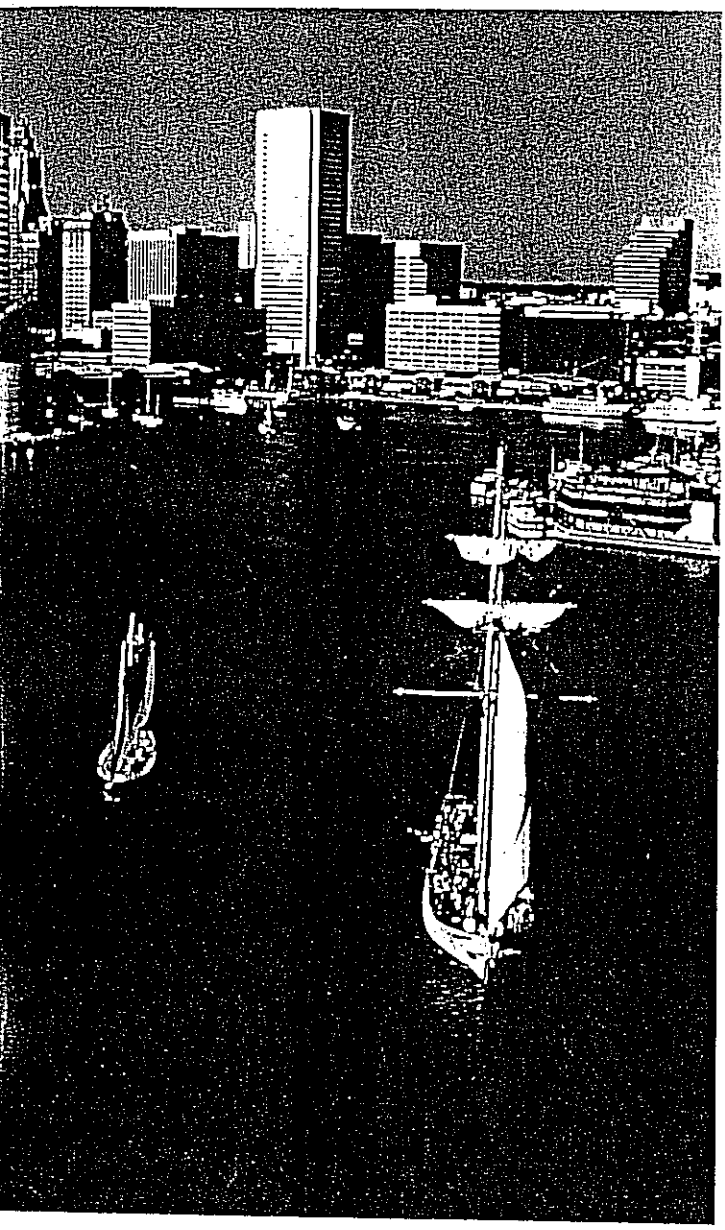
AUTO-CARTO 9

Proceedings

Ninth International Symposium on Computer-Assisted Cartography

Baltimore, Maryland
April 2 — 7



Object-Oriented Modeling in GIS: Inheritance and Propagation*

Max J. Egenhofer
Andrew U. Frank
National Center for Geographic Information and Analysis
and
Department of Surveying Engineering
University of Maine
Orono, ME 04469, USA
MAX@MECAN1.bitnet
FRANK@MECAN1.bitnet

Abstract

The relational data model has proven to be too restrictive for applications with spatial data, such as Geographic Information Systems (GIS). In particular, the absence of techniques to form complex objects and represent spatial objects at different abstraction levels makes it difficult to model geographic situations properly. The object-oriented approach, which has been recently promoted for similar engineering applications, such as CAD/CAM, VLSI design, or molecular models in chemistry, seems to overcome some of the deficiencies. By incorporating the abstraction mechanisms *generalization* and *aggregation*, the data model gets richer and more powerful than the relational model, and the application designer is given more and better tools to model complex situations.

