Effective glue between geoscience concepts, data, and modeling systems

Mark Verschuren[1] and David M. Butler[2]

Abstract

We describe the use of a novel mathematical data model, the sheaf data model, to support management and integration of diverse data sources and modeling tools used in the process of subsurface interpretation. We first introduce a simple abstract framework for the interpretation process. We follow with a review of the notion of data model and the requirements a data (meta-)model must meet to effectively support data integration in geoscience applications. We introduce the sheaf model and give a mostly non-mathematical overview of the principle features of the model, showing how it meets the requirements and how it can be used to make the abstract interpretation framework operational. We then describe the application of the framework and model to several specific interpretation topics. We finish by comparing the framework presented here to two existing frameworks.

Introduction

Geoscience increasingly requires integration of diverse conceptual formulations, incompatible modeling systems, and disparate data types to successfully address the evermore complex geologic structures and questions confronting the geoscientist. There is a growing consensus that integration of the many available data sources and tools is a critical problem for subsurface interpretation in particular.

The purpose of this paper is to draw attention to the potential of a novel mathematical data model, the sheaf data model, to provide a unified conceptual framework for representation, management, and integration of diverse data sources and to enable implementation of advanced modeling tools. The intended audience is primarily researchers and advanced developers because the approach we describe here is only starting to be reduced to practice, but we also hope that practicing geoscientists can gain some insight from the presentation.

Organization

The discussion is organized into four sections. In the next section, "An abstract framework for interpretation and modeling," we present a simple conceptual framework for thinking about the components and tasks of subsurface interpretation. Making this abstract framework operational requires a metamodel. This leads to a discussion in the following section "Data models and data integration" of data models in general, the conection between data models and data integration, and the sheaf data model in particular. In the section "Effective glue for interpretation," we describe the application of the conceptual framework and sheaf data model to various aspects of subsurface interpretation. In the section "Comparison to other modeling frameworks," we then compare the framework presented here to two existing frameworks: the knowledge-oriented framework described by Perrin and Rainaud (2013) and the data-oriented framework defined by the RESQML version 2 standard (Energistics and the RESQML SIG, 2014a, 2014b).

An abstract framework for interpretation and modeling

We start with a question: What is an interpretation? In our view, an interpretation consists of two components:

1) the interpreter's evolving conceptual model of some object or objects of interest and
2) a collection of concrete computer representations of the conceptual model.

This framework, depicted in Figure 1, is simple and abstract, yet, as we will show, it also captures critical aspects of the interpretation process and its products.

Conceptual model

The conceptual model identifies conceptually important parts of the object and includes a notion of shape

[1]Shell, Houston, Texas, USA. E-mail: mark.verschuren@shell.com.
[2]Limit Point Systems, Inc., Livermore, California, USA. E-mail: d.m.butler@limitpoint.com.

Manuscript received by the Editor 29 November 2014; revised manuscript received 27 May 2015; published online 24 February 2016. This paper appears in *Interpretation*, Vol. 4, No. 1 (February 2016); p. T103–T121, 18 FIGS.

for the object along with various other attributes or properties. These properties can be of numerous different types, but the most important category of such types is what is referred to in mathematical physics as a *field*. A field is a mathematical entity that describes some physical property as a function of position in the object and/or in time. Examples of fields include:

- rock density as a function of position,
- sound velocity as a function of position,
- rock porosity as a function of position,
- fluid velocity as a function of position and time, and
- rock displacement as a function of position and time.

There are of course a large number of other such fields that occur routinely in subsurface interpretation; any physical property is a candidate as well as its estimated error or uncertainty.

A simple example will help to clarify the notion of the conceptual model. Figure 2 shows a simple model of a well that will be used as the object in a continuing example throughout the paper. The model is deliberately extremely simple so that diagrams we will present shortly fit the constraints of publication. Nevertheless, the example allows us to illustrate the essential aspects of interpretation and modeling. We will also show a more complex example below.

