1994

Bambi Meets Godzilla: Object Databases for Scientific Computing

David Maier
Oregon Graduate Institute of Science & Technology

David Hansen
George Fox University, dhansen@georgefox.edu

Bambi Meets Godzilla: Object Databases for Scientific Computing

David Maier

Computer Science & Engineering
Oregon Graduate Institute
Portland, OR 97291-1000

David M. Hansen

Computer Science & Engineering
Oregon Graduate Institute
Portland, OR 97291-1000

Abstract

Object-oriented databases (OODBs) are in many ways a better match for scientific data management than conventional record-oriented database systems. User-defined datatypes reduce the encoding going from a scientific domain to the database. Direct support for complex objects is useful for capturing hierarchical structures, such as molecules. OODBs generally have collection types, such as lists and arrays, that are a better basis than sets for the dimensional data common in scientific applications. Their inherent extensibility seem a good match for handling new kinds of metadata, and having behavior definable in the database permits transparent access to existing data in multiple formats via a common object model.

We begin by recounting our experiences with using OODBs for scientific data, in the domains of computational chemistry, and materials science. The bulk of the talk, however, deals with areas that need improvement for OODBs to support scientific applications well, among them:

- *Management of massive data sets and tertiary storage*
- *Data loading and archiving*
- *Querying over ordered collection types*
- *Availability on appropriate computing platforms*
- *Application programming interfaces, particularly for FORTRAN and parallel environments*
- *Supporting data interchange formats*

1 Introduction

Scientific computing has been underserved by database systems. Many reasons come to mind for scientific applications avoiding commercial database management systems: they are not available on the right computing platforms or with APIs (application programming interfaces) for the right languages; the data model and data manipulation languages do not match scientific data structures and operations; they are tuned wrong for the access patterns of scientific applications; they lack gateways to existing data sets.

We begin with a brief discussion on why object-oriented databases might be a better match for scientific computing than record-oriented databases. The main benefit is in data model expressivity. We next consider projects that have used OODBs for scientific data management, and comment on our own experience with commercial systems. We have found a great advantage in using OODBs as "middleware" that implements a common domain model for connecting multiple applications to multiple data sources, including unmanaged data.

We then turn to the main section of the paper how OODBs could be improved for supporting scientific computing. They are far from the ideal vehicles for scientific data management currently, and we outline the most important areas for work.

2 Why Object-Oriented Databases?

We construe the term "scientific computing" broadly here beyond computational science to all uses of computers in science: experiment management and result capture, data compilation and exchange, publication and bibliographic searching, and so forth. Our own work has been directed more at support for the individual scientist than at large projects. We see such a scientist managing relatively modest amounts of data, but organized into many data sets. He or she wants to easily retrieve that data and use it with

a range of tools, both specialized to a scientific domain and more generic software, such as statistical packages, plotting routines and spreadsheets. Such an individual will frequently want to work with external data that are produced and published by others.

The main advantages that OODBs hold forth for scientific data management derive from the more expressive data model. Rather than being constrained to a predefined collection of data structures and operations, database designers can create new datatypes through a class-definition mechanism. Entities of interest can then be modeled directly with these types, rather than being encoded into record structures. In our own use, we have readily modeled a variety of scientific structures, such as molecules, basis sets, and crystal unit cells. The behavioral modeling capabilities of OODBs of course make possible a broader range of modeling, to include actions on entities and derived properties.

While not an inherent feature of an OODB, current products provide more than one collection or "bulk" type. Typically, multisets, lists and one-dimensional arrays are supported, in addition to sets. (The proposed ODMG-93 standard includes all these bulk types [4], so they are likely to appear in future commercial offerings as well.) Ordered types, particularly arrays, are largely unsupported in record-oriented models, but valuable in scientific applications for modeling dimensional data (with spatial or temporal components). Arrays are clumsy to encode and manipulate in conventional data models. The bulk type constructors in OODBs also tend to be more flexible. They can be nested, and an element can belong to multiple collections, unlike, say, the relational model where every tuple belongs to a unique relation. This latter property is useful in representing alternative groupings of specimens, classifications and working sets.

