Reduced Data Model for Storing and Retrieving Geographic Data

Andrew U. Frank

Dept. of Geoinformation and Cartography
Technical University Vienna
frank@geoinfo.tuwien.ac.at

Abstract. The 'industry-strength' data models are complex to use and tend to obscure the fundamental issues. Going back to the original proposal of Chen for Entities and Relationships, I describe here a reduced data model with Objects and Relations. It is mathematically well founded in the category of relations and has been implemented to demonstrate that it is viable. An example how this is used to structure data and load data is shown.

1 Introduction

Numerous groups investigate how to structure data for use in Geographic Information Systems (GIS); they use tools prepared for 'real life' applications using the tried and trusted methods of the past—mostly object concepts based on languages like C++ or Java and the relational data model. Design is typically using UML despite the known lack of formal definition. These methods and tools are useful to build GIS applications but they are not appropriate as foundations for GIScience research. Shortcomings have been identified years ago, but convincing solutions are still missing (Dijkstra 1976). This paper addresses the fundamental question of structuring data for permanent storage and proposes a reduction of data models to *Objects* and *Relations* with 6 operations. Building the program to store and retrieve the data for a graph shows that the reduction is viable and has preserved the essence.

Current GIScience research focuses—among other things—on

- adding support for temporal data and processes to the GIS (Langran 1992);

- drive the design of a geographic application from an ontological perspective (Fonseca and Egenhofer 1999) and incorporating the results of the 'GIS ontology discussion' (Mark, Smith et al. 2000; Frank 2003);
- integrating data from different sources (Nyerges 1989; Bishr 1998);
- use an agent oriented design and construct agent based simulations (Ferber 1998; Bittner 2001; Raubal 2001);
- improve usability by connecting the user interface design with the ontological analysis (Achatschitz 2005).

For these and other similar research efforts, the construction of a repository for the data used in the application code is very time consuming. The performance orientation of the 'industry-strength' systems impose restrictions and limitations, and—last but not least—the theoretical bases on which these industry systems are built are in conflict with the research goals. It is hard to build new tools if the machinery to build them is imposing assumptions that we try to overcome!

Over the past years we have built many programs to store the data necessary for experiments in handling spatial data. We learned to reduce the data model to a minimum, which gives maximum flexibility and the least restrictions. This paper describes a usable set of functions for storage and retrieval of data for experimentation with advanced concepts of geographic data handling, especially research focused on temporal data, ontology, and integration. It achieves:

- programs are structured around objects,
- a data structure and the data collection can be changed and extended,
- modules describing an object type can be freely combined.

A number of limitations in programming languages had to be overcome to translate a clean design founded in a mathematical theory into executable code that validates the design. The code is now ready for use by others and made available from our CVS repository (gi07.geoinfo.tuwien.ac.at/CVSroot/RelDB).

Before starting with a new design, I review in section 2 the achievements in Computer Science that are used for GIS, explain their rationale and the shortcomings relevant for GIS. Section 3 then restricts the various concepts employed to two fundamental ones: objects and binary relations. Section 4 explains the bundling of data description and object code. Section 5 gives examples with code and the concluding section touches on some of the limitations of the current code compared to full object-orientation or database concepts; it describes directions for future work as well.

2 Issues with the Current Technology

The current technology used to build applications that manage large collections of geographic data, gives the stability and the performance necessary for the GI industry, but the resulting systems are complex to install and manage. The flexibility to fulfill novel requirements or to adapt to changing situations is limited and numerous restrictions apply. New theoretical foundations, especially the development of the theory of programming languages, allow new approaches and we must rethink our design choices (Chen 2006). In this section I argue for the specific choices behind the reported solution.

2.1 Database concept

To consider the data as a resource of an organization was a first step towards the information age in the 1970s. Data management became a central task and generalized database management software developed. The three schema model separates the stable description of the data stored from the often changing application programs. Unfortunately, the ability to describe the data in the schema is limited and many important aspects of data descriptions are dispersed in the application programs.