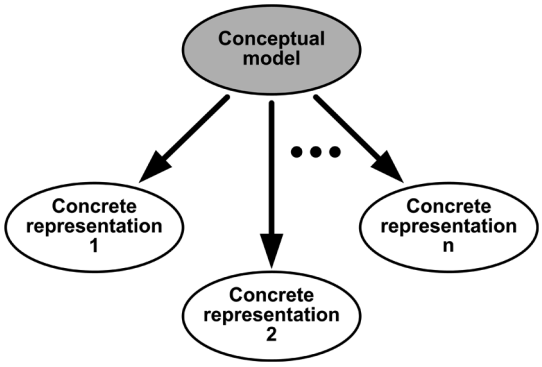

Figure 1. Schematic of abstract framework for interpretation. An interpretation consists of the interpreter's evolving conceptual model and one or more concrete computer representations. Each of the arrows depicts an *is represented by* relationship.

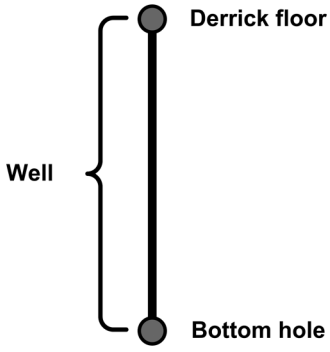

Figure 2. A simple well. The well has three conceptual parts, identified in the diagram.

One essential aspect of interpretation is that even at the conceptual level, an interpretation usually identifies that the object under consideration is decomposed into several parts. Our well example identifies three parts: the well as a whole along with the derrick floor and the bottom hole. It is important to emphasize two points with regard to this example. First, the parts are not disjoint: The well part contains the derrick floor and bottom hole parts, but it is not equal to their sum. Second, these parts are in the conceptual model, not generated as a by-product of any computer representation.

In addition to this part decomposition, the conceptual model identifies which physical properties are of interest in the interpretation. We assume the following properties are of interest for the well example:

- $\mathbf{r}(s)$, the well path (the physical position of the well as a function of distance s down the well),
- $\gamma(s)$, the gamma log as a function of distance down the well,
- $\rho(s)$, the density as a function of distance down the well, and
- various sonic properties.

Figure 3 shows a second example, a salt structure. The structure contains several bounding surfaces: the water bottom, two additional horizons, two crosscutting faults, and a salt boundary. The horizons bound the upper, middle, and lower strata. The faults break the middle stratum into four subvolumes: inside (between

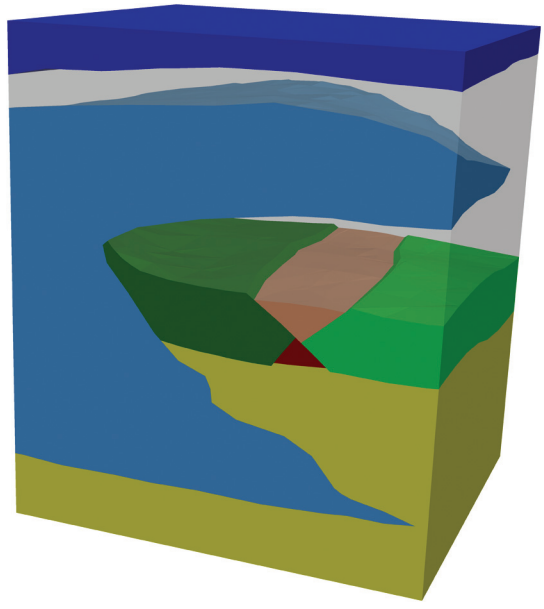

Figure 3. A salt structure. The structure contains eight subvolumes: water (dark blue), upper stratum (transparent light gray), middle stratum inside (dark green), middle stratum above faults (light red), middle stratum below faults (dark red), middle stratum outside (light green), lower stratum (dark yellow), and salt (light blue). The upper stratum is rendered transparent to make the faults more visible.

the salt and the faults), above the faults, below the faults, and outside (between the faults and the boundary of the region). Thus, there are eight subvolumes altogether: water, upper stratum, middle stratum inside, middle stratum above faults, middle stratum below faults, middle stratum outside, lower stratum, and salt.

In this example, the properties of interest are as follows:

- the shapes of the subvolumes, in particular the position of their bounding surfaces and
- the acoustic velocity as a function of position within the subvolumes.

In each of these examples, the part decomposition and the properties are considered as abstract mathematical entities at the conceptual level. The conceptual level asserts that the parts and the properties exist, but it does not typically assign concrete, quantitative values to them. That role is left to the concrete representations.

