Enabling Interoperation of High Performance, Scientific Computing Applications:

Modeling Scientific Data With The Sets & Fields (SAF) Modeling System

Mark C. Miller[1], James F. Reus[1], Robb P. Matzke[1], William J. Arrighi[1],
Larry A. Schoof[2], Ray T. Hitt[2] and Peter K. Espen[2]

[1]Lawrence Livermore National Laboratory, 7000 East Ave, Livermore, CA. 94550-9234
{miller86, reus1, matzke1, arrighi2}@llnl.gov
[2]Sandia National Laboratories, PO Box 5800 Albuquerque, NM 87185
{laschoo, rthitt, pkespen}@sandia.gov

Abstract. This paper describes the *Sets and Fields* (SAF) scientific data modeling system. It is a revolutionary approach to interoperation of high performance, scientific computing applications based upon rigorous, math-oriented data modeling principles. Previous technologies have required all applications to use the same data structures and/or meshes to represent scientific data or lead to an ever expanding set of incrementally different data structures and/or meshes. SAF addresses this problem by providing a small set of mathematical building blocks—sets, relations and fields—out of which a wide variety of scientific data can be characterized. Applications literally *model* their data by assembling these building blocks. A short historical perspective, a conceptual model and an overview of SAF along with preliminary results from its use in a few ASCI codes are discussed.

1 Introduction

This paper describes the *Sets and Fields* (SAF, pronounced "safe") scientific data modeling system: a revolutionary approach to storage and exchange of data between high performance, scientific computing applications. It is being developed as part of the *Data Models and Formats* (DMF) component of the *Accelerated Strategic Computing Initiative* (ASCI) [15].

ASCI-DMF is chartered with developing a suite of software products for reading, writing, exchanging and browsing scientific data and for facilitating interoperation of a diverse and continually expanding collection of scientific computing software applications. This amounts to solving a very large scale scientific software integration problem. Small scale integration has been addressed by a number of existing products: SEACAS[20], PACT[4], Silo[18], Exodus II[19], CDMLib[1], netCDF[16], CDF[12], HDF[11],[3] and PDBLib[4] to name a few. There are many others. Numerous isolated successes with these products on the small scale lead many to believe any one is sufficient for the large scale. It is merely a matter of comming to agreement. However, because these products force applications to express their data in terms of handfuls of data structures and/or mesh types, this has never been practical. Consequently, these approaches have succeeded in integration on the small scale but hold little promise for the large scale.

The key to integration and sharing of data on the large scale is to develop a small set of building blocks out of which descriptions for a wide variety of scientific data can be constructed. Each new and slightly different kind of data involves the use of the same building blocks to form a slightly different *assembly*, to literally *model* scientific data.

To achieve this, the building blocks must be at once, primitive and abstract. They must be primitive enough to model a wide variety of scientific data. They must be abstract enough to model the data in terms of *what* it represents in a mathematical or physical sense independent of *how* it is represented in an implementation sense. For example, while there are many ways to represent the airflow over the wing of a supersonic aircraft in a computer program, there is only one mathematical/physical interpretation: a field of 3D velocity vectors over a 2D surface. This latter description is immutable. It is independent of any particular representation or implementation choices. Understanding this *what* versus *how* relationship, that is *what* is represented versus *how* it is represented, is key to developing a solution for large scale integration of scientific software.

These are the principles upon which SAF is being developed. In the remaining sections, we present a brief history of scientific data exchange, an abstract conceptual view of scientific data, the data modeling methodology developed for and implemented in SAF and preliminary results from integration of SAF with some ASCI applications.

2 Historical Perspective

In the early days of scientific computing, roughly 1950 - 1980, simulation software development at many labs invariably took the form of a number of software “stovepipes”. Each big code effort included sub-efforts to develop supporting tools for visualization, data differencing, browsing and management. Developers working in a particular stovepipe designed every piece of software they wrote, simulation code and tools alike, to conform to a common representation for the data. In a sense, all software in a particular stovepipe was really just one big, monolithic application, typically held together by a common, binary or ASCII file format. Data exchanges across stovepipes were achieved by employing one or more computer scientists whose sole task in life was to write a conversion tool called a *linker* and keep it up to date as changes were made to one or the other codes that it linked. In short, there was no integration. Furthermore, there was duplication of effort in the development of similar tools for each code.

