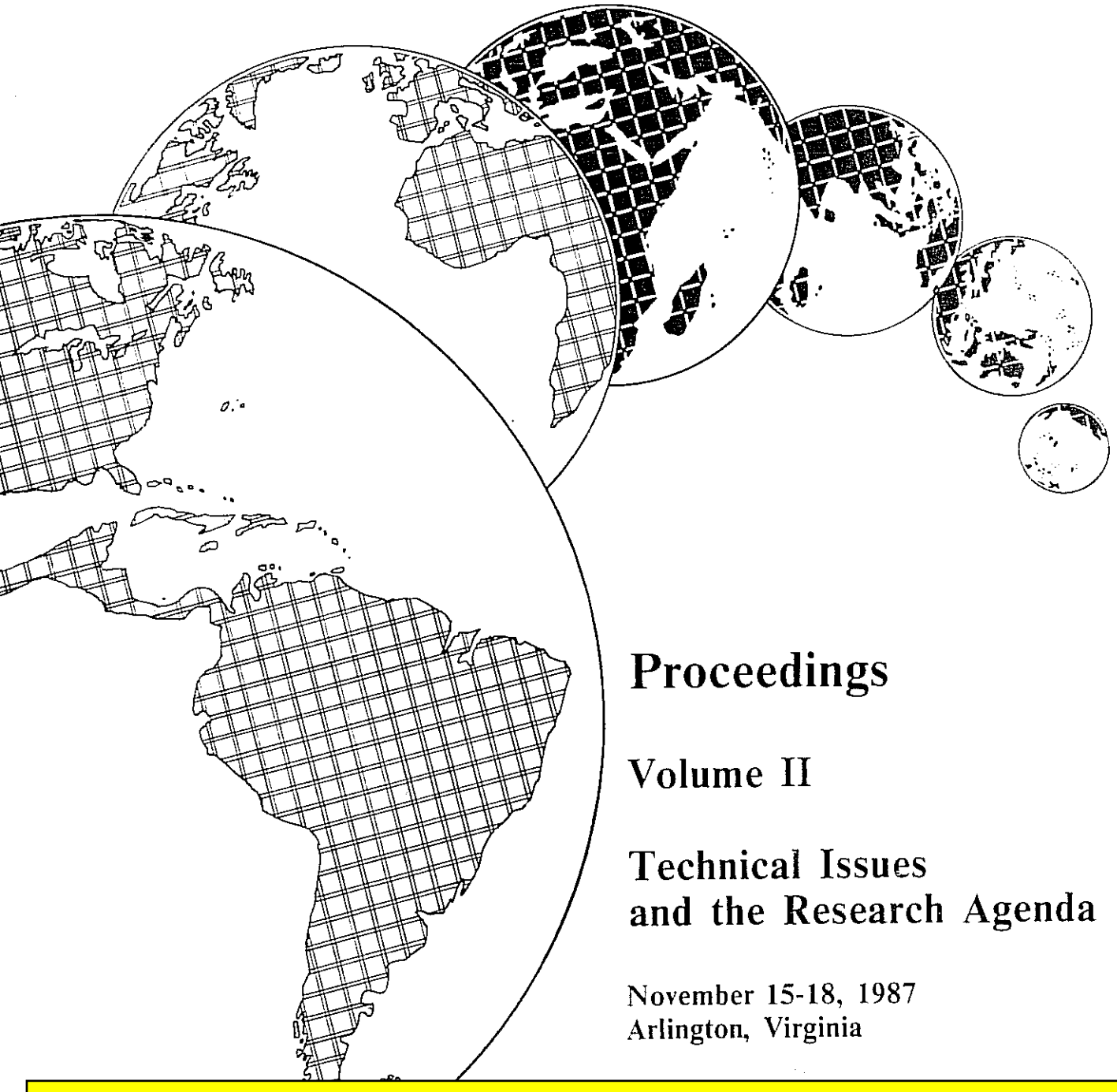
International Geographic Information Systems (IGIS) Symposium: The Research Agenda



Proceedings

Volume II

Technical Issues and the Research Agenda

November 15-18, 1987
Arlington, Virginia

Egenhofer, M.J., and A. U. Frank. "Object-Oriented Databases: Database Requirements for GIS." Paper presented at the International Geographic Information systems (IGIS) Symposium: The Research Agenda, Arlington, Virginia, USA, 15-18 November, 1987 1987.

Object-Oriented Databases: Database Requirements for GIS*

Max J. Egenhofer
Andrew U. Frank
Surveying Engineering Program
University of Maine
Orono, ME 04469, USA
MAX@MECAN1.bitnet
FRANK@MECAN1.bitnet

Abstract

Conventional database management systems have proven to be insufficient to model and perform non-standard applications such as spatial information systems, CAD/CAM, etc. The object-oriented approach overcomes some crucial deficiencies with refined methods for data abstraction and suitable tools to structure data allowing the usage of complex object types. For the implementation of an object-oriented database management system appropriate tools in programming languages are needed. A Geographic Information System (GIS) which treats spatial data asks for additional object-oriented extensions such as storage clusters to provide fast access on spatial objects, concurrent management of spatial and non-spatial data, and treatment of properties of spatial objects in query languages.

1 Introduction

Geographic information systems (GIS) contain substantial amounts of data which must be stored in computer readable and accessible form. Computer scientists have studied database management systems (DBMS) for several years and database management systems for commercial usage are currently available; however, several studies have shown that these systems are not suitable for non-standard applications, such as GIS, Land Information Systems (LIS) [Frank 1981] [Frank 1984],

*Work on these concepts was partially funded by a grant from NSF under No. IST-8609123

and CAD/CAM [Eastman 1980] [Stonebraker 1982]. Several experimental database management systems have been tried, but a complete system is not yet available commercially and additional research is needed.

