SAF 1.0 Tutorial

Mark C. Miller

1 Introduction

This is a tutorial for the Sets and Fields (SAF, pronounced “safe”) parallel I/O and scientific data modeling system. This tutorial describes all versions of SAF with a one (‘1’) in the major digit of the version number, that is 1.A.B. This tutorial is aimed at teaching readers how to use and apply SAF to storing and exchanging scientific data. Nonetheless, since SAF is aimed at modeling *fields*, this tutorial also serves as a detailed primer on *numerical fields*.

SAF represents a revolutionary approach to storage and exchange of data between scientific computing applications. The primary purpose of SAF is to enable *Very Large Scale Integration* (VLSI[1]) in scientific computing. That is to support the storage and seamless exchange of an unprecedented variety of scientific data amoung an unprecendented variety of applications. Small scale integration has been achieved with a number of existing products: SEACAS, PACT, Silo, Exodus II, CDMLib, netCDF, CDF, HDF and PDBLib to name a few. There are many others.

Isolated successes with these products in integrating on the small scale lead many to believe that any one is sufficient for the large or very large scale. It is merely a matter of comming to agreement and picking one we all should use. This belief rests primarily on the notion that the problems with large scale integration are largely social in nature and not technical. However, there are indeed substantial technical hurdles in achieving integration on the very large scale. SAF’s revolutionary design is aimed at addressing many of these.

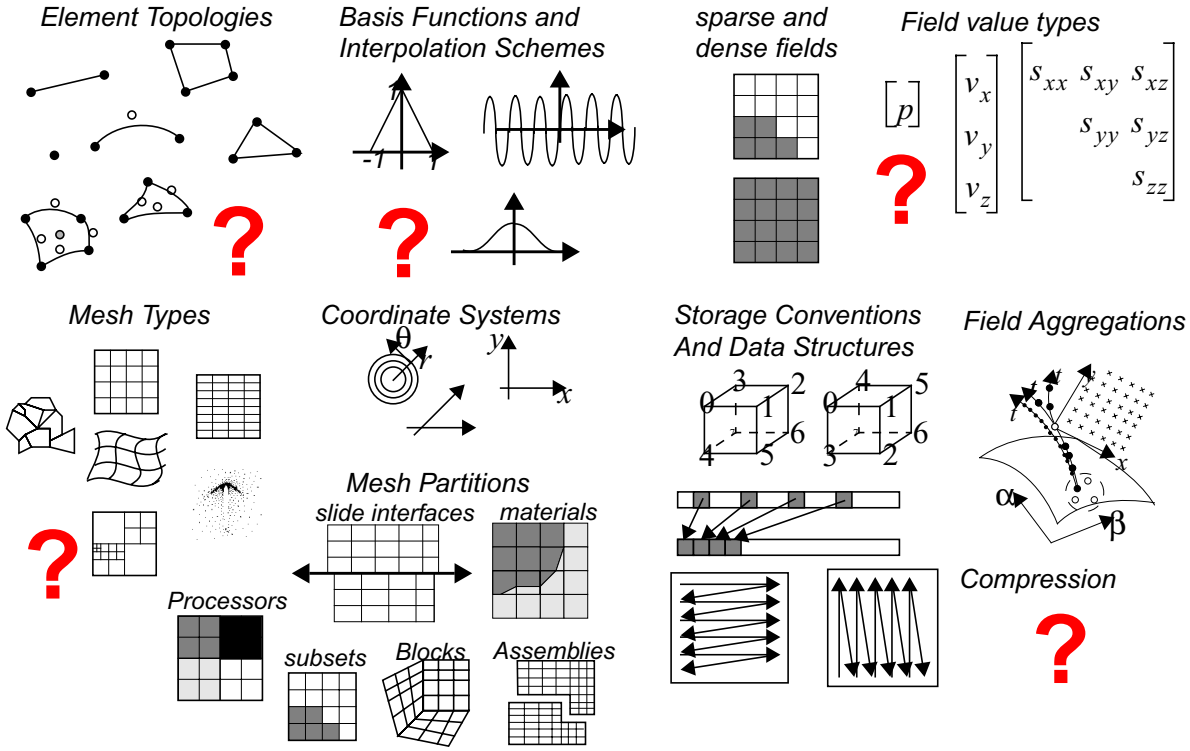
Unfortunately, since the purpose of this tutorial is to instruct readers on how to *use* SAF, a complete explanation of the motivations for *why* SAF is designed as it is is beyond the scope of this tutorial. Only a brief introduction is provided here in the next sub-section. Following that, we cull out a few of the key concepts helpful to keep in mind as you learn to use SAF.