Both generalization and aggregation involve the derivation of attribute values at different levels of detail and abstraction. Two methods for the derivation of properties are introduced: (1) *inheritance* describing properties and methods of subclasses in is_a hierarchies, and (2) *propagation* deriving properties in part_of hierarchies. While inheritance acts in a top-down fashion along the generalization hierarchy, propagation can derive values from parts to the aggregates (bottom-up). Frequently aggregate functions, such as SUM, MIN, or MAX are involved to pass on properties of composed objects to its parts.

1 Introduction

The relational model upon which most current GIS software systems are built has been acknowledged as an insufficient model for applications that deal with spatial data [Frank 1984] [Härder 1985] [Frank 1988b]. The application of object-oriented techniques for the design of future Geographic Information Systems has been proposed on several stages, e.g., such as message-passing programing language [Kjerne 1986], object-oriented database management systems [Egenhofer 1987], and object-oriented software engineering techniques [Egenhofer 1989a] [Egenhofer 1989b]. This paper focuses on advanced object-oriented techniques to model the dependencies of properties, operations, and values in generalization- and aggregation hierarchies. These methods are ideal for applications with spatial data, because they enforce natural phenomena, such as the fact that the area of a subdivision is exactly the sum of the areas of the partitions.

*This research was partially funded by grants from NSF under No. IST 86-09123 and Digital Equipment Corporation. The support from NSF for the NCGIA under grant number SES 88-10917 is gratefully acknowledged.

Barrera and Buchmann introduced the derivation of attributes in hierarchies of spatial inclusion and aggregates to geographic applications [Barrera 1981]. Since then, many controverse discussions have been among researchers in the areas of object-oriented modeling, object-oriented database management systems [Dittrich 1986b] [Dittrich 1988], and object-oriented programming languages [OOPSLA 1986a] [OOPSLA 1986b] about various types of inheritance, such as behavior vs. abstract implementation, single vs. multiple, automatic vs. on-demand, and upwards vs. downwards.

The paper begins with an overview of the principles of object-oriented modeling. Examples of GIS applications show the benefits of the abstraction mechanisms classification, generalization, association, and aggregation, for spatial modeling. There are two methods for relating properties from one object to another. In section 3 the data model is extended by *inheritance* linking properties of objects in a generalization hierarchy. In section 4 *propagation* is introduced linking the values of objects that are linked by association or aggregation. The separation of these two different methods is important; therefore, clearly distinct terms are chosen. Propagation is sometimes called in the literature *upward inheritance* [Barrera 1981] [Brodie 1984a], but it should become clear from this paper that there exist two different concepts which must not be mixed. The paper concludes that inheritance and propagation are important for GIS applications and need efficient support from object-oriented programming languages.

2 Object-Oriented Model

This chapter introduces the notation of objects and the abstraction tools available to deal with them. A definition of object-orientation is that an entity of whatever complexity and structure can be represented by exactly one object [Dittrich 1986a]. No artificial decomposition into simpler parts should be necessary due to technical restrictions, e.g., normalization rules [Codd 1972]. The object-oriented data model is built on the four basic concepts of abstraction [Brodie 1984b]: classification, generalization, association, and aggregation.

2.1 Classification

Classification is the mapping of several objects (instances) to 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, every object is an instance of a class. A type characterizes the behavior of its instances by describing the operators that are the only means to manipulate those objects [O'Brien 1986]. All objects that belong to the same class are described by the same properties and have the same operations. Classification is often referred to as the *instance of* relationship because the individuals are *instances of* the corresponding class.

For example, the GIS model for a town may include the classes *residence*, *commercial building*, and *street*. A single instance, such as the residence with the address '30 Grove Street' is an instance of the class *residence*. Operations and properties are assigned to object types, so for instance the class *residence* may have the properties *number of bedrooms* and *address* which are specific for all residences.