During the last years, research in software engineering has promoted an object-oriented design method by which real world objects and their relevant operations are modelled in a program. This approach is most useful for application areas like GIS, because it naturally supports treatment of complex, in this case geometric, objects [Kjerne 1986]. Unlike conventional data models, an object-oriented design is more flexible to describe the complex data structures, primarily by refined techniques for conceptual modeling, such as generalization and inheritance.

Spatial information systems will benefit from the use of object-oriented databases in various forms:

- The system architecture will become clearer and easier to maintain so that software systems will have a longer life cycle.
- Programmers need not worry about aspects of the physical location where to store data; instead, a unified set of commands provides the functionality to store and retrieve data.
- By using a database, data is treated by its properties; by using an object-oriented database, these properties are logically combined to objects which can be often very complex.

The paper starts with a review on existing database systems and their deficiencies in serving as a suitable tool for modeling space related applications. In particular, the theory for data structuring techniques is compared to the tools in programming languages available to implement them. Following this chapter, an object-oriented data model is introduced which is built upon the three major abstraction methods of classification, generalization, and aggregation. The important concepts of hierarchical and multiple inheritance are explained. Software engineering aspects are investigated in chapter 4. Tools, such as programming languages and programming environments, are seen as especially crucial to implement object-oriented designs. In chapter 5, properties and requirements of object-oriented database management systems suitable for GIS are discussed. The implications of object-oriented structures to query languages are sketched. The paper closes with a summarization of the GIS requirements for a database management system.

2 Databases for Non-Standard Applications

Databases have become an accepted tool for storing data in readable and accessible form, and systems for CAD/CAM or spatial information (Geographic or Land

Information Systems) integrate a database system into a larger software system [Frank 1983].

Unlike conventional systems, spatial information systems deal with data describing the real world, and they combine several domains which have not been studied sufficiently. Think of a Geographic Information System in which a large variety of diverse tasks, such as

- sophisticated treatment of real-world geometry
- measurements of different resolution and accuracy
- uncertain measurements and attribute classifications
- legal aspects
- management of time series of veriable attributes
- representation of data in different generalization levels

are combined in a single system. This combination of difficult areas may be a reason why spatial information systems have been so slow to come and only similar, but more restricted applications are used successfully [Tripp 1987] [Beckstedt 1987].

Spatial information systems can benefit from using database management systems because these provide a unique form of storing and accessing structured data. The problems of low-level data management are removed from the programmer's and end user's responsibility, and data can be descibed by their properties, not their physical structure.

Unfortunately, existing commercial database systems are not sufficient for applications to engineering tasks, often called non-standard applications, such as CAD/CAM for VLSI design or cartographic and geographic information systems. Each of these areas struggle with the same kind of problem: they contain substantial amounts of 'real world' data including geometric aspects which must be managed by a computer. Their composition is too complex to be managed in conventional database systems efficiently. Interactive systems require a certain degree of performance, independent of the size of the data sets—a demand which cannot be fulfilled by existing systems.

2.1 Deficiencies of Conventional Database Systems

Members of the CAD/CAM-community have complained that there are currently no database systems which are appropriate for their demands [Buchman 1985] [Sidle 1980] [Udagawa 1984]. They state that standard database systems do not fulfill requirements which are crucial for engineering applications:

- Performance is unacceptable when a database is populated with large amounts of data [Wilkins 1984] [Härder 1985] [Maier 1986]. Dealing with spatial data, this deficiency is particularly visible during interactive graphic sessions.

- The treatment of complex objects [Lorie 1983], such as molecules in chemistry [Batory 1984], spatial objects in GIS/LIS [Frank 1984], or circuits in VLSI is not supported. Spatial objects, for example, should not be forced to be artificially decomposed in smaller parts.

- Appropriate mechanisms for data structuring [Johnson 1983], especially for structuring real-world data, are missing.

Most conventional database management systems are built on the relational model [Codd 1982]. In the relational model, data are organized in tables or relations. Columns of the tables are called attributes and all values in an attribute are elements of a common domain; rows are records, tuples, or relation elements.

While this concept is suitable for modeling commercial data, such as bank accounting, it is too simplistic for modeling data describing the real world. Geographic data cannot be modelled as strings, reals, and integers in table format—the necessary data types are more complex. One property of complex object types is that an object type can be part of another (more) complex object type. Data types for two-dimensional coordinates, for example, consist of an x- and y-value. A more complex object type includes some error value stating the accuracy of the coordinates. Such 'error-affected' coordinate types may be part of a point type which combines the coordinates with a number and a theme.

Moreover, the relational model lacks the powerful concept of recursion which is crucial for modeling complex situations such as spatial data and their subdivisions: areas can be decomposed into several sub-parts which themselves can be continuously decomposed further. For example, the area of a town is composed of several other areas, such as the house lots, streets, etc.

Only recently, the design of databases for spatial information system has become a research topic [Lipeck 1986] [Schek 1986] and only a few experimental database systems exist which pursue new concepts [Frank 1982a] [Dayal 1986] [Batory 1986].

In existing GIS software systems, a trend towards sophisticated software engineering methods [Aronson 1983] and architecture [Smith 1986] can be observed, stressing object-oriented concepts for geometric data handling [Herring 1987]; however, database management systems, and especially object-oriented ones, have not yet been incorporated into commercial GIS systems.

What are the technical reasons that database systems have failed to support complex non-standard situations? First, database requirements are not a hardware problem. It is commonly known that hardware currently develops much faster than software; however, faster and less expensive CPUs, larger hard disks, and more

memory do not overcome certain problems in database technology for engineering. Solutions which can be achieved by exploiting additional and faster hardware are not the problems which will be addressed in this paper. For example, new technologies for hard disks provide more storage capacity at less expensive costs, but the disk access has not become faster. This is important for database management systems which struggle with growing data collections which get too complex to be managed efficiently.