Concrete computer representation

A computer representation consists of data combined with software that together represent some aspect of the conceptual model. A representation is typically partial; it may address just one or a few of the properties, and it may cover only part of the object. A representation is also typically approximate; it may implement some physical or numerical approximation and may have missing data. Finally, a representation is typically optimized for some particular purpose.

The partial, approximate, and optimized nature of the computer representations means that the conceptual model inherently requires multiple representations, which in turn requires that the interpretation process must track the status of the various representations, the relationships between the representations, and the relationships between the representations and the conceptual model.

Continuing the well example, we assume that we have a computer representation that specifies the well path for the entire well and that we have several well logs, sampled at different rates, covering various separated and/or overlapping intervals along the well. A schematic for this interpretation is shown in Figure 4.

For the salt structure example, we assume that we have triangle mesh representations for the faults, horizons, and other surfaces defining the various subvolumes. We assume that we have three representations for the velocity field:

1) velocity as a constant or given algebraic function of depth between surfaces,
2) velocity linearly interpolated over tetrahedral meshes covering each subvolume and conforming to the triangles on the bounding surfaces, and
3) velocity on an imaging grid overlaying the entire volume.

The schematic for the salt structure example is shown in Figure 5.

As discussed so far, the schematic describing an interpretation is just an abstract framework. However, we can make it operational and implementable by introducing a metamodel capable of describing the conceptual model and the representations. The metamodel must describe the geometry, property fields, and other types of attributes at the abstract, conceptual level and the concrete, data representation level.

Data models and data integration

We will shortly describe a metamodel that meets the above requirements, namely, the sheaf data model. However, we have to first explain what we mean by the term *data model* and establish the connection between data models and data integration.

Definitions of data model

The term data model has at least three definitions in common usage. All three of these meanings will be needed in the course of this paper:

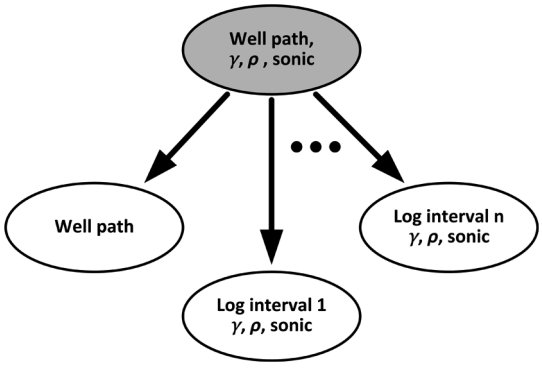

Figure 4. Schematic for the well example. The computer representations include the well path and well logs on various separated or overlapping intervals along the well.

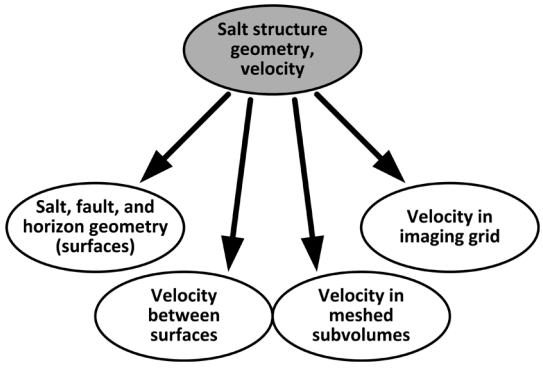

Figure 5. Schematic for the salt structure example. The computer representations include triangle meshes for the surfaces bounding the subvolumes, velocity as a given algebraic function of depth between the surfaces, velocity linearly interpolated over a surface conforming tetrahedral mesh for each subvolume, and velocity on an imaging grid overlaying the entire structure.

1) In the database theory community, the term data model means a theory of data. More precisely, a data model specifies a class of abstract mathematical objects and the operations on those objects, with the understanding that all data will be interpreted as instances of the class and manipulated using the operations. The best known example of such a data model is the relational data model, which is the theoretical basis for many (until recently, essentially all) commercially available database management systems. We will refer to this definition with the term *data metamodel*.
2) In the database practice community, the term data model refers to a specification of the explicit data that represents some particular organization or activity. A data model in this sense is usually restricted to a description of static aspects of the data; that is, operations are not part of the model. We will reserve *data model* for this meaning.
3) In the application development community, the term data model is often used to refer to the collection of data structures and operations used to implement an application, especially those aspects that are visible to users of the application. This use of the term is much less formal than the other two. We will refer to it as the *application data model*.