The extensibility of OODBs, through adding types, operations and objects, in principle seems well matched to keeping up with scientific models and theories as they evolve. However, until schema modification utilities become more capable, this advantage is largely illusory. Direct support of object identity in OODBs would seem to hold advantage for easily linking results to annotations or bibliographic references, but this potential is largely negated by the lack of support in current products for relationships external to objects. Metadata, both denotative and annotative, is of utmost concern to scientists, to ensure that their data is properly interpreted in the future and by others. The flexibility of object models appears to bode well for capturing metadata. However, while some OO languages (CLOS and Smalltalk) support modification of the class definition mechanism, such changes in a database context wreak havoc with query optimization and integrity constraints. Further, annotative metadata tends to be attached at the individual object level, rather than associated with a whole class.

A perhaps unexpected advantage of OODBs is their suitability for serving as "adaptors" of external datasets. While much has been written (and implemented) about DBMSs serving as gateways to data in other DBMSs, existing scientific data sets are largely *unmanaged*. That is, they exist as sets of files, not under the control of any DBMSs. Our experience, which we recount shortly, is that OODBs are an effective means to impose some management on such unmanaged data. Having a DML (data manipulation language) with general computational capabilities allows us to parse, reformat and combine external data to make it conform to a common object interface. Further, it lets us compute properties of data items if they are not explicitly stored. For example, the density of a crystal lattice can be computed from the geometry of its unit cell and a list of its atomic constituents.

We note here that it is certainly possible to use an object-oriented modeling discipline, but use a record-oriented database or even simply files as an implementation vehicle. Such is the case with an implementation of the *E. Coli* Genetic Stock Center (CGSC) database where an object-oriented model for E. Coli bacteria was implemented using a relational DBMS [16]. While some of the benefits above obtain in such approach, we see potential problems. In the cited reference, 11 "objects" were mapped into over 100 relations. This proliferation of entities bodes ill for the manageablility and efficiency of querying.

3 OODBs in Science

When we examine the literature for applications of OODBs to the management of scientific data, two things are apparent. First, the primary reason for choosing to use an OODB is the richness of the modeling constructs. Domains that work with complex data structures such as image data [5, 2], geographic data [9], experiment management [23], and protein structures [3, 11] seem to be common testbeds for OODB technology.

The second observation is that many users are still building their own OODBMS. Researchers working with the Brookhaven Protein Data Bank (PDB) have

constructed an OODBMS for managing protein structure data [11, 23]. Others have built object-oriented layers on top of a relational DBMS such as the APRIL object model for image data that is layered atop an Oracle database [2].

3.1 Our Work

We have been applying OODB technology to the scientific domains of computational chemistry and materials science [6]. The common approach in each domain is to use OODB as an integrating technology, bringing together application programs and relevant sources of data (Figure 1). A central premise

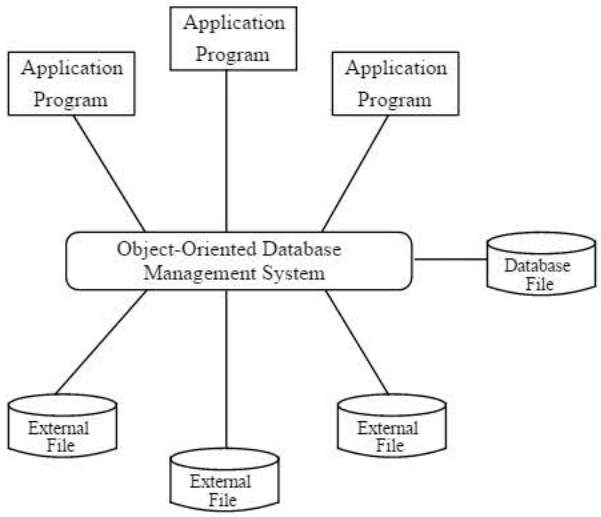

Figure 1: Integrating Programs and Data

of our work is that each domain can be modeled using a domain-specific object-oriented data model. The domain-specific model gives users and their application programs a single, unifying view of data from databases and other application programs. We note that this "hourglass" architecture, with a common object-oriented domain model through which tools can access multiple data sources, is essentially the same as that adapted by Bourne and Pu [3] for work with the Protein Data Bank.

