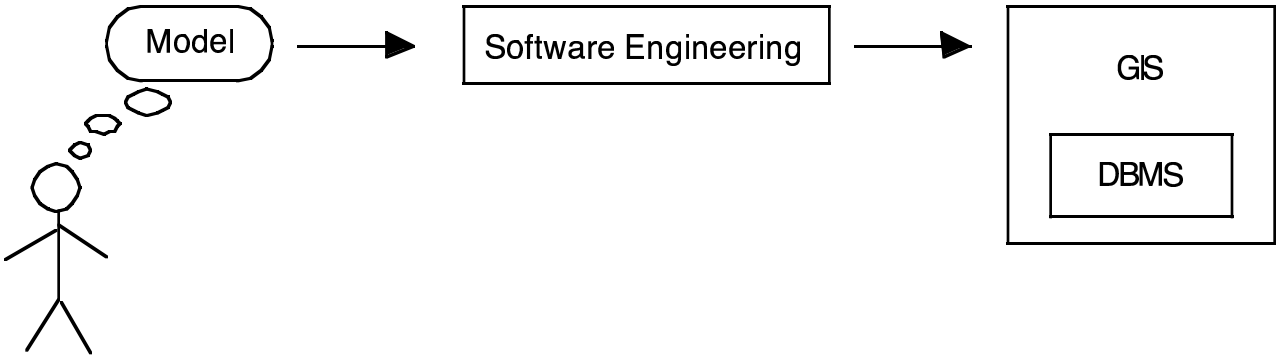


Figure 1: Model, Software Engineering, and DBMS for GIS.

## 2.2 Data Models and Abstraction Mechanism

An important part of the model-building facilities is contributed by the data model used for the database management system in the GIS. A data model is a collection of (1) data structure types, (2) operators or inference rules, and (3) general integrity constraints (Codd 1981). It provides the tools, i.e., the language available, to describe the meta model of a database (Date 1986). Examples of data models are the relational model (Codd 1970), the Entity-Relationship-Model (Chen 1976), first-order predicate calculus (Gallaire *et al.* 1984), and object-oriented models (Manola and Dayal 1986; Bancilhon *et al.* 1988).

The data model of a database management system must provide for abstraction mechanisms that are also carried forward into the programming language and the software engineering environment; therefore, software engineering must provide the users with tools to apply the abstraction mechanisms immediately in a programming language. For example, if a model offers a mechanism to establish objects that can be composed of other objects, then the programming language used for the implementation must offer a construct to get all components that are part of another component. The lack of appropriate constructs in programming languages often leads to simulations that make software systems complex and difficult to maintain.

It is standard practice that an information system is designed using an Entity-Relationship-Model (Chen 1976), then implemented in an Algol-like programming language, e.g., Pascal or C, which was extended to access a relational database management system through an embedded query language, such as Embedded SQL or Embedded Quel. Incompatibilities occur at each interface between two sets of tools due to the different models for information representation. This confusion of various data models results in what has been called *impedance mismatch*. Powerful features of one component must frequently be simulated in another. These simulations reduce efficiency and lead to discrepancies between the design and the implementation. The resulting product, based upon an incoherent design, consumes undue resources and is difficult to maintain.

For example, the calculation of the area of a 100-foot buffer along Route 1A has to be broken down into the following steps:

- Retrieving the line segments of Route 1A from the relational database with an SQL query into a “cursor”:

```
DECLARE    road CURSOR FOR
SELECT     start.x, start.y, end.x, end.y
FROM road, edge, node start, node end
WHERE      road.name = "1A" and
           road.lines = edge.id and
           edge.start = start.id and
           edge.end = end.id;
```

- Assigning the result of the SQL query to variables in a programming language, tuple by tuple, and building a data structure for the nodes (e.g., a linked list):

```
// create an empty list "edgeL" ;
OPEN road ;
WHILE (SQLCODE == 0)
{
FETCH road INTO :edge.start.x, :edge.start.y, :edge.end.x,
:edge.end.y ;
// add the variable edge to edgeL ;
};
CLOSE road ;
```

- Processing the linked list in a function "buffer zone" that was previously implemented in a standard programming language such as C:

```
{
roadBuffer = buffer (edgeL, 100);
}
```

Experience shows that these incompatibilities cannot be completely hidden at internal and user interfaces, making the programmers' tasks and, ultimately, the users' tasks, more difficult. This is an important consideration, because teaching and training costs for new systems are very considerable parts of the overall costs of introducing a new information system.

### 2.2 Object Orientation

The basic idea of object orientation is the observation that the world is often perceived as consisting of *objects*, which interact in specific ways. The interaction among the objects can be seen as a command, or message, given to an object—either verbally or by some action such as physical force. Based upon common operations that can be applied to the objects—or commands to which they respond—they are grouped into classes. This concept was originally introduced into programming as part of the simulation language SIMULA (Dahl and Nygaard 1966), and later found to be generally valid and applicable.

In software engineering, object orientation has become a design method that focuses as a first line of structuring on modeling objects as humans perceive them in reality. Unlike previously used approaches, it combines modeling of the *structure* and the *behavior* of the objects. Procedural abstractions model primarily the operations, while methods used for designing database schemas concentrate on the structure of the entities.

The object-oriented method corresponds closely to the mathematical concept of *multi-sorted* or *heterogeneous algebras* (Birkhoff and Lipson 1970). From this point of view, the description of an object consists of a name for its type, a set of operations that are applicable to objects of this type, and a set of axioms that define the behavior of the operations. The important idea is that it is possible to write the axioms in terms of the operations, i.e., one defines the behavior of an operation in terms of other operations.

## 3. Object-Oriented Abstraction Mechanisms

This section introduces the notions of objects and the abstraction tools available to deal with them, following Dittrich's (1986) synthesis. A definition of object orientation is that

- any entity, independent of whatever complexity and structure, may be represented by exactly one object.

No artificial decomposition into simpler parts due to technical restrictions should be necessary. This is referred to as *structural object orientation*. Complex data types *per se*, modeling large objects such as entire cities (with all details about streets, buildings, etc.), do not overcome the problem of data structuring, and only the combination of complex object types and operations upon such instances provides the necessary view of objects. This second component of object orientation is called *operational object orientation* and requires that

- operations on complex objects are possible without having to decompose the objects into a number of simple objects.

The third notion of *behavioral object orientation* states that

- a system must allow its objects to be accessed and modified only through a set of operations specific to an object type.

The object-oriented data model is built on the four basic concepts of abstraction (Brodie *et al.* 1984): classification, generalization, association, and aggregation.

## 3.1 Classification

Classification is the mapping of several objects (instances) onto a common class. The word *object* is used for a single occurrence (instantiation) of data describing something that has some individuality and some observable behavior. The terms *object type*, *sort*, *type*, *abstract data type*, or *module* refer to types of objects, depending on the context. In the object-oriented approach, for every object, there exists at least one corresponding class, i.e., every object is an instance of a class; therefore, classification is often referred to as the *instance_of* relationship.

A type characterizes the behavior of its instances by describing the common *operators* that can manipulate those objects (O'Brien *et al*. 1986). These operations are the only means to manipulate objects. All objects that belong to the same class are described by the same properties and have the same operations. For example, the model for a Town may include the classes RESIDENCE, COMMERCIALBUILDING, STREET, and LANDPARCEL. A single instance, such as the building with the address "26 Grove Street," is an object of the corresponding object type, that is, the particular object is an instance of the class RESIDENCE. Operations and properties are assigned to object types; for instance, the class RESIDENCE may have the property NumberOfBedrooms, which is specific for all residences. Likewise, the class STREET may have an operation to determine all adjacent Parcels (Figure 2).


Figure 2: Graphical representation for the two *classes* RESIDENCE and STREET.

Differences between objects of the same class are based upon their property values. Property values describe the individual characteristic of each object. For example, two LANDPARCELS may be distinguished by their addresses, different values of their areas, or specific LandUseTypes.

