

A Scalable DBMS for Large Scientific Simulations*

John L. Pfaltz
Dept. of Computer Science, Univ. of Virginia
jlp@virginia.edu

Ratko Orlandic
Dept. of Computer Science, Illinois Inst. of Tech.
ratko@cns.iit.edu

September 7, 1999

Abstract

Scientific simulations evolve rather fast. Both the logical organization of the underlying database and the scientist's view of data change rapidly. The underlying DBMS must provide appropriate support for the evolution of scientific simulations, their rapidly increasing computational intensity, as well as the growing volumes and dimensionality of scientific data. ADAMS is a dynamic and scalable academic DBMS for scientific applications designed to address the evolutionary processes and the problems of scale that have a tendency to cause system overhauls and, thereby, potentially significant disruptions in the course of scientific investigation. This paper presents innovative solutions adopted in ADAMS allowing the system to accommodate the increasing computational intensity of scientific simulations as well as the growing volumes and dimensionality of scientific data.

1 Scientific Simulations

Databases have traditionally been used to represent the state of a system with well defined parameters. Examples include accounting systems, inventory systems and personnel systems, where it is important to record minute to minute modifications to the system. The transaction concept ensures that these modifications preserve consistency— especially the underlying structure of the system.

*Research supported in part by DOE grant DE-FG05-95ER25254.

Scientific databases are quite different. First, the basic scientific observations are never changed¹, so transaction concepts are relatively unimportant. But, scientific databases can grow without bound. The data rate of some scientific experiments is measured in gigabytes, or even terabytes, per day [31]; scientific simulations, which are our particular interest, can also generate similar amounts of data. This is an important characteristic that must be accommodated. A scientific database must scale well.

Second, scientific databases may be only weakly structured. A characteristic of scientific enquiry is that we examine large collections of facts, observations and data whose structure we do not fully understand. Shifting these bits of data, aggregating some subsets of data, juxtaposing other bits with one another and generally trying to find patterns within the data which will suggest its underlying structure is one aspect of scientific study. Database systems that permit scientists to rearrange their data and view it from very different perspectives provide a powerful analytic tool.

Another aspect of science is hypothesizing an underlying structure, that is creating a model, and then re-examining the data with respect to this new structure. This second characteristic must also be accommodated. Scientific databases should support a wide range of system evolution.

A final characteristic of science lies in its tension between individual and collaborative work. Many investigations are essentially based on the data collected by a scientific team; yet, their work may rely heavily on and reference other data in a more public domain. Finally, this data which was initially private will want to be shared with others. A good scientific database should facilitate both the individual's definition and management of his own data within a protected environment; yet provide a mechanism whereby the same data can also be shared widely among colleagues. An approach that relies on a database administrator to define and manage the structure of the database will not work in a scientific context.

In the following sections, we describe a database approach which we believe can effectively represent the kinds of global change simulations we are engaged in, as well as many other scientific problems. Of the many important issues, we will address scalability in greatest detail.

2 Support for Scientific Simulations

In this section we assume that we are discussing object-oriented databases. Ever since the seminal discussion in [2] there has been mounting evidence for the superiority of the object-

¹Changing one's scientific data to fit one's model is called scientific fraud. While *never* is a bit too strong; it is only under conditions of clear recording error, or other apparent error, that one is justified in changing the data.

oriented paradigm in complex scientific enquiries. Whenever we talk about *data* we are implicitly referring instantiated objects, all of which belong to a class which defines them. Thus to us, an *observation* which records, say, the *temperature* at a specific *place* and *time*, is a datum belonging to the class OBSERVATION where *temperature*, *place*, and *time* are attributes of this OBSERVATION instance, or object.²

We would assert that certain capabilities belong in any database which supports scientific simulations. These will include support for temporal data, for data and structure evolution, and for scaling to very large data sets. Since so much has been written about temporal databases, *c.f.* [1, 29, 5], we skip over this important aspect in this paper.