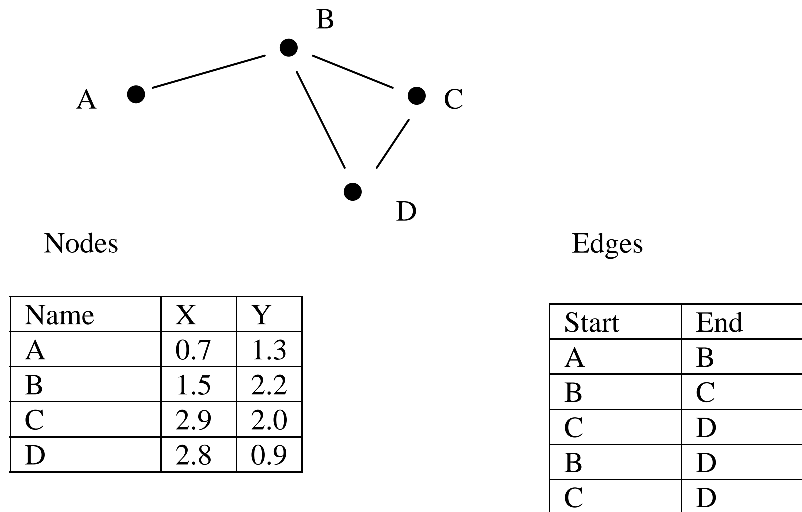
Computer science research investigates how the data descriptions can be embedded with the data using the XML language (Ceri, Fraternali et al. 2000); ontology languages try to include more from the data description (Dieckmann 2003) but seem to lack abilities to describe processes.

2.2 Relational data model

Codd invented the relational data model to facilitate the management of administrative data, specifically data that was ordinary represented on paper as tables. The relational data model was very successful because it gave a formal base to the intuitive and widely used concept of tables; Codd defined a small number of operations on tables that are closed and 'relationally complete' (Codd 1982). For applications, the SQL query language presented a human readable interface. The relational data model is 'value based', which means that all operations compare just values and there is no concept of objects in the formal model.

Härder observed already 20 years ago that geographic data, similar to data from CAD and other applications that relate to space and time, require for their representation multiple tables, connected by common values of

their identifiers (Härder 1986). The representation of a graph—which is at the core of most spatial applications, e.g., transportation, cadastral data—can become easily inconsistent by changing the names of nodes and not maintaining the corresponding edge data (Figure 1):



Name	X	Y
A	0.7	1.3
B	1.5	2.2
C	2.9	2.0
D	2.8	0.9

Start	End
A	B
B	C
C	D
B	D
C	D

Fig. 1. A graph with 2 tables for nodes and segments

Codd himself has seen this limitation and suggested the use of substitutes (i.e., identifiers) to establish the relations between tuples (rows) of tables (Codd 1979). It is possible to store geographic data under the relational data model but code to maintain the data consistent is necessary.

It was found that structuring data in relational tables leads to anomalies in updates. Redundancy can be hidden within the tables and leads to inconsistencies when changes are applied. Dependencies between values within a tuple (row of a table) must be avoided and some multi-column tables must be broken in smaller tables to eliminate such dependencies. It was not possible to give a small set of rules to identify all harmful dependencies and to normalize a set of relational tables.

2.3 Object-orientation

The structuring of design and code centering on objects and operations applicable to them is the dominant paradigm of software engineering the past 20 years. Code for objects and their interactions as operations can hide the internals of an object (so-called encapsulation). Inheritance of object be-

havior into subclasses gives extensibility (Borning 1977). Object-orientation was welcomed in the GIS community to help with the analysis of the complex structure of geographic reality (Egenhofer and Frank 1987; Worboys, Hearnshaw et al. 1990).

Programming languages offer methods to structure data into objects for processing, but different languages have selected slightly different approaches (Cardelli and Wegner 1985); the controversy, regarding contra- vs. co-variance has not yielded a usable and implementable solution (Lämmel and Meijer 2005). Particular difficulties arise with multiple inheritance, i.e., cases where an object is a specialization of two (parent) classes, which is important for geographic data (Frank 1988; Frank and Timpf 1994) (Figure 2) but convincing solutions to model the difference in meaning of concepts like *boat-house* and *house-boat* are missing (Goguen and Harrell 2006). I use here a class-bounded (parametric) concept of polymorphism because the difficulties with subtyping polymorphism seem insurmountable (Abadi and Cardelli 1996; Lämmel and Meijer 2005).