2.2 Generalization

Generalization groups several classes of objects, which have some properties and operations in common, to a more general superclass [Dahl 1966] [Goldberg 1983]. The terms *subclass* and *superclass* characterize generalization and refer to object types which are related by an *is_a* relation. The converse relation of superclass, the *subclass*, describes a specialization of the superclass. For example, the object type *residence* is a *building*; *residence* is a subclass of

building, while *building* is its superclass[1].

Generalization may have an arbitrary number of levels in which a subclass has the role o a superclass for another, more specific class. The *building/residence*-generalization can b extended with the class *rural residence*. While *residence* is a subclass of *building*, it is at th same time a superclass for *rural residences*.

It is important to note that superclass and subclass are abstractions for the same object and do not describe two different objects. The residence with the address '30 Grove Street' for example, is at the same time an instance of the class *residence* as well as of its superclas *building*.

2.3 Association

Association is a form of abstraction in which a relationship between similar objects is considered a higher level set object [Brodie 1984b]. The term *set* is used to describe the association, and the associated objects are called *members*. Hence, this abstraction is referred to as the *member_of* relation, but is also often called *grouping* or *partitioning*. For example, a subdivision divides one parcel into several parcels.

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. An operation over a set consists of one operation repeated for each member of the set, e.g., a FOR EACH loop structure, found in some modern programming languages, such as CLU [Liskov 1981].

For example, the *city* Orono and the *building* with the address '30 Grove Street' are associated by the relationship *inside*.

2.4 Aggregation

A similar abstraction mechanism to association is aggregation which models composed objects, i.e., objects which consist of several other objects [Smith 1977]. The term *composit* object describes the higher-level object, while *subpart* or *component* refers to the parts of the composit object. The relationship among the components and the composit object is the *part_of* relationship, and the converse relationship is *consists_of*. For example, the class *building* is an aggregate of all *walls*, *windows*, *doors*, and *roofs* which are *part of* it.

When considering the aggregate, details of the constituent objects are suppressed. Every instance of an aggregate object can be decomposed into instances of the component objects. Each part keeps its own functionality. Operations of aggregates are not compatible with operations on parts.

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.

3 Inheritance

In generalization hierarchies, the properties and methods of the subclasses depend upon the structure and properties of the superclass(es). Inheritance is a tool to define a class in terms of one or more other, more general classes [Dahl 1966]. Properties which are common for superclass and subclasses are defined only once—with the superclass—and inherited by all objects of the subclass, but subclasses can have additional, specific properties and operations which are not shared by the superclass. Inheritance is the transitive transmission of the properties from one

[1] Frequently, the terms *parent* and *child* are 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*.

superclass to all related subclasses, and to their subclasses, etc. This concept is very powerful, because it reduces information redundancy [Woelk 1987] and maintains integrity. Modularity and consistency are supported since essential properties of an object are defined once and are inherited in all relationships in which it takes part.

Operations of the superclass are applicable to all objects of the subclass because each object of the subclass is at the same time an object of the superclass; however, operations which are specifically defined for a subclass are not compatible with superclass objects.

3.1 Single Inheritance

The inheritance relation can be restricted to form a strict hierarchy and is then often referred to as *single* inheritance. Single inheritance requires that each class has at most a single immediate superclass. This restriction implies that each subclass belongs only to a single hierarchy group and one class cannot be part of several distinct hierarchies.

Figure 1 shows an example of inheritance along a generalization hierarchy. *Residence* is the general superclass and *city residence* and *rural residence* are the specific subclasses. All properties of the class *residence* are inherited to the two subclasses. For example, *residentName* is a property of the class *residence* which applies to all *city residences* and *rural residences*, and hence is inherited to them. Likewise, all operations defined upon *residence*, such as *moving into a residence*, are applicable both to *city residences* and *rural residences*. On the other hand, the operations defined specifically for a subclass are not applicable for objects of the superclasses. For example, *nextSubwayStop* is a property which applies only to city residences.

The common representation of hierarchies as trees is used for strict inheritance with the most general superclass at the top, and the most specific subclasses at the bottom. Each class is modelled as a node, while the *is_a* relation between two nodes is visualized as a vector pointing from the node of the superclass to the node of the subclass. The direction of the vector is to emphasize the top-down concept of inheritance—from the general to the specific.