Both DBMS evolution and scalability are complex problems. Under the general term of *evolution* we include all those mechanisms that a scientist may use to dynamically structure and then restructure his data. These include namespaces into which a scientist may freely insert the persistent names of new classes and new class attributes, sets of data of particular interest, or experimental multi-dimensional spaces in which to organize and view the data. Just as a programmer can freely create and name variables, procedures and classes in a program space, one should be able to name arbitrary portions of a data space. Individual scientists and/or teams should be able to name objects, create classes or modify classes in their data spaces without any administrative intervention.

The attributes associated with a data object and certain kinds of relationships with other objects are described in its *class*. But, sometimes the real life objects denoted by these simulation objects change. For example, additional readout may be obtained from an established instrument class to which a new sensor has been added. It is valuable if one can modify these class descriptions without having to recompile code or reformat very large data files. There has been a considerable literature exploring schema evolution [25], but actual implementation is not easy.

A scientific database can evolve as a result of various kinds of schema modifications, but there are other ways in which the researcher can evolve his view³ of the data. Perhaps the most basic of these is the simple aggregation of data of interest into sets. One set of objects may consist of those items satisfying a certain predicate; another set may have been defined by a clustering; while a final set may have simply been enumerated by the scientist in an interactive, one by one manner. However formed, it should be possible to put these sets aside as persistent entities and name them for later reference when one might compare them, iterate through them, or use them as base sets for other operators. Set aggregation seems basic, yet it is not available in traditional relational systems because a tuple would have to be replicated if it were to be in more than one set.

²One may informally equate objects with a relational "tuple" of values; but, in fact, the object concept is much more general.

³We are using this term in its generic sense as an *understanding*, not in the narrowly technical sense of relational *views*.

Another mechanism for organizing one's data is to linearly order it. Database systems generally provide the capability of sorting one's data with respect to any specified attribute(s). The easiest way to make this re-organization persistent is to create a secondary index over the data set. Most traditional systems provide this capability. The key is to integrate an ordering operator into the language so that it can dynamically interact with the other data organization operators, such as aggregation mentioned in the preceding paragraph and multi-dimensional space facility described below, in a graceful manner.

One of the most powerful analytic tools a scientist can use is the organization of items in an n -dimensional space with respect to independent variables. Figure 1 illustrates a representative biological space in which *leaf area index* is plotted with respect to *photon*

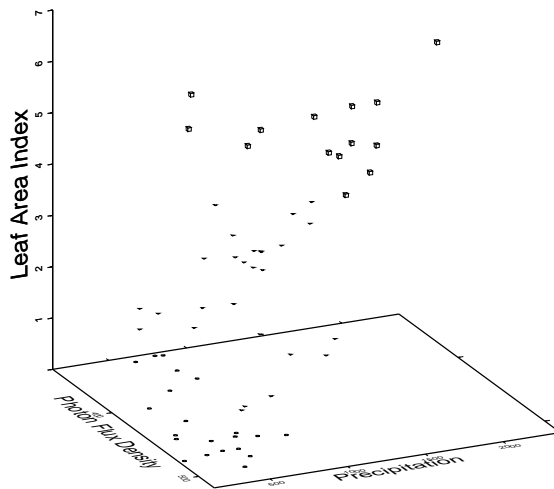


Figure 1: A plot of objects with respect to 3 numeric axes

flux density (or sunlight) and *precipitation* for 60 randomly selected land areas in the U.S. Parameter relationships are often revealed through the kinds of spatial clusters illustrated here.

Even though it is the visualization of such n -dimensional spaces that provides a researcher with the most intuition, we do not believe it is the function of a database system to generate the display itself. There is abundant graphical rendering software for this. But, maintenance of such spaces is within the purview of the database system. In particular, we believe the database system should provide for the dynamic creation of such spaces, together with an efficient population of them with objects in the database and retrieval of specified subsets of the space. One subsetting mechanism that is quite useful is a convex hull operator; others include a variety of clustering algorithms. Since, we view database retrieval as simply the identification of subsets of objects of interest, these mechanisms for

defining subsets should also be available within the database system.

