Representing Metadata in the ADAMS Database System

John L. Pfaltz †
jlp@cs.virginia.edu

David R. Mikesell
drm9f@cs.virginia.edu
Computer Science Department

William R. Emanuel ‡
wre6s@virginia.edu
Department of Environmental Science

University of Virginia, Charlottesville VA

Computer Science TR-98-21

July, 25 1998

Abstract:

A major concern of environmental scientists, and others with long term data requirements, has been the establishment of metadata standards so that data recorded today will be accessible 50 to 100 years hence. We contend that more important than the standards themselves will be a context in which they can be represented and can evolve as new requirements and technologies emerge. In this paper, we discuss an object oriented language which facilitates both the representation of metadata and its graceful evolution.

This research was supported in part by
† DOE Grant # DE-FG05-95ER25254, and
‡ NASA's Mission to Planet Earth through Grant NAGW 2669.

1. Introduction

In the IEEE Metadata Workshop of 1993, Francis Bretherton asked a fundamental question, "if scientists fifty, one hundred, or two hundred years in the future look at the data we are recording now, will there be sufficient metadata for them to use it effectively?" [Bre93]. The answer, which was almost universally echoed by the participants, has been a resounding "no". In [Bre96], the wording of this question was refined to what he calls the 20 year test: "what will our successors think 20 years from now when they observe changes between now and then and are trying to decide whether such changes are real, or artifacts of the way in which we took, processed, or analyzed the data". These kinds of questions, by Bretherton and a growing community of researchers concerned with metadata issues, *e.g.* [DiM91, FJP90, GAL96, Les95], have led to an evolving understanding of the nature and purpose of metadata. One categorization has been into *guide* metadata and *control* metadata, where the former had been assumed to be natural language notes intended for interpretation by humans, and the latter were machine readable values, such as field lengths, intended to control data manipulation operations and data transfer. For example, netCDF [Ful88, ReD90] and others provide a way of specifying control metadata needed to exchange data files. To be machine readable, control metadata must be restricted to a controlled vocabulary with a well defined syntax. Gradually, it has become apparent that guide metadata becomes more valuable if it too is expressed with respect to a structured language having a common vocabulary, that is, if it more closely resembles control metadata. Only then does guide metadata facilitate automatic *searching, browsing* and *retrieval*. This suggests another taxonomy of metadata based on its intended usage including those above, and: *ingest, quality assurance, reprocessing, machine to machine transfer, storage* and *archival* functions [BrS94]. The reader can easily identify examples of metadata associated with each of these functions.

The interest in metadata has also generated considerable pressure to develop a set of standards specifying "what metadata should be provided" (a) to facilitate access to distributed data sets [Bar94], (b) to interpret complex spatial data [FGD94], or (c) to ensure that long term data sets can be accurately interpreted in the future [Bla96, KGM96, Rot96]. The authors have encountered this metadata problem in the specific context of a global change simulation. We are implementing a large global environmental change system that had been written in Fortran using flat files for storage to one written in C using object-oriented database technology for storage. What metadata must we incorporate into our database to make it intelligible to other global change researchers? to global change researchers 50 years from now?

But trying to establish "what" metadata is necessary may be the wrong issue on which to initially focus. We are unlikely to be able to precisely specify *all* the metadata our successors will regard as necessary. Perhaps a better question would be "can we identify the facilities needed to express metadata concepts in an evolving family of languages?" We are familiar with the slow evolution of languages with which to express computational procedures; a similar process seems appropriate for the representation of data and its metadata. NetCDF would be one member of this family, as would SQL. The ADAMS database system [PfF93, Pfa93], which facilitates access in shared, distributed databases provides another syntax for expressing metadata. We are convinced that metadata is a conceptually deep problem which will never be *solved*, only managed incrementally; and that progress will only come from dispersed nuclear groups developing their own solutions [Bre96]. The emergence of database languages that facilitate the expression of metadata concepts will be necessary to coalesce such evolving standards.