Rather than hardware, software engineering is an impediment for better-suited engineering databases. We claim that theoretical and conceptual problems are the subject of major improvements. For example, even with much faster access to storage devices, internal data structures organizing spatial data will be needed in order to provide adequate performance for range queries.

2.2 Deficiencies of Software Engineering Tools

Programming is still seen as an art in which the programmer can accomplish a goal by whatever means he believes to be suitable. The contrary is true: each program is a very formal piece of code following strict rules; programs must not only be compatible between hardware, but also 'compatible' among programmers, i.e., different programmers must be able to read and understand each others' code. By following strict rules and restrictions, these goals can be achieved. Programming languages tend towards offering a large variety of tools, while the opposite is needed, namely restrictions, such that several programmers come up with very similar solutions.

The other deficiency observed is that concepts in software engineering do not match with database models or are not suitable for them. Theory in software engineering provided some suitable techniques, such as the concept of abstract data types [Guttag 1977] [Parnas 1978] [Zilles 1984], in which each module encapsulates an object type with all its pertinent operations. Standard database systems, based on networks [CODASYL 1971] or the relational [Codd 1970] data model do not fit easily into this concept.

2.3 Deficiencies of Implementing Data Structures

A number of methods to capture more semantics in the data model have been proposed in the literature (for an overview and critique see [Brodie 1984] which contains an extensive list of references), but most of these methods have not been implemented, and none of them is readily available. Complex tasks require complex structuring tools. 'Complex', however, need not mean 'complicated': the support of data structuring must be powerful without sacrificing the ease of use, and it must be extensive without becoming excessive.

In the past, considerable efforts were made to enrich existing data models with

facilities to treat complex objects. 'QUEL as a Datatype' [Stonebraker 1983a] and ADT-INGRES [Stonebraker 1983] extend the relational model with features to define more complex types; DAPLEX [Shipman 1981] is a functional language which includes hierarchical relationships and transitive closure; the NF^2 model [Schek 1985] supports composite attribute types being tuples or relations performing a hybrid of relational and hierarchical data model. Recently, it was investigated, how geometry could be modelled by using these extended data models [Kemper 1987], and it was shown that only partial remedies are provided with these extensions.

3 An Object-Oriented Data Model

This chapter introduces the notation of objects and the abstraction tools. An entity of whatever complexity and structure can be represented by exactly one object in the database [Dittrich 1986], meaning that no artificial decomposition into simpler parts should be necessary due to some technical restrictions. Complex data types for large objects, such as an entire city (with its details about streets, houses, and their details such as owners, neighbors, etc.), do not overcome the problem of data structuring.

The object-oriented data model is built on the three basic concepts of abstraction [Brodie 1984a]: classification, generalization, and aggregation. Furthermore, inheritance describes how the properties of a class are derived from properties of related classes.

3.1 Classification

Classification can be expressed as 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 behaviour of its instances by describing the operators that can manipulate those objects [O'Brien 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*, *commercial building*, *street*, *park*, etc. A single instance, such as the house with the address '30 Grove Street', is an object of the corresponding object type, i.e. the particular object is an instance of the class *residence*. Operations and properties are assigned to object types, so for instance the class *residence* may have the property *number of bedrooms* which is specific for all residences. Similarly, the class *street* may have an operation to determine all adjacent parks.

In the implementation, classes translate to abstract data types or modules.

3.2 Generalization

Generalization is a well-known term in cartography for varying representations of objects according to the level of detail needed in a given scale. Similarly, generalization as an abstraction mechanism provides views in different levels of detail.

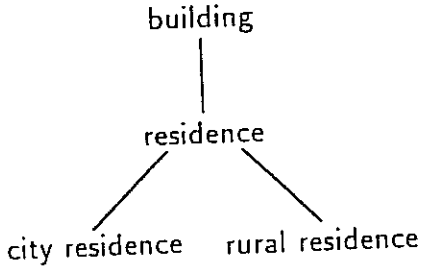
Several classes of objects which have some operations in common are grouped together into a more general *superclass* [Dahl 1968] [Goldberg 1983]. The converse relation of superclass, the *subclass*, describes a specialization of the superclass. *Ancestor* and *descendant* are often used as equivalent terms for superclass and subclass. Subclass and superclass are related by an *is_a* relation. For example, the object type *residence* is a *building*; *residence* is a subclass of *building*, while *building* is its superclass.

3.2.1 Inheritance

The properties and methods of the subclasses depend upon the structure and properties of the superclass(es). Inheritance defines a class in terms of one or more other classes. 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 from the superclass. Inheritance is transitive propagating the properties from one superclass to all related subclasses, to their subclasses, etc. This concept is very powerful, because it reduces information redundancy [Woelk 1987]. For example, *building* shares properties and operations with *residence*, such as having an *area* and being *neighbor* of some other building. The class *residence* inherits all operations from its superclass *building* without the need to redefine them explicitly. Additional properties, such as the *number of bedrooms*, are specific for the *residence*. Specializations of *residences*, such as *rural residences* and *city residences* inherit the specific properties of the *residences*, i.e. *number of bedrooms*, and by transitivity the properties of the super-superclass *building*, i.e., *area* and *neighbor*.