Scalability can be used in two senses. Informally, it just means the ability to accommodate a wide range of problem magnitudes. In a more restricted sense, scalability implies that one can double the size of the data sets without sacrificing performance *provided* we also double the number of processors and associated resources. Such linear scalability is theoretically impossible, but very well designed systems can approach it asymptotically. Consequently, our approach to very large scientific simulation systems envisions distributed, concurrent processing [11].

Scalability and evolution are not orthogonal. While some data sets simply grow in size over time, others grow because of evolution. Creating new object classes, or only adding new attributes to an existing class, forces system growth. Scaling to handle this kind of data volume require incremental additions of new processing power and some form of data migration.

Figure 1 illustrates a 3-dimensional data space. But, evolution permits the creation of arbitrary n -dimensional spaces for display or other analytic purposes. However, representation of higher dimensional spaces is computationally much more complex. Similarly, one may want to partition a data set into many more, but smaller, subsets. This entails significantly more overhead. Both kinds of situation illustrate what we call *dimensional scalability*.

Evolution and scalability are fundamental to support scientific simulation.

3 ADAMS Support for Scalability

Under a grant from the Department of Energy, the authors are using component design [16, 15] to develop a database system called ADAMS (Advanced DATA Management System) that will incorporate the capabilities of the preceding section. Its design is specifically targeted to scientific applications, especially scientific simulations. In particular, it will be scalable to accommodate very large data sets, and provide a variety of data organization capabilities, including aggregation, linear ordering and multi-dimensional clustering. In this short paper we will only sketch those aspects which are most relevant to issues of scalability.

3.1 Scaling to Increase Processing Power

Can an object oriented database with abundant links between classes be effectively implemented as a distributed, concurrent system? To answer this fundamental question, we have already created a parallel version of ADAMS as described in [11, 23]. In this implementation, objects are distributed to one of the n sites solely on the basis of their *oid* (unique object identifier) in what amounted to a nearly round robin manner. The sequential script constituting the client's program is then broken into n multiple and independent threads,

each running on a separate processor. Relatively long operator sequences involving set, retrieval, and update operators can be executed without synchronization, provided they do not involve references (links) to objects at other sites. When such cross references (which are equivalent to relational joins) are encountered, it is only object *oid*'s which need to be communicated over the network. Attribute values are never transmitted unless requested by the client for display purposes. Keeping message traffic to a minimum is crucial for a successful distributed implementation.

Repeated tests of ADAMS were performed on shared-nothing configurations using 1, 2, 4, or 8 Sun SPARC20 processors, each with an attached disk, and all interconnected on an Ethernet LAN. This parallel execution yielded significant speed up (as one might expect). More importantly, the system demonstrated nearly linear *scale up*, as seen in Figure 2. Linear scale up occurs when

$$time_n(n \cdot task_size) / time_1(task_size) = 1$$

where $time_1$ and $time_n$ denote the time to perform the task on 1 and n processor systems respectively, The plot indicates the average time of 10 independent executions of the query

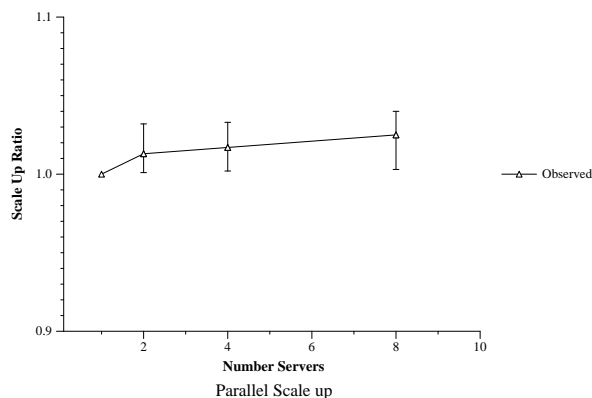


Figure 2: Scale up of a moderately complex ADAMS query

shown below, with the error bars indicating the variance.