Each of these domains imposes limits on the degree of integration due to the presence of legacy components (applications and databases). In the computational chemistry domain, we are free to create and modify the data, but the legacy applications are not necessarily developed by the scientist using them, are large and complex, and have not been structured to make replacement of I/O functions easy. Hence we have taken an approach of connecting applications to the database without recoding them. Conversely, in the materials science domain, we can modify applications to access the OODB, but legacy databases are too large to be entirely converted and loaded into the OODB, and so must be integrated in some other way. Our work has explored solutions to these integration problems as well as the general applicability of OODBs to managing scientific data.

3.1.1 Computational Chemistry

The focus of our work in computational chemistry is to integrate large, stand-alone computational chemistry "codes" using a single object-oriented data model for input and output [8]. A common data model for computational experiments provides a mechanism for sharing data among applications, and provides a uniform view of experimental results output from different programs (Figure 2).

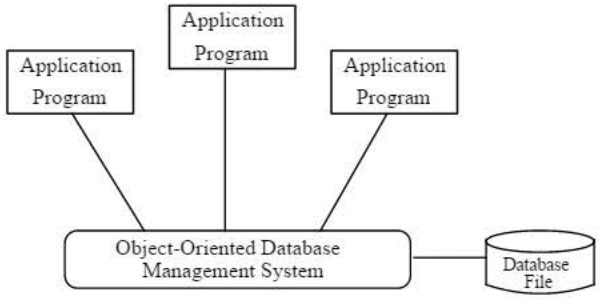

Figure 2: Program Integration

We use a "computational proxy" for specifying and conducting a computational chemistry experiment [7]. A proxy contains information about the computational chemistry code to be run and its parameters. As we are dealing with codes that are problematic to modify, they are left intact, and the proxy provides mappings to construct input files from database objects and to parse outputs into objects.

The database is used for:

Experiment set-up allowing the user to set-up a computational experiment by building a proxy and specifying the experimental parameters such as molecular structure, basis set, and computational code.

Experiment monitoring using the proxy to provide the user with near real-time information on the state and progress of a computational experiment.

Experiment analysis — letting the user browse the end results of computational experiments in a common form.

3.1.2 Materials Science

Our work in materials science has two aspects. One of the objectives is to integrate large, stand-alone databases of materials property data using a single object-oriented data model. The second objective is to develop completely new object-oriented applications and databases where the data is stored in the OODB and the applications are developed using object-oriented programming techniques and languages.

We are using an OODB to integrate and provide access to large, unmanaged databases of materials property data [13]. A space-efficient, tunable representation of the data in external files is stored within the OODB and the objects in the OODB transparently access data from external files on demand (Figure 3). We use a multi-layer architecture of objects within the OODB to hide the external data sources from the objects that users access. Data extracted from external files may be cached within the objects of the OODB, and one of the research issues being explored is how to control and manage cached data to improve query-processing.

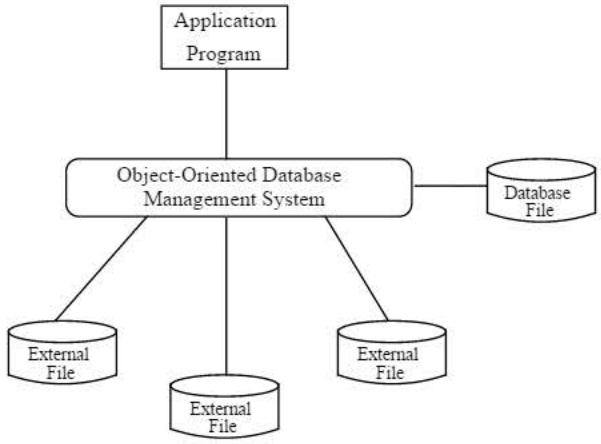

Figure 3: Data Integration

The OODB provides a common object-oriented data model through which users and their application programs access data from external files. The OODB is accessed by application programs using the language-specific API provided by the OODB and directly by users through an interactive query interface.