1.1 Briefly, Why SAF?

In Figure 1 we illustrate a number of examples of diversity in scientific data. The examples in the figure are by no means exhaustive. They don’t illustrate all of the categories of diversity in scientific data we deal with today. Furthermore, as the large, red question marks are meant to suggest, we don’t know what new variations we will need to deal with tomorrow.

1. Here, we borrow from integrated circuit technology where the push for larger scales of integration, such as *Very Large Scale Integration*, continues to enable ever more powerful processors.

Figure 1: Examples of Diversity In Scientific Data



Addressing large scale integration involves developing an application programming interface (API) for managing scientific data that solves two problems. On the one hand, it must permit developers to express numerous details about how their data is represented such as those exemplified in Figure 1. On the other hand, it must also enable developers to manipulate their data in a *representation independent* manner. By "representation independent" we mean that data can treated 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 literally thousands of ways to represent the airflow over the wing of a supersonic aircraft in a computer program, there is only one mathematical/physical interpretation; a 3D velocity field over a 2D domain. This latter description is immutable and independent of any particular representation or implementation choices.

Representation independence is made possible only through a sufficiently general abstraction. For scientific data, the most appropriate abstraction is one that employs the mathematical and physical notions inherent in the application domain. That is the mathematical and physical notions of *fields* defined on infinite point *sets*. This is where SAF gets its name.

In SAF, the term "field" is used to describe any phenomena that can be mathematically represented, at least locally, as a function over some infinite point set as its domain. The term "set" is used to describe an infinite point set, typically continuous, with a topological dimension over which fields are defined. Thus, the abstraction consists of three primary entities, fields and sets, and relationships between these entities.

Fields may represent real physical phenomena such as pressure, stress and velocity. Fields may be related to other fields by integral, derivative or other algebraic equations. Fields are defined on sets. Sets may represent real physical objects such as parts in an assembly, materials and slide interfaces. And, sets may be related to other sets by equations involving union, intersec-

tion and difference operations.

Unfortunately, detailed arguments justifying why this abstraction is essential to very large scale integration in computational science is outside the scope of this tutorial. However, the reader should pause for a moment and confirm in his or her own mind just how general the notions of field and set are in describing scientific data. The columns of an Excel spreadsheet are fields. A time history is a field. The coordinates of a mesh is a field. A plot dump is a whole bunch of related fields. An image is a field. A video is a field. A load curve is a field. Likewise for sets. An individual node or zone is a set. A processor domain is a set. A material decomposition is a collection of sets. An element block is a set. A slide line or surface is a set. A part in an assembly is a set. And, so on.

For a more detailed discussion of issues in scientific database design, the reader is refered to ???. No further motivation for SAF's design is presented here.

1.2 Key Concepts

In this section we cull out key concepts the reader will need to keep in mind throughout the rest of this tutorial. SAF is different enough from other technologies that it is useful to take a moment and address any potential preconceptions the reader might have about how SAF works.

1.2.1 SAF is Model-Oriented not Menu-Oriented

An important difference between SAF and other products is that SAF is a *model*-oriented system while most other products are *menu*-oriented.

A menu-oriented system such as Silo or Exodus II, offers what amounts to a menu of objects. From among the objects on the menu, a user chooses an object that most closely matches her data. If the user cannot find an appropriate object, her choices are to ask the "chef" to cook up a new object (that is, extend the system) or eat at a different restaraunt (that is, for example, switch from Exodus II to Silo).

There are unique API calls to handle each object in a menu-oriented system. Therefore, as a menu-oriented system is employed to support a wider variety of applications and reach larger and larger scales of integration, its API and system complexity grow right along with its menu. This is a key consequence of a menu-oriented approach.

By contrast, a model-oriented system such as SAF offers a small set of primitive building blocks. A user assembles these building blocks to literally model her data. The same set of building blocks can be assembled in a variety ways to model a wide variety of data. As a model-oriented system is employed to reach larger and larger scales of integration, its key advantage over a menu-oriented system is that the API and system complexity remain fixed. On the other hand, a model-oriented system does have some drawbacks.

Learning to use a menu-oriented system is relatively simple. It involves browsing the menu (e.g the API reference manual) for an object that most closely matches the data in an application and then learning the handful of API calls (often just one) to read and write that object. However, learning to use a model-oriented system is more involved.