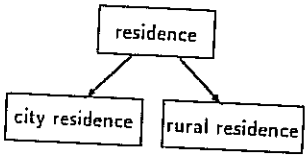


Figure 1: An example of inheritance along a generalization hierarchy with the more general class at the top and more specialized classes at the bottom.

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. Figure 2 shows a more complex generalization hierarchy with 3 levels of classes. The properties of a *building*, such as *address* and *owner*, are inherited to the subclass *residence*, and also transitively to the sub-subclasses *rural residence* and *city residence*.

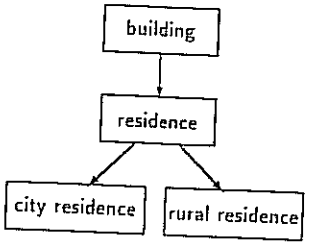


Figure 2: Properties are transitively inherited from a superclass to all its subclasses, the sub-subclasses, etc.

3.2 Formalism

The inheritance relation can be described formally in First Order Predicate Calculus. First Order Predicate Calculus is a language based upon a set of primitive symbols composed of (1) variables, constants, and predicate symbols; (2) logical connectors NOT, AND, OR, implication, and equivalence; (3) quantifiers FOR ALL and EXISTS; and (4) parentheses. A combination of constants and variables (called predicates), linked with the logical connectors, is called a well-formed formula (*wff*).

Subsequently, constants (or facts) are capitalized, while variable names are lower cased. Facts and rules (axioms) will be marked by an asterisk (*) to distinguish from inquiries about predicates (hypothesis). Each property of a class is expressed as a predicate of the form *p (class, property)*. Generalization is described as the *is_a*-predicate of the form *is_a (subclass, superclass)*. The following facts describe the model depicted in figure 2.

```
*p (Building, Address).
*p (Building, Owner).
*p (Residence, Resident).

*is_a (RuralResidence, Residence).
*is_a (CityResidence, Residence).
*is_a (Residence, Building).
```

Inheritance is then defined by a predicate *properties* which recursively derives the properties associated with a class and all its superclasses.

```
*properties (class, property) IF p (class, property).
*properties (class, property) IF is_a (class, superclass),
                                 properties (superclass, property).
```

All properties of the class *cityResidence* can then be determined with the predicate

```
properties (CityResidence, prop).
```

which is fullfilled with the following values for the variable *prop*:

```
prop = Resident
prop = Address
prop = Owner
```

Likewise, it can be determined to which classes a certain property belongs, e.g.,

```
property (class, Resident).
```

yields

```
class = Residence
class = RuralResidence
class = CityResidence
```

This example showed the need for the definition of properties only once, that is with the most general superclass. All dependent properties are derived with the transitivity rule.

3.3 Multiple Inheritance

The structure of a strict hierarchy is an idealized model and fails frequently when applied to real world data. Most 'hierarchies' have at least a few non-hierarchical exceptions in which one subclass has more than a single, direct superclass. Very often more than one hierarchy of classes exists which is used concurrently. Again, the one-superclass-per-subclass rule is violated. Thus, pure hierarchies are not always the adequate structure for inheritance. Instead, the concept of *multiple inheritance* [Cardelli 1984] permits to pass operations or 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. Figure 3 shows the simplest case of multiple inheritance with a subclass inheriting properties from two distinct superclasses.

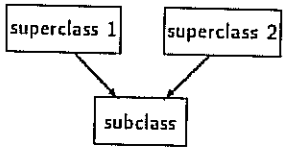


Figure 3: Multiple Inheritance: The properties of 2 distinct superclasses are passed to a common subclass.

An example from geography shows how multiple inheritance combines often two distinct hierarchies. One hierarchy is determined by the separation of *artificial* and *natural* transportation links, whereas the other hierarchy distinghuishes *water bodies*. Classes with properties from both hierarchies are *channels*, that are *artificial transportation links* and *water bodies*, and *navigable rivers*, that are *rivers* and *natural transportation links*. Other classes, such as *highway* or *pond* belong only to one hierarchy (Figure 4).