Between 1980 and 2000, an important innovation emerged, the general purpose I\O library. In fact, two variants emerged each working at a different level of abstraction. One focused on computer science objects such as arrays, structs and linked lists. The other focused on computational modeling objects such as structured and unstructured meshes and zone- and node-centered variables. Examples of the former are CDF, HDF and PDBLib (early 80’s). Examples of the latter are EXODUS (1982), Silo (1988) and CDMLib (1998).

Unfortunately, both kinds of I/O libraries focused more upon the *how*s of scientific data representation than the *what*s. That is, they characterize *how* the data is represented in terms of arrays, structs and linked-lists or in terms of structured and unstructured meshes with zone- and node-centered variables. To use these products across the gamut of scientific data, a set of conventions in *how* these objects are applied to represent various kinds of data must invariably be adopted. And, without a lot of cooperative effort,

different organizations wind up using a different set of conventions for the same kinds of data. Worse still, for many kinds of scientific data, both qualitative and quantitative information can be completely lost in the resultant representations.

A related innovation that emerged during this same time is also worth mentioning, the visualization tool-kit. Examples are AVS[21], DX[14] and Khoros[13]. While these products were developed primarily for visualization, they included functionality to exchange scientific data between software modules via common data representations. From this point of view, most notable among these is DX for its implementation of a data model based upon abstract, mathematical building blocks [14], [5].

Large scale integration of scientific computing software requires a *common abstraction* capable of supporting a wide variety of scientific computing applications. An appropriate abstraction is the continuous mathematical setting from which the majority of scientific computing software is ultimately derived.

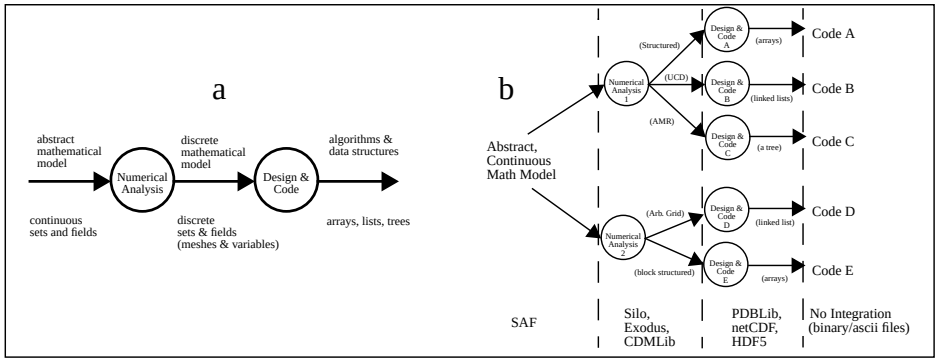

Fig. 1 a) typical scientific computing application design process, b) example of different codes that result from different decisions in each phase of the design

In Fig. 1a[8], we illustrate the basic stages in the development of a scientific computing application. The process begins with an abstract mathematical characterization of the problem or class of problems to be solved. From there, a thorough numerical analysis is performed resulting in a discrete approximation to the original continuous problem or problem class. Invariably, this results in particular kinds of meshes and interpolation schemes for dependent variables defined over them. Next, the software design and coding stage is entered resulting in a set of data structures and software components implementing the numerical model. At each stage, there are numerous design and implementation decisions to be made, each resulting in a different implementation of a solution for the same problem or problem class. This is illustrated in Fig. 1b.

Of course, the most important point of Fig. 1b is that the only salient feature of the data that is truly a *common abstraction* is the abstract mathematical setting from which all the different implementations are derived. This is the only context which is immutable across all design and implementation decisions. Unfortunately, historically all of the rich mathematical information necessary to support this abstraction is routinely lost, left in design documents, programmer's notebooks or the heads of the computational engineers. This is so because there has never existed an I/O library that allowed software developers to model their data in these fundamental, abstract, mathematical terms.

3 The Abstract Mathematical Setting: Fields

SAF is designed to provide the abstract mathematical setting alluded to in the previous section. It is based upon representing *fields*[8]. Fields serve as the standards for real (or imagined) observables in the real, continuous world. In terms of scientific computing, fields are the dependent variables of a simulation. In fact, a field is really just a generalization of a *function*. Like functions, fields are specified in three logically distinct parts: something akin to the domain of a function called the *base-space*, something akin to the range of a function called the *fiber-space* and a *map* that relates points in the base-space to points in the fiber-space. Furthermore, both the base-space and fiber-space are infinite point-sets with topological dimensions called the *base-dimension* and *fiber-dimension*. This terminology comes from the mathematics of *fiber-bundles*[5].