First, the primitive building blocks of a model-oriented system are more abstract, less familiar and require greater time in learning to use than the objects in a menu-oriented system.

Second, it is often the case that a user cannot get by with learning only a portion of a model-oriented API. To do anything useful in a model-oriented system, a user needs to become familar with a majority of the API.

Third, the user needs to learn how to apply the modeling primitives to build up expressions

for his or her data in terms of assemblies of these primitives. In other words, the user has to become a data modeler. Furthermore, each time the user encounters new data s/he has not dealt with before, s/he has to re-engage in the modeling activity to decide how best to represent it with the given primitives.

For these reasons, the reader should not expect using a model-oriented system like SAF to be anything like experiences s/he may have had using menu-oriented systems like Silo, Exodus II or similar products.

Another useful way of thinking about the difference between using menu- and model-oriented systems is that using menu-oriented systems is like answering multiple choice questions while using model-oriented systems is like answering essay questions.

In fact, a good way to think about SAF is that it is like a language designed specifically for descriptions of scientific data. We call that language a *data model*. A data model is a formal, conceptual tool for describing data, data relationships, data semantics and consistency constraints[1].

Throughout this tutorial, we will often have the need to use the term "model" both as a noun and as a verb. For example, SAF employs a data model (noun) and we use it to model (verb) data. We mention this point now to reduce the possibility of confusion.

A comparison of the strenghts and weaknesses of menu-oriented and model-oriented systems is summarized in Table 1.

Table 1: Comparison of menu- and model-oriented systems

Menu-oriented Systems		Model-oriented Systems	
Strengths	Shortcomings	Strengths	Shortcomings
More development experience	Often need to "cast" data	No "casts"	Less development experience
Familiar/simple objects	Fragile extensibility	Robust extensibility	**Unfamiliar/abstract math**
Simple design/interface	Ever expanding menu	Handful of building blocks	Complex design/interface
	Partially self-describing	Wholly self-describing	
	Ad-hoc interoperability	**Formal interoperability**	
Silo, Exodus, CDMLib		**SAF, LibSheaf**	

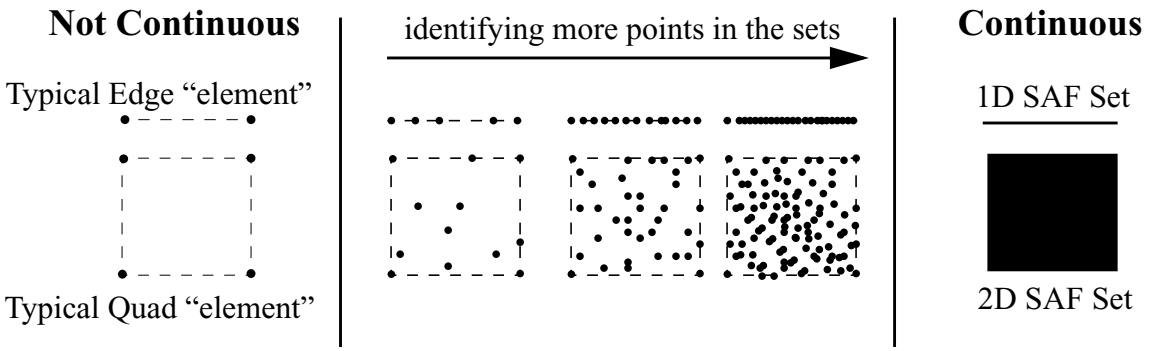
1.2.2 SAF Represents Objects in a Real World Continuum.

For readers with experience with products like netCDF, HDF, Exodus II or Silo, another hurdle in learning to use SAF is the fact that SAF is designed to represent objects in a real world *continuum*. That is, continuous sets and continuous fields.

A set object in SAF represents an infinite point set. Set objects are used to represent the computational domain, and portions thereof, over which dependent (and independent) variables of a simulation (or experiment) are defined. Every node, edge, face, volume and/or higher dimensional element in a computational simulation as well as processor pieces, parts in assemblies, materials, slide surfaces, boundary conditions, etc. are ultimately represented as sets in SAF. However, it is important to emphasize that a set in SAF is a *continuous* thing. It is an infinity of points. As an example, the picture in Figure 2 illustrates the difference between the objects we traditionally think of when we talk about edge or quad elements and their counterparts as SAF sets.