## 3.2 Generalization

*Generalization*—not to be confused with the same term used in cartography—groups several classes of objects with common operations into a more generalsuperclass (Dahl and Nygaard 1966; Smith and Smith 1977b; Goldberg and Robson 1983). The term *superclass* characterizes this grouping and refers to object types that are related by an *is-a* relation. The converse relation of superclass, the *subclass*, describes a specialization of the superclass. Frequently, the terms *parent* and *child* are also used for superclass and subclass, respectively. Though this terminology is helpful to clarify the dependency of subclasses from superclasses, it is not accurate with respect to the abstraction, because the relationship between parent and child is not *is-a*. Subclass and superclass are abstractions for the same object and do not describe two different objects. For example, each RESIDENCE is a BUILDING; RESIDENCE is a subclass of BUILDING, while BUILDING

is its superclass (Figure 3). The residence with the address "26 Grove Street," for example, is simultaneously an instance of the classes RESIDENCE and its superclass BUILDING.

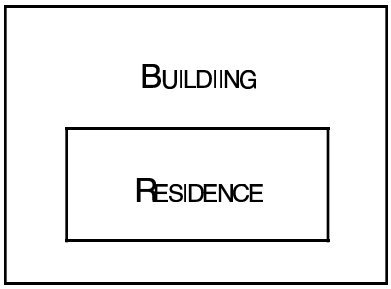


Figure 3: A generalization hierarchy with the more general class (Building) depicted around the more specialized class (Residence).

Two properties of generalization should be mentioned in more detail:

- A superclass may encompass multiple subclasses. For example, besides Residences, there may be other building types such as HOSPITALS and COMMERCIALBUILDINGS (Figure 4).

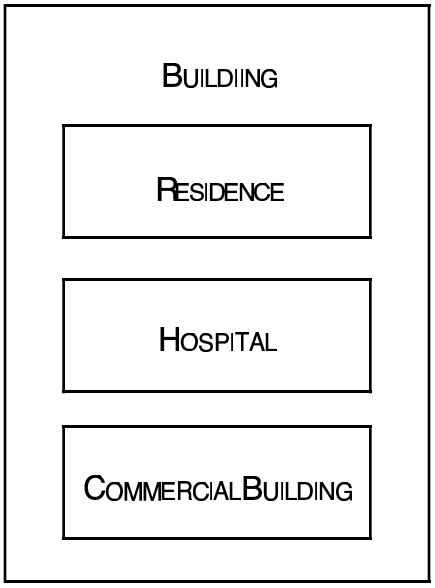


Figure 4: A generalization hierarchy with multiple subclasses of a superclass.

- Generalization may have an arbitrary number of levels in which a subclass has the role of a superclass for another, more specific class. For example, the specialization from Buildings to Residences can be extended with the classes RURALRESIDENCE and URBANRESIDENCE, both being subclasses of RESIDENCE. While RESIDENCE is a subclass of BUILDING, it is at the same time a superclass for RURALRESIDENCE and URBANRESIDENCE (Figure 5).

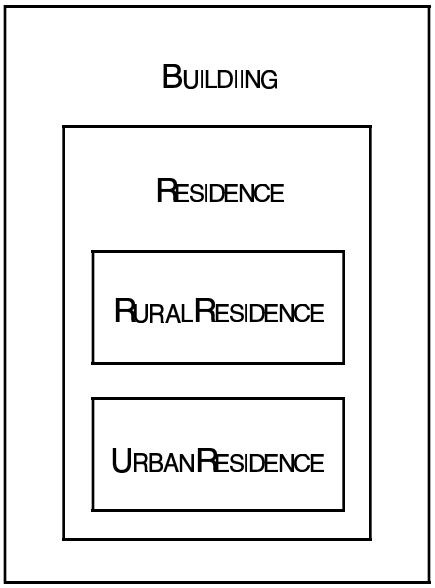


Figure 5: Transitivity of generalization.

### 3.3 Association

An *association* relates two or more independent objects and the relationship between objects is considered a higher level set object (Brodie 1984). The term *set* is used to describe the association, and the associated objects are called *members*. Hence, this abstraction mechanism is referred to as the *member-of* relation, but is also often called *grouping* or *partitioning*. An example of an association in the GIS domain is neighborhood, which relates a land parcel with its adjacent house lots.

The details of a member object are suppressed and properties of the set object are emphasized. An instance of a set object can be decomposed into a set of instances of the member object. Association applied to objects (members) produces a set data structure. Operations over sets are normally operations that are repeated for each member of the set. The implementation of the repetition may be a FOR EACH loop structure, as found in some modern programming languages, e.g., CLU (Liskov *et al.* 1981).

### 3.4 Aggregation

An abstraction mechanism similar to association is *aggregation*, which models composed objects, i.e., objects that consist of other objects (Smith and Smith 1977a). Several objects can be combined to form a semantically higher-level object, called *aggregate* or *composite object*, where each part has its own functionality. The terms *subpart* or *component* refer to the parts of the composite object. Operations of aggregates are not compatible with operations on parts, and vice-versa. When considering the aggregate, details of the constituent objects are suppressed. Every instance of an aggregate object can be decomposed into the instances of the corresponding component objects.

The relation established by aggregation is often called the *part-of* relation since aggregated instances are parts of the aggregate; the relationship converse to *part-of* is called *consists-of*. For example, a CITY may be modeled as an aggregate of all HOUSELOTS, STREETS, and PARKS—they are *part-of* a CITY or, conversely, a City *consists-of* them (Figure 6).

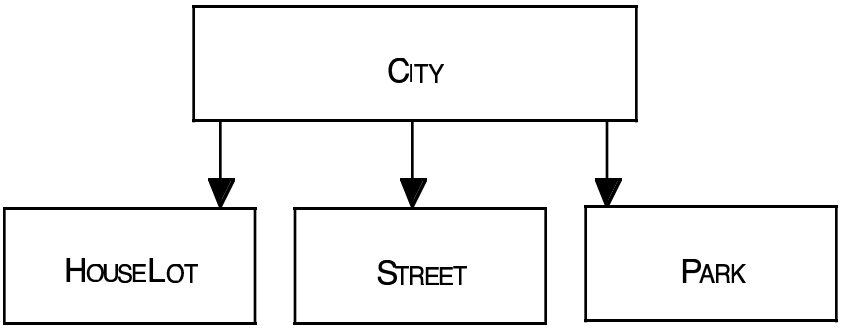


Figure 6: A CITY modeled as the *aggregate* of HOUSELOTS, STREETS, and PARKS.

Aggregation applied to objects (components) produces an aggregate (or record) type data structure. An operation over an aggregate consists of a fixed number of different operations in sequence or in parallel, one for each component. Hence, aggregation relates to sequence or parallel control structures.

## 4. Formalizations of Behavior

The literature concerned with object orientation has evolved considerably over the last few years and the initial proposals for formalizations of complexly structured objects (Batory and Buchmann 1984) have progressed to detailed descriptions of the tools to model the behavior of objects. Since there is still a lack of consensus among different researchers in the field of object-oriented modeling about some details of these concepts, this section will attempt to synthesize the mainstream ideas of enhancing the semantic model with the crucial concepts of inheritance and propagation. The first form describes the derivation of properties in generalization hierarchies, while the second one deals with values in aggregation hierarchies. Sometimes propagation is also called upward inheritance (Barrera and Buchmann 1981; Brodie and Ridjanovic 1984), but it should become clear from this paper that these are two different concepts which must be separated.

### 4.1 Inheritance

In a generalization hierarchy, the properties and methods of the subclasses depend upon the structure and properties of the superclass or superclasses. Inheritance is a method to define a class in terms of one or more other, more general classes (Dahl and Nygaard 1966). Properties that are common to a class and its subclasses are defined only once (with the superclass), and inherited to all objects of the subclass. Subclasses may have additional, specific properties and operations that are not shared with the superclass, but they have strictly all operations and properties of the superclass. Operations of the superclass are compatible among objects of the superclass and all its subclasses. Every operation on an object of a superclass can be carried out on the subclass as well; however, operations specifically defined for the subclass are not compatible with superclass objects.