The ADAMS database system we discuss in the next section is being incorporated into GCSYS (Global Change SYStem) to manage its persistent data. The two related questions we address are: "are the kinds of metadata we are recording for global change sufficient?" and "is such a functional language model an appropriate mechanism for representing metadata"?

2. The Decomposed Storage Model

By a decomposed storage model, we mean one in which all data, or object values, are decomposed into distinct attribute functions. Data is *not* regarded as a string of bits packed in some storage structure. Rather data associated with any object is obtained by evaluating the attribute function given the object identifier, or *oid*, as argument. As long ago as 1974, Lorie proposed much the same decomposed structure in XRM [Lor74]. Copeland and Khoshafian [CoK85, KCJ87] have also employed such a data representation. They first coined the term DSM, or decomposed storage model. A primary difference between ADAMS and other object-oriented systems is that in our language we treat functions as first class objects, just like any other data object. We also make a clear distinction between data values and data objects.[1] A data value is a bit string which is treated as an atomic, scalar value (which may be treated as a numeric, string, image *etc.*) but whose internal structure, if any, is not expressible within the language. In contrast, data objects may have considerable structure. All data objects are uniquely identifiable, whereas data values are not.

In spite of this object-oriented flavor, we hesitate to call this an object-oriented model because we do not associate general methods with the data objects[2]. This database language is primarily concerned with the representation of, and access to, data and metadata in persistent secondary and tertiary storage; but because methods can only be executed in volatile primary storage ADAMS only provides implicit fetch and put methods. We assume that the host language, in which applications are expressed, is the proper medium for expressing most methods and data processing procedures. ADAMS is then embedded within this host language to provide high-level support for persistent data structures and their metadata. We believe there are three advantages to this approach. First, scientists will always elect to use the language most appropriate for their domain specialty, for example C to express global change processes. An embedded language can permit scientists using different programming languages to share data.[3] Second, when one thinks of data in functional terms, it is quite natural to let the function access disparate, distributed data and transform it by means of filters to the form expected by the host process. For example, Kemp and Gray [KDG96], whose system is based on the functional data model, have been quite successful in combining distributed genome data. Third, this treatment of attribute functions as first-class objects enhances several forms of class, or schema, evolution. On the other hand, a "seamless" database

[1] Many object-oriented languages make this distinction explicitly, or in a *de facto* manner [ZdM90]; others do not [KGB90].

[2] There seems to be a consensus that object-oriented languages must support methods [ElN94, ZdM90].

[3] If properly designed, the language will facilitate the sharing of data models and the linking of modules implemented in different languages [PKW96]. One example of this kind of module interconnection can be found in [AIL96].

language, using for example the syntax of C++, need not gracefully migrate to some different, higher-level object-oriented language of the future. If an embedded metadata language can interface with two, or more, current programming languages, it can probably accommodate yet another ten years hence. In this paper we illustrate both evolution of metadata and evolution of the underlying database structure itself.

ADAMS assumes that all objects belong to user defined classes, of which there are four generic base classes — ordinary objects, set objects, scalar valued function objects, and object valued function objects. Treating sets and functions as objects in their own right is central to the ADAMS approach because these three latter kinds of objects have all the properties, capabilities, and restrictions of *ordinary* objects, such as class inheritance. In addition, a "set" object is comprised of zero, or more, distinct objects *of the same class*; a "scalar valued function" object returns a data value when invoked with an object *oid* as argument; and an "object valued function" returns an object identifier. In accordance with traditional usage in both the relational and object-oriented models, we will call the scalar valued function, an *attribute*. Many authors also use the term "attribute" for object valued functions as well; we will call them *obj_attributes*.