Another aspect of our work in materials science is to develop completely new object-oriented applications and databases. The Engel-Brewer Correlation method of calculating phase diagrams is being developed as an object-oriented application that is tightly coupled with an OODB [21]. As depicted in Figure 4, some of the application code is stored in the database as methods associated with the database objects. These methods are accessible by other applications and by interactive user queries.

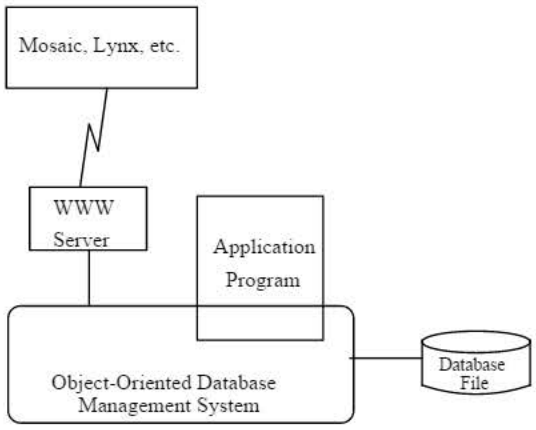

Figure 4: Object-Oriented Approach

The database contains phase diagrams (and associated meta-data) that have been calculated using the Engel-Brewer Correlation, as well as phase diagrams that have been transcribed from the literature, and phase diagrams derived experimentally. The database is currently being made available to others through a World-Wide-Web (WWW) server that provides predefined hyper-text access paths through the database. The object-oriented paradigm is providing a powerful mechanism for modeling, computing, storing, and querying a complex entity such as a phase diagram.

4 Room for Improvement

While OODBs are gaining use in support of applications in scientific computing, we see many aspects that could be improved, which we outline in the following subsections.

4.1 Massive Data Sets

The issues of massive data sets and tertiary storage management have been documented for data management systems in general [1]. Here we cover issues particular to object-oriented databases.

Support for tertiary storage means the ability to move objects from online secondary storage to nearline or offline media. OODBs that tie object identifiers to physical locations will have a hard time with that migration. Some of these systems support dividing the persistent store into multiple databases, and moving an entire database between secondary and tertiary storage should be possible. However, the idea that all objects in a database should migrate simultaneously is suspect to us. The ability to have multiple implementations of a single type suggests an approach to object movement. Using an internal representation where an object is structured as a small header plus one or more data segments, the header could remain resident in secondary storage while the main part of the object is archived. This approach requires an object potentially to change its internal structure during its lifetime, but such a capability has been implemented before.

A second problem is the treatment of schema data for objects moved to tertiary storage or archives. Restructuring all objects when a class definition is modified is no longer feasible. Class definitions will likely have to be versioned and stored with offline data. A final point is that using collections as the unit of migration is problematic when an object can participate in more than one collection. Using collections in this role seems to require the ability to determine what collections a given object is in.

4.2 Bulk Loading

The operative model of object creation for OODBs has been that new objects are created during an interactive session, and therefore the number of objects created during a transaction and their sizes are modest. This model is not borne out by the scientific user, who may want to load a large external dataset during a single sitting. Such a large chunk of data might arise as an extract from a public database, the output of a scientific simulation or be produced as a result of an experiment or observation with an instrument. Our experience is that large loads experience poor performance with current database products. The problem is probably due to a combination of factors. Some commercial OODBs have been architected towards the case where the working set of a transaction fits in main memory, which can easily fail to be the case for bulk loads. Memory management structures, such as a resident object table, might themselves grow beyond the bounds of physical memory and incur paging overhead. If object identifiers and storage space are allocated in small units, the process that manages them could be a bottleneck during load. Recovery mechanisms might not be well tuned to transactions where many objects are created. Even with amelioration of these problems, bulk loading is an innately I/O intensive task. Load utilities that take advantage of parallel writes are a route to better performance here.