The implementations of an operation common to several classes may be different for each class, but must have the same properties in terms of the axioms of the superclass. For example, two-dimensional and three-dimensional coordinates are both representations for POINTS. From this superclass POINT both subclasses, 2D-POINT and 3D-POINT, inherit operations such as calculating the distance and direction between two points. The implementation of these operations is different; however, from the outside they behave the same in terms of the axioms of distance and direction.

In terms of the algebra, a superclass is the intersection of all the axioms of all its subclasses, or, the reverse, a subclass contains all the axioms of the superclass plus some additional ones. In order to maintain such simplicity, one has to exclude that a subclass may inherit only parts of the operations prescribed by the superclass. Otherwise, complex exception rules apply (Borgida 1988).

Inheritance is transitive, i.e., the properties are passed along from a superclass to all related subclasses, and to their subclasses, etc. This concept is very powerful, because it reduces information redundancy and maintains integrity (Woelk and Kim 1987). Modularity and consistency are supported since essential properties of an object are defined only once and inherited in all relationships in which it takes part.

### 4.1.1 Single Inheritance

Inheritance can be strictly hierarchical; it is then often referred to as *single inheritance*. Single inheritance requires that any class has at most one single immediate superclass. This restriction implies that each subclass belongs only to a single hierarchy group and that one class cannot be part of several distinct hierarchies.

The following example shows inheritance along a generalization hierarchy (Figure 7). RESIDENCE is the general superclass and URBANRESIDENCE and RURALRESIDENCE are the specific subclasses. All properties and operations of the class RESIDENCE are inherited to its two subclasses. For example, resident and movingIn are associated with the class RESIDENCE and inherited to all URBANRESIDENCES and RURALRESIDENCES. They are compatible with URBANRESIDENCES and RURALRESIDENCES. On the other hand, the operations defined specifically for a subclass are not applicable to objects of the superclasses. For instance, nextSubwayStop is a property that applies only to that.

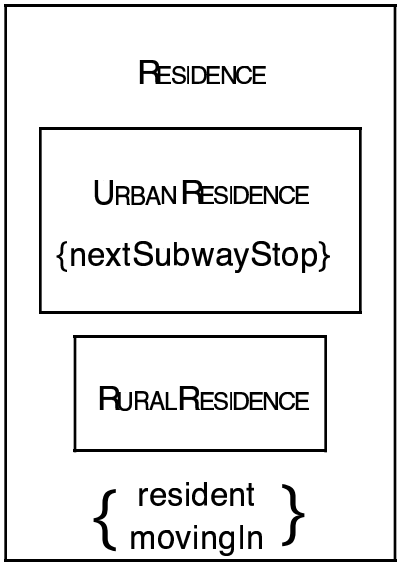


Figure 7: Inheritance of the property resident along the generalization hierarchy.

The transitive property of inheritance implies that any property is passed not only from the superclass to the immediate subclasses, but also to their sub-subclasses, etc. For example, the properties of a BUILDING, such as address and owner, are inherited to the subclass Residence, and also transitively to the sub-subclasses RURALRESIDENCE and URBANRESIDENCE (Figure 8).

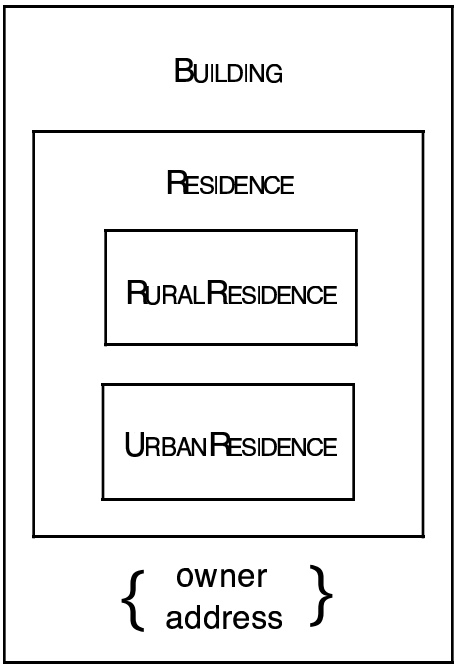


Figure 8: Transitively inheriting the properties address and owner to all subclasses of Building.

### 4.1.2 Multiple Inheritance

The structure of a strict hierarchy is an idealized model and often fails when applied to real world data. Most "hierarchies" have a few non-hierarchical exceptions in which one subclass has more than a single, direct superclass. Thus, pure hierarchies are not always the adequate structure for inheritance. Instead, the concept of *multiple inheritance* (Cardelli 1984) permits one to pass properties from several higher-level classes to another class. This structure is not hierarchical, because—in terms of the parent-child relation—one child can have several parents. In the simplest case of multiple inheritance, a subclass inherits properties from two distinct superclasses. For example, the different roles of a LANDPARCEL as a TAXABLEITEM and a REALESTATEOBJECT can be effectively modeled by multiple inheritance (Figure 9).

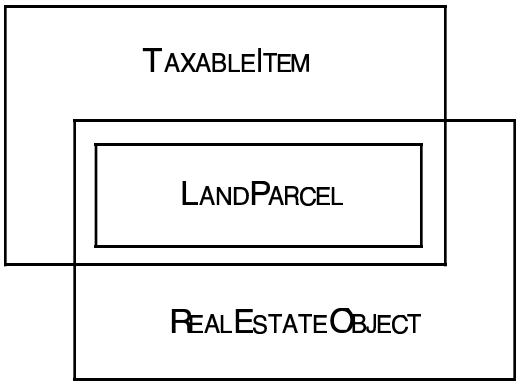


Figure 9: A LANDPARCEL modeled with multiple inheritance as a TAXABLEITEM and a REALESTATEOBJECT.

A more complex GIS example shows how multiple inheritance combines two distinct hierarchies (Figure 10). The first hierarchy is determined by the separation of TransportationLinks

into ARTIFICIALLINKS and NATURALLINKS. HIGHWAYS and CHANNELS are ARTIFICIALTRANSPORTATIONWAYS, and NAVIGABLERIVERS are NATURALTRANSPORTATIONLINKS. WATERBODIES with PONDS, CHANNELS, and RIVERS form a second hierarchy, in which two types of rivers are distinguished: NAVIGABLERIVERS and UNNAVIGABLERIVERS. Classes with properties from both hierarchies are CHANNELS that are ARTIFICIALTRANSPORTATIONLINKS and WATERBODIES, and NAVIGABLERIVERS that are RIVERS and NATURALTRANSPORTATIONLINKS. These hierarchies cannot be compared with each other, because a WATERBODY is not necessarily a TRANSPORTATIONLINK and, vice-versa, not every TRANSPORTATIONLINK is a WATERBODY either; however, the hierarchies share common subclasses, because Channels are both WATERBODIES and ARTIFICIALTRANSPORTATION links, and NAVIGABLERIVERS are RIVERS and NATURALTRANSPORTATIONLINKS. Other classes, such as HIGHWAY or POND, belong only to a single inheritance hierarchy in this schema.