2.1. The Global Change Application

To illustrate some of the features we believe are appropriate in a metadata language, we examine a module that describes incident solar radiation at the terrestrial surface. [WSE95]. The basic unit of this model is a map pixel, or an unit of earth area 50km $\times$ 50km, which has a known *latitude, longitude*, and *elevation*. We may also associate the *average solar radiation* with each element.[4] Consequently, we might declare our basic element as:

[4] Actually we can calculate incident solar energy from latitude, elevation, and time of day. (In the actual model we use a monthly average; in this paper we suggest how such refinements can be made to the database.) Standard solar radiation formulas come from [Jon92].

```
EARTH_ELEMENT  isa  CLASS
      having attributes = { latitude, longitude, elevation, s_rad }
      having obj_attributes  = { east, west, south, north }
```

Figure 2-1
A class declaration

Here, the elements *east, west, south* and *north* denote those earth elements standing in that relation to the current one. If *x* denotes an *earth_element* object, that is its *oid,* then *x.elevation* will denote its numeric elevation and *x.west* will denote the *earth_element* object immediately to the west of this one.[5] It is not difficult to write corresponding declarations for this item of structured data in Pascal, C, or C++. In a *struct* the attributes would be represented by numeric variables of appropriate type; the four directions would be pointer variables.

This emphasizes the fundamental characteristic of the language we are describing. In a decomposed model, neither *latitude, longitude, elevation, s_rad, east, west, south* or *north* would be variables of any type; they are functions, which given an object identifier of the *earth_element* class, have a single scalar or object value; but otherwise are objects with all the aspects of any object.

2.2. Providing Metadata

An immediate observation based on the class declaration of Figure 2-1 is that it provides absolutely NO metadata by which it might be comprehensible 20 years from now! Because the research scientists chose some rather clear mnemonic names, a future reader would probably know what the numeric values *latitude, longitude,* and *elevation* mean in general. But, there is no indication of the units of measurement. Is *elevation* measured in *feet* or *meters*? Is the convention that *lat* = -47.25 corresponds to 47° 15′ S latitude obvious? The variable *s_rad* is used in this model, but its meaning, "average solar radiation", is not at all apparent. All the necessary metadata does exist in separate documentation; but one has to paw through a fair amount of paper notes and source listings to find it. To be effective metadata should exist in whatever data space contains the class declaration and data sets themselves.

[5] We employ a postfix application of attribute functions, as in SQL. Such suffix notation supports a simple left to right parsing and evaluation of complex expressions.

A primary reason for choosing a decomposed storage model to represent data is that it is usually the attributes that need documentation. When the attribute is a functional object, then we can associate attributes with it, as in:

```
MEASUREMENT  isa  REAL_ATTRIBUTE
      having attributes = { units, long_name }
      having obj_attributes  = { }

latitude  instantiates_a  MEASUREMENT
latitude.units  <- "degrees of arc"

elevation instantiates_a  MEASUREMENT
elevation.units <- "meters"

s_rad     instantiates_a  MEASUREMENT
s_rad.long_name <- "average solar radiation"
s_rad.units     <- "W / meter^2"
```

Figure 2-2
Attributes declared as objects with attributes

One detail to note is that the code of Figure 2-2 is essentially assigning attributes defined in the EARTH_ELEMENT class. That is, every instance of an EARTH_ELEMENT object does not carry the *units* and *long_name* with its attributes; rather, the attributes *latitude*, *elevation* and *s_rad* are themselves treated as entities which have these attribute properties. This mechanism allows metadata to be associated with any attribute (or table column, using the relational paradigm). Furthermore, since these string valued attributes *units* and *description* are in the data space itself, they are available to any process that can also access attributes of *earth_element* objects. In particular, one can imagine a subsequent process examining these attribute values also performing a dimensional analysis on their associated *units* as a means of quality assurance.

It can be argued that just knowing the units of measurement and a full (unabbreviated) name of an attribute may still provide insufficient metadata. True! But, we would respond that (1) this represents essentially all of the metadata that is available to us as documentation of this simulation project, and (2) we are only providing a mechanism by which metadata can be attached to the data items that it describes, were it to be available. Our experience is that a large class of important metadata is more naturally associated with the attributes as a whole than with individual measurements. The ADAMS language facilitates this.

2.3. Database Evolution

In [Bre96], it was emphasized that the representation of metadata should be evolutionary in nature. Because of the importance of these solar radiation models, it may turn out that we should indicate a maximum expected error in the average solar radiation (*s_rad*) and to also document with a set of external references the methodologies used to derive this average.