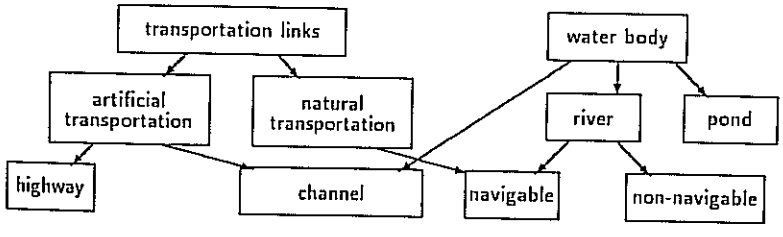


Figure 4: An example of multiple inheritance.

3.4 Inheritance in GIS

Inheritance plays an important role for the clear and concise definition and implementation of very large software systems,such as Geographic Information Systems. The tools to implement such a system should be at least as powerful as the tools used for the conceptual model.

The advantages are similar to the ones of semantic models: Complex situations can be described concisely; consistency can be achieved by avoiding redundancy; and systems can be

maintained easier. A specific problem of the implementation of a GIS is the coexistence of a number of fairly complex tasks, such as the treatment of geometry, graphical representation, concurrent sharing of data, management of history and versions, etc.

Inheritance can be used as a software engineering design tool to describe the structure and properties of a GIS. One part of an application model is the definition of a set of classes as the abstraction of objects with common properties. Traditionally, for each class the appropriate operations and relationships must be defined, including operations which combine objects of different classes. For example, the class *building* has the operation *inside* which checks whether a building is located inside a parcel. Since *inside* applies also to many other objects, such as *cities* with respect to *countries*, many similar, often highly redundant operations are defined and implemented which make modifications difficult and yield frequently inconsistencies.

The application of inheritance overcomes these problems. 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. Such a framework may consist of general superclasses, such as *spatial*, *graphical*, *temporal*, and *db-persistent*.

For example, the superclass *spatial* defines the geometric properties, such as location, spatial relationships, and spatial operators. A class in the user model can be defined as a subclass of *spatial* inheriting all its properties. For example, the class *building* is a spatial object. *Building* can be described as the subclass of *spatial* inheriting all spatial properties, such as the operation *inside* (Figure 5).

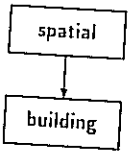


Figure 5: Creating a spatial class *building* by inheriting the spatial properties from a superclass *spatial*.

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 *db-persistent*. The general database operations, such as store, delete, retrieve, and modify, are defined for the class *db-persistent* and passed to the specific object classes. If the class *building* is a db-persistent class, then buildings can be stored, deleted, retrieved, and modified.

It is obvious that this type of modeling requires multiple inheritance [Frank 1988a]. A class can have a multitude of diverse properties to be inherited. Important properties for GISs are *db-persistent* providing database bahavior, *spatial* inheriting a common geometric concept, *graphical* providing graphical display, and *temporal* for the description of history of data [Egenhofer 1988]. Figure 6 shows the creation of the class *building* with two properties: *spatial* and *db-persistent*.

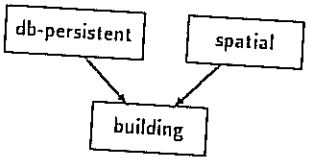


Figure 6: Creating a class *building*, the instances of which being both spatial and persistent.

The population of the county is a value which is propagated from the values of the population of the associated settlements.