To give you a slight flavor of the syntax of the ADAMS language, the query generating the performance shown in Figure 2 is:⁴

```
result <- x in observations |
          p1 <= x.pressure <= p2 and
```

⁴Queries in ADAMS are simply expressions of the form $\{x \text{ in } some\ set | P[x]\}$, where $P[x]$ is any first order predicate with free variable x . The query expression may then be assigned to a set, as in this case, or used wherever a set can be used, as in iteration statements.

```
t1 < x.temperature < t2 or
x.instrument.elevation > 1000
```

The final term of the query is an *implicit join* involving a link from objects in the OBSERVATION class to objects in an INSTR_PACKAGE class. Data volume consisted of 250K objects in the single processor configuration; 500K objects with 2 processors; and 1M, 2M in the 4 and 8 processor configurations respectively. Query size (the number of objects satisfying the query expression) was 76,430 objects in the 1M object database of the 4 node configuration. To illustrate the importance of network message traffic, no more than 65 data messages were sent by any single processor in the course of identifying these 76 thousand objects. When we doubled the database size on the 8 node configuration, the same query yielded 149,473 objects in nearly the same time.

3.2 Scaling to Handle Data Volume

An important characteristic of scientific simulations is the unpredictable volume of data used and/or generated by the simulations. In the initial stages of its evolution, a simulation typically begins with a small data volume. However, as the need for scientific accuracy increases over time, so does the volume of statistically-viable data that must be considered or produced. Typically, the volume of data that is stored in the underlying database grows far beyond the initial expectations.

To deal with the problem of growing data volumes of scientific simulations, ADAMS is designed to allow distributed processing in wide-area networks and support an on-line gradual migration of data across the network. Present implementation of ADAMS supports distributed processing with full location transparency, while data migration will be supported in the next version. The migration scheme, described below, will enable an asynchronous log-based migration of data (while the updates and forward processing on the migrated collection still go on) and ensure a high-degree of fault tolerance.

The processes of wide-area distributed processing and data migration determine the global architecture of ADAMS illustrated in Figure 3. The architecture assigns each site in the network a *distribution server* (DIS) and a *data server* (DAS). The distribution server maintains a complete replica of the data dictionary (global data dictionary) indicating the whereabouts of each dictionary object. The data server at each site maintains only a portion of the data space depicted in its local dictionary. Each client communicates with the nearby distribution server that coordinates the execution of the client's requests and is responsible for global query processing. Local processing against the objects named in the request is done at the data servers maintaining the objects.

In most general terms, the process of data migration applies the following protocol. Upon receiving the information from the local data server (called DAS1 in Figure 3) that a

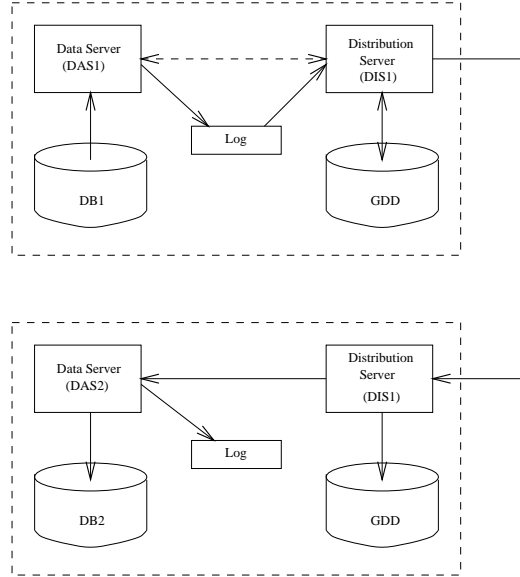


Figure 3: Data Migration in ADAMS.