The first is easiest. Since maximum expected error can be a meaningful attribute for all MEASUREMENT attributes, we can simply evolve the schema of this attribute class by instantiating a real attribute *max_exp_error*, and then executing the ADAMS statement

```
insert  max_exp_error  into  MEASUREMENT->attributes
```

By inserting this new attribute into the set of attributes associated with the MEASUREMENT class, we have effectively changed it to that shown in Figure 2-3.

```
MEASUREMENT  isa  REAL_ATTRIBUTE
      having attributes = { units, long_name, max_exp_error }
      having obj_attributes  = { }
```

Figure 2-3
Newly evolved MEASUREMENT class

Henceforth, all code referencing this class will see this new attribute schema. Adding, or deleting, attributes from the set of associated attributes is fundamental to schema evolution. Some have suggested that real schema evolution must involve more than just adding or deleting attributes, *c.f.* [BKK87, MoS93, Rod92]. However, evolving schema constraints and class methods is much more complex, and this ability to evolve schema "on the fly" *without* recompilation of code and *without* reformatting storage seems to be an important first step.

Adding documentation is more complex. We must declare three new object classes. If a reference citation is an object in the class

```
CITATION  isa  CLASS
          having attributes = { author, title, journal, volume
                                year, pages }
```

and we declare the set

```
CITATION_SET  isa  SET of  CITATION  elements
```

then we may create another class of object functions by

```
REFERENCES  isa  OBJ_ATTRIBUTE  with  image  CITATION_SET
```

A set of references of this form is created explicitly by instantiating the set, then iteratively reading in each reference citation, creating a corresponding citation object which is then inserted into the *citation_set*. Finally, we will associate this set of citations with the *s_rad* attribute by executing the assignment

```
s_rad.references  <-  citation_set
```

But, before this is possible, we must make *references* an object attribute of the MEASUREMENT class. As with the new attribute *max_exp_error* of the preceding paragraphs we use the ADAMS statement

```
insert  references  into  MEASUREMENT->obj_attributes
```

to insert *references* into the set of *obj_attributes* associated with MEASUREMENT, so that the class declaration now looks like

```
MEASUREMENT  isa  REAL_ATTRIBUTE
      having attributes = { units, long_name, max_exp_error }
      having obj_attributes  = { references }
```

Figure 2-4
Final form of MEASUREMENT objects

This kind of flexibility can be used to evolve the associated metadata. It can also be used to modify the underlying structure of the modeled data itself. For example, while incident solar radiation is a fundamental measurement, a more important variable in this study of the primary productivity of land plants is average photon flux density in the photosynthetically active portion of the solar spectrum, measured in 10^{-6}*mol photons/m*^{2}*/sec*. Over a day, both solar radiation and photon flux density are dependent on the length of daylight at this particular location on the earth's surface. And, the day length is in turn a function of the time of year. In the actual global change model we have been using as our running example, solar radiation, photon flux density and day length are calculated for each month of the year, using the middle day of the month as representative. Consequently, we choose to subscript these attributes (with respect to the months, 1 to 12). Since attribute functions are simply objects, of course they can be subscripted [PfF90].

By inserting the subscripted attributes *s_rad*, *pfd* and *day_length* and a generic string valued attribute called *comments* into the set *EARTH_ELEMENT→attributes* we can achieve the following class declaration

```
EARTH_ELEMENT  isa  CLASS
      having attributes = { latitude, longitude, elevation,
                            s_rad[1..12], pfd[1..12],
                            day_length[1..12], comments }
      having obj_attributes  = { east, west, south, north }
```

Figure 2-5
Evolved EARTH_ELEMENT class declaration

A functional approach, in which data is not regarded as fixed tuples of stored bits, but rather the result of applying functional processes to which the object identifier is only an argument, opens up some very different ways of thinking about data and its metadata. If these processes are themselves objects, then we have a representational language that is capable of easily modifying the way data objects are related to each other and metadata is attached to the various components of the representation. For example, solar radiation and photon flux density are only two intermediate values in the global change model. The real quantity of interest is *npp*, net primary production (of carbon), illustrated in Figure 2-6 and which is in turn a function of vegetative density, or leaf area index.