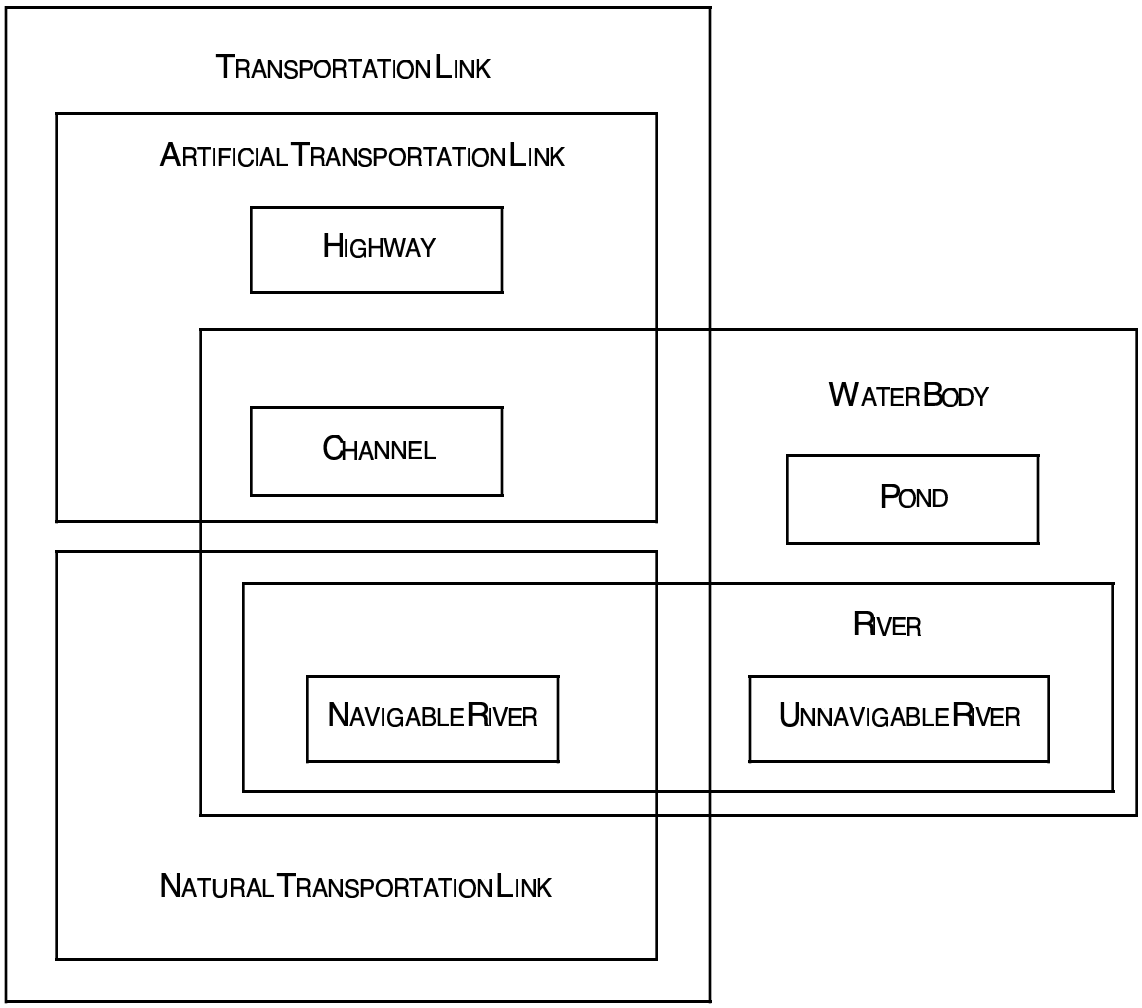


Figure 10: A GIS example of the use of multiple inheritance.

The problem of *name clashes* or *inheritance conflicts* has received much attention. If a class has several superclasses, it may inherit distinct operations with the same name, but different meanings. For instance, a LANDPARCEL has a value as a REALESTATEOBJECT and as a TAXABLEITEM. Both values are based on different assessments and used for different purposes. Single inheritance has a simple rule to resolve such name conflicts; it gives preference to the most specific method (i.e., the one associated with the most detailed superclass). This selection may not necessarily be what was intended with the model, but it is at least a simple and consistent rule. For multiple inheritance,

there are no such simple conceptual rules that could capture the intended meaning. Frequently, the conflict is resolved by giving preference to the methods in the order they are listed in the data definition (Stonebraker and Rowe 1986); however, this would not be a valid solution for the value of the LANDPARCEL. Since the two names actually describe two different properties, it is necessary to distinguish between them by tagging the property names with their class names, e.g., REALESTATEOBJECT.value and TAXABLEITEM.value.

### 4.1.3 Inheritance for GIS Modeling

Usually, a GIS contains many application-specific classes of objects such as CITIES, RIVERS, ROADS, BUILDINGS, HOUSEOWNERS, PARCELS, SOILS, and their detailed subclasses. A number of operations is associated with each class. For example, owners may sell their land, a new road may be constructed, or a building may be demolished. Some of these operations are similar, e.g., the operation to determine all CITIES that lie within a COUNTY and the operation determining all BUILDINGS within a CITY can be both interpreted as the geometric operation inside (Egenhofer and Herring 1990). Identifying and describing formally similar objects with common operations forms one of the goals of conceptual modeling, which allows the users' models of their mini-worlds to be implemented with the least redundancy.

One part of an application model is the definition of a set of classes as the abstraction of objects with common properties. For each class the appropriate operations and relationships must be defined, including operations that combine objects of different classes. For example, the class BUILDING has the operation onParcel, which checks whether a building is located inside a parcel. Since inside applies also to many other objects, such as CITIES with respect to COUNTIES, many similar, often highly redundant operations are defined and implemented which make modifications difficult and yield frequent inconsistencies.

Inheritance is an extremely effective means to model such situations in a GIS, formalizing the structure and properties of the object classes. By the definition of a general superclass for each specific concept, common properties may be defined in a single high-level class and inherited to the classes of the GIS application. For example, the superclass GEOMETRIC defines geometry with properties, e.g., location, spatial relationships such as neighborhood, inclusion, intersection, distance, and direction (Egenhofer and Herring 1990). A class in the user model can be defined as a subclass of Geometric inheriting all these properties. For example, the class BUILDING is a SPATIAL object. BUILDING can be described as the subclass of GEOMETRIC inheriting all spatial properties (Figure 11). Other properties can be defined in a similar way. For example, database properties, such as persistency, multi-user access, and transaction control, can be inherited from a superclass Persistent. The general database operations, such as store, delete, retrieve, and modify, are then defined for the class PERSISTENT and passed to the specific object classes. If the class BUILDING is a PERSISTENT class, BUILDINGS can be stored, deleted, retrieved, and modified.

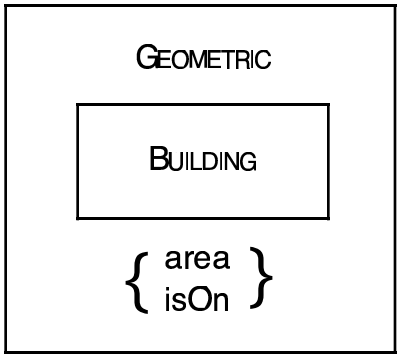


Figure 11: The class BUILDING inherits the geometric properties from the superclass GEOMETRIC.

It is obvious that this type of modeling requires multiple inheritance (Frank 1988b). A class can have a multitude of diverse properties to be inherited. Important GIS properties are PERSISTENT providing database behavior, GEOMETRIC inheriting common geometric concepts, GRAPHICAL providing graphical display, and TEMPORAL for the description of the history of data (Frank and Egenhofer 1992; Frank 1988b). For example, buildings with geometric, graphical, and database behavior can be modeled by creating a class Building with properties, such as address and owner, and then inheriting Geometric, Graphical, and Persistent properties from the corresponding superclasses (Figure 12).