4.3 Ordered Datatypes

Initial OODBs offerings were greatly influenced by the market for storage managers for CASE and CAD tools. The archetypal data structure for these applications is something like a parse tree a hierarchical decomposition of a design artifact. While scientific applications certainly posses compound hierarchic structures to represent, ordered datatypes abound, such as sequences, time series, matrices, grids and images. Ordered types are common because scientific applications often deal with some kind of coordinate space with time or distance dimensions. Here it is not just membership of a collection but its "topology" that is important. Information is encoded in the nearness of one element to another.

We have detailed the issues in supporting ordered types in a database elsewhere [18]. As mentioned earlier, commercial OODBs provide multiple bulk types, including ordered types such as lists and arrays. However, system support for such types is limited. There are limited (generally one) options for physical layout of an ordered collection instance. Within the query languages, ordered collections are treated little differently than sets or multisets, and auxiliary access methods are either absent or only support set like access.

4.4 Models and Tools

Besides ordered collections, there are other areas in which type and modeling capabilities of OODBs are limited. A common misperception is that OODBs are realizations of semantic data models (SDMs). In fact, there are great differences in the modeling capabilities supported by OODBs and SDMs. The type systems of OODBs mainly derive from those of a particular OO programming language, such as C++ or Smalltalk. These type systems are focused on the semantics of operations and support for subtyping (substitutability). Type checking of expressions is a main concern. A class or type hierarchy in an OODB falls short of being a full database schema, because it does not declare (usually) what are the named instances of objects and collections that comprise the database. SDMs, in contrast, are more state oriented, and the hierarchies of entity types they support are for constraint or clas-

sification. Much of what they define is not statically checkable.

Others have noted the limitations of OO programming languages as the basis for data modeling [10]. Few OODBs have direct support for part-of relationships or for taxonomies. While it is sometimes possible to map a taxonomy onto the class hierarchy of an OODB, such a representation presents problems. Changes or extensions to the taxonomy then mean class modification after the database is deployed. Further, querying about taxonomic relationships reduces to determining subclass relationships, which is not supported in all OO languages. Enforcement of keys is mainly lacking in OODBs. (An important point is that keys are a state-based property of a collection, and not a property of the element class. Remember that one object can be in several collections, and what would constitute a key in, say, a collection of human genome fragments, will not necessarily be a key in a multi-organism collection.) Another useful modeling feature lacking in OODBs is the ability to constrain the value of a property to lie in a particular collection (a form of referential integrity). Support for relationships is also severely limited. At best, OODBs support binary relationships represented internally to objects; ideally one wants n-way external relationships.

What seems needed here is a data modeling layer on top of the type definition layer, either as part of the OODB or provided through design tools. Whichever form is provided, it should be well enough integrated to use in formulating and processing queries.

Metadata management (in the sense of descriptive or annotative information for datasets) is not directly supported in OODB products. A good example for a starting place is the Aurora Dataserver [15], which has an extended relational model. Aurora stores metadata about dimensional datasets, and has a generic facility for annotating such datasets.

Another modeling issue is how much behavioral capability should the database be able to capture. While DML's for OODBs tend to be computationally complete, hence capable of capturing procedural knowledge, in practice they are limited. Some OODB query languages disallow methods in queries, or turn off optimization when they appear. Some OODBs impose enough performance overhead on method execution in the database to make capturing computationally intensive procedures as methods unattractive. Some database researchers have proposed providing uniform access to stored data and data that comes from simulation models [23]. However, one can meet this requirement without necessarily having the simulation models be part of database class definitions.

One of the selling points of conventional DBMSs are the development tools available that work with them, which allow certain stereotypical applications to be constructed rapidly, often with little programming. For example, almost all commercial relational DBMSs have tools for constructing data entry forms and for report generation. Database tools of high utility for scientific data management would be workflow systems specialized for experiment and laboratory management, collaboration environments and data converters to formats used by visualization systems, statistical packages and spreadsheets.