certain threshold of its data volume has been reached, the distribution server DIS1 decides which data collection should be migrated and where. Following the decision, DIS1 informs the distribution server at the targeted site (DIS2) of the pending migration and initiates the process by sending the request to the data server DAS1 to begin the migration of the indicated data collection. DIS2 orders its local data server DAS2 to create the targeted collection in its space, while DAS1 initiates an asynchronous activity that reads the items of the source collection and puts each in its roll-forward log. DIS1, which monitors the log, picks up from the log the items intended for migration and forwards them to DIS2 one element at a time. Acting as a client of DAS2, DIS2 propagates the items to DAS2, which, in turn, inserts them into the targeted collection of its database as usual.

After all items of the source collection have been transmitted, the data server DAS1 inserts into its log a special “message” informing DIS1 of the end of migration and begins purging the source collection from its database as a background activity. (In the present protocol, following this action, DAS1 fails all requests involving the source collection.) Upon receiving the message, DIS1 indicates the new location of the migrated aggregate object in its dictionary and broadcasts the change to all distribution servers in the network.

The process of migration is very similar to that of asynchronous replication [30], but there are several important differences. On one hand, this data migration is a simpler process than asynchronous replication in that its protocol need not be concerned with

the resolution of conflicting updates on different replicas [30]. On the other hand, unlike asynchronous replication, the process of migration has a sharp endpoint which, in principle, should be handled as an atomic action of very short duration. Our present protocol incurs a certain span of time in which the updates on the migrated collection are simply rejected. Better solutions of handling the end of a migration process are possible, and they will be investigated in the future.

3.3 Scaling to Accommodate High Dimensional Spatial Data

Multi-dimensional spaces can be implemented in several different ways for scientific research. Multi-dimensional arrays are employed in statistical analyses (see [26, 13] for two good treatments). For the most part, these assume relatively dense arrays with most cells filled. We are more interested in sparse multi-dimensional spaces. These have been widely investigated; grid files [14, 12], *kdB*-trees and R^* -trees [24, 10, 3] are representative examples. Both offer substantial capabilities. But, as data availability has increased, so has the dimensionality of problems to be solved. Like many other advanced database applications (e.g. information retrieval and data mining), scientific simulations must often cope with data sets of very high dimensionality. An ideal *spatial access method* (SAM) [8] for these applications must support queries against both points and regions in these high-dimensional spaces. Unfortunately, traditional SAMs impose a practical limit on the number of dimensions, which is typically very low.

All three groups of contemporary spatial access methods (region-overlapping, clipping, and transformation schemes) [8, 27] suffer from major conceptual flaws that have a tendency to aggravate as the number of dimensions grows. Region overlap (percentage of space covered by more than one rectangle) in R -trees [10] and its variants [3, 17, 4] require the traversal of multiple index paths, which increases the number of disk accesses. As demonstrated in [4], overlap in these schemes increases with the growing number of dimensions. Object clipping [9] creates multiple index entries for a single spatial object, which increases the size of the structure. With the increased storage overhead, the search performance deteriorates, because a greater number of index pages must be traversed to answer a spatial query. Since the probability of clipping an object increases with the growing number of dimensions, these adverse effects of clipping become more pronounced in high-dimensional spaces. In addition to numerous other problems [8], object transformation [27] doubles the size of index entries, thus reducing the fan-out of index pages and increasing the size of the structure.

By employing an original spatial access method, called the *QSF*-tree [32], ADAMS will be able to support multi-dimensional spaces and scale well to the increasing dimensionality of scientific data. This novel structure was carefully designed to eliminate the problem of region overlapping without object clipping or object transformation. Unlike the region-

overlapping schemes [10, 3, 4], which represent object MBRs (minimum bounding rectangles) as an ordered list of intervals projected on individual axes of the multi-dimensional space, *QSF*-trees represent each MBR with its two opposing vertex (point) vectors. However, unlike spatial access methods that apply endpoint transformation of MBRs [8], MBRs are not mapped to points in higher-dimensional space. Instead, *QSF*-trees apply an original query transformation calculating search and filtering regions from the given query and using these to search the tree and filter the result set [32].