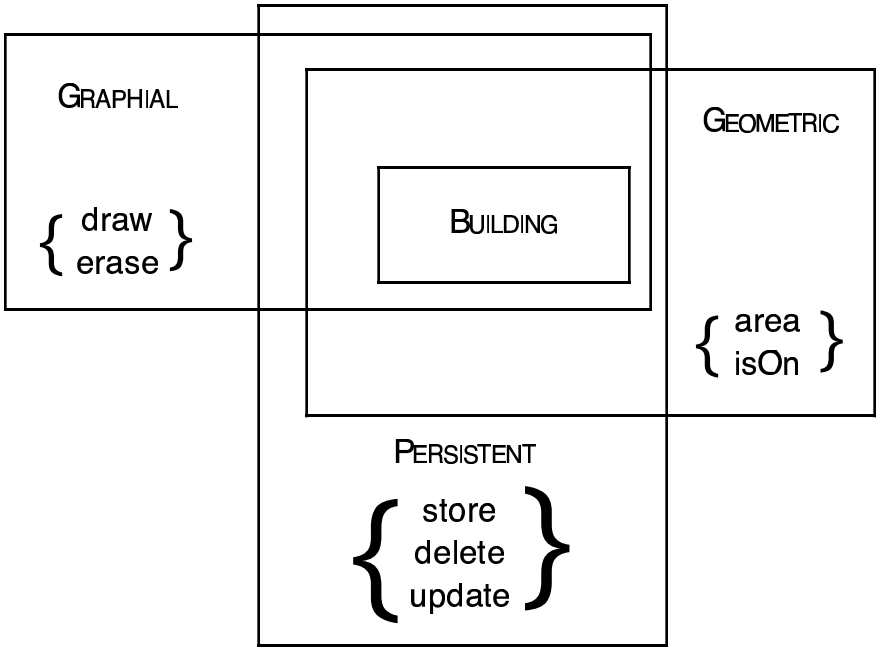


Figure 12: The class BUILDING inherits the various properties from multiple subclasses.

### 4.2 Propagation

Frequently, complex objects are not independent and have some property values that rely upon values of other objects (Kim *et al.* 1987). In aggregation hierarchies, for example, some values of a composite object depend on values of the properties of its components. These dependencies are meaningful and in order to guarantee consistency and integrity, their correct modeling is crucial. Of course, a composite object may have property values that it owns specifically and which are independent from those of their components. In contrast to less powerful models which require redundant storage of such values, the object-oriented model allows objects to have properties with values which rely on values of other objects and models these dependencies consistently. This model is superior, because it enforces integrity by constraints. These derived values frequently describe geometric or statistical properties. Particularly in GISs, a large number of attribute values at one level of abstraction depend upon values from another level and must be derived from them. When combining local and regional data, this concept of modeling data at different levels of resolution must be used to furnish consistency among dependent values. The population of a county, for example, depends on the populations of all related settlements; therefore, the value for the property population of a county must be derived from all values of the property population owned by the settlements (Figure 13).

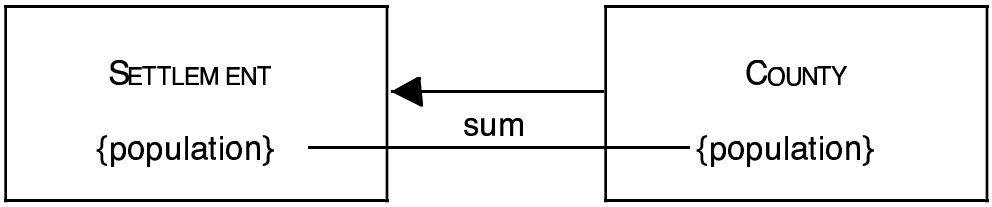


Figure 13: The value of the COUNTY population is propagated as the sum of the population of the aggregated Settlements.

While inheritance describes properties of subclasses (types and operations), *propagation* describes how a *value* of a property of one class is derived from values of properties of another class (Egenhofer and Frank 19986). The notion of propagation is sometimes also used for modeling the behavior of operations, such as copy, destroy, print, and save, upon composite objects and how these operations propagate to their components (Rumbaugh 1988), and consistency of actions (Ellis *et al.* 1990). Here, propagation describes dependencies in the reverse direction—from the components to the composite object. Formal definitions of propagation, also demonstrating the differences between inheritance and propagation, have been given in terms of first-order predicate calculus (Egenhofer and Frank 1989; Egenhofer and Frank 1990) and are also part of more comprehensive algebras for complex objects (Shaw and Zdonik 1989; Beeri and Kornatzky 1990). There may be multiple methods to deduce a value; a value may also be recorded explicitly even if it could be derived from others. It may be necessary to resolve discrepancies and errors resulting from such "redundancies," e.g., by assessing the quality of the different results.

Propagation becomes trivial if the complex object happens to be composed of a single part and the value of the aggregate refers to a single value of the part; however, in most cases propagation involves values of multiple components. If more than a single value contributes to the derived value, the combination of the values must be described by an aggregate function. Aggregate functions combine the values of one or several properties of the components to a single value. This value reduces the amount of detail available for a complex object. It may determine the sum or union of values of the components, or define a specific, outstanding part such as the greatest, heaviest, or conversely, the smallest or lightest one. On the other hand, it may be representative such as the average or weighted average of the values of a specific property. Common aggregate operations are minimum, maximum, sum, average, and weighted average. For example, the population of the biggest city in a county is the maximum of the populations of all its cities; the area of a state is the sum of the areas of all its counties; the population density of the state is the average of the population density of its counties weighted by the county areas.

The concept of propagation guarantees consistency, because data is only stored once and the dependent values of the aggregate are derived; therefore, derived aggregate values need not be updated every time the components are changed. Of course, updates underlie the common rules for updates of views (Dayal and Bernstein 1978), i.e., no derived properties can be updated explicitly, but only the fundamental properties. For example, modifying the population of Penobscot county by assigning the value 65,000 to the county population if the town population of Orono grows by 5,000 is not allowed. Instead, the population of the settlements must be modified which implicitly updates the county population.

Two characteristics of propagation are observed: (1) the propagation of an aggregate value may involve several values from different classes, and (2) propagation may be transitive, i.e., propagated values may be used to derive further aggregated values. The following two examples clarify these characteristics. The area of a County depends on the areas of its LandParcels, Roads, Lakes, and Rivers and must be derived as the sum of all areas of these components. An example for the transitivity of propagation is the population of the largest county (populationOfLargestCounty) in a STATE, which depends on the population of the COUNTIES, which in turn is the sum of the populations of their SETTLEMENTS. Implementation considerations recommend that these computationally expensive aggregate operations are reduced to a minimum to

improve query performance (Blakeley *et al*. 1986), sometimes by introducing redundant storage of aggregated values.

## 5. Alternatives to Object-Oriented Data Models

With the knowledge and understanding of the object-oriented model it is now possible to assess its benefits and trade-offs when looking at other data models.

### 5.1 The Relational Data Model

Today, the relational data model is the most common one. It seems to be suitable for modeling commercial data for which humans may have the mental model of tables, such as bank accounts; however, it is too simplistic for modeling data that describe spatial phenomena. Though geographic data may be modeled with strings of characters, reals, and integers in tables (Osborn and Heaven 1986; Waugh and Healey 1987; Abel 1988; von Meyer 1989; Herring 1992), a higher-level model is desired in which more complex data types can be used (Nyerges 1980; Egenhofer and Frank 1987). The relational model lacks, for instance, the powerful concept of recursion, which is crucial for modeling multi-scale situations such as spatial data and their subdivisions. Regions can be decomposed into several smaller regions, each of which may be continuously decomposed further. This recursive subdivision applies, for instance, to the subdivision of land parcels. Likewise, existing commercial database management systems are not sufficient for applications of *non-standard database applications* (Härder and Reuter 1985) such as geographic information systems or automated cartography. Each of these applications contains substantial amounts of "real world" data with geometric aspects. Their composition it too complex to be efficiently managed in conventional database management systems. Current database management technology falls short of the requirements for engineering (Sidel 1980; Udagawa and Mizoguchi 1984; Buchmann and de Celis 1985) and scientific (French *et al*. 1990) database applications, because the treatment of complex objects (Lorie and Plouffe 1983), such as spatial objects in GIS/LIS (Frank 1984), circuits in VLSI, or molecules in chemistry (Batory and Buchmann 1984), is not supported and appropriate mechanisms for data structuring are missing. As a consequence, performance is unacceptable when a database is populated with large amounts of data (Wilkins and Wiederhold 1984; Härder and Reuter 1985; Maier 1986). When applied to geographic data handling, the same problems have been observed (Smith and Frank 1990; Guenther and Buchmann 1990).