The relational data metamodel

To help clarify the notion of a data metamodel and for reference later in this paper, we briefly describe the relational model.

The objects in the relational model are *relations* on sets, and the operations are referred to as *relational algebra*. We briefly summarize the mathematical definitions for sets and relations in Appendices A and B, but here we rely on a well-known metaphor, the table metaphor, to describe the structure of the relational model.

An n-ary relation can be viewed as a table with n columns and an arbitrary number of rows. Each table is equipped with a schema that specifies the name of each column and the type of data that can appear in the column. A given table schema is interpreted as defining a type, each column defines an attribute of the type, and each row defines an instance of the type. The operations defined by the relational algebra are viewed as operations that manipulate tables: select rows, select columns, merge two tables with the same columns into a single table, create a new table that has the columns of two or more input tables, and so on.

The data metamodel development paradigm and data integration

The notion of a data metamodel carries with it a paradigm for software development. The metamodel can be used directly as the application data model for software libraries, tools, and languages. These components collectively form a platform, often referred to as a *database management system*, on which actual applications can be built.

This development paradigm can produce numerous benefits. It increases the level of abstraction for application development. It increases the capability of applications. It facilitates interoperation and integration between diverse data sources. All of these benefits combine to increase the productivity of application developers and application users.

The relational paradigm delivered all of these advantages to business data management, revolutionizing the discipline in numerous ways. The mathematics of the model raised the level of abstraction of applications developed on top of it. Storage-specific procedural queries were replaced by declarative queries written in the standardized query language SQL. Application-independent tools such as report generators replaced untold quantities of custom programming.

Most important in the context of the current paper, the relational model became the conceptual framework for defining data integration. The relational database management system, and especially SQL, became the enabling technology for integration and interoperation of diverse sources and formats via data warehouses.

The lesson to be learned from the relational model example is that successful data integration in a given application domain requires an effective data metamodel.

Failure of the relational metamodel for scientific computing

The current state of "disintegration" in geoscience and other scientific computing disciplines is due in large part to the failure of the relational model to provide the same efficacy in these disciplines as it has in traditional business data management. The fundamental reason for this is that, as we have already noted above, geoscience data are predominately field data. Succinctly put, the relational model can not describe the way we want to manipulate field data, nor can it describe the way we want to store field data. It is useful to describe the mathematical origins of the storage problem.

Mathematically, at least in simple cases, a field is what is called a *map* or *function* in set theory. (See Appendices A and B for a brief summary of the relevant concepts of set theory and maps.) A map is an association between two sets, the domain set and the range set, that assigns to each member of the domain some member of the range. For a typical field, the domain is the set of points in some object and the range is some property type. For the purpose of this discussion, we assume the domain is a finite set with n points.

There are two ways of defining a map mathematically. In the first approach, a map is defined as a *binary relation*. Described in the table metaphor, a binary relation is a table with two columns: one for domain and one for range. Each row in the table is a pair (domain member and range member). A map is a collection of n rows, exactly one row for each domain member. If we

have more than one such map, then the table contains multiple such collections of rows.

In the second approach, the map is defined as a single member of the *Cartesian power*, denoted range$^{\text{domain}}$. In the table metaphor, the Cartesian power is a table with n columns, one for each member of the domain, and the map corresponds to a single row in this table. If we have multiple maps, then the table contains multiple rows.

The first approach has several disadvantages. The first is that the map cannot be represented as a single entity, as an individual object. Instead, the map must be "shredded" into a collection of (domain and range) pairs. This is undesirable conceptually because the map does not actually exist as an explicit entity in the schema of the database. It is also undesirable practically because a map as an entity must be "rediscovered" by appropriate queries each time it is needed, which requires complex programming and creates performance problems.

The second approach removes these disadvantages. The map type appears explicitly in the database schema. Each instance of the map type is represented by a single row that can be stored and retrieved as a unit. This method requires less storage because the association between domain point and column is part of the schema and hence need not be duplicated if there are multiple maps.