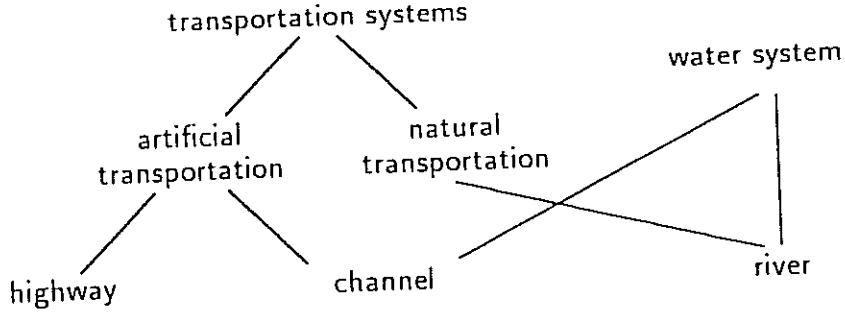
Operations of the superclass are compatible between objects of the superclass and subclass. Every operation on an object of a superclass can be carried out on the subclass as well; however, operations specifically defined for the subclass are not compatible with superclass objects. For example, *neighbor* is an operation of the superclass *building*, and it is thus compatible with objects of the type *residence*. On the other hand, the operation *number of bedrooms* is specific fo the subtype *residence* and thus not applicable for objects of the superclass *building*.

Hierarchies: Inheritance can be strictly hierarchical as in the example of *building*, *residences*, etc. Hierarchical inheritance implies that each subclass belongs only to a single group of hierarchies; one class cannot be part of several distinct hierarchies, i.e., in hierarchical generalization, a class can have only one direct superclass.



Multiple Inheritance: The structure of a strict hierarchy is an idealized model and fails most often 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, class lattices [Woelk 1987] with acyclic directed graphs are more suitable. This concept, allowing a multitude of distinct superclasses for a single class, is called multiple inheritance [Nguyen 1986].

For example, *highways* and *channels* are *artificial transportation ways*, while *rivers* are *natural transportation systems*; however, *channels* and *rivers* both belong to another hierarchy, the *water system* as well. So, *channel* and *river* participate in both *water systems* and *transportation ways*, two distinct hierarchies which cannot be compared with each other.



3.3 Aggregation

Several objects can be combined to form a semantically higher-level object where each part has its own functionality. This is different from the way generalization is defined: operations of aggregates are not compatible with operations on parts.

Aggregation establishes a relation which is often called a 'part-of'-relation since aggregated classes are 'parts of' the aggregate. For example, the class *county* is an aggregate of all related *settlements*, *forests*, *lakes*, *streets*, etc.

3.3.1 Dependencies Among Values of Different Object Types

Complex objects often do not own independent data, but properties which rely upon values of other objects. In GIS and LIS, for example, a large amount of attribute values is propagated from one level of abstraction to another. When combining local and regional data, this concept must be used to pursue the dependencies among data of different levels of resolution [Egenhofer 1986]. The population of a county, for example, is the sum of the population of all related settlements; therefore, the property *population* of the aggregate *county* is derived by adding all values of the property *population* owned by the class *settlement*.

While inheritance is the propagation of operations, functional dependencies describe how *values* of one class are derived from values of another class. Often, a value is directly propagated from the property of one class to another property of a different class. In the object-oriented model, objects may have properties with values which rely on values of other objects. Dependent values must be derived.

If more than a single value contributes to the derived value, the combination of the values must be described by a function. Common operations are minimum, maximum, sum, average, and weighted average.

3.4 Tools for Modeling

Even for relatively small models of a mini-world, the structure can become too confusing to be presented with pure alphanumeric means. Graphical methods have proven to be suitable tools for a better and clearer understanding of data structures; representations such as entity-relationship diagrams [Chen 1976] are essential tools in an object-oriented design. By using the power of graphical representation, modeling becomes more clear and understandable compared to pure alphanumerical descriptions.

The concept of multiple inheritance must be incorporated into the entity-relationship model by using some simple graphical means.

4 Software Engineering Aspects

A database is only one part in the large effort to produce a CAD/CAM system, a land information system, or any of the other non-standard applications. It is thus necessary that database methods and techniques fit well within a software

engineering environment. Theoretically founded methods have proven to be well-suited for modularization based on abstract data types.

4.1 Abstract Data Types

The software engineering method for abstraction and data structuring is based on the work on formal specifications using abstract data types. An abstract data type (ADT) or a multi-sorted algebra is a mathematical structure which fully defines the behaviour of objects, i.e., the semantics of the object-type and its operations. Abstract data types are specified as an algebra describing what sorts of objects (types) are dealt with and what kinds of operations they are subject to. A set of axioms determines the effects of the operations. It is up to the designer to assure that the sorts and their operations form a reasonable and meaningful object.

Abstract data types can be combined in layers, where higher-level abstract data types are first described independently (specifications) and it is then shown how this behaviour can be achieved using other, hierarchically lower abstract data types (called an 'abstract implementation' [Olthoff 1985] [Frank 1986]). Such abstract implementations can be formally checked for correctness, by proving that the axiomatically specified behaviour of the upper abstract data type follows from the abstract implementation and the axiomatically specified behaviour of the lower abstract data type.

Abstract data types are very useful in the design of large systems. They allow a much more comprehensive and complete way of specifying routines together with their axioms than an extensive programming language does.

4.2 Object-Oriented Programming

The programming language must support the abstraction methods. The development of object-oriented databases has had problems, mostly because suitable software tools are missing or old and inappropriate tools have been used. Certain programming tools must be available to implement object-oriented databases and data structures.

- In order to implement complex objects, language tools, such as RECORD structures in Pascal [Jensen 1978], Ada [Ada 1983], etc., are needed to create user-defined object types. For example, in a spatial information system a large amount of the data are points with coordinates, either 2-dimensional or 3-dimensional. By combining the coordinates to a RECORD, a complex data type, such as a `pointType`, can be created.

```
TYPE pointType = RECORD
                    x, y, z: coordType;
                 END;
```

- In object-oriented programming, objects consist of a type definition and a collection of operations. Objects can be only accessed or manipulated by using these operations. For example, the object type `pointType`, as introduced above, has a specific operation to shift a point in x-, y-, or z-direction (translation); the user can access a point only via the interface of the `pointType`.