4.5 Availability

Most commercial OODB products targeted CASE and CAD tools as a major initial market. Hence, those systems were first offered on engineering workstations. While workstations are not uncommon in scientific research, personal computers are probably more prevalent, and for many, access to data from parallel processors and supercomputers is imperative. Most commercial OODBs can support access from a personal computer to a database hosted on a workstation, and some now can run in a personal-computer-only environment. However, there are still some personal computer architectures where OODB offerings are nearly absent. A few OODBs are now available on multiprocessors, but are absent on most high-end machines. This lack is understandable, as the development environment is expensive to acquire and the installed base of potential buyers is limited. Further, some OODBs have operating system dependencies and are hard to port to another OS. As OODBs do appear on parallel platforms, and interesting question is how to move beyond process parallelism in tapping the computational resources of such machines.

4.6 The Application Programming Interface

From our conversations with scientific users of computers, the most pragmatic reason that they have not attempted to use OODBs in their work is the lack of appropriate APIs (application programming interfaces). To our knowledge, no OODB product offers a FORTRAN API, which easily excludes half the possible audience. APIs from visualization and statistical analysis packages would also increase the attractiveness of OODBs. An initial prototype that connected the newS environment for data analysis and graphics to a commercial OODB [19] convinces us that such

a linkage could be made quite seamless, and might greatly enhance the ability of such an analysis tool to handle large datasets.

Besides specific languages, the form of the API is important. A model of data movement between database and application other than cursors or single message sends is needed. It should be possible to map complex structures into memory with one call to the database. In transferring a large structure, such as a multi-dimensional array, the API mechanism needs to minimize copying and crossings of procedure boundaries. For supporting data access from parallel programs, the API ought to give help with efficient data movement when the program wants to distribute a data structure in one manner and the database has it partitioned on disk in another. Such support may involve interaction between the database and a parallel compiler.

4.7 Data Interchange Formats

Data interchange formats (DIFs), such as CDF, HDF, CIF and ASN.1, were originally devised for moving data between programs and between groups of researchers, in a platform-independent file format. They are mostly self-describing, containing data element definitions along with the base data, though in some cases they make use of standard schemas. DIFs allow exchange of data structures between programs, rather than just byte streams. They do not support the exchange of objects, in that there is no behavioral component, though some groups have written object layers over them [20]. They are typically implemented as a procedure library that is linked with an application.

An interesting phenomenon is that DIFs are being used for data management, though they were intended for data exchange. Research groups keep their data in files using one of these formats, and reprogram their tools, or write adaptors, to use the data in that format. Certain bits of data management capability are appearing in the form of cataloging and browsing facilities for files in a particular DIF, and even some rudimentary ability to query over a collection of DIF files on certain attributes. Some DIFs are being extended with alternative access methods. For example, recent versions of netCDF [14] allow record-like reads and writes to files.

The question then comes, why bother with a DBMS rather than storing data in DIF files? There are a number of advantages to the latter approach. The software for accessing these files is easy to port and has been widely ported. The link libraries exist for languages of interest, for example, FORTRAN. Since the access routines are linked with the application, data access calls can read and write data directly from and to program structures. Some commercial scientific software is appearing that can access certain DIF files directly, and particular DIFs, or schemas within DIFs, are being standardized in some scientific domains. Probably most important, DIFs directly support structures of interest in scientific computing, such as multi-dimensional arrays.

The DIF-file approach to data management is not without its drawbacks, though. When the number of files grows large, a DIF offers no help in managing or grouping datasets. Sharing between DIF files is not supported the semantics is strictly copy-in, copy-out, with no notion of anything like object identity between files. While DIF files are self-describing, there is no external schema manager, hence no support to make sure that a particular dataset conforms to a predefined structure. Query facilities are primitive, if they exist at all. The program interface to DIFs operate mainly at the granularity of whole files, and are not oriented to many reads and writes of small pieces of data. Obviously, data management amenities of a DBMS are absent: concurrency control, recovery, authorization. In terms of representation, DIFs do not support alternatives for physical layout of data, nor much in the way of auxiliary access paths.