The reason that the relational model has proven ineffective in scientific computing is that it cannot support the Cartesian power representation. The schema for a Cartesian power table depends on the domain. In practice, the domain is not a fixed object, defined outside the database and known when we define the database schema. Instead, the domain is itself represented by a table, with a row for each point. Hence, the schema for the map depends on the data in the domain table. The relational model cannot represent such data-dependent schema: Schema must be defined independent of the rows of any table. Hence, the relational model forces one to use the binary relation representation for maps. The resulting conceptual and practical difficulties outweigh any other benefits of using the relational model in scientific computing.

Fiber bundle metamodel

Over the past several decades (since the introduction of the relational model), there have been many attempts to define a data metamodel that can effectively support fields and the other mathematical data types common in scientific computing. One approach to define such a model leverages a strategic advantage presented to the would-be data metamodel developer by the physical sciences in general and the geosciences in particular: The quantitative aspects of these problem domains have already been mathematized. They inherit from mathematical physics a basic conceptual framework for describing objects and their properties mathematically. The data modeler does not have to discover or invent abstractions. Scientists and mathematicians have spent literally thousands of years sorting out what the abstractions are and distilling them into their most general and powerful forms. The metamodeler can transliterate the mathematical concepts into metamodel form. The object-oriented paradigm is particularly well suited to this task.

The fiber bundle data metamodel (Butler and Pendley, 1989a, 1989b; Butler and Bryson, 1992) was based on this strategy. The objects and operations were translated more or less literally from modern mathematical physics into an object-oriented data metamodel.

In modern mathematical physics, a physical object is conceptualized as a *topological space*, that is, an abstract, featureless, shapeless space with just enough mathematical structure to define the notion of continuity. (See Appendix C for a summary of the essential definitions for topological space.) Any additional features of the physical object are explicitly attached to this "naked" physical object, typically by defining property fields. The fully "clothed" physical object thus typically consists of a topological space and a collection of property fields, in particular a global coordinate field that gives shape to the object.

The various physical property kinds are conceptualized as algebraic types, principally various kinds of abstract vector spaces that fall into the category known as *multilinear algebra*, that is, scalars, vectors, and tensors of various types. (See Appendices D and E for a summary of vector spaces.)

A property field, that is, the dependence of some property on position within a physical object, is conceptualized using the theory of fiber bundles. (See Appendix F for a summary of essential definitions from the theory of fiber bundles.)

The theory of fiber bundles is a generalization of the simple notion of function, intended to deal with complexities that arise when the physical object has a complicated topology. The theory identifies three roles in defining the dependence of property on position:

1) The physical object is referred to as the *base space*.
2) The property type, the set of all possible property values, is referred to as the *fiber space*.
3) The association of each base space point with a property value is referred to as a *section*. The set of all possible sections with a given base space and fiber space is referred to as a *section space*.

In simple cases, the terms base space, fiber space, and section are equivalent to the terms domain, range, and function, respectively. The change in terminology emphasizes that these concepts are intended to also apply to topologically complex objects for which the simple notion of function is inadequate.

Typical of such topological complexities is the well-known fact that one cannot cover the surface of a sphere with a single coordinate system. Any attempt to do so will produce singularities somewhere. For instance, the latitude-longitude coordinate system breaks down at the poles. The fiber bundle formalism deals with such issues by decomposing the base space (physical object) into

overlapping parts and representing the section as a simple function on each part. Thus, a section is in general a collection of ordinary functions, each defined on a part of the base space. We have to immediately ask: What do we mean by "the" value of the section (the property value) at any point where the parts overlap? The bulk of the mathematical machinery contained in the definition of a fiber bundle is directed at answering precisely this question. Fortunately, we will not need it in the context of this paper.

The operations of the fiber bundle model incorporate all the operations associated with the types of its constituents. This includes various topological and geometric operations on the base space, the algebraic operations of the property types, and the algebraic and calculus operations associated with the section type.

Casting the metamodel at the level of generality provided by fiber bundle theory is mathematically required to ensure that the model can deal with any topological complexity and property type that can occur in practice. However, the generality of the model, especially the decomposition into parts that is the central feature of the fiber bundle structure, has also proven very useful for practical reasons even when not strictly required for topological reasons.