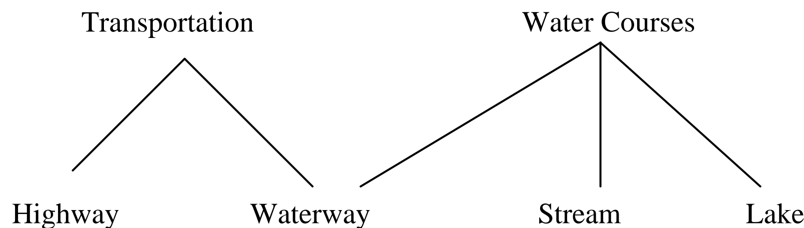


Fig. 2. A Waterway inherits properties from the transportation system and the water bodies

2.4 Object-oriented databases

An impedance mismatch was observed between data handling in an application program written in an imperative language, which is organized 'a piece of data at a time', and the relational database, which operates on whole relations. Object-oriented databases (OODB) combine database concepts with the object-orientation in data structuring (Atkinson, Bancilhon et al. 1989; Lindsay, Stonebraker et al. 1989).

Practically, the subtle differences between variants of object-oriented concepts in programming languages and databases led to difficulties with structuring application data: only the concepts available in both the OODB and the object-orientation programming languages could be used; most

OODB systems are tied to specific object-oriented languages. This makes the integration of data that is organized under different OODBMS very difficult. The tight coupling of the object concept in the application program and the long term view of the database seems to be fundamentally different and impedes evolution of a database over time.

Mapping the data structure from the program view to a simple structure maintained by the data storage system on secondary (disk) storage seems to be the answer. The object-relational approach combines a relational database with an object-oriented programming language (Stonebraker, Rowe et al. 1990), but simpler solutions, going directly to storage emerge (e.g., db4o, objectStore).

2.5 Desired solution

Current GIS applications use an improved, but not theory based, "object-relational" database for storage of data. For experimentation with programs that handle spatial data I felt the following aspects important:

- combination of object description in the schema with the operations to handle the object instances (the data);
- binary relations to avoid dependencies;
- composability of object definitions;
- object orientation, with multiple inheritance using parametric polymorphism;
- focus on long-term secondary storage and direct connection to object-orientation structure of programming language;
- formal, mathematically sound framework.

The next section describes how these goals were achieved. The suggested solution is designed for experimentation and leaves out a number of questions important for processing large amounts of data.

3 Concepts to retain

The desired solution should rather contain less than more concepts and the concepts should be simpler and more generally applicable, more oriented towards the user or the world ontology. The basic concepts, *Entities* and *Relationship*, to structure data but also the conceptualization of the world was described by Peter Chen in a landmark paper (Chen 1976).

3.1 Types and instances

The world contains individuals and we structure our concepts of the world in entities, things that are thought to exist independently. The representation of these entities we will call *instances*. The discussion of data models concentrates on collections of similar instances, which in programming are called types (Cardelli 1997).

3.2 Objects

Objects represent entities that have permanence in time. The difference to the relational data model is that
1. the values and operations applicable to an object may change; but also
2. two objects may coincide in their values but are still distinct objects.
The term ‘object’ is generally used both to describe object types or classes and object instances, which are specific objects, representing individuals; compare the class ‘dog’ and my dog ‘Fido’, which is an individual. Object classes can be seen as algebras, with domains and operations (Ehrich, Gogolla et al. 1989; Loeckx, Ehrich et al. 1996).

3.3 Relations

Object (instances) are related to values. A city has a name, a coordinate to describe its position, and the name of the state it is in. These can be described formally as functions from the object instance to the values (Shipman 1981), but this is not general enough. For example a person can have several children, the mapping from person to children is therefore not a function but a relation. Relations have an advantage over functions as they have always a *converse*: a city is related to the state it is in, the converse relations relates the state to the cities it contains (some functions have inverses, but not all of them!).

Relations can be used to store two different aspects, namely (1) the values associated with an object and (2) a relationship to another object. An example for the first is the name of the city; an example for the second is the state the city is in, which is usually not stored as a name, but as the identifier of an object of type *state*. This use of relations to store relationships between objects solves the problem of maintaining the representation of a graph and other geometric data structures.

4 Description of the solution

A first step towards simplification and flexibility is to select a modern Functional Programming (FP) language (Peyton Jones, Hughes et al. 1999), because FP languages are closely connected to the mathematics of computers (Asperti and Longo 1991; Walters 1991). The resulting conceptual simplicity is demonstrated by the restriction to two concepts and 6 functions to manipulate the data.