Next, if a field is intended to represent a real (or imagined) observable in the real, continuous world, then it has infinitely precise value at an infinite number of points. How do we represent it on a computer? More importantly, how do we formulate a representation that is common across many scientific computing applications? One common representation is given in Eq.1[9]

$$F(x) = \sum_i f_i b_i(x) \tag{1}$$

where we represent a continuous field, $F(x)$, by a *basis-set*, $\{b_i(x)\}$ and a *dof-set*, $\{f_i\}$. We call the f_is *degrees of freedom* or *dofs*[8]. To call them *values* would imply that they are in fact *equal* to the field for certain points, *x*, and this is only true when the basis functions are *interpolating*. They are most appropriately viewed as the degrees of freedom in the representation of the field.

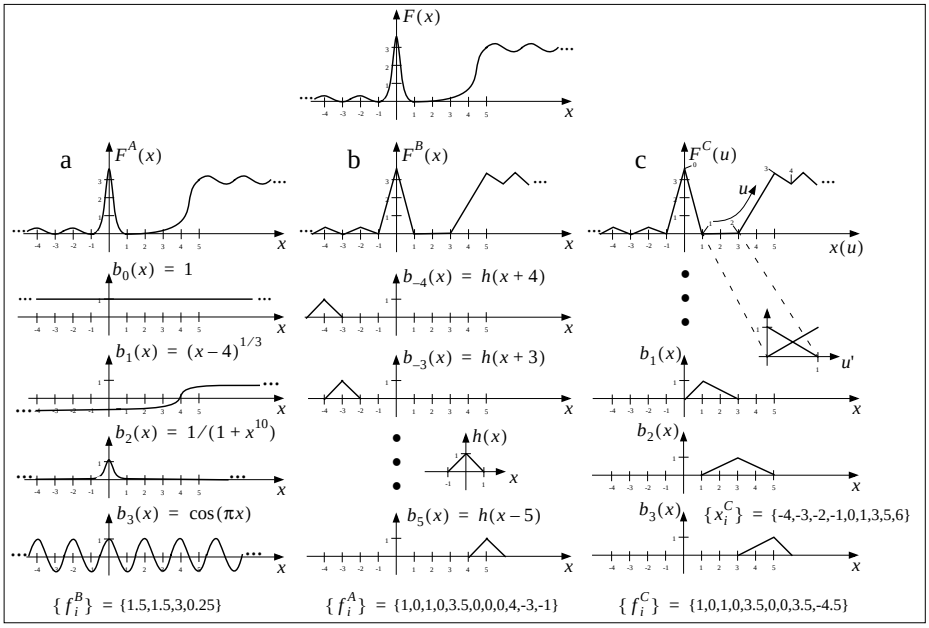

Fig. 2 An arbitrary one dimensional field and possible representations.

Three examples are illustrated in Fig. 2. In this figure, the horizontal axes represent the base-space. The vertical axes represent the fiber-space and the wiggly lines we've plotted represent the map between the two. While there is much to be learned by comparing these examples in detail, we will only touch on a few key points.

First, Fig. 2c introduces a level of indirection by re-parameterizing on a new base-space, *u*. In some sense, *u* represents the infinite point-set *along* the wiggly line itself, as opposed to either axis. On *u*, even the independent variable, *x*, is a field. It is a special field because it is a *coordinate field*. Nonetheless, this illustrates the point that even the independent variable(s) of a simulation are fields like any other field. This also means that apart from the fields *F* and *x*, *u* is nothing more than a topological space. And, statements about *u* and its various subsets are concerned solely with set-algebraic relationships of infinite point-sets. Next, observing that *u* is broken into subsets each of which is independently mapped onto an elemental base-space, *u'*, leads to another key point in modeling a field, the *subset inclusion lattice* or *SIL*[8].