  Object-oriented programming needs (1) tools to tie types and their operations closely together (modularisation), and (2) methods to hide internal parts of the routines from unauthorized external usage (encapsulation). Modula-2 [Wirth 1982], Smalltalk-80 [Goldberg 1983], and C++ [Stroustrup 1986] are examples for languages which support modularization and encapsulation. Encapsulation provides implementation-independent code outside of a module.

- Genericity [Cardelli 1985] has proven to be a suitable and powerful method to reduce redundant definitions of ADTs. A generic type or object is a definition which is a backbone for a series of detailed and specified definitions. A *pair*, for instance, is a generic object type which combines two objects; *pair* can be applied to two integers forming a *pair of integers*. Operations which are common for all *pairs*, such as *equality*, are defined only once, while their implementation might vary from type to type.

  Programming code is considerably reduced by using generic types, because operations which apply to all instances are coded only once.

- Only recently, inheritance has been recognized as a crucial issue in object-oriented approaches. Two categories of inheritance are identified, hierarchical (or straight) and multiple inheritance.

  Hierarchical inheritance deals with a strict structure in which for each child only a single direct ancestor exists. Some programming languages, such as Smalltalk-80 and C++, support straight inheritance, other languages with variant RECORD structures, such as Pascal or Ada, can at least simulate hierarchical inheritance of types, but not of operations. For example, two-dimensional and three-dimensional coordinates are both representations for points. From this superclass, they both inherit operations such as calculating the distance between two points. The implementation of these operations may be different, however, from outside they look the same. With a variant RECORD structure, the inheritance can be implemented as follows:

```
TYPE pointType = RECORD
                   CASE dimension: dimensionType of
                      twoD: (x, y: coordType);
                      threeD: (x, y, z: coordType);
                 END;
```

Multiple inheritance allows one object to be part of several distinct hierarchies. Only a few languages, such as some LISP dialects (CommonObjects [Snyder 1986], Zeta Lisp [Weinreb 1981]), ObjectLOGO [Davidson 1987], and Trellis/Owl [O'Brien 1986] allow the definition of multiple inheritance.

These concepts for object-oriented programming do not include a message-passing paradigm [Goldberg 1983]; message-passing is based on passive objects which can receive messages directing what actions can be executed on an object. Message-passing is often cited as necessary for an object-oriented design; in our opinion it is not an essential feature, but certainly a desirable and helpful paradigm. It was outlined that message-passing is rather a pedagogical than a semantical difference to conventional routine calls, and any procedure call in an Algol-like language could be seen as message-passing [Storm 1986].

Object-oriented programming of object-oriented applications depends on the choice of the programming language; only a few languages support object-orientation sufficiently. We claim that object-oriented applications cannot be achieved without object-oriented tools and concepts.

4.3 Support System

Database systems are large software systems which must be embedded in some environment supporting development and maintainance. Especially the latter is crucial for the life cycle of a system. Such an environment must be tailored to the object-oriented approach by automatically propagating object definitions to other objects, controlling which object uses which other objects, and which objects are used elsewhere.

In software engineering, abstract data types may be represented by modules which encapsulate a type and its operations. Facilities such as packages and use clauses in Ada or modules and import commands in Modula-2 allow the programmer to easily translate abstract data types into executable code. For other languages with capabilities for separate compilation of modules, these concepts can be applied by using a precompiler which propagates the ADT-definitions to the modules where they are used.

5 Database Management Systems Tailored to GIS

This chapter investigates how the object-orientation reflects in the architecture of a database management system for GIS. Ideally, an object-oriented database consists of subsystems which can be added or exchanged to support specific tasks [Batory 1986]. Before investigating the GIS-specific parts of a tailored object-oriented database, subsystems of standard database systems are compiled which can be adopted from an object-oriented database.

5.1 Applicable Subsystems of a Conventional Database

5.1.1 Disk Storage Subsystem

A database must provide persistent storage for objects modelled in the information system. Due to the large size of the data sets, use of magnetic or optical disk systems for permanent storage is required.

A storage subsystem must provide operations to store and retrieve data. The storage subsystem does not know anything about other operations owned by the objects or their internal structure. Disk access is expensive and improvements in performance can be achieved by reducing the number of disk accesses by buffering the storage elements.

5.1.2 Multi-User Facilities

In multi-purpose information systems, a database is shared by a large variety of users who often want to access the same data in parallel. The known straight forward techniques are sufficient as long as concurrent users only access the data without changing them. This is, however, usually not appropriate to accomodate users and their needs. Most organizations cannot restrict update operations to a single user without severe distortion of their flow of work.

A database management system must provide control mechanisms in order to guarantee concurrent access for multiple users to all data and to prevent users from accessing data during inconsistent states, i.e., the database management system guarantees that no user sees the preparative stages of a change until the change is completed and made visible to all other users. This is done by a dedicated subsystem, the so-called transaction manager. A transaction is a sequence of operations by one user which transforms the database from one consistent state into another one, such that outside of a transaction a consistent image of the mini-world exists at all times. A transaction can either be committed, i.e., all modifications will be made visible, or aborted, i.e., none of the modifications started during the transaction will become effective. Modified data are isolated during the transaction preventing other users from uncontrolled access. The transaction subsystem can fulfill with essentially

the same mechanisms other goals as well. It is usually assumed that a transaction system will maintain all the consistency of the database across hardware or software malfunctionings, and prevents loss of data. A transaction has the following four classical properties (ACID) [Härder 1985]:

- Atomicity states that in a transaction either all modifications become effective, or no change at all are made in the data set.
- Consistency is guaranteed before and after the transaction.
- Isolation preventes from unauthorized access during a transaction.
- Durability keeps comitted transactions permanently.