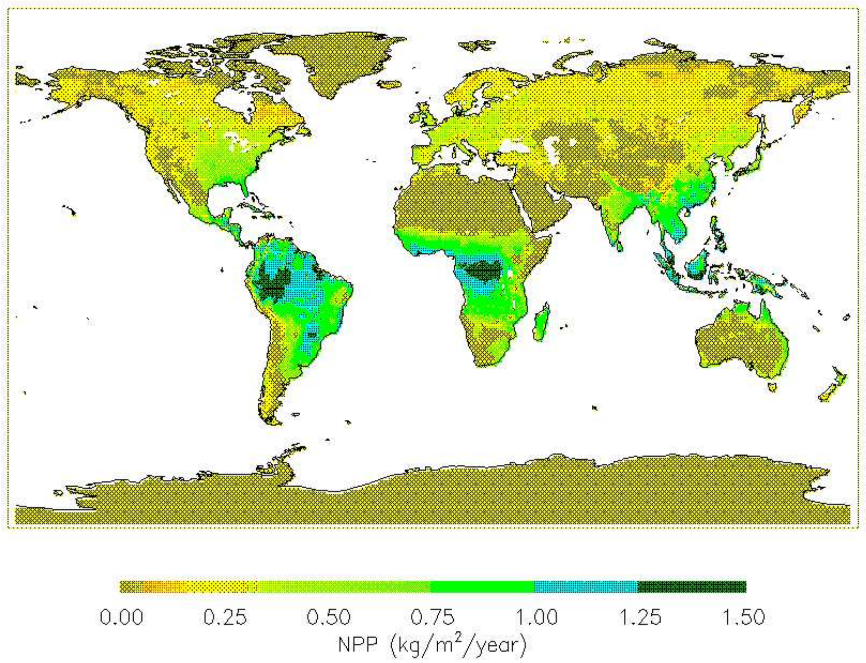

Figure 2-6

As the model evolves these attributes have been added to the underlying persistent database. The application code generates the values for each earth element using intermediate parameters that have first been calculated and collects them into 2-dimensional image objects. Then it displays these image objects as color maps (black and white in this copy) as shown in Figure 2-6. Maps of this nature make the data visually comprehensible and constitute a major way of comparing the output of different global change models. They, too, are objects in the database. What metadata must be provided to make a map comprehensible? Can it be gracefully added to your database system?

3. Metadata and Evolution in Practice

All of the concepts in the preceding section have been implemented; some such as class evolution for over five years. This facility was reported in [PFG92] where we described the gradual evolution of a tactical battlefield database written in ADAMS. It is now routine to let the structure of an ADAMS database, such as a rather complex medical records prototype, evolve as operational experience accumulates. All of the illustrated code fragments have been drawn from working programs. However, we have only begun to explore the potential of embedded metadata and its evolution.

The decomposed storage model used by ADAMS provides the ability to create self-documenting, evolvable databases. But it does so at a cost. Each data reference requires the evaluation of the attribute function; retrieval of a single n-tuple of values requires n function evaluations and possibly n disk accesses in contrast to a single disk access in most relational or object-oriented database configurations. A moderately complex retrieval in ADAMS can be two orders of magnitude slower than the equivalent query in a well implemented relational database. This has limited the use of ADAMS in many conventional applications.

On the other hand, ADAMS provides superior performance in multi-threaded database applications. The global change simulation described here is one such application because the computation and data access associated with one earth element can be performed asynchronously with relatively little cross talk between threads.

In [Had95, Had97, HaP97, PHF98], it is shown that a distributed, shared nothing ADAMS can obtain nearly linear scale[6] up of moderately complex queries involving implicit joins over 2, 4, or 8 processors. This scale up was achieved using a relatively large database of over 1 million objects. Discussion of the full details is inappropriate here. We will only note that we are in the process of implementing an ADAMS configuration over Legion [GrW97] which is an emerging operating system for managing globally distributed resources and that our department has just acquired a cluster of 64 DEC Alpha's (soon to be expanded to 128) on which we can continue our study of parallel performance. The very characteristics which make a decomposed storage system slow in a tightly coupled system appear to make it quite competitive for multi-threaded applications over loosely coupled distributed systems.