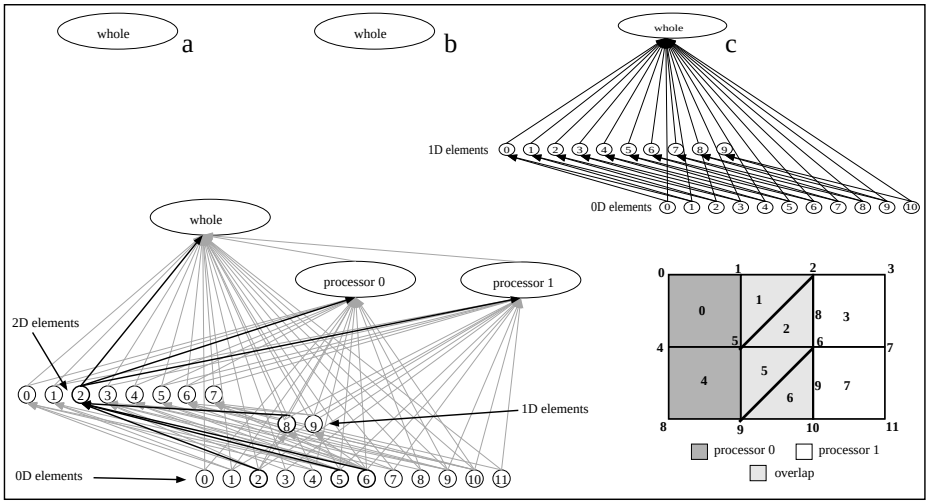

Fig. 3 The SILs in the upper half of the figure are for each of the example fields in Fig. 2. The SIL in the lower half is drawn to highlight relationships for element 2 of the 2D mesh at right.

The SIL is a directed graph, indicating how various subsets of the base-space are related. Examples of SILs are illustrated in Fig. 3 where an ellipse represents an infinite point-set and an arrow represents the fact that the point-set at its tail is a subset of the point-set at its head. Fig. 3c illustrates the SIL for the field of Fig. 2c. The whole one dimensional base-space is decomposed into ten one dimensional pieces. Each of these pieces is, in turn, bounded by two zero dimensional pieces. These pieces are often referred to, respectively, as the *zones* and *nodes*. However, a better terminology is *N dimensional element* where *N* is the base-dimension of an elemental piece. For commonly used elements such as points, edges, faces and volumes, *N* is 0, 1, 2 or 3, respectively. The lower half of Fig. 3 represents the SIL for a simple, 2 dimensional mesh composed of quad, triangle and a couple of edge elements as well as its decomposition into different processor pieces.

Across all possible representations for a field, there are subtle trade-offs between how a representation is split between the dof-set, basis-set and SIL. There is probably some *conservation of information* principle at work. To see this, observe that the basis-set in Fig. 2a is rather complicated, involving rational polynomials and trigonometric functions while the dof-set is small. The basis-set in Fig. 2b involves only a single function, *h(x)*, shifted about integral positions on the *x* base-space. The dof-set is larger. And, in both of these cases, the SIL contains only a single set. Further comparison of Fig. 2b with Fig. 2c. reveals that while both are piecewise linear representations, in Fig. 2c, we have an additional set of numbers, $\{x_i^C\}$. These can be viewed either as an entirely new dof-set for the independent variable, *x*, as a field defined on the *u* base-space or as parameters that control the shape of the member functions of a basis-set for the field *F* defined on the *x* base-space. Ultimately, the difference is determined by how the basis-set is defined. If it is defined on the *x* base-space with member functions such as $b_1(x)$, $b_2(x)$ and $b_3(x)$ in Fig. 2c, then the SIL contains just one set for the whole base-space. If, on the other hand, the *u* base-space is decomposed into elemental subsets, *u'*, then the basis-set contains just two functions, *u'* and *1-u'*, and the SIL is more involved because it must specify how these pieces fit together. This latter view is illustrated in Fig. 3c. Finally, the field in Fig. 2b could be characterized as a special case of that of Fig. 2c when $u=x$ and $\{x_i^B\} = \{i\}$. Again, the difference is in the basis-set.

3.1 The Data Model Enables Development of Set and Field Operators

There is much more to modeling fields than there is room to discuss here. Furthermore, the ability to model fields is only the first, small step. Of much greater importance is the plethora of operations on sets and fields such a data model enables. Therefore, we conclude this section by postulating a number of these operations[8].

With such a model, we have all the information necessary to evaluate a field at any point in its base-space. We can efficiently describe fields that are defined on only a subset of a mesh instead of having to define them with a bunch of zeros over the whole mesh. We can easily postulate a *restriction* operator which restricts a field to a particular subset of its base-space. Such an operation is invaluable when we are interested in looking at only a portion of some large data set. Likewise, we can postulate a number of set operations such as *union*, *intersection*, *difference*, *boundary-of*, *copy-of*, and *neighbor-of*.