The population of a county is the sum of the population of the settlements. This dependency is expressed by the following rule, meaning the population of a specific county is the sum of the population of all settlements which are part of it.

```
*propagates (PartOf, SettlementPopulation, CountyPopulation, BySumming).
```

The generic rule for propagation is the following predicate. It describes the value of the property of an aggregate in terms of the values of the components using a specific aggregation function.

```
*p (aggregateClass, aggregateProperty, aggregateValue) IF
    propagates (relation, componentProperty, aggregateProperty, operation),
    p (componentClass, relation, aggregateClass),
    p (componentClass, componentProperty, componentValue),
    p (operation, componentValue, aggregateValue).
```

For example, the value of the property *countyPopulation* is then evaluated with

```
p (County, CountyPopulation, x).
```

and results in

```
x = 60,000
```

Propagation guarantees consistency because data is only stored once and derived from there. Updates underlie the common rules for updates of views [Dayal 1978], i.e., no derived properties can be updated explicitly, but only the fundamental properties. For example, it is not allowed to update the population of the Penobscot county by asssigning the value 65,000 to the property *countyPopulation* if the town population of Orono grows by 5,000. Instead, the population of the settlements must be modified, e.g., `*p (Orono, SettlementPopulation, 15,000)`, which implicitly updates the countyPopulation.

5 Conclusion

The object-oriented model has powerful tools for data structuring, such as classification, generalization, aggregation, and association. In order to model dependencies of properties, operations, and values in hierarchies of generalized and aggregated objects, the concepts of inheritance and propagation are introduced. By using these techniques, complex situations in Geographic Information Systems can be modeled more naturally than with relational tables.

Three important conceptual differences exist between inheritance and propagation: (1) Inheritance is defined in generalization (is_a) hierarchies, while propagation acts in aggregation (part_of) or association (member_of) hierarchies. (2) Inheritance describes properties and operations, while propagation derives values of properties. (3) Inheritance is a top-down approach, inheriting from the more general to the more detailed class. Propagation on the other hand acts bottom-up.

Implementations need efficient support for these techniques. For example, programming languages must include object-oriented language constructs to model generalization and inheritance; to loop over aggregation parts; and to defined propagation.

6 Acknowledgement

Thanks to Renato Barrera and Alex Buchmann for many stimulating discussions which contributed to our understanding of object-orientation and propagation.

References

[Barrera 1981] R. Barrera and A. Buchmann. Schema Definition and Query Language for a Geographical Database System. IEEE Transactions on Computer Architecture: Pattern Analysis and Image Database Management, 11, 1981.

[Brodie 1984a] M.L. Brodie and D. Ridjanovic. On the Design and Specification of Database Transactions. In: M.L. Brodie et al., editors, On Conceptual Modelling, Springer Verlag, New York, NY, 1984.

[Brodie 1984b] M.L. Brodie. On the Development of Data Models. In: M.L. Brodie et al., editors, On Conceptual Modelling, Springer Verlag, New York, NY, 1984.

[Cardelli 1984] L. Cardelli. A Semantics of Multiple Inheritance. In: G. Kahn et al., editors, Semantics of Data Types, Springer Verlag, New York, NY, 1984.

[Codd 1972] E.F. Codd. Further Normalization of the Data Base Relational Model. In: R. Rustin, editor, Data Base Systems, Prentice-Hall, Englewood Cliffs, NJ, 1972.

[Dahl 1966] O.-J. Dahl and K. Nygaard. SIMULA—An Algol-based Simulation Language. Communications of the ACM, 9(9), September 1966.

[Dayal 1978] U. Dayal and P. Bernstein. On the Updatability of Relational Views. In: S. Bing Yao, editor, Fourth International Conference on Very Large Data Bases, West-Berlin, Germany, 1978.

[Dittrich 1986a] K. Dittrich. Object-Oriented Systems: The Notation and The Issues. In: K. Dittrich and U. Dayal, editors, International Workshop in Object-Oriented Database Systems, Pacific Grove, CA, 1986.

[Dittrich 1986b] K. Dittrich and U. Dayal, editors. Proceedings of the International Workshop in Object-Oriented Database Systems, Pacific Grove, CA. Springer-Verlag, New York, NY, 1986.