4.1 Objects map to identifiers

As already suggested by Codd (Codd 1979) the objects are represented in the long term data storage as identifiers, which are permanent and never reused. When creating a new object in the database, only a new identifier is assigned to it.

Representing objects by identifiers, not data fields, is arguably the most radical decision here, which is sensible only in an environment where data is primarily stored on secondary (disk) storage. Giving up the combination of data storage and object representation, cuts away many of the complexities of object management.

4.2 Materialized (stored) relations

The data associated with the objects are stored in binary relation. Breaking all data into binary relations from object identifier to value removes all potential for anomalies and gives automatically the highest level of normalization.

That operations on relational tables could compose is a major strength of the relational data model and must be preserved. Operations on relations must have inputs and results that are sets of values, not just single values. Thus operations can compose, i.e., the result of one operation can be the input for the next one. A n isomorphic mapping, called the power transpose (Bird and de Moor 1997, 108), is necessary because the category of relations cannot be directly implemented. The power transpose maps from relations to functions over the powerset, which can be implemented.

4.3 Bundles of functionality are modules

Bundles of functionality, e.g., support for graphs, ownership cadastre, or agents moving, etc. must be designed and coded separately. They are rep-

resented by modules, which are independently compilable units and care will be necessary to keep their dependencies minimal. In particular, the data structure and the related operations must be included in the same module, allowing different applications to use different combinations of bundles.

4.4 Operations for data handling

To store data, the operation is *assert*, which adds a new entry to a relation. The operation *change* takes a function that is applied to the currently stored value and the result is then stored (this can be used to *set* a value to *v* by passing the function *const v*). The function *delete* will delete the corresponding entries.

To retrieve two base functions are sufficient:

to: given a set of identifiers and a relation label, find the values related to the identifiers.

from: given a set of values and a relation label, find the identifiers related to the values. (Note: *to* and *from* are not inverse to each other but *from = to.conv*!) For the special case of searching in functions (which are a special kind of relations) with a single value and expecting a single result, specialized forms of *to* and *from* are given as *to'* and *from'*. *To* and *from* are total functions—they produce always a result, but *to'* and *from'* can fail and produce then a descriptive error message.

5 Example: How to Use RelDB

This section shows code for storing and retrieving data describing a simple graph: the entities are NODE and EDGE; there are relations from NODE to the value of its number and to the value of its coordinate. EDGE has its length as a value and two relations to the points where it starts and where it ends. The ER diagram in Figure 3 shows that there are five relations, 3 to values (ellipses) and 2 between entities (diamonds).

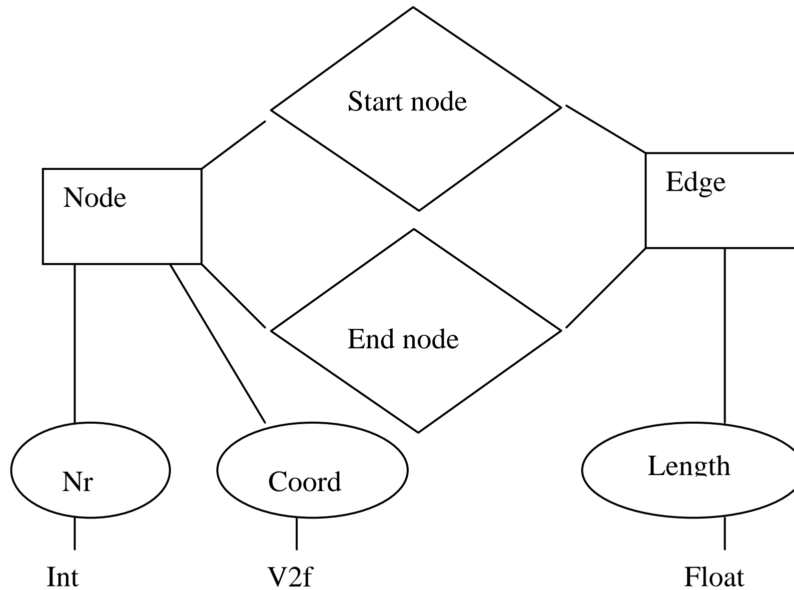


Fig. 3. The ER diagram in the original style (Chen 1976)