[6] Recall "scale up" is the ratio $time_n(n * problem\_size) / time_1(problem\_size)$. It is a far more conservative, and accurate measure of parallel performance than "speed up" which assumes a fixed problem size. That is, speed up = $time_n(problem\_size) / time_1(problem\_size)$. Because of the extra cache storage and other resources, observed speed up of ADAMS programs is markedly superlinear.

4. Summary

In a database system, it should be possible to associate metadata either with individual data points or with attributes. In addition, this metadata, regardless of what it is associated with, should be accessible and searchable by the users of the database. This would provide the flexibility necessary for users with different metadata requirements to use the same database system.

Given a relational model of data, it is not difficult to associate metadata with any row, or tuple, of the relation *r*. One either adds a metadata attribute to the schema *R*, or creates a separate relation *r_meta*, with the key of *r* as a foreign key. This metadata is accessible in the same manner as any other data item. For example, in our global change model, associating each leaf area index datapoint with metadata describing the type of vegetation predominating in a given area could easily be done in this manner. This can be useful since leaf area index value is not necessarily associated with the type of vegetation, but observing how they both change over time may give greater insight into the global change problem. This can be implemented in our decomposed storage model by simply adding a metadata attribute to the class definition that includes the leaf area index attribute. Metadata with object level granularity is not hard to implement, even though it can be expensive in terms of storage.

But, associating metadata with the columns, or attributes, of the relation is more difficult. One must create a relation *c_meta*, whose keys are the schema attribute names or column headings. Unfortunately, such an implementation severely limits browse capability because (1) retrieving the metadata requires prior knowledge of the attribute name, and (2) given an attribute with appropriate metadata characteristics, there is no mechanism (such as a join) for introducing the corresponding attribute name into a selection process.

This problem is relevant because most metadata is about attributes. In [Len94], Lenz describes a set of typical metadata associated with the Berlin Census of 1987.[7] Here, *every* example he cites is with

[7] We would note that demographers have different metadata requirements than environmental scientists, who have different requirements than geneticists.

respect to an attribute of the statistical table. These include (using our notation):

household.definition	"All the people who live there together and have a joint budget ..."
salary.type	summary_attribute
profession.type	category_attribute
salary.domain	decimal
city.footnote	"until 1990 Berlin as Berlin(-West)"

He then goes on to list more than 40 categories of metadata that would be reasonable in the context of census statistics. In our model, this set of metadata can be easily associated with the attributes of the statistical table using the same mechanism described in Section 2.2.

Use of an object-oriented database does not, by itself, ameliorate this problem. Normally, it is a variable name, or sometimes a method name, that functions as the attribute name. These names are defined in `.h` include files along with the class declarations. They are not part of the data space and are not normally "browseable". It is our contention that only by making attributes themselves first class objects can one make metadata accessible, searchable, and evolutionary.

In Section 1, we observed that there are two general kinds of metadata, *guide* and *control*. The former is intended for use by humans and is expressed in natural language; the latter is intended for use in computer system operations and may be expressed using a controlled vocabulary. In [Bre96], it was noted that the boundary is not fixed and that "a strategy for improving our handling of guide metadata should be to move that boundary towards the control category in an evolutionary manner". All of the metadata examples of this paper have been of the *guide* variety (primarily because this paper is intended for use by humans), yet all is machine accessible. By providing well-defined linguistic constructs to express metadata, we facilitate its movement towards becoming *control* metadata which can be employed in automatic, and possibly remote, retrieval and processing operations. We believe this is a crucial step in incorporating adequate, and usable, metadata with future data sets.

5. References

[AIL96] V. Anjur, Y. E. Ioannidis and M. Livny, FROG and TURTLE: Visual Bridges Between Files and Object-Oriented Data, in *8th Intern'l Working Conf. on Scientific and Statistical Database Management*, P. Svensson and J. C. French (editors), IEEE Press, Stockholm, June 1996, 76-85.