Limitations of the fiber bundle model

The fiber bundle model was successfully applied in a number of projects and products, most notably the IBM Data Explorer visualization system (Haber et al., 1991), now known as OpenDX (Visualization and Imagery Solutions, Inc., 2006) and the more recent F5 library (Benger, 2009, 2011).

However, extensive experience with the fiber bundle model in a variety of settings has shown that although the very high level concepts and operations defined by the model provide a good interface for using the data, the model does not address how to actually represent and store the data.

As with our discussion of the relational model, it is useful to investigate the mathematical origins of this problem. To do so, we return to an assumption we made previously when discussing the representation of a simple map in the relational model. Specifically, we assumed that the domain was a *finite* set of points. However, a physical object conceptualized as a topological space has an infinite number of points, and hence we cannot store the value of the map at each point. Doing so would require a table with either an infinite number of rows (Cartesian product representation) or an infinite number of columns (Cartesian power representation).

The common practice for dealing with this problem is to first discretize the base space, that is, decompose it into a finite collection of simple parts, then approximate the map by a simple map on each simple part. The discretization and approximation have to have the following features:

- Each simple part has to be equipped with some mechanism for specifying any one of the infinite number of points in the part. We will refer to this mechanism as *local coordinates*.
- The simple map has to have, in general, some collection of stored data, although for some specific map types this collection may be empty.
- The simple map has to have an evaluation method that takes as input the local coordinates of a point and uses those coordinates and the stored data to compute the value of the map at the point as output.

We will refer to such a simple map type as a *local section evaluator*, or just *evaluator* for short. Typically, the stored data are the values of the map at some points in the part and the evaluation method is interpolation. But whatever the specific data and evaluation method for each part, the result for the entire physical object is a finite amount of data that we can represent in a table. However, note that we have to maintain the association between part, evaluator type, and data to evaluate the map.

A consequence of this common practice is that, in general, we need at least two decompositions of the base space: the topological decomposition defined by the fiber bundle model and another for the field discretization. In practice, we need additional decompositions for a wide variety of purposes from application-related features such as stratigraphy to implementation-related features such as parallel programming and distributed computing.

In short, we need general support for multiple, concurrent decompositions of the base space and support for the association of evaluator type and data with the parts in the decomposition. Fiber bundle theory does not address either of these requirements. As a result, implementation of the model requires ad hoc extensions and these extensions inevitably limit the scope and utility of the implementation.

Summary of data integration requirements

Data integration for scientific computing requires an effective data metamodel. The relational model, on the one hand, is of too low a level to be effective. Its table structure will not support the (mathematical) field as a type, and the table operations are not useful for manipulating fields. The fiber bundle model, on the other hand, is of too high a level; it cannot describe how a field type is represented by data.

Successful data integration for scientific computing requires a data metamodel that supports the following four features:

1) data-dependent schema,
2) multiple concurrent decompositions,
3) the association of evaluator type with parts in a decomposition, and
4) the association of data with parts in a decomposition.

We will refer to these requirements repeatedly.

The sheaf data metamodel

The sheaf data metamodel directly addresses these requirements using two mathematical types: finite distributive lattices (FDLs) and sheaves. We will attempt to give an intuitive, informal description of these types in the body of this paper. See Appendix G for a summary of the mathematical definitions of partially ordered sets and lattices and Appendix H for a summary of the definitions for sheaves. Additional mathematical details, tutorials, and the full source code for the open source SheafSystem™ implementation of the model are available online (Limit Point Systems Inc., 2015).

FDL

The FDL type is based on the notion of a *partial order relation*. A partial order relation is a generalization of the subset relation between two sets. The subset relation $\subseteq$ satisfies the following properties:

1) $S \subseteq S$ for any set S (*reflexivity*),
2) $S1 \subseteq S2$ and $S2 \subseteq S3$ implies $S1 \subseteq S3$ for any sets S1, S2, S3 (*transitivity*), and
3) $S1 \subseteq S2$ and $S2 \subseteq S1$ implies $S1 = S2$ (*antisymmetry*).

An abstract partial order relation is any relationship that has these three properties, regardless of whether it involves subsets or not.