The index structure of *QSF*-trees is a simple modification of a point access method [24, 28] indexing the low endpoints of the MBRs and using their high endpoints to filter the result set once the search reaches the lowest level of the tree. The underlying point access method determines the complexity of updates of *QSF*-trees, which is typically much lower than the update complexity of a traditional SAM. Furthermore, because typical point access methods partition spatial universe into non-overlapping regions, the problem of region overlap is automatically eliminated. As a result, *QSF*-trees enable significant improvements in the performance of search operations and adapt gracefully to higher dimensions.

Another important characteristics of *QSF*-trees is that they fully discriminate search operations involving different topological relations. Each query with a different topological relation [32] is mapped onto a different (search-window, search-predicate) pair used in traversing the upper-levels of the tree. This mapping allows greater selectivity of search operations at the upper levels, which further increases the performance of spatial selections.

We have conducted an extensive set of experiments to verify the search performance of *QSF*-trees implemented as modifications of *kdB*-trees [24]. The performance was measured in terms of the number of page accesses. For an average query, *QSF*-trees outperform *R**-trees [3] by up to 65% for 2-dimensional and 80% for 4-dimensional objects. They also outperform the topological *R**-trees (an improvement over the original *R**-trees) [17] by up to 40% and 70% for 2- and 4-dimensional objects, respectively [32]. More recent experiments show that the performance improvements over both the original and topological *R**-trees increase even further with the growing number of dimensions of the spatial universe. Figure 4 shows the relative performance of *QSF*-trees and topological *R**-trees in 2- to 16-dimensional spaces for an average spatial query performed on sets of 65,536 randomly-generated objects. Relative to the length of the universe along each axis, the lengths of object MBRs along the axis were between (a) 0.1% and 0.5% (*small objects*), (b) 15% and 30% (*large objects*), and (c) 0.5% and 30% (*widely-varying objects*).

4 Present Capabilities

For most of this paper we have been describing features that we believe belong in a database that will support scientific simulation and indicating how they will be incorporated into the

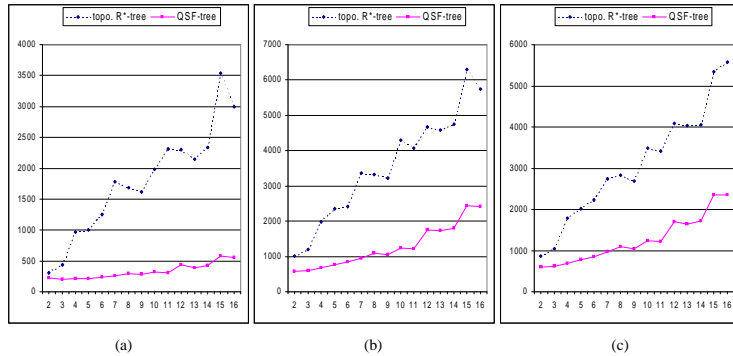


Figure 4: Average number of page accesses per query for (a) small, (b) large, and (c) widely-varying objects with different dimensionality.

ADAMS system. But, not all of these features are just design concepts; many of them already exist. ADAMS is a working language [21, 19] which has been used to implement non-conventional database applications for more than 8 years. Some of its more interesting applications have been a representation of both friendly and enemy forces in a military C^3 system [20], a mamography database for breast cancer research, and an interactive data repository for a large scale global change simulation conducted by a colleague in Environmental Science [6, 7], and innumerable student projects. It also supported the very popular "Oracle of Bacon" web site, that was cited by *TIME* magazine as one of the 10 best in 1998.