The coordinates are formatted as type *V2f* (Vector with *x* and *y* coordinates represented as floats), the number is an integer (*Int*), the length of the edge is a *Float*. There are five functions, which will be represented as relations:

```
coord :: PointID-> V2f
nr :: PointID -> Number
startNode :: EdgeID -> PointID
endNode ::EdgeID -> PointID
length :: EdgeID -> Float.
```

5.1 Definition of relations

The five relations are defined each in 3 lines of code. Consider the relation from the node to the coordinate value: Define a type for the relation label and define a value of it. This creates two entries in the namespace of the module, for example *ID2Coord* and *id2coord*. These will be used as the relation label; for convenience it is also used to contain a descriptive string that is used when printing the relation.

```
newtype ID2Coord = ID2Coord String
id2coord = ID2Coord "coordinates of points"
```

A function *id2c* to add this relation to the empty database is declaring the type of the values, namely *V2f*:

```
id2c = HCons (id2coord, zero::RelVal V2f)
```

This must be repeated for the 4 other relations: for the number, for the two relation edge to node and finally for the length:

```
newtype ID2Nr = ID2Nr String
id2nr = ID2Nr "identification of nodes"
id2n = HCons (id2nr, zero::RelVal Nr)

newtype ID2StartNode = ID2StartNode String
id2startnode = ID2StartNode "start node of edge"
id2s = HCons (id2startnode, zero:: RelID)

newtype ID2EndNode = ID2EndNode String
id2endnode = ID2EndNode "end node of edge"
id2e = HCons (id2endnode, zero:: RelID)

newtype ID2EdgeLength = ID2EdgeLength String
id2edgelength = ID2EdgeLength "length of edge"
id2d = HCons (id2edgelength,
                    zero:: RelVal Float)
```

5.2 Construction of database (schema)

The database is constructed by adding these relations to the empty database (*emptydb1* exported from the generic database module). A database with support for node with number and coordinates would be:

```
pointdb1 = id2c (id2n emptydb1).
```

Remember: the constructs *id2n*, *id2c* are functions that can be applied or composed (with "."). To construct a database for graphs is:

```
graphdb1 = (id2c . id2n . id2s . id2e) emptydb1
```

or equivalently using the already existing pointdb1, which may already contain data:

```
graphdb1 = (id2s . id2e) pointdb1.
```

5.3 Handling object data

5.3.1 Points

Assuming that the data describing the points is in a file as a sequence of pairs consisting of number and coordinate values. A single entry describing a point looks as follows: *(3, V2f 4.1 2.9).*

The function *loadNode* takes a pair of number and coordinate as input. It creates first a new object and gets the identifier into the variable *i*. Then it asserts that this identifier *i* has for the relation *id2nr* the value of *n* (the node number from the input) and then asserts that this identifier *i* has for the relation *id2coord* the value of *p* (the coordinate from the input).

```
loadNode (n, p) = do    i <- createM    nodeT
                        assertM  id2nr i n
                        assertM  id2coord i p
```

To load a series of points, stored in list *fh*, to the *pointdb1* is achieved with:

```
pointdb2 = (mapM loadNode fh) *** pointdb1.
```

To find for a given point number the corresponding identifier, we use the function *from'* that is specialized for cases where we expect only a single value as a result (as intended, the function from point number to identifier is an isomorphism):

```
identifyByNr db nr =
          from' "identifyNr: not found" db id2nr
```

where the message is printed if for this point number no point is found in the data. The function to retrieve the point coordinate from a given identifier is very similar:

```
pos  db i = to' "position i pl in loadFreihaus"
                                    db  id2coord i.
```

5.3.2 Edges

Loading an edge is somewhat more involved: Assume that the external file contains pairs of numbers of the start and end points for each edge. The relations between the edges and the nodes however are based on identifiers; it is necessary to find the identifier with *identifyByNr* to enter in the relation. To compute for an edge the length from start to end, we have to retrieve the position of the two nodes using *pos*. A function *dist*' to compute the distance between to coordinate pairs exists and is extended (overloaded) with a new instance, such that it computes the distance between to points given by their identifier (in the context of the current data):

```
instance  (FromTos a ID2Coord V2f)
             =>  Vec2x ID (State a Float) where
    dist' a b = State $ \s ->
             let      ap = pos s a
                      bp = pos s b
             in ( (dist' ap bp), s)
```