5.1.3 Distributed Systems

Central databases are important resources and access to them must be available for many operations in parallel and at the same time. For organizational reasons, large databases are often not centralized, but distributed over several computer systems. This may improve their availability during local hardware failures, reduce data transfer cost, etc. In our opinion, support for distribution should be built into this layer of the subsystem.

5.2 Specific GIS requirements

In this chapter it is proposed which additional concepts must be integrated into an object-oriented database management system, such that it would fulfill the high expectations and requirements on a persistent management system for very large spatial data sets.

5.2.1 Performance Enhancements for Spatial Data

Large spatial data collections should be managed without artificially subdividing them into several user-visible map sheets. Ideally, the user should have a single database covering the entire area of his or her application; however, the performance penalties are serious when very large amounts of spatial data should be managed. Interactive graphic representations of spatial data with map output require especially adequate performance.

Generally, database performance depends upon the number of disk accesses needed to retrieve the data. Conventional systems do not pay attention to the distribution of spatial data and store them 'as they are collected'. This treatment is not appropriate for spatial data collections, because the access on a set of data to

be drawn as a map will take a long time due to too many disk accesses. The larger the data collection grows, the slower the system will perform.

Access on spatial data must be enhanced by integrating spatial storage clusters [Frank 1983a]. Such techniques take advantage of the spatial distribution of data and store them such that spatial neighbors are neighbors on the storage device. It is assumed that (1) data used on one drawing are likely to be re-used soon for further display or interactive modification, and (2) not only a single object, but several adjacent objects will be used on a drawing. Spatial access is organized such that not a single instance, but a set of adjacent data is read at a time from a storage device and afterwards kept in main memory for future access. This buffering scheme needs to be organized according to some rules, e.g., a least-frequently-used strategy. In combination with spatial access methods, fast response time can be guaranteed essentially independent from the size of the data collection.

It is worthwile to study how storage structures for heterogeneously distributed spatial data can be improved such that access on less frequent objects performs well [Egenhofer 1987].

5.2.2 Complex Geographical Objects

The complexity of geographical objects, primarily spatial objects, requires methods to define and use appropriate data types and operations. The object-oriented model is tailored to this task. Data structures for recursive object definitions, such as areas being subdivided in other areas, and transitive closure operations are necessary.

Geometric and non-spatial data should not be forced to be physically separated from each other or managed in different types of databases. Instead, both categories must be integrated in one single system. This requires that

- user-defined, complex types can be stored in the database.
- a buffering schema is incorporated to reduce physical disk access; the geometric neighborhood should be exploited for a system with primarily graphical output.

Since objects can be more complex, their transactions will last longer and may be nested. A transaction concept is necessary which goes beyond the classical properties atomicity, consistency, isolation, and durability [Härder 1985].

5.3 Object-Orientation in Query Languages

Query languages have been disregarded for a long time, while they are one of the most crucial issues of an object-oriented database. Query languages are expected to benefit from the object-oriented approach by providing object operations at the user interface. Standard query languages for conventional database mangement

systems, such as SQL [Chamberlin 1976], QUEL [Stonebraker 1976], or Query-By-Example [Zloof 1977], are not suited to deal with spatial data, because they do not include the specific properties of spatial objects. Non-standard database management systems must be furnished with query languages which support the treatment of complex objects including their specific properties. Proposals have been made to develop completely new query languages which are tailored to their specific application [Frank 1982]. A database system for GIS which is able to manage spatial objects efficiently is incomplete if it does not support spatial data and their properties in the query language.

We do not believe that natural language interfaces overcome the problems in query languages for non-standard applications and rather promote structured, object-oriented query languages.

5.3.1 Support of Spatial Relations

In order to treat spatial objects in query languages, properties must be included which are specific for spatial data. Typical properties among spatial objects are topological and metrical relations describing neighborhood, inclusion, distance, and direction. Object-oriented spatial relations must be dimension-independent, i.e., such that they can be applied to any spatial object. 'Disjoint', for example, is a relation which can hold for any two spatial objects (two points, two lines, two areas, two volumes, but also a point and a line, a point and an area, etc.).

Currently, we are investigating how the syntax of a standard query language must be extended such that it includes an adequate treatment of spatial objects [Egenhofer 1987a].

5.3.2 High-Level Object-Oriented Operators

Conventional query languages address subparts of objects explicitly and compare them with standardized operations. This is the reason for the 'artificial' style of queries such as

```
RETRIEVE all roads
WITH road.width > 12
```

In natural language, humans would combine the operation ¿ with the type of the object-subpart to some meaningfull term such as *broader than*. The object-oriented approach pursues this concept since the structure of the object and its operations are closely combined. Consequently, query languages must apply this concept, too, and express relations between objects such as

```
RETRIEVE all roads
WHICH ARE broader than 12
```

By combining object parts with operators and mapping them onto high-level operators, an object-oriented view of operators can be presented.

5.3.3 Graphical Display

Conventional query languages deal only with alphanumerical data, consequently, the graphical display for spatial objects cannot be specified. The specification is more complex for graphical output than for alphanumerical output where the sequence of columns in a table is described. Methods are needed to specify colors, patterns, and symbols.

Furthermore, interactive sessions with graphical output need mechanisms for user feedback, such as input via mouse on a drawing. The concept of direct manipulation on objects is more advanced and object-oriented than conventional input methods, because it does not reference a specific value, but screen coordinates which correspond to some object.

The output system directed by a query language has some other interesting problems, such as the selection of context [Frank 1982] which is dispalyed to make the graphical output understandable. Other proposals investigate, how methods from artificial intelligence like LOBSTER [Frank 1984a] can be incorporated into an intelligent interface for query languages.