Among the features that already exist in ADAMS, and on which our project will be building include: a practical, dynamic **namespace** with multiple levels of visibility that support both private and shared names [18]; complete, dynamic **set** manipulation facilities which support both the temporary and persistent aggregation in to various set configurations; and **dynamic schema evolution** which supports the modification of class declarations "on the fly" with no need for recompilation [22]. This latter capability has proven itself repeatedly in the design and implementation of large scientific databases. Almost never does one get the design of a scientific database exactly right. The ability to change an object's class structure after the entire database has been populated is invaluable.

ADAMS can be implemented as a distributed, parallel system, as demonstrated in [11, 23]. We have emphasized the importance of database scalability in large scientific applications. The underlying structure of an ADAMS database is scalable in a nearly linear fashion. Consequently, although our project to develop an effective database architecture for scientific applications is quite ambitious, it is at least being constructed over a proven, working foundation.

References

- [1] Khaled K. Al-Taha, Richard T. Snodgrass, and Michael D. Soo. Bibliography on Spatiotemporal Databases . *SIGMOD RECORD*, 22(1):59–67, Mar. 1993.
- [2] Malcolm Atkinson, Francois Bancilon, David DeWitt, Klaus Dittrich, David Maier, and Stanley Zdonik. The Object-Oriented Database System Manifesto. In *Proceedings of the First International Conference on Deductive and Object-Oriented Databases.*, Kyoto, Japan, December 1989.
- [3] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R^* -tree: An Efficient and Robust Access Method for Points and Rectangles . In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 322–331, 1990.
- [4] S. Berchtold, D.A. Keim, and H. Kriegel. The X -tree: An Index Structure for High-Dimensional Data. In *Proc. 22th Int. Conf. on Very Large Data Bases*, pages 28–39, 1996.
- [5] James Clifford, Curtis Dyreson, Tomas Isakowitz, Christian S. Jensen, and Richard T. Snodgrass. On the Semantics of "Now" in Databases . *TODS*, 22(2):171–214, June 1997.
- [6] W. R. Emanuel, A. W. King, and W. M. Post. A Dynamic Model of Terrestrial Carbon Cycling . In M. Heimann, editor, *The Global Carbon Cycle*. Springer-Verlag, Berlin, 1994.
- [7] William R. Emanuel. Modeling carbon cycling on disturbed landscapes. *Ecological Modelling*, 89:1–12, 1996.
- [8] V. Gaede and O. Gunther. Multidimensional Access Methods. *ACM Computing Surveys*, 30(2):170–231, 1998.
- [9] O. Gunther and J. Bilmes. Tree-Based Access Methods for Spatial Databases: Implementation and Performance Evaluation. *IEEE Trans. Knowledge and Data Engineering*, 3(3):342–356, 1991.
- [10] A. Guttman. R -trees: A Dynamic Index Structure for Spatial Searching. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 47–54, 1984.
- [11] Russell F. Haddleton and John L. Pfaltz. Client/Server Architecture in the ADAMS Parallel, Object-Oriented Database System. In Yutaka Ishikawa, Rodney Oldenhoef, John Reynders, and Merydell Tholburn, editors, *Proc. 1st Intern'l Conf. in Scientific Computing in Object-Oriented Parallel Environments*, Lecture Notes in Computer Science, #1343, pages 257–266, Marina del Rey, CA, Dec. 1997. Springer-Verlag.
- [12] Hans Hinterberger, Kathrin Anne Meier, and Hans Gilgen. Spatial Data Reallocation Based on Multi-dimensional Range Queries. In *Seventh International Working Conference on Scientific and Statistical Database Management*, pages 228–239, Charlottesville, Virginia., September 1994.
- [13] Leonid Libkin, Rona Machlin, and Limsoon Wong. A Query Language for Multidimensional Arrays: Design, Implementation, and Optimization Techniques . In *1996 ACM SIGMOD Conf.*, pages 228–239, Montreal, Quebec, June 1996.