Looking, again, at the SILs in Fig. 3, we see that different types of mesh structure can be captured in terms of different patterns of subset relations (arrows) between the *N* dimensional elements. Only the pattern need be captured. With this approach, various kinds of structured grids such as rectangular, triangle-strips, hexagonal, could be easily and efficiently represented. In a completely arbitrarily connected grid, there is no such pattern and the subset relations (arrows) must be explicitly characterized.

We can represent *inhomogeneous* fields. For example, we can represent a symmetric tensor field which is a 2D tensor over an infinitely thin, 2D wing but is a 3D tensor over a 3D fuselage. Furthermore, we can characterize multiple, independent decompositions of a base-space into element-blocks, materials, processor domains, parts in an assembly, etc. In the case of processor decompositions, the SIL provides the information to enable operators that seamlessly change between different decompositions to run

a problem on 32 processors, for example, and restart it on 48. Finally, we can describe multiple, different representations for the same field and seamlessly exchange between them. Operations such as these are fundamental to large scale, scientific software integration.

4 The Sets and Fields (SAF) Scientific Data Modeling System

SAF represents a first cut at a portable, parallel, high performance application programming interface for reading and writing shareable scientific data files based upon abstract data modeling principles. Our primary goal has been to demonstrate that we can apply this technology to do the same job previously achieved with mesh-object I/O libraries like Silo and EXODUS, that is to simply describe and store our data to files. Our belief is that if we cannot demonstrate this basic capability, there is little point in trying to address more advanced features of scientific data management made possible with this technology.

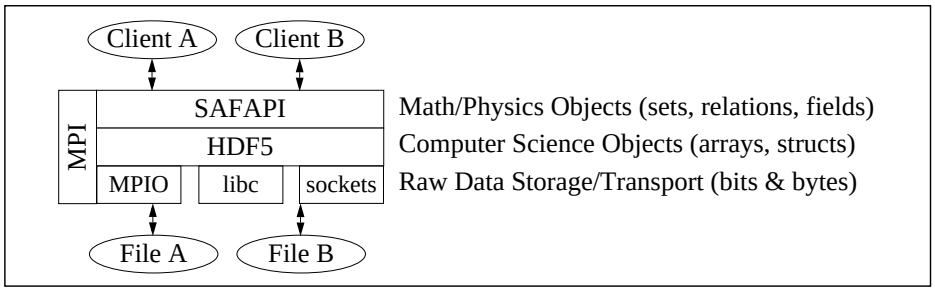

Fig. 4 Overall SAF System Software Architecture

A coarse view of the architecture is illustrated in Fig. 4. SAF is built upon the Hierarchical Data Format, version 5 (HDF5) and the Message Passing Interface (MPI). It is a procedural implementation of a design based upon object oriented techniques. Based on the conceptual model, *set*, *relation* and *field* objects were defined along with the following operators:

- **`declare`**: create a handle to a new object and define its parameters.
- **`describe`**: get the declaration parameters of an object.
- **`write`**: put out "raw" data (typically problem-sized) that populates an object
- **`read`**: get "raw" data of an object.
- **`find`**: retrieve objects based on matching criteria and/or traversing the SIL.

4.1 Sets and Collections

Sets are used to define the base-space for fields. In theory, every subset of the base-space, even every node, edge, face and volume element is a full-fledged set. However, in the implementation of SAF, we have had to differentiate between two kinds of sets: *aggregate* and *primitive*. Aggregate sets are the union of other sets in the base-space. As such, aggregate sets are used to establish *collections* of primitive sets as well as other aggregate sets. Primitive sets are not the union of any other sets in the base-space. Typ-

ically, primitive sets represent computational elements: nodes, edges, faces and volumes. Primitive sets are never instantiated as first class sets in SAF. Instead, they only ever appear as members of collections.

4.2 Relations

A relation describes how members of a *domain* collection are set-algebraically related to members of a *range* collection. There are numerous possibilities for different kinds of relations, but currently only two have been implemented: *subset relations* and *topology relations*. A subset relation defines a subset relationship between two aggregate sets by identifying the members in a range collection, on the superset, that are in the domain collection on the subset. For example, a processor subset is specified by a subset relation which identifies all the elements in the whole that are on the processor. A topology relation defines how members of a domain collection are knitted together to form a network or mesh by virtue of the fact that they share members of a range collection. For example a collection of zones is knitted together by identifying the nodes each zone shares.