Partial order relations are very common in abstract mathematics and in practical applications. In particular, partial ordering is considered the starting point for formalizing the *part of* relation (Varzi, 2015). Indeed, we can informally think of an FDL using the *part space* metaphor (Butler, 2013a, 2013b). This metaphor interprets an FDL as a part space: a space consisting of a set of *basic parts* and all possible distinct *composite parts* that can be constructed from the basic parts. A part is composite if it is precisely equal to the sum of its parts, and it is basic if it has no parts or is greater than the sum of its parts. The partial order of the FDL is interpreted as the part of relation in this metaphor, but we have to emphasize that the notion of part and part of in this metaphor is abstract. What specific kind of object a part is and the concrete meaning of part of will depend on the application at hand. For instance, in the following we will see examples in which part of means spatial inclusion, data member inclusion, and model dependency.

Visualizing an FDL

We will leave the mathematical definitions to the appendix as much as possible, and in the body of this presentation, we will rely on the part space metaphor and a method of visualizing an FDL that mathematicians call a *Hasse diagram*. A Hasse diagram is a collection of boxes, which we refer to as *nodes*, connected by arrows, referred to as *directed edges*. Our Hasse diagrams contain two types of nodes: one for basic parts and one for composite parts. The basic nodes are drawn as filled boxes, and the composite nodes are unfilled. The edges in the diagram represent the *covers* relation. Part A covers part B if and only if B is part of A and there is no other part C such that B is part of C and C is part of A. In other words, A covers B if B is a direct part of A. (Note that part of and covers "point" in opposite directions.) We always draw the diagram such that if A covers B, B is below A on the page. Therefore, following the edges down the page takes us to smaller, more deeply nested parts. Part C is part of part A if there is a path from A down to C.

We can easily draw the Hasse diagram for our well example, as shown in Figure 6. In this example, the parts are "chunks" of space and the partial order is spatial inclusion. The whole well covers the derrick floor and the bottom hole. All the parts are basic. In particular, the well is basic; it includes the derrick floor and bottom hole but is not equal to their sum.

We can also draw the Hasse diagram for the salt structure example, as shown in Figure 7. The diagram is more complex, and we intend it to give the reader some feel for how the Hasse diagrams scale up to more complex structures. The salt, the water, the top stratum, and the bottom stratum are each basic parts. The middle stratum is divided into parts by the two faults, as described above. Each of these parts is basic, and the middle stratum is precisely their sum, so it is composite. The two faults are crosscutting, so they split both themselves and two of the three horizons into parts. The faults and horizons are each the sum of their respective parts, so they are composite. The truncation of the horizons against the salt also splits the salt boundary into parts, but we do not include the parts of the salt boundary, faults, or horizons in the diagram. The full diagram is sufficiently complex that it is difficult to draw by hand and would be too large for this publication.

The part spaces visualized in the Hasse diagrams in Figures 6 and 7 describe the well and salt structure, respectively, at the conceptual level. The diagrams encode the existence of the various parts and their relationships but do not provide any specific geometric representation for the parts.

The size of a part space depends exponentially on the number of basic parts in general; hence, even a few basic parts can generate a very large part space. As a

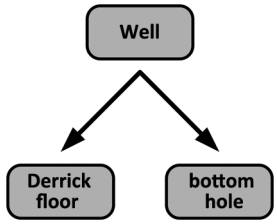

Figure 6. Hasse diagram for the well example. The whole well covers the derrick floor and the bottom hole. All the parts are basic.

result, we rarely visualize an entire part space. Instead, we visualize the basic parts and only those composite parts we are specifically interested in, as we have shown in the examples. For instance, derrick floor + bottom hole is a possible composite part for the well example, but it is not interesting; hence, we have not put it in the diagram. Every application will have such a set of basic parts and interesting composite parts. We refer to this interesting portion of the part space as the *part structure* for the application, but remember, these are parts in the abstract sense.

Although the FDLs for realistic cases tend to be too complex to illustrate by hand, they can be easily represented and manipulated on the computer. The covers relation depicted in a Hasse diagram corresponds to a well-known data structure: the *directed acyclic graph* (DAG). DAGs have been exhaustively studied in computer science and are widely used in computer programming. The SheafSystem implementation of the sheaf data model uses DAGs to directly represent FDLs. Thus, using the SheafSystem, we can describe the conceptual models shown in Figures 6 and 7 on the computer. We shall see below how this description extends from the conceptual model to its concrete representations.