6 Conclusion

It has been investigated how object-oriented database management systems can serve as suitable tools for spatial information systems. It was outlined that current database technology is not sufficient for the specific tasks when dealing with large amounts of spatial data. Recently, research in non-standard database environments promoted an object-oriented model which looks promissing to overcome some problems that make conventional database management systems unsuitable, such as the lack of modeling power to adequatly describe complex objects and the unacceptably slow performance of current implementations. This paper presented an object-oriented data model based on the abstraction concepts of classification, generalization, and aggregation. Hierarchical and multiple inheritance are crucial for modeling complex object types. The implementation of object-oriented data structures was investigated from the software engineering aspects (object-oriented programming languages).

With respect to geographic applications, several components of an object-oriented database system were identified which are mandatory for treating very large collections of spatial and non-spatial data. In particular, data structures for complex spatial data types, internal storage clusters for fast access on spatial data, and concurrent treatment of spatial and non-spatial objects must be incorporated in the database management system. Finally, the impact of the object-oriented design on query languages was investigated. Query languages for spatial information systems need high-level relations and operators for spatial objects, methods for specifying graphical display, and language tools for direct manipulation.

Acknowledgement

Thanks to Doug Hudson and Jeff Jackson who helped prepare this paper.

References

[Ada 1983] United States Department of Defense. Reference Manual of the ADA Programming Language. Springer Verlag, New York (NY), 1983.

[Aronson 1983] P. Aronson and S. Morehouse. The ARC/INFO Map Library: A Design for a Digital Geographic Database. In: Auto-Carto VI, 1983.

[Batory 1986] D.S. Batory. GENESIS: A Project to Develop an Extensible Database Management System. In: International Workshop in Object-Oriented Database Systems, Pacific Grove (CA), 1986.

[Batory 1984] D.S. Batory and A.P. Buchmann. Molecular Objects, Abstract Data Types, and Data Models: A Framework. In: 10th VLDB conference, Singapore, 1984.

[Beckstedt 1987] M. Beckstedt. Clark County Land Parcel Mapping System Pilot Study. In: Proceedings AM/FM International Automated Mapping/Facilities Management Conference X, Snowmass (CO), 1987.

[Brodie 1984] M.L. Brodie et al., editors. On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages. Springer Verlag, New York (NY), 1984.

[Brodie 1984a] M.L. Brodie. On the development of data models. In: M.L. Brodie et al., editors, On Conceptual Modelling, Perspectives from Artificial

Intelligence, Databases, and Programming Languages, Springer Verlag, New York (NY), 1984.

[Buchman 1985] A.P. Buchmann and C.P. de Celis. An Architecture and Data Model for CAD Databases. In: 11th VLDB conference, Stockholm, 1985.

[Cardelli 1985] L. Cardelli and P. Wegner. On Understanding Type, Data Abstraction, and Polymorphism. ACM Computing Surveys, 17(4), 1985.

[Chamberlin 1976] D.D. Chamberlin et al. SEQUEL 2: A Unified Approach to Data Definiton, Manipulation, and Control. IBM Journal of Research and Development 20(6), 1976.

[Chen 1976] P.S.S. Chen. The Entity-Relationship model: towards a unified view of data. ACM Transactions on Database systems, 1(1), March 1976.

[CODASYL 1971] Data Base Task Group report. CODASYL. Technical Report, New York (NY), April 1971.

[Codd 1982] E.F. Codd. Relational data base: a practical foundation for productivity. Communications of the ACM, 25(2), February 1982.

[Codd 1970] E.F. Codd. A relational Model for large shared data banks. Communications of the ACM, 13(6), June 1970.

[Dahl 1968] O.-J. Dahl et al. SIMULA 67 Common Base Language. Technical Report, Norwegian Computing Center, Oslo, 1968.

[Davidson 1987] L.J. Davidson et al. ObjectLogo. Coral Software Corporation, Cambridge (MA), 1987.

[Dayal 1986] U. Dayal and J.M. Smith. PROBE: A Knowledge-Oriented Database Management System. In: M.L. Brodie and J. Mylopoulos, editors, On Knowledge Base Management Systems: Integrating Artificial Intelligence and Database Technologies, Springer Verlag, New York (NY), 1986.

[Dittrich 1986] K. Dittrich. Object-Oriented Systems: the Notation and the Issues. In: International Workshop in Object-Oriented Database Systems, Pacific Grove (CA), 1986.

[Eastman 1980] C. Eastman. System facilities for CAD databases. In: 17th Design Automation Conference, Minneapolis, 1980.

[Egenhofer 1987] M. Egenhofer. Managing Spatial Storage For Structured Data. In: Congres Conjoint Carto-Québec / A.C.C. / C.C.A., Québec, 1987.

[Egenhofer 1987a] M. Egenhofer. An Extended SQL Syntax For Treating Spatial Objects. In: Y.C. Lee, editor, Proceedings of the Second International Seminar on Trends and Concerns of Spatial Sciencies, Fredericton (NB), 1987.

[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, 1986.

[Frank 1986] A. Frank and W. Kuhn. A Provable Correct Method for the Storage of Geometry. In: Second International Symposium On Spatial Data Handling, Seattle (WA), 1986.

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

[Frank 1984a] A. Frank. Extending a Database with Prolog. In: L. Kerschberg, editor, Proceedings of the First International Workshop on Expert Database Systems, Kiawah Island (SC), October 1984.

[Frank 1983] A. Frank. Data Structures for Land Information Systems—Semantical, Topological, and Spatial Relations in Data of Geo-Sciences. PhD thesis, Swiss Federal Institute of Technology, Zurich (Switzerland), 1983.

[Frank 1983a] A. Frank. Problems of Realizing LIS: Storage Methods for Space Related Data: The Field Tree. Technical Report 71, Swiss Federal Institute of Technology, Zurich (Switzerland), 1983.