- [14] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The Grid File: An Adaptable, Symmetric Multikey File Structure . *TODS*, 9(1):38–71, Mar. 1984.
- [15] Ratko Orlandic. Introducing MESSIA: A Methodology for Developing Software Architectures Supporting Implementation Independence . In *First Working IFIP Conf. on Software Architecture, WICSA1*, San Antonio, TX, Feb. 1999.
- [16] Ratko Orlandic. Methodical Design of Mismatch-Free Architectures for Information Management . In *Proc. 5th Int. Conf. on Information Systems Analysis and Synthesis, ISAS'99*, Orlando, FL, Aug. 1999.
- [17] D. Papadias, Y. Theodoridis, T. Sellis, and M.J. Egenhofer. Topological Relations in the World of Minimum Bounding Rectangles: A Study with *R*-trees. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 92–103, 1995.
- [18] John L. Pfaltz. Programming over a persistent data space. Technical Report IPC TR-92-008, Sept. 1992.
- [19] John L. Pfaltz. The ADAMS language: A tutorial and reference manual. Technical Report IPC TR-93-003, Institute for Parallel Computation, Univ. of Virginia, Apr. 1993. At <http://www.cs.virginia.edu/adams>.
- [20] John L. Pfaltz and James C. French. Representation of enemy dispositions. JTFO Report, Royal Signals Estab., Great Malvern, UK, Mar. 1993.
- [21] John L. Pfaltz and James C. French. Scientific database management with ADAMS. *Data Engineering*, 16(1):14–18, Mar. 1993.
- [22] John L. Pfaltz, James C. French, Andrew S. Grimshaw, and Rodney D. McElrath. Functional data representation in scientific information systems. In *Intern'l Space Year Conf. on Earth and Space Science Information Systems (ESSIS)*, pages 788–799, Pasadena, CA, Feb. 1992.
- [23] John L. Pfaltz, Russell F. Haddleton, and James C. French. Scalable, Parallel, Scientific Databases. In *10th International Conf. on Scientific and Statistical Database Management*, pages 4–11, Capri, Italy, July 1998.
- [24] J.T. Robinson. The kdB Tree: A Search Structure for Large Multidimensional Dynamic Indexes. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 10–18, 1981.
- [25] John F. Roddick. Schema Evolution in Database Systems — An Annotated Bibliography . *SIGMOD Record*, 21(4):35–39, DEC 1992.
- [26] Kent E. Seamons and Marianne Winslett. Physical Schemas for Large Multidimensional Arrays in Scientific Computing Applications. In *Seventh International Working Conference on Scientific and Statistical Database Management*, pages 218–227, Charlottesville, Virginia., September 1994.
- [27] B. Seeger and H.P. Kriegel. Techniques for Design and Implementation of Efficient Spatial Access Methods. In *Proc. 14th Int. Conf. on Very Large Data Bases*, pages 10–17, 1988.
- [28] B. Seeger and H.P. Kriegel. The Buddy-tree: An Efficient and Robust Access Method for Spatial Data Base Systems. In *Proc. 16th Int. Conf. on Very Large Data Bases*, pages 590–601, 1990.
- [29] Arie Segev, Christian S. Jensen, and Richard T. Snodgrass. Report on the 1995 Intern'l Workshop on Temporal Databases . *SIGMOD Record*, 24(4):46–60, Dec. 1995.
- [30] D. Stacy. Replication: DB2, Oracle, or Sybase? *SIGMOD Record*, 24(4):95–101, 1995.
- [31] W. Tang, T. Dunning, D. Leith, and W. McCurdy. DOE Strategic Simulation Plan: Basic Science Component . Dept. of Energy, Oct. 1998.
- [32] Byunggu Yu, Ratko Orlandic, and Martha Evans. Simple QSF-Trees: An Efficient and Scalable Spatial Access Method . In *8th Int. Conf. on Information and Knowledge Management, CICK'99*, Kansas City, MO, 1999.