Combining these support functions to form a single loadEdge function:

```
loadEdge (s, e)  =
         do   i <- createM edgeT
               si <- identifyNrM s
```

```
                  se <- identifyNrM e
                  assertM id2startnode i si
                  assertM id2endnode i se
                  (c :: Float) <- dist' si se
                  assertM id2edgelength i c.
```

It takes a pair of node numbers as input and creates first an edge with an identifiers. Then it retrieves the identifiers for the start (*si*) and the end node (*se*). It asserts that these are the values for the *id2startnode* and the *id2endnode* relations respectively. Then the distance between the two nodes (given by their identifiers) is computed and the result asserted for the relation *id2edgecost*.

6 Conclusion

The complex issues of designing a database schema has been reduced to a very small number of essential concepts. Additional tools may be necessary to achieve better performance, to install spatial access methods and to connect with a transaction management system, tasks left for future research. The goal was to identify what is essential, and to separate it from the desirable aspects. I have found it necessary to provide more than a 'paper and pencil' analysis but to implement the result and to show how it can be done in a running program.

- The restriction to binary relations simplifies the query language to 2 commands, one to find the related terms to an entry (*to*) and the other to find the identifiers related to a value (*from*), which is using the converse of the relation.
- Application programming in a functional programming language using the monadic style is straightforward.
- Modularization such that the schema information and the code to operate on an object class can be bundled and an application can use multiple of these bundles without interference.

A number of questions remain open for future work:

- Should the identifiers be typed?
- Consistency constraints: if we know that a relation is a function, where is the best place to enforce this restriction?

Acknowledgements

Many of my colleagues and students have helped me to advance to this point. Particular thanks go to Werner Kuhn for many valuable discussions during the past years. John Herring has asked years ago very valuable questions that have pushed me in this direction, but special recognition goes to Peter Chen for his early insight into the fundamental aspects of data modeling. An anonymous reviewer made useful suggestions to improve the paper.

References

Abadi, M. and L. Cardelli (1996). A Theory of Objects. New York, Springer-Verlag.

Achatschitz, C. (2005). Identifying the Necessary Information for a Spatial Decision: Camping for Beginners. CORP 2005 & Geomultimedia05, Vienna, Austria, Selbstverlag des Institutes für EDV-gestützte Methoden in Architektur und Raumplanung.

Asperti, A. and G. Longo (1991). Categories, Types and Structures - An Introduction to Category Theory for the Working Computer Scientist. Cambridge, Mass., The MIT Press.

Atkinson, M., F. Bancilhon, et al. (1989). The Object-Oriented Database System Manifesto. First International Conference on Deductive and Object-Oriented Databases, Elsevier.

Backus, J. (1978). "Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs." CACM 21: 613-641.

Bird, R. and O. de Moor (1997). Algebra of Programming. London, Prentice Hall Europe.

Bishr, Y. (1998). "Overcoming the Semantic and Other Barriers to GIS Interoperability." International Journal of Geographical Information Science 12(4): 299-314.

Bittner, S. (2001). An Agent-Based Model of Reality in a Cadastre. Department of Geoinformation. Vienna, Technical University Vienna.

Borning, A. (1977). ThingLab - An Object-Oriented System for Building Simulations Using Constraints. IJCAI 1977. vol. 1.: 497 - 498.

Cardelli, L. (1997). Type Systems. Handbook of Computer Science and Engineering. A. B. Tucker, CRC Press: 2208-2236.

Cardelli, L. and P. Wegner (1985). "On Understanding Types, Data Abstraction, and Polymorphism." ACM Computing Surveys 17(4): 471 - 522.

Ceri, S., P. Fraternali, et al. (2000). XML: Current Development and Future Challenges for the Database Community. Advances in Database Technology - EDBT 2000 (7th Int. Conference on Extending Database Technology, Kon-

tanz, Germany). C. Zaniolo, P. C. Lockemann, M. H. Scholl and T. Grust. Berlin Heidelberg, Springer-Verlag. 1777: 3-17.

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

Chen, P. P. (2006). "Entity-Relationship Modeling:Historical Events, Future Trends, and Lessons Learned." Retrieved 01.09.06, 2006.