[Frank 1982] A. Frank. MAPQUERY—Database Query Language for Retrieval of Geometric Data and its Graphical Representation. In: D. Bergeron, editor, Proceedings SIGGRAPH '82, ACM Computer Graphics, July 1982.

[Frank 1982a] A. Frank. PANDA—A Pascal Network Database System. In: G.W. Gorsline, editor, Proceedings of the Fifth Symposium on Small System, Colorado Springs (CO), 1982.

[Frank 1981] A. Frank. Applications of DBMS to Land Information Systems. In: C. Zaniolo and C. Delobel, editors, Seventh International Conference on Very Large Data Bases, Cannes (France), 1981.

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

[Guttag 1977] J. Guttag. Abstract Data Types And The Development Of Data Structures. Communications of the ACM, June 1977.

[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 (NJ), 1985.

[Herring 1987] J. Herring. TIGRIS: Topologically Integrated Geographic Information Systems. In: N.R. Chrisman, editor, Eighth International Symposium on Computer-Assisted Cartography, Baltimore (MD), 1987.

[Jensen 1978] K. Jensen and N. Wirth. Pascal User Manual and Report. Springer-Verlag, New York (NY), 1978.

[Johnson 1983] H.R. Johnson et al. A DBMS Facility for Handling Structured Engineering Entities. In: ACM Engineering Design Applications, 1983.

[Kemper 1987] A. Kemper and M. Wallrath. An Analysis of Geometric Modeling in Database Systems. ACM Computing Surveys, 19(1), March 1987.

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

[Lipeck 1986] U.W. Lipeck and K. Neumann. Modelling and Maipulation of Objects in Geoscientific Databases. In: 5th International Conference on the Entity-Relationship Approach, Dijon, 1986.

[Lorie 1983] R. Lorie and W. Plouffe. Complex Objects and Their Use in Design Transactions. In: ACM Engineering Design Applications, 1983.

[Maier 1986] D. Maier. Why Object-Oriented Databases Can Succeed Where Others Have Failed. In: International Workshop in Object-Oriented Database Systems, Pacific Grove (CA), 1986.

[Nguyen 1986] V. Nguyen and B. Hailpern. A generalized Object Model. SIGPLAN Notices, 21(10), October 1986.

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

[Olthoff 1985] W. Olthoff. An Overview on ModPascal. SIGPLAN Notices, 20(10), October 1985.

[Parnas 1978] D.L. Parnas and J.E. Share. Language facilities for supporting the use of data abstraction in the development of software systems. Technical Report, Naval Reasearch Laboratory, Washington (DC), 1978.

[Schek 1986] H.-J. Schek and W. Waterfeld. A Database Kernel System for Geo-scientific Applications. In: Second International Symposium on Spatial Data Handling, Seattle (WA), 1986.

[Schek 1985] H.-J. Schek. Towards a Basic Relationsal NF^2 Algebra Processor. In: Conference on Found. Data Org. (FODO), Kyoto, 1985.

[Shipman 1981] D. Shipman. The functional data model and the data language DAPLEX. ACM Transactions on Databse Systems, 6(1), March 1981.

[Sidle 1980] T.W. Sidle. Weakness of Commercial Database Management Systems in Engineering Applications. In: 17th Design Automation Conference, 1980.

[Smith 1986] T. Smith et al. KBGIS-II: A Knowledge-Based Geographic Information System. Technical Report TRCS86-13, Dept. of Computer Science, University of California, Santa Barbara (CA), May 1986.

[Snyder 1986] A. Snyder. CommonObjects: An Overview. SIGPLAN Notices, 21(10), October 1986.

[Stonebraker 1983] M. Stonebraker et al. Application of abstract data types and abstract indices to CAD databases. In: Proceedings of ACM SIGMOD Conference on Engineering Design Applications, San Jose (CA), 1983.

[Stonebraker 1983a] M. Stonebraker et al. QUEL as a datatype. Technical Report UCB/ERL M83/73, University of California, Berkley (CA), 1983.

[Stonebraker 1982] M. Stonebraker and A. Guttman. Using a Relational Database Management System for CAD data. IEEE Database Engineering, 5(2), June 1982.

[Stonebraker 1976] M. Stonebraker et al. The design and implementation of INGRES. ACM Transactions on Database systems, 1(3), September 1976.

[Storm 1986] R. Storm. A comparison of the Objet-Oriented and Process Paradigms. SIGPLAN Notices, 21(10), October 1986.

[Stroustrup 1986] B. Stroustrup. The C++ Programming Language. Addison-Wesley Publishing Company, 1986.

[Tripp 1987] R.E. Tripp. AM/FM Progress at Carolina Power & Light Company. In: Proceedings AM/FM International Automated Mapping/Facilities Management Conference X, Snowmass (CO), 1987.

[Udagawa 1984] Y. Udagawa and T. Mizoguchi. An Extended Relational Database System for Engineering Data Management. IEEE Database Engineering, 7(2), June 1984.

[Weinreb 1981] Lisp Machine Manual. Symbolics, Inc., 1981.

[Wilkins 1984] M.W. Wilkins and G. Wiederhold. Relational and Entity-Relationship Model Databases and VLSI Design. IEEE Database Engineering, 7(2), June 1984.

[Wirth 1982] N. Wirth. Programming in Modula-2. Springer-Verlag, New York (NY), 1982.

[Woelk 1987] D. Woelk and W. Kim. Multimedia Information Management in an Object-Oriented Database Systm. In: 13th VLDB conference, Brighton, 1987.

[Zilles 1984] S.N. Zilles. Types, algebras, and modelling. In: M.L. Brodie et al., editors, On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages, Springer Verlag, New York (NY), 1984.

[Zloof 1977] M.M. Zloof. Query-by-Example: a database language. IBM Systems Journal 16(4), 1977.