How should OODBs interact with DIFs? The minimum seems utilities for reading and writing particular DIF formats, and converting data elements to and from object classes. A more ambitious approach is to adapt an OODB to serve as a replacement for a DIF-file database. It should provide an API that includes the current procedural interface, but could go beyond that to provide query capabilities and additional access methods. For the second approach to be successful, OODBs will need to support ordered types better, for every DIF we have investigated contains either lists or arrays or both as a data structuring mechanism.

5 Conclusion

OODBs are gaining use in scientific data management, but there is a great distance between current commercial offierings and a real scientific data management system. We have listed areas in which OODBs could be improved to bring them towards the ideal. However, we are not sanguine about many of these changes happening. Scientific data management is a small market, despite large data-intensive undertakings, such as the Human Genome Project. It is not

clear that a commercial investment in these extensions could be recouped.

The best hope is that the same features will be needed for other, larger markets. Ordered types, particularly time series, have great utility for financial modeling and management. Support for image data will likely be motivated by health care. Database connections to data interchange formats will be driven by document management and manufacturing. DBMSs that operate in parallel environments will probably be driven more by decision support than scientific applications.

Bambi may never be an equal match to Godzilla, but these other areas should add a few steroids to the little fellow's diet.

6 Acknowledgements

The authors wish to acknowledge the collaboration of Ramachandran Venkatesh and Judy Cushing in preparing this paper, and the comments of Weidong Chen on an earlier draft. This work is supported in part by NSF Grant IRI-9117008.

References

[1] *Proceedings of the Massive Digital Data Systems Workshop*, Reston, Virginia, February 1994.

[2] Peter Baumann. Database support for multidimensional discrete data. In David Abel and Beng Chin Ooi, editors, *Advances in Spatial Databases*, volume 692 of *Lecture Notes in Computer Science*, pages 191 – 206. Springer Verlag, New York, June 1993.

[3] Philip E. Bourne and Calton Pu. Tools for the management of data from the protein data bank. In Wesley W. Chu, A. F. Cardenas, and Ricky K. Taira, editors, *Proceedings of the NSF Scientific Database Projects*, pages 8 15, Boston, Massachussetts, February 1993. AAAS, NSF.

[4] R. G. G. Cattell. *The Object Database Standard: ODMG - 93*. Morgan Kaufmann, San Mateo, California, 1994.

[5] Wesley W. Chu, Alfonso F. Cardenas, and Ricky K. Taira. A knowledge-based multimedia medical distributed database system – KMeD. In Wesley W. Chu, A. F. Cardenas, and Ricky K. Taira, editors, *Proceedings of the NSF Scientific Database Projects*, pages 2 – 7, Boston, Massachussetts, February 1993. AAAS, NSF.

[6] Judith Bayard Cushing, David Hansen, David Maier, and Calton Pu. Connecting scientific programs and data using object databases. *IEEE Bulletin on Data Engineering*, 16(1):9 – 13, March 1993.

[7] Judith Bayard Cushing, David Maier, Meenakshi Rao, Don Abel, D. Michael DeVaney, and David Feller. Computational proxies: Modeling scientific applications in object databases. In *Proceedings of the Seventh International Working Conference on Scientific and Statistical Database Management*, Charlottesville, Virginia, September 1994. IEEE Computer Society Press.

[8] Judith Bayard Cushing, David Maier, Meenakshi Rao, D. Michael DeVaney, and David Feller. Object-oriented database support for computational chemistry. In Hans Hinterberger and James C. French, editors, *Proceedings of the Sixth International Working Conference on Scientific and Statistical Database Management*, pages 58 76, Ascona, Switzerland, June 1992. Institute for Scientific Computing, ETH Zurich.

[9] Benoit David, Laurent Raynal, Guylaine Schorter, and Véronique Mansart. GeO_2: Why objects in a geographical DBMS? In David Abel and Beng Chin Ooi, editors, *Advances in Spatial Databases*, volume 692 of *Lecture Notes in Computer Science*, pages 264 276. Springer Verlag, New York, June 1993.