Codd, E. (1979). "Extending the Database Relational Model to Capture More Meaning." ACM TODS 4(4): 379-434.

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

Dieckmann, J. (2003). DAML+OIL und OWL XML-Sprachen für Ontologien. Berlin: 21.

Dijkstra, E. W. (1976). A Discipline of Programming. Englewood Cliffs, NJ, Prentice Hall.

Egenhofer, M. J. and A. U. Frank (1987). Object-Oriented Databases: Database Requirements for GIS. International Geographic Information Systems (IGIS) Symposium: The Research Agenda, Crystal City, VA, NASA.

Ehrich, H.-D., M. Gogolla, et al. (1989). Algebraische Spezifikation abstrakter Datentypen. Stuttgart, B.G. Teubner.

Ferber, J., Ed. (1998). Multi-Agent Systems - An Introduction to Distributed Artificial Intelligence, Addison-Wesley.

Fonseca, F. T. and M. J. Egenhofer (1999). Ontology-Driven Geographic Information Systems. 7th ACM Symposium on Advances in Geographic Information Systems, Kansas City, MO.

Frank, A. U. (1988). Multiple Inheritance and Genericity for the Integration of a Database Management System in an Object-Oriented Approach. 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, New York, NY, Springer-Verlag.

Frank, A. U. (1999). One Step up the Abstraction Ladder: Combining Algebras - From Functional Pieces to a Whole. Spatial Information Theory - Cognitive and Computational Foundations of Geographic Information Science (Int. Conference COSIT'99, Stade, Germany). C. Freksa and D. M. Mark. Berlin, Springer-Verlag. 1661: 95-107.

Frank, A. U. (2003). Ontology for Spatio-Temporal Databases. Spatiotemporal Databases: The Chorochronos Approach. M. Koubarakis, T. Sellis and e. al. Berlin, Springer-Verlag: 9-78.

Frank, A. U. and S. Timpf (1994). "Multiple Representations for Cartographic Objects in a Multi-Scale Tree - An Intelligent Graphical Zoom." Computers and Graphics Special Issue on Modelling and Visualization of Spatial Data in GIS 18(6): 823-829.

Goguen, J. and D. F. Harrell (2006). Information Visualization and Semiotic Morphisms. 2006.

Härder, T. (1986). New Approaches to Object Processing in Engineering Databases. Proceedings on the 1986 International Workshop on Object-Oriented

Database Systems, Pacific Grove, California, United States, IEEE Computer Society Press.
Lämmel, R. and E. Meijer (2005). Mappings Make Data Processing Go' round, Redmond, USA, Microsoft Corp.
Langran, G., Ed. (1992). Time in Geographic Information Systems. Technical Issues in GIS, Taylor and Francis.
Lindsay, B., M. Stonebraker, et al. (1989). "The Object-Oriented Counter Manifesto."
Loeckx, J., H.-D. Ehrich, et al. (1996). Specification of Abstract Data Types. Chichester, UK and Stuttgart, John Wiley and B.G. Teubner.
Mark, D., B. Smith, et al. (2000). Ontological Foundations for Geographic Information Science: 18.
Nyerges, T. (1989). "Schema Integration Analysis for the Development of GIS Databases." International Journal of Geographical Information Systems 3(2): 153 - 183.
Peyton Jones, S., J. Hughes, et al. (1999). Haskell 98: A Non-Strict, Purely Functional Language.
Raubal, M. (2001). Agent-Based Simulation of Human Wayfinding: A Perceptual Model for Unfamiliar Buildings. Institute for Geoinformation. Vienna, Vienna University of Technology: 159.
Shipman, D. W. (1981). "The Functional Data Model and the Data Language DAPLEX." ACM Transactions on Database Systems 6 (March).
Stonebraker, M., L. A. Rowe, et al. (1990). Third-generation Data Base System Manifesto, UC Berkeley: Electronics Research Lab.
Walters, R. F. C. (1991). Categories and Computer Science. Cambridge, UK, Carslaw Publications.
Weiss, G. (1999). Multi-Agent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge, Mass., The MIT Press.
Worboys, M. F., H. M. Hearnshaw, et al. (1990). "Object-Oriented Data Modelling for Spatial Databases." International Journal of Geographical Information Systems 4(4): 369-383.