### 5.2 Extensions of Traditional Models

A number of methods to capture more semantics in the data model have been proposed in the literature, often referred to as *semantic data models* (Hull and King 1987). Most of these methods have not been implemented, and only a few of them, such as SIM by Unisys, are readily available. It is suspected that if the transition from the first scientific publications to commercial products happesn at the same pace as with relational technology, then the first commercial products based upon semantic data models may become reality in the mid '90s (Peckham and Maryanski 1988).

In the past, considerable effort was made to enrich existing data models with facilities to treat complex objects. ADT-INGRES (Stonebraker *et al*. 1983) extends the relational model with features to define more complex types. DAPLEX (Shipman 1981) is a functional language that includes hierarchical relationships and transitive closure. The extension of the relational model with a surrogate concept provides hierarchical relationships between relations (Meier and Lorie 1983). The $NF^2$ model (Schek 1985) supports composite attribute types being tuples or relations performing a hybrid of the relational and hierarchical data models. The POSTGRES data model (Stonebraker and Rowe 1986) is based on the relational data model and ties complex objects and the corresponding operations close together. The molecule-atom data model (MAD and the molecule algebra (Mitschang 1989) extend relational algebra to model aggregates as complex objects. Investigations of how geometry could be modeled using these extended data models showed that only partial remedies are provided with these extensions (Kemper and Wallrath 1987), because they lack sophisticated object orientation.

Only recently has the design of database management systems specifically for spatial information systems become a research topic (Frank 1981; Lipeck and Neumann 1986; Schek and Waterfeld 1986; Smith and Frank 1990) and a few experimental spatial database management systems exist, such as PANDA (Frank 1982), or database management systems with extensions for spatial data handling, such as DASDBS (Waterfeld *et al*. 1988). Current commercial GIS software systems tend toward sophisticated software engineering methods (Aronson and Morehouse 1983) and object-oriented concepts for geometric data handling (Herring 1987; Chance *et al*. 1990); however, database management systems, and especially object-oriented ones, have not yet been widely incorporated into commercial GISs.

## 6. Conclusions

This paper presented a synthesis of an object-oriented data model based on the abstraction concepts of classification, generalization, association, and aggregation. Inheritance and propagation are crucial for modeling the behavior of complex objects. It has been demonstrated how object-oriented modeling can serve as a suitable method for the design of spatial information systems. Current data models and database technology are not sufficient for the specific tasks of dealing with large amounts of spatial data. Recently, research in non-standard database environments promoted an object-oriented model, which promises to overcome some problems that make conventional database management systems unsuitable, such as the lack of modeling power to adequately describe complex objects and the unacceptably slow performance of current implementations. Recent benchmarks (Cattel and Skeen 1992) of some object-oriented database management systems have given evidence that they can perform more than an order or magnitude better than their relational predecessors.

Object orientation is an encompassing methodology that can be applied to all stages of designing and implementing a large, complex software system. Commercial products for object-oriented database management systems have become available (Maier *et al*. 1986; Fishman 1986), with more to appear on the market in the near future (Deuz *et al.* 1991; Lamb *et al*. 1991). Future GISs are excellent candidates for applying object-oriented concepts in order to improve their design, implementation, and maintenance. All three object-oriented concepts—modeling, software engineering, and database management systems—are important for GIS personnel:

- The object-oriented abstraction mechanisms are necessary to model the complex situations, such as geometric objects, which can change over a period of time. The complexity of spatial objects requires methods to define and use appropriate spatial data types and operations. The object-oriented model supports this task with appropriate abstraction mechanisms. In particular, the powerful concept of inheritance gives rise to consistent and concise definitions for properties such as geometry, graphics, and persistency. Data structures are necessary for recursive object definitions, such as areas being subdivided into other areas, and transitive closure operations.
- Object-oriented programming languages will be needed to implement the future GIS most efficiently. In recent years, research in software engineering has promoted an object-oriented design method by which real world objects and their relevant operations can be modeled in a program. This approach is most useful for application areas like GISs because it naturally supports the treatment of complex, in this case geometric, objects (Kjerne and Dueker 1990). Compared with conventional data models, an object-oriented design is more flexible and better-suited to describe complex data structures.
- Object-oriented database management systems must be used to exploit the modeling power and performance to manage and retrieve spatial data. Spatial information systems will benefit from the use of object-oriented database management systems in various ways: (1) The architecture of a GIS will become clearer such that the maintenance of GIS software will be easier and its life cycle will be longer. (2) Programmers should not worry about aspects of the physical location of data; instead, a unified set of commands provides the

functionality for storage and retrieval of data. (3) By using a database management system, data is treated by its properties; the object-oriented approach groups these properties into possibly complex objects and corresponding operations.

## 7. Acknowledgments

This work was partially supported by a grant from Intergraph Corporation and grants from Digital Equipment Corporation under TP-765536 and Sponsored Research Agreement No. 414. The support from NSF for the National Center for Geographic Information and Analysis (NCGIA) under grant number SBE-8810917 is gratefully acknowledged.

Our understanding of object orientation has been improved during many discussions with Renato Barrera and Alex Buchmann. John Isner made us familiar with object-oriented concepts in C++. Robert Cicogna, Doug Hudson, and Jeff Jackson read and commented on an earlier version of this paper.

## 8. References

D. Abel (1988) Relational Data Management Facilities for Spatial Information Systems. in: D. Marble (Ed.), *Third International Symposium on Spatial Data Handling*, Sydney, Australia, pp. 9-18.

S. Abiteboul and R. Hull (1987) IFO: A Formal Semantic Database Model. *ACM Transactions on Database Systems* 12(4): 525-565.

K. Al-Taha and R. Barrera (1990) Temporal Data and GIS: An Overview, *GIS/LIS '90*, Anaheim, CA, Vol. 1, pp. 244-254.

P. Aronson and S. Morehouse (1983) The ARC/INFO Map Library: A Design for a Digital Geographic Database, *Auto-Carto VI*, Ottawa, pp. 346-382.

F. Bancilhon, G. Barbedette, V. Benzaken, C. Delobel, S. Gamerman, C. Lecluse, P. Pfeffer, P. Richard, and F. Velez (1988) The Design and Implementation of O2, an Object-Oriented Database System. in: K. R. Dittrich (Ed.). *Advances in Object-Oriented Database Systems—Proceedings of the 2nd International Workshop on Object-Oriented Database Systems, Bad Muenster am Stein-Ebernburg, F.R. Germany,*. Lecture Notes in Computer Science 334**:** 1-22. New York, NY, Springer-Verlag.

R. Barrera and A. Buchmann (1981) Schema Definition and Query Language for a Geographical Database System. *IEEE Transactions on Computer Architecture: Pattern Analysis and Image Database Management* 11**:** 250-256.

C. Beeri and Y. Kornatzky (1990) Algebraic Optimization of Object-Oriented Query Languages. in: S. Abiteboul and P. Kanellakis (Eds). *Third International Conference on Database Theory, Paris, France*. Lecture Notes in Computer Science 470**:** 72-88. New York, NY, Springer-Verlag.

G. Birkhoff and J. Lipson (1970) Heterogeneous Algebras. *Journal of Combinatorial Theory* 8: 115-133.

J. Blakeley, P.-A. Larson, and F. Tompa (1986) Efficintly Updating Materialized Views, *ACM-SIGMOD, International Conference on Management of Data*, Washington, D.C., Vol. 15, pp. 61-71.

A. Borgida (1988) Modelling Class Hierarchies with Contradictions. in: H. Boral and P.-A. Larson (Eds), *1988 ACM SIGMOD, International Conference on Management of Data*, Chicago, IL, pp. 434-443.

M. Brodie, J. Mylopoulos, and J. Schmidt (1984) *On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages*. New York, NY, Springer-Verlag.

M. L. Brodie (1984) On the Development of Data Models. in: M. Brodie, J. Mylopoulos, and J. Schmidt (Eds). *On Conceptual Modelling***:** 19-48. New York, NY, Springer-Verlag.