[BKK87] J. Banerjee, W. Kim, H. Kim and H. F. Korth, Semantics and Implementation of Schema Evolution in Object-Oriented Databases, *Proceedings 1987 SIGMOD Conference*, San Francisco, CA, December 1987, 311-322.

[Bar94] J. Barnett, The Schlumberger Data Model: A Meta Based Data Modeling Methodology, *IEEE Workshop on Metadata for Scientific and Technical Data Management*, Washington, DC, May 1994.

[Bla96] G. Bland, EOSDIS Core System Metadata Design, *1st IEEE Metadata Conference*, Bethesda, MD, April 1996.

[Bre93] F. P. Bretherton, Reference Model for Metadata: A Strawman, *IEEE Workshop on Metadata for Scientific and Technical Data Management*, College Park, MD, May 1993. (position paper).

[BrS94] F. P. Bretherton and P. T. Singley, Metadata: A User's View, in *7th Intern'l Working Conf. on Scientific and Statistical Database Management*, J. C. French and H. Hinterberger (editors), IEEE Press, September 1994, 166-174.

[Bre96] F. P. Bretherton, Reinventing the Wheel, *1st IEEE Metadata Conference*, Bethesda, MD, April 1996. (Keynote address).

[CoK85] G. P. Copeland and S. N. Khoshafian, A Decomposition Storage Model, *Proc. ACM-SIGMOD 1985 Conf. on Management of Data*, Austin, TX, May 1985, 268-276.

[DiM91] J. Diederich and J. Milton, Creating Domain Specific Metadata for Scientific Data and Knowledge Bases, *IEEE Trans. on Knowledge and Data Engineering 3*,4 (December 1991), 421-434.

[ElN94] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, Benjamin/Cummings, Reading MA, 1994.

[FGD94] *Content Standards for Digital Geospatial Metadata*, Federal Geographic Data Committee, Washington, 1994.

[FJP90] J. C. French, A. Jones and J. L. Pfaltz, A Summary of the NSF Scientific Database Workshop, *IEEE Data Engineering Bulletin 13*,3 (September 1990), 55-61.

[Ful88] D. W. Fulker, The netCDF: Self-Describing, Portable Files—a Basis for Plug-Compatible Software Modules Connectable by Networks, *ISCU Workshop on Geophysical Informatics*, Moscow, USSR, August 1988.

[GAL96] D. W. Gillman, M. V. Appel and W. P. LaPlant, Jr., Design Principles for a Unified Statistical Data/Metadata System, in *8th Intern'l Working Conf. on Scientific and Statistical Database Management*, P. Svensson and J. C. French (editors), IEEE Press, Stockholm, June 1996, 150-155.

[GrW97] A. Grimshaw and W. Wulf, The Legion Vision of a Worldwide Virtual Computer, *Communications of the ACM 40*,1 (January 1997), 39-45.

[Had95] R. F. Haddleton, An Implementation of a Parallel Object Oriented Database System, TRCS-95-49, Department of Computer Science, University of Virginia, August 1995.

[Had97] R. F. Haddleton, *An Implementation of a Parallel Object Oriented Database System*, Ph.D. dissertation, Department of Computer Science, University of Virginia, Charlottesville, VA, August 1997.

[HaP97] R. F. Haddleton and J. L. Pfaltz, Client/Server Architecture in the ADAMS Parallel, Object-Oriented Database System, *Proc. 1st Intern'l Conf. in Scientific Computing in Object-Oriented Parallel Environments*, Marina del Rey, CA, December 1997, 257-266.

[Jon92] H. G. Jones, *Plants and Microclimate*, Cambridge University Press, New York, 1992. (2nd edition).

[KDG96] G. J. L. Kemp, J. Dupont and P. M. D. Gray, Using the Functional Data Model to Integrate Distributed Biological Data Sources, in *8th Intern'l Working Conf. on Scientific and Statistical Database Management*, P. Svensson and J. C. French (editors), IEEE Press, Stockholm, June 1996, 176-185.