1. Paraphrased from Silbershatz, Korth and Sudarshan, "Database System Concepts"

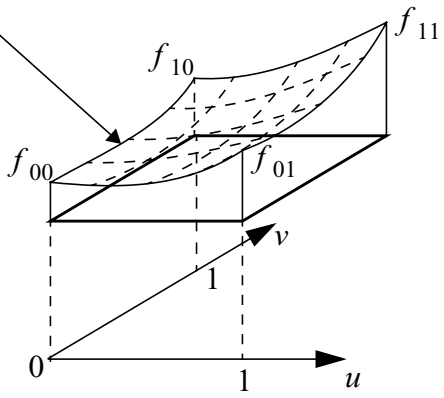
Figure 2: Illustration to facilitate learning what an infinite point set is



Likewise, fields in SAF are defined on sets. As such, a field has value at every point in the set it is defined on. Since a set consists of an infinity of points, a field has an infinity of values, one for every point in the set on which it is defined. The picture in Figure 3 illustrates a continuous field, a bilinear field in this example, defined on a set representing an individual quad element. The equation for *f(u,v)* defines the field for all u and v over the region defined by the quad, $0 \le (u, v) \le 1$. Thus, just as there are an infinite number of points in the set representing the quad, the field f is defined and has value for each such point.

Figure 3: Continuous field on an individual quad element

$$f(u, v) = v[uf_{11} + (1-u)f_{10}] + (1-v)[uf_{01} + (1-u)f_{00}$$



To summarize then, a key difference between SAF and other products is that SAF is designed to represent continuous fields defined on continuous, infinite point sets.

1.2.3 SAF's Data Model is Distinct from its Implementation

The design of SAF is logically decomposed into a few key pieces; its data model specification, its API specification and its implementation.

The data model specification is a white paper, natural language specification of the various model primitives and their attributes. The data model specification is highely technical in nature and defines the mathematical foundations of the data model.

The API specification is automatically generated from the library source code. The library source code is more than just code. It includes specially formatted embedded comments that describe public symbols of the library including compile-time and run-time constants, API calls,

their behavior, their arguments, their pre-, post- and invariant conditions. Although items are routinely *added* to the API specification, the existing specification is rarely *changed*. The API specification is also known as the *Library Reference Manual*. The implementation is, of course, the rest of the source code other than its API specification.

The reason for explaining the distinction between these parts of SAF's design is that the data model specification *leads* the API specification which, in turn, *leads* the implementation. This has two consequences. First, if new functionality outside the scope of the current design becomes necessary, it is first addressed in the data model, then in the API and finally in the implementation. Second, the data model often suggests functionality the current API spec. and implementation do not support. So, the reader should not be surprised if upon learning the data model, s/he encounters limitations in the current API or implementation s/he did not expect. Likewise, the reader should not be surprised if the API specification includes functionality that is not available in the current implementation[1].

1.2.4 SAF Enables Dramatically More Functionality Than It Currently Delivers

SAF enables development of many useful capabilities existing products cannot. In fact, the development of SAF's data model was driven for more by its potential to enable generally useful software to opereate on scientific data than its potential to simply store that data. Nonetheless, a solution to the latter is a pre-requesite for the former. That is, writing tools and operators to process a variety of scientific data is not possible until there exists a common means for managing that variety of data. Consequently, all of the effort in developing SAF has focused on the latter problem. That is developing a data model sufficiently general enough to manage an unprecedented variety of scientific data.

No doubt, as the reader learns more about SAF, useful capabilities will come to mind that are currently missing but could rightly be argued should be part of or built into SAF. For example, although SAF deals with infinite point sets, it does not currently provide much functionality in the way of set algebraic manipulations. There are no API calls to compute set intersections, unions or differences. There simply are not enough resources to add all the functionality to SAF that is made possible by its data model.

This brings us to the last and most important thing the reader should keep in mind as s/he learns about SAF. Each time you think of, wish for or otherwise expect a given functionality to be part of SAF and are dissapointed because it is not, just remember, none of that functionality would even be implementable without the intial foundations SAF currently provides. In time, as SAF evolves, much of this functionality will likely be added to SAF.

1.3 Summary

SAF is a model-oriented system while most existing products are menu-oriented. SAF is designed to deal with objects in the real, continuous world. Its key objects are continuous fields and continuous, infinite point sets as well as relationships between them. Conceptually, SAF's design is broken into its data model specification, its API specification and its implementation. The data model is more general than the API specification which is, in turn, more general than the implementation. SAF enables more functionality than its early implementations deliver.

---

1. The API specification (e.g. library reference manual) clearly identifies these situtations.