[Dittrich 1988] K. Dittrich, editor. Advances in Object-Oriented Database Systems—Proceedings of the 2nd International Workshop on Object-Oriented Database Systems, Bad Münster am Stein-Ebernburg, F.R. Germany. Springer-Verlag, New York, NY, September 1988. Lecture Notes in Computer Science, Vol. 334.

[Egenhofer 1986] M. Egenhofer and A. Frank. Connection between Local and Regional: Additional 'Intelligence' Needed. In: FIG XVIII. International Congress of Surveyors, Commission 3, Land Information Systems, Toronto, Ontario, Canada, 1986.

[Egenhofer 1987] M. Egenhofer and A. Frank. Object-Oriented Databases: Database Requirements for GIS. In: International Geographic Information Systems Symposium: The Research Agenda, Crystal City, VA, November 1987.

[Egenhofer 1988] M. Egenhofer. Graphical Representation of Spatial Objects: An Object-Oriented View. Technical Report 83, Surveying Engineering Program, University of Maine, Orono, ME, July 1988.

[Egenhofer 1989a] M. Egenhofer and A. Frank. PANDA: An Extensible DBMS Supporting Object-Oriented Software Techniques. In: Database Systems in Office, Engineering, and Scientific Environment, Springer-Verlag, New York, NY, March 1989.

[Egenhofer 1989b] M. Egenhofer and A. Frank. Why Object-Oriented Software Engineering Techniques are Necessary for GIS. In: International Geographic Information Systems (IGIS) Symposium, Baltimore, MD, March 1989.

[Frank 1984] A. Frank. Requirements for Database Systems Suitable to Manage Large Spatial Databases. In: International Symposium on Spatial Data Handling, Zurich, Switzerland, August 1984.

[Frank 1988a] A. Frank. Multiple Inheritance and Genericity for the Integration of a Database Management System in an Object-Oriented Approach. In: K.R. Dittrich, editor, Advances in Object-Oriented Database Systems—Proceedings of the 2nd International Workshop on Object-Oriented Database Systems, Bad Münster am Stein-Ebernburg, F.R. Germany, Springer-Verlag, New York, NY, September 1988. Lecture Notes in Computer Science, Vol. 334.

[Frank 1988b] A. Frank. Requirements for a Database Management System for a GIS. Photogrammetric Engineering & Remote Sensing, 54(11), November 1988.

[Goldberg 1983] A. Goldberg and D. Robson. Smalltalk-80. Addison-Wesley Publishing Company, 1983.

[Härder 1985] T. Härder and A. Reuter. Architecture of Database Systems for Non-Standard Applications, (in German). In: A. Blaser and P. Pistor, editors, Database Systems in Office, Engineering, and Scientific Environment, Springer Verlag, New York, NY, March 1985. Lecture Notes in Computer Science, Vol. 94.

[Kjerne 1986] D. Kjerne and K.J. Dueker. Modeling Cadastral Spatial Relationships Using an Object-Oriented Language. In: D. Marble, editor, Second International Symposium on Spatial Data Handling, Seattle, WA, 1986.

[Liskov 1981] B. Liskov et al. CLU Reference. Lecture Notes in Computer Science, Springer Verlag, New York, NY, 1981.

[O'Brien 1986] P. O'Brien et al. Persistent and Shared Objects in Trellis/Owl. In: K. Dittrich and U. Dayal, editors, International Workshop in Object-Oriented Database Systems, Pacific Grove, CA, 1986.

[OOPSLA 1986a] OOPSLA '86—Object-Oriented Programming Systems, Languages, and Applications, Conference Proceedings. Portland, OR, September 1986.

[OOPSLA 1986b] OOPSLA '87—Object-Oriented Programming Systems, Languages, and Applications, Conference Proceedings. Orlando, FL, October 1986.

[Smith 1977] J.M. Smith and D.C.P. Smith. Database Abstractions: Aggregation. Communications of the ACM, 20(6), June 1977.

[Woelk 1987] D. Woelk and W. Kim. Multimedia Information Management in an Object-Oriented Database System. In: P. Stocker and W. Kent, editors, 13th VLDB conference, Brighton, England, 1987.