B. Bruegger and A. Frank (1989) Hierarchies over Topological Data Structures, *ASPRS-ACSM Annual Convention*, Baltimore, MD, pp. 137-145.

B. Buttenfield (1989) *Multiple Representations: Initiative 3 Specialist Meeting Report*. Technical Report 89-3, National Center for Geographic Information and Analysis, Santa Barbara, CA.

B. Buttenfield and R. McMaster (1991) *Map Generalization: Making Rules for Knowledge Representation*. London, Longman.

L. Cardelli (1984) A Semantics of Multiple Inheritance. in: G. Kahn, D. McQueen, and G. Plotkin (Eds). *Semantics of Data Types*. Lecture Notes in Computer Science 173**:** 51-67. New York, NY, Springer-Verlag.

R. Cattel and J. Skeen (1992) Object Operations Benchmark. *ACM Transactions on Database Systems* 17(1): 1-31.

A. Chance, R. Newell, and D. Theriault (1990) An Object-Oriented GIS—Issues and Solutions, *EGIS '90*, Amsterdam, The Netherlands.

P. S. S. Chen (1976) The Entity-Relationship Model: Towards a Unified View of Data. *ACM Transactions on Database Systems* 1(1): 9-36.

E. Codd (1981) Data Models in Database Management. *SIGMOD RECORD* 11(2): 112-114.

E. F. Codd (1972) Further Normalization of the Data Base Relational Model. in: R. Rustin (Ed.). *Data Base Systems***:** 33-64. Englewood Cliffs, NJ, Prentice-Hall.

E. F. Codd (1982) Relational Data Base: A Practical Foundation for Productivity. *Communications of the ACM* 25(2): 109-117.

O.-J. Dahl and K. Nygaard (1966) SIMULA—An Algol-based Simulation Language. *Communications of the ACM* 9(9): 671-678.

Data Base Task Group Report (1971) *CODASYL*. Technical Report, New York, NY.

C. J. Date (1986) *An Introduction to Database Systems*. Reading, MA, Addison-Wesley Publishing Company.

E. Davis (1986) *Representing and Acquiring Geographic Knowledge*. Los Altos, CA, Morgan Kaufmann Publishers, Inc.

U. Dayal and P. Bernstein (1978) On the Updatability of Relational Views. in: S. B. Yao (Ed.), *Fourth International Conference on Very Large Data Bases*, West-Berlin, Germany.

O. e. a. Deux (1991) The O2 System. *Communications of the ACM* 34(10): 34-48.

K. Dittrich (1986) Object-Oriented Database Systems: The Notation and The Issues. in: K. Dittrich and U. Dayal (Eds), *International Workshop in Object-Oriented Database Systems, Pacific Grove, CA*, Washington, D.C., pp. 2-4, IEEE Computer Society Press.

M. Egenhofer (1990) Interaction with Geographic Information Systems via Spatial Queries. *Journal of Visual Languages and Computing* 1(4): 389-413.

M. Egenhofer and A. Frank (1986) Connection between Local and Regional: Additional `Intelligence' Needed, *FIG XVIII. International Congress of Surveyors, Commission 3 Land nformation Systems*, Toronto, Ontario, Canada.

M. Egenhofer and A. Frank (1987) Object-Oriented Databases: Database Requirements for GIS, *International Geographic Information Systems (IGIS) Symposium: The Research Agenda*, Arlington, VA, Vol. II, pp. 189-211.

M. Egenhofer and A. Frank (1988) A Precompiler For Modular, Transportable Pascal. *SIGPLAN Notices* 23**:** 22-32.

M. Egenhofer and A. Frank (1989) Object-Oriented Modeling in GIS: Inheritance and Propagation. *AUTO-CARTO 9, Ninth International Symposium on Computer-Assisted Cartography***:** 588-598. Baltimore, MD.

M. Egenhofer and A. Frank (1990) LOBSTER: Combining AI and Database Techniques for GIS. *Photogrammetric Engineering & Remote Sensing* 56(6): 919-926.

M. Egenhofer, A. Frank, and J. Jackson (1989) A Topological Data Model for Spatial Databases. in: A. Buchmann, O. Günther, T. Smith, and Y. Wang (Eds). *Symposium on the Design and Implementation of Large Spatial Databases.* Lecture Notes in Computer Science 409**:** 271-286. New York, Springer-Verlag.

M. Egenhofer and J. Herring (1990) A Mathematical Framework for the Definition of Topological Relationships. in: K. Brassel and H. Kishimoto (Eds), *Fourth International Symposium on Spatial Data Handling*, Zurich, Switzerland, pp. 803-813.

H. Ellis, S. Demurjian, F. Maryanski, G. Beshers, and J. Peckham (1990) Extending the Behavioral Capabilities of the Object-Oriented Paradigm with an Active Model of Propagation, *18th Annual Computer Science Conference*, Washington, D.C.

D. Fishman, D. Beech, H. Cate, E. Chow, T. Connors, J. Davis, N. Derrett, C. Hoch, W. Kent, P. Lyngbaek, B. Mahbod, N. Neimat, T. Ryan, and M. Shan (1986) Iris: An Object-Oriented Database Management System. *ACM Transactions on Office Information Systems* 5**:** 48-69.

A. Frank (1981) Applications of DBMS to Land Information Systems. in: C. Zaniolo and C. Delobel (Eds), *Seventh International Conference on Very Large Data Bases*, Cannes, France, pp. 448-453.

A. Frank (1982) PANDA—A Pascal Network Database System. in: G. W. Gorsline (Ed.). *Fifth Symposium on Small Systems*. Colorado Springs, CO.

A. Frank (1984) Requirements for Database Systems Suitable to Manage Large Spatial Databases. *International Symposium on Spatial Data Handling***:** 38-60. Zurich, Switzerland.

A. Frank (1988a) Multiple Inheritance and Genericity for the Integration of a Database Management System in an Object-Oriented Approach. in: K. R. Dittrich (Ed.). *Advances in Object-Oriented Database Systems—Proceedings of the 2nd International Workshop on Object-Oriented Database Systems, Bad Muenster am Stein-Ebernburg, F.R. Germany,*. Lecture Notes in Computer Science 334**:** 268-273. New York, NY, Springer-Verlag.

A. Frank (1988b) Requirements for a Database Management System for a GIS. *Photogrammetric Engineering & Remote Sensing* 54(11): 1557-1564.

A. Frank and M. Egenhofer (1992) Computer Cartography for GIS: An Object-Oriented View on the Display Transformation. *Computers and Geosciences* 18(8): 975-987.

A. Frank, M. Egenhofer, and W. Kuhn (1991) A Perspective on GIS Technology in the Ninties. *Photogrammetric Engineering & Remote Sensing* 57(11): 1431-1436.

A. Frank and W. Kuhn (1986) Cell Graph: A Provable Correct Method for the Storage of Geometry. in: D. Marble (Ed.), *Second International Symposium on Spatial Data Handling*, Seattle, WA, pp. 411-436.

A. Frank and D. Mark (1991) Language Issues for GIS. in: D. Maguire, M. Goodchild, and D. Rhind (Eds). *Geographical Information Systems, Volume 1: Principles***:** 147-163. London, Longman.

H. Gallaire, J. Minker, and J. Nicolas (1984) Logic and Databases: A Deductive Approach. *ACM Computing Surveys* 16(2): 153-185.

A. Goldberg and D. Robson (1983) *Smalltalk-80*. Reading, MA, Addison-Wesley Publishing Company.

O. Günther and A. Buchmann (1990) Research Issues in Spatial Databases. *SIGMOD RECORD* 19(4): 61-68.

J. Guttag (1977) Abstract Data Types And The Development Of Data Structures. *Communications of the ACM* 20(6): 396-404.

T. Härder and A. Reuter (1985) Architecture of Database Systems for Non-Standard Applications (in German). in: A. Blaser and P. Pistor (Eds). *Database Systems in Office, Engineering, and Science*. Informatik Fachberichte 94**:** 253-286. New York, Springer-Verlag.