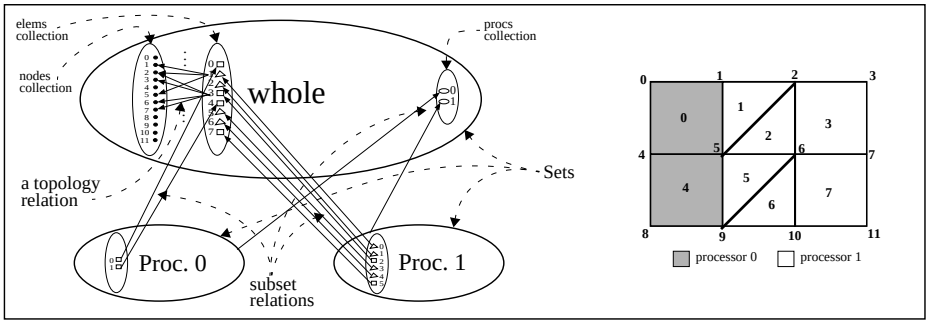

Fig. 5 An example of sets, collections, subset and topology relations.

4.3 Field-Templates and Fields

Fields are defined in SAF in two steps. First, a field-template is defined by attaching a fiber-space to a set. A fiber-space is specified by its fiber-dimension (e.g. the number of components in the field), an algebraic-type such as scalar, vector or tensor, a basis-type such as Euclidean and a physical quantity such as length, time or charge. In fiber-bundle theory, the Cartesian product of the fiber-space and the base-space it is attached to defines the *bundle*[5]. A bundle serves to identify a class of fields. Thus, a field-template defines a bundle.

In the second step, a field is defined as an instance of a field-template. The basis-set is defined along with the specific units of measure such as meters, seconds or coulombs. Finally, the dof-set is defined by specifying a mapping of dofs to pieces of the base-space. Currently, SAF supports mappings of dof-sets which are *n:1* associated with members of collections. That is, for every member there are *n* dofs that influence the field over it. These mappings are a generalization of the older notions of node- and zone-centered variables commonly used in I/O libraries and visualization tools.

5 Results

A key result and milestone for SAF is that sub-organizations in ASCI who currently use EXODUS or Silo, have demonstrated that SAF is general enough to replace *both* of these older scientific database technologies. This is a critical result. It validates our approach to large scale integration. In addition, SAF is currently being integrated with several ASCI applications and support tools including SNL's SIERRA[10] and LLNL's Ale3d[22] and MeshTV[17]. The results in Fig. 6 compare SAF's I/O performance with Silo in the Ale3d simulation code. They are highly preliminary but encouraging. SAF's performance suggests scalability and appears to compare favorably with Silo.

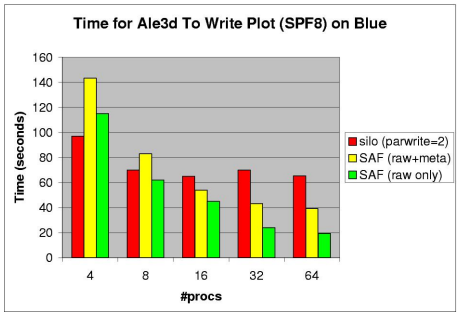

Fig. 6 Preliminary performance results

6 Further Work

Our primary focus so far has been to develop a sufficiently general data model and then to prove that we can produce an easy to use, scalable, parallel I/O library of acceptable performance that describes scientific data in these terms. From here, there are many directions to take. Some are enumerated below.

- The ability for SAF clients to define new element types and basis-sets at run time.
- The ability to register specific assemblies of SAF objects in an object registry.
- The ability to characterize derived fields along with the specific derivation rules.
- A publish/subscribe paradigm for exchanging data in-situ instead of by I/O to a file.
- MPI-like messaging pitched in terms of sets and fields instead of arrays and structs.
- The addition of set and field operators.
- A query language analagous to SQL used in relational database systems.