[KGM96] L. Kerschberg, H. Gomaa, D. Menasce and J. P. Yoon, Data and Information Architectures for Large-Scale Distributed Data Intensive Information Systems, in *8th Intern'l Working Conf. on Scientific and Statistical Database Management*, P. Svensson and J. C. French (editors), IEEE Press, Stockholm, June 1996, 226-233.

[KCJ87] S. N. Khoshafian, G. P. Copeland, T. Jagodits, H. Boral and P. Valduriez, A Query Processing Strategy for the Decomposed Storage Model, *Proc. Third International Conference on Data Engineering*, Los Angeles, CA, February 1987.

[KGB90] W. Kim, J. F. Garza, N. Ballou and D. Woelk, Architecture of the ORION Next-Generation Database System, *IEEE Trans. on Knowledge and Data Engineering 2*,1 (March 1990), 109-124.

[Len94] H. Lenz, The Conceptual Schema and External Schemata of Metadabases, in *7th Intern'l Working Conf. on Scientific and Statistical Database Management*, J. C. French and H. Hinterberger (editors), IEEE Press, September 1994, 160-165.

[Les95] M. Lester, Toward a Taxonomy of Metadata: The User's Perspective, *Proceedings of SDM-92 Planning Workshop*, Salt Lake City, UT, June 1995, 85-90.

[Lor74] R. A. Lorie, XRM — An Extended (*n*-ary) Relational Memory, Tech. Rpt. G320-2096, IBM Scientific Center, Cambridge, MA, January 1974.

[MoS93] S. Monk and I. Sommervile, Schema Evolution in OODB's Using Class Versioning, *SIGMOD RECORD 22*,3 (September 1993), 16-22.

[PfF90] J. L. Pfaltz and J. C. French, Implementing Subscripted Identifiers in Scientific Databases, in *Statistical and Scientific Database Management*, Z. Michalewicz (editor), Springer-Verlag, Berlin-Heidelberg-New York, April 1990, 80-91.

[PFG92] J. L. Pfaltz, J. C. French, A. S. Grimshaw and R. D. McElrath, Functional Data Representation in Scientific Information Systems, *Intern'l Space Year Conference on Earth and Space Science Information Systems (ESSIS)*, Pasadena, CA, February 1992, 788-799. AIP Conf. Proc. #283.

[PfF93] J. L. Pfaltz and J. C. French, Scientific Database Management with ADAMS, *Data Engineering 16*,1 (March 1993), 14-18.

[Pfa93] J. L. Pfaltz, The ADAMS Language: A Tutorial and Reference Manual, IPC TR-93-003, Institute for Parallel Computation, Univ. of Virginia, April 1993.

[PHF98] J. L. Pfaltz, R. F. Haddleton and J. C. French, Scalable, Parallel, Scientific Databases, *10th International Conf. on Scientific and Statistical Database Management*, Capri, Italy, July 1998, 4-11.

[PKW96] W. M. Post, A. W. King and S. D. Wullschleger, Soil organic matter models and global estimates of soil organic carbon, in *Evaluation of Soil Organic Model Models Using Existing Long-Term Datasets*, P. Smith, J. Smith and D. Powlson (editors), Springer-Verlag, Berlin, 1996.

[ReD90] R. K. Rew and G. P. Davis, The Unidata netCDF: Software for Scientific Data Access, *6th International Conference on Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology*, Anaheim, CA, February 1990.

[Rod92] J. F. Roddick, Schema Evolution in Database Systems — An Annotated Bibliography, *SIGMOD Record 21*,4 (December 1992), 35-39.

[Rot96] J. Rothenberg, Metadata to Support Date Quality and Longevity, *1st IEEE Metadata Conference*, Bethesda, MD, April 1996.

[WSE95] F. I. Woodward, T. M. Smith and W. R. Emanuel, A global land primary productivity and phytogeography model, *Global Biochemical Cycles 9*(1995), 471-490.

[ZdM90] S. B. Zdonik and D. Maier, *Readings in Object-Oriented Database Systems*, Morgan Kaufmann, San Mateo, CA, 1990.