J. Herring (1987) TIGRIS: Topologically Integrated Geographic Information Systems. in: N. Chrisman (Ed.), *AUTO-CARTO 8 Eighth International Symposium on Computer-Assisted Cartography*, Baltimore, MD, pp. 282-291.

J. Herring (1991) TIGRIS: A Data Model for an Object-Oriented Geographic Information System. *Computers and Geosciences* 18(4): 443-452.

R. Hull and R. King (1987) Semantic Database Modeling: Survey, Applications, and Research Issues. *ACM Computing Surveys* 19(3): 201-260.

A. Kemper and M. Wallrath (1987) An Analysis of Geometric Modeling in Database Systems. *ACM Computing Surveys* 19(1): 47-91.

W. Kim, J. Banerijee, H.-T. Chou, J. Garza, and D. Woelk (1987) Composite Object Support in an Object-Oriented Database System. *OOPSLA '87***:** 118-125. Orlando, FL.

D. Kjerne and K. J. Dueker (1986) Modeling Cadastral Spatial Relationships Using an Object-Oriented Language. in: D. Marble (Ed.). *Second International Symposium on Spatial Data Handling*. Seattle, WA.

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

G. Langran (1989) Temporal GIS Design Tradeoffs. *URISA Journal* 2(2): 16-25.

U. W. Lipeck and K. Neumann (1986) Modelling and Manipulation of Objects in Geoscientific Databases. in: S. Spaccapietra (Ed.), *5th International Conference on the Entity-Relationship Approach*, Dijon, France, pp. 67-86.

B. Liskov, R. Atkinson, T. Bloom, E. Moss, J. C. Schaffert, R. Scheifler, and A. Snyder (1981) *CLU Reference Manual*. New York, NY, Springer-Verlag.

R. Lorie and W. Plouffe (1983) Complex Objects and Their Use in Design Transactions. in: D. DeWitt and G. Gardarin (Eds). *ACM Engineering Design Applications***:** 115-121.

D. Maier (1986) Why Object-Oriented Databases Can Succeed Where Others Have Failed. in: K. Dittrich and U. Dayal (Eds). *International Workshop in Object-Oriented Database Systems, Pacific Grove, CA***:** 227. Washington, D.C., IEEE Computer Society Press.

F. Manola and U. Dayal (1986) PDM: An Object-Oriented Data Model. in: K. Dittrich and U. Dayal (Eds). *International Workshop in Object-Oriented Database Systems, Pacific Grove, CA***:** 18-25. Washington, D.C., IEEE Computer Society Press.

A. Meier and R. A. Lorie (1983) A Surrogate Concept for Engineering Databases. *9th International Conference on Very Large Data Bases*. Florence, Italy.

B. Meyer (1988) *Object-Oriented Software Construction*. New York, Prentice Hall.

B. Mitschang (1989) Extending the Relational Algebra to Capture Complex Objects. in: P. Apers and G. Wiederhold (Eds), *15th International Conference on Very Large Data Bases*, Amsterdam, pp. 297-305.

T. Nyerges (1980) Representing Spatial Properties in Cartographic Data Bases, *ACSM-ASPRS Annual Convention*, St. Louis, MO, pp. 29-41.

P. O'Brien, B. Bullis, and C. Schaffert (1986) Persistent and Shared Objects in Trellis/Owl. in: K. Dittrich and U. Dayal (Eds). *International Workshop in Object-Oriented Database Systems, Pacific Grove, CA*: 113-123. Washington, D.C., IEEE Computer Society Press.

S. Osborn and T. Heaven (1986) The Design of a Relational Database System with Abstract Data Types for Domains. *ACM Transactions of Database Systems* 11(2): 357-373.

J. Peckham and F. Maryanski (1988) Semantic Data Models. *ACM Computing Surveys* 20(3): 153-189.

J. Rumbaugh (1988) Controlling Propagation of Operations using Attributes on Relations. *OOPSLA '88*: 285-296. San Diego, CA.

H.-J. Schek and W. Waterfeld (1986) A Database Kernel System for Geoscientific Applications. in: D. Marble (Ed.). *Second International Symposium on Spatial Data Handling*: 273-288. Seattle, WA.

K. Schmucker (1986) MacApp: An Application Framework. *Byte* 11(8): 189-193.

A. Sernadas (1980) Temporal Aspects of Logical Procedure Definition. *Information Systems* 5: 167-187.

G. Shaw and S. Zdonik (1989) An Object-Oriented Query Algebra. *Data Engineering* 12(3): 29-36.

D. Shipman (1981) The Functional Data Model and the Data Language DAPLEX. *ACM Transactions on Database Systems* 6: 140-193.

T. W. Sidle (1980) Weakness of Commercial Database Management Systems in Engineering Applications. *17th Design Automation Conference*: 57-61. Minneapolis, MN.

J. M. Smith and D. C. P. Smith (1977a) Database Abstractions: Aggregation. *Communications of the ACM* 20(6): 405-413.

J. M. Smith and D. C. P. Smith (1977b) Database Abstractions: Aggregation and Generalization. *ACM Transactions of Database Systems* 2(2): 105-133.

T. Smith and A. Frank (1990) Very Large Spatial Databases: Report from the Specialist Meeting. *Journal of Visual Languages and Computing* 1(3): 291-309.

R. Snodgrass (1987) The Temporal Query Language TQUEL. *ACM Transactions on Database Systems* 12: 247-298.

M. Stonebraker, R. Katz, D. Patterson, and J. Ousterhout (1988) The Design of XPRS. in: D. DeWitt and F. Bancilhon (Eds). *14th International Conference on Very Large Data Bases*. Los Angeles, CA.

M. Stonebraker, B. Rubenstein, and A. Guttman (1983) Application of Abstract Data Types and Abstract Indices to CAD Databases. in: D. DeWitt and G. Gardarin (Eds). *ACM SIGMOD Conference on Engineering Design Applications*: 107-113. San Jose, CA.

B. Stroustrup (1986) *The C++ Programming Language*. Addison-Wesley Publishing Company.

C. D. Tomlin (1990) *Geographic Information Systems and Cartographic Modeling*. Englewood Cliffs, NJ, Prentice-Hall.

D. Tsichritzis and F. Lochovsky (1977) *Data Base Management Systems*. New York, NY, Academic Press.

Y. Udagawa and T. Mizoguchi (1984) An Extended Relational Database System for Engineering Data Management. *IEEE Database Engineering* 7.

J. Ullman (1982) *Principles of Database Systems*. Rockville, MD, Computer Science Press.

P. van Oosterom (1990) *Reactive Data Structures for Geographic Information Systems*. Ph.D. thesis, Leiden, The Netherlands, University of Leiden.

N. von Meyer (1989) A Conceptual Model for Spatial Cadastral Data in a Land Information System.

W. e. a. Waterfeld (1988) How to Make Spatial Access Methods Extensible. in: D. Marble (Ed.). *Third International Symposium on Spatial Data Handling*: 321-335. Sydney, Australia.

T. Waugh and R. Healey (1987) The GEOVIEW Design: A Relational Database Approach to Geographical Data Handling. *International Journal of Geographical Information Systems* 1: 101-118.

M. W. Wilkins and G. Wiederhold (1984) Relational and Entity-Relationship Model Databases and VLSI Design. *IEEE Database Engineering* 7.

D. Woelk and W. Kim (1987) Multimedia Information Management in an Object-Oriented Database System. in: P. Stocker and W. Kent (Eds). *13th International Conference on Very Large Data Bases*: 319-329. Brighton, England.

M. Worboys, H. Hearnshaw, and D. Maguire (1990) Object-Oriented Data Modelling for Spatial Databases. *International Journal of Geographical Information Systems* 4(4): 369-383.

S. Zdonik and D. Maier (1990) Fundamentals of Object-Oriented Databases. in: S. Zdonik and D. Maier (Eds). *Readings in Object-Oriented Database Systems*: 1-32. San Mateo, CA, Morgan Kaufmann Publishers, Inc.