[10] Nathan Goodman. An object-oriented DBMS war story: Developing a genome mapping database in C++. In W. Kim, editor, *Modern Database Management: Object-Oriented and Multidatabase Technologies*. ACM Press, New York, 1994.

[11] P. M. D. Gray, N. W. Paton, G. J. L. Kemp, and J. E. Fothergill. An object-oriented database for protein structure analysis. *Protein Engineering*, 3(4):235 243, 1990.

[12] David Hansen, David Maier, James Stanley, and Jonathan Walpole. An object-oriented heterogeneous database for materials science. *Scientific Programming*, 1(2):115 131, Winter 1992.

[13] David M. Hansen and David Maier. Using an object-oriented database to encapsulate hetero-

geneous scientific data sources. In Jay F. Nunamaker and Ralph H. Sprague, editors, *Proceedings of the Twenty-Seventh Annual Hawaii International Conference on System Sciences*, volume III, pages 408 – 417, Maui, Hawaii, January 1994. The University of Hawaii, IEEE Computer Society Press.

[14] H. L. Jenter and R. P. Signell. NetCDF: A freely-available software-solution to data-access problems for numerical modelers. In *Proceedings of the American Society of Civil Engineers Conference on Estuarine and Coastal Modeling*, Tampa, Florida, 1992.

[15] Gregory A. Jirak. Aurora dataserver. In *IEEE Visualization '93 – Workshop on Database Issues for Data Visualization*. IEEE Computer Society Press, July 1993.

[16] S. I. Letovsky and M. B. Berlyn. Issues in the development of complex scientific databases. In Lawrence Hunter, editor, *Proceedings of the Twenty-Seventh Annual Hawaii International Conference on System Sciences*, volume V, pages 5 – 14, Maui, Hawaii, January 1994. The University of Hawaii, IEEE Computer Society Press.

[17] David Maier, Judith Bayard Cushing, David M. Hansen, George D. Purvis III, Raymond A. Bair, D. Michael DeVaney, David F. Feller, and Mark A. Thompson. Object data models for shared molecular structures. In R. Lysakowski, editor, *First International Symposium on Computerized Chemical Data Standards: Databases, Data Interchange, and Information Systems*, Atlanta, GA, May 1993. ASTM.

[18] David Maier and Bennet Vance. A call to order. In Kenneth A. Ross, editor, *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 1 – 16, Washington, DC, May 1993. ACM Press.

[19] Hitomi Ohkawa. *Object-Oriented Database Support for Scientific Data Management: A System for Experimentation*. PhD thesis, Oregon Graduate Institute of Science & Technology, Portland, OR, April 1993.

[20] Dong-Guk Shin, Rohit Gupta, and T. C. Ting. Achieving interoperability between heterogeneous object-oriented genomic databases. In Lawrence Hunter, editor, *Proceedings of the Twenty-Seventh Annual Hawaii International Conference on System Sciences*, volume V, pages 77 – 86, Maui, Hawaii, January 1994. The University of Hawaii, IEEE Computer Society Press.

[21] R. Venkatesh, David M. Hansen, James T. Stanley II, and David Maier. Applications of object database technology in thermodynamics and materials science. To appear in *Proceedings of the Symposium on Applications of Thermodynamics in the Synthesis and Processing of Materials*. Rosemont, IL, October 1994. TMS Materials Week.

[22] Sandar Walther and Richard Peskin. Object-oriented data management for interactive visualization of scientific databases. In Wesley W. Chu, A. F. Cardenas, and Ricky K. Taira, editors, *Proceedings of the NSF Scientific Database Projects*, pages 131 – 136, Boston, Massachussetts, February 1993. AAAS, NSF.

[23] Janet L. Wiener and Yannis E. Ioannidis. A Moose and a Fox can aid scientists with data management problems. In Catriel Beeri, Atsushi Ohori, and Dennis E. Shasha, editors, *Database Programming Languages*, Workshops in Computing, pages 376 – 398. Springer Verlag, New York, 1993.