SAF is currently being prepared for a full public release around the end of March, 2001. At that time, there will be publicly viewable web pages at or near http://www.ca.sandia.gov/asci-sdm/ (follow the "Data Models and Formats" link)

7 Acknowledgements

Special thanks to David Butler for his early and important contributions in defining the data model [7]. In addition, we thank our alpha users and early adopters: Nancy Collins, Doug Speck, Rob Neely and Greg Sjaardema. Our sincere thanks, also, to numerous Livermore and Sandia management advocates who have supported this work: Linnea Cook, Celeste Matarazzo, Tom Adams, Terri Quinn, Charles McMillan, John Zepper, James Peery and Constantine Pavlakos. Thanks also to Mike Folk and the HDF team for the HDF5 product, upon which SAF is built, and continued enthusiastic collaboration in ASCI-DMF. This work was performed under the auspices of the U.S. De-

partment of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.

References

[1] Ambrosiano, J., Holten, J., Medvick, P., Parker J., Reed, T., "CDMlib: Simplifying Data Exchange Among Simulation Codes", to be published report of the Los Alamos National Laboratory (1998).

[2] Baum, J. D., "Elements of Point-Set Topology", Dover Publications Inc., 1991.

[3] Brown, S., Folk, M., Goucher, G., Rew, R.,"Software for Portable Scientific Data Management," *Computers in Physics*, Vol. 7, No. 3, pp. 304-308, (1993).

[4] Brown, S. A., "PACT User's Manual", University of California Research Lab Report, Lawrence Livermore National Laboratory, UCRL-MA-112087 (1999).

[5] Butler, D. M., Pendley, M. H., "A Visualization Model Based on the Mathematics of Fiber Bundles", *Computers in Physics*, V3 N5, pp. 45-51 (1989).

[6] Butler, D. M., Bryson, S., "Vector Bundle Classes Form Powerful Tool for Scientific Visualization", *Computers in Physics*, V6 N6, pp. 576-584 (1992).

[7] Butler, D. M., "ASCI Data Models and Formats Committee Array API Specification" aka "VB-1.0", working draft, Limit Point Systems and Sandia National Laboratories, (1997).

[8] Butler, D. M., various consultations on the ASCI-DMF project (1997-1999)

[9] Cheney, E. W., "Introduction to Approximation Theory", McGraw-Hill, 1966.

[10] Edwards, H. C. "A Look at SIERRA: A Software Environment for Developing Complex Multi-physics Applications", http://www.cfd.sandia.gov/sierra.html (2000).

[11] Fortner, Brand, "HDF: the Hierarchical Data Format. " (Technology Tutorial) Dr. Dobb's Journal V23, N5, pp. 42-48 (1998).

[12] Goucher, G., Mathews, J., "The National Space Science Data Center Common Data Format," *R&T Report*, NASA/Goddard Space Flight Center Publication, (December 1994).

[13] Grygo, G., "Khoros public domain environment integrates stages of data visualization. (product announcement)", *Digital Review* v8, n34 (Nov 4, 1991)

[14] Haber, R., Lucas, B. and Collins., N. "A Data Model for Scientific Visualization with Provisions for Regular and Irregular Grids", *Proceedings IEEE Visualization '91 Conference*, pp. 298-305, October, 1991.

[15] Larzelere, A.R., II, "Creating simulation capabilities," *IEEE Computational Science and Engineering*, Jan.-March, 5, 1 (1998).

[16] Rew, R. K., Davis, G., "NetCDF: An Interface for Scientific Data Access" *IEEE Computer Graphics and Applications*, V4, pp. 72-82, July (1990).

[17] Roberts, L., Brugger, E. S.,Wookey, S. G., "MeshTV: scientific visualization and graphical analysis software", University of California Research Lab Report, Lawrence Livermore National Laboratory, UCRL-JC-118600 (1999).

[18] Roberts, L. J., "Silo User's Guide", University of California Research Lab Report, Lawrence Livermore National Laboratory, UCRL-MA-118751-REV-1 (2000).

[19] Schoof, L. A., Yarberry, V. R., "EXODUS II: A Finite Element Data Model," Tech. Rep. SAND92-2137, Sandia National Laboratories (1994).

[20] Sjaardema, G. D., "Overview of the Sandia National Laboratories Engineering Analysis Code Access System", SAND92-2292, Sandia National Laboratories (1993).

[21] Upson, C., et. al., "The Application Visualization System: a computational environment for scientific visualization", *IEEE Computer Graphics and Applications*, July, 1989.

[22] Wallin, B. K., Nichols III, A. L., Chow, E., Tong, C., "Large multi-physics simulations in ALE3D", *Tenth Society for Industrial and Applied Mathematics Conference on Parallel Processing and Scientific Computing*, Portsmouth, VA, March 12-14, 2001.