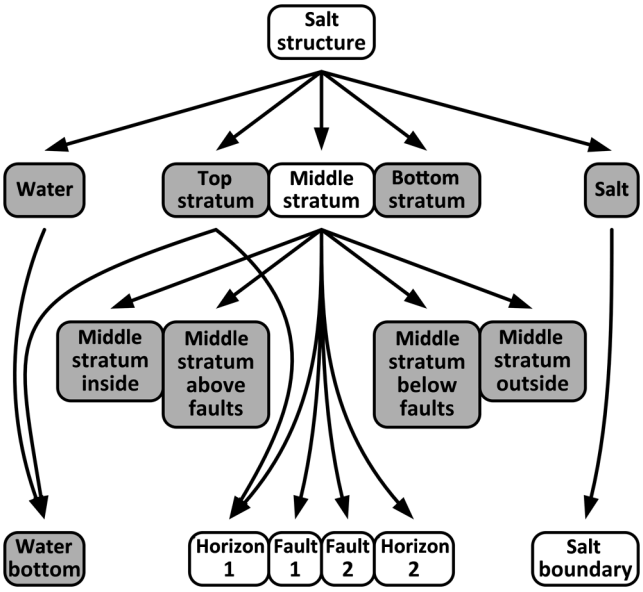

Figure 7. A partial Hasse diagram for the salt structure example. The diagram shows the subvolumes, salt boundary, faults, and horizons, but it leaves out the smaller and lower dimension parts.

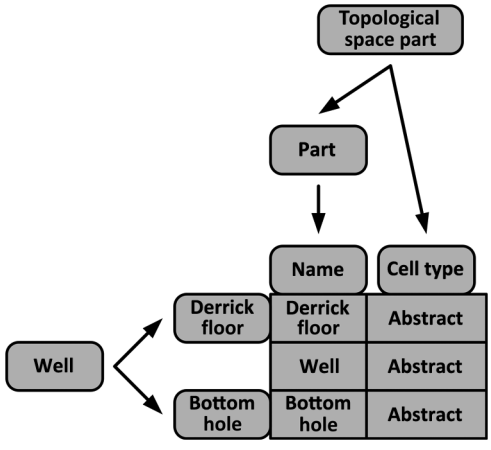

Figure 8. Table metaphor for the well example.

FDLs and data representation

We have so far introduced the FDL as a mathematical type and presented a method for visualizing the type, but how do we actually use it to represent data? To answer this question, we return to the relational model and the table metaphor. As we described above, in the relational model a table represents a data type and each row is an n-tuple that corresponds to an object or instance of the type. Thus, the table as a whole corresponds to a set of n-tuples. The sheaf data model follows this pattern, but instead of the rows being a *set* of tuples, the rows are a *lattice-ordered set* of tuples. In other words, the sheaf data model represents data as a part space (FDL) in which the parts are n-tuples and the part of relation is an arbitrary user-defined partial order. The order relation is data, like the tuples themselves.

We refer to this structure as a *lattice-ordered relation*. It satisfies data metamodel requirement 2: representation of multiple, concurrent decompositions. Any decomposition or collection of decompositions defines a set of basic parts, and a lattice-ordered relation can represent these basic parts as well as any and all composite parts created from the basic parts.

We can visualize a lattice-ordered relation as a table with a Hasse diagram attached to the rows. The table corresponding to the well example is shown in Figure 8. The well parts are represented using a schema with two attributes: *name*, a string; and *cell type*, also a string. Ignore for the moment the Hasse diagram attached to the columns; we will describe it in the next section. The Hasse diagram attached to the rows in the figure is just Figure 6 turned on its side. When we draw a Hasse diagram on its side like this, we will sometimes still refer to the direction the edges point as "down", even though they point to the right.

Schema

We satisfy data metamodel requirement 1, data dependent schema, directly. The schema for each table is defined by some other table in the database. That is, the columns in one table are defined by the rows in another table. This means that there is a Hasse diagram for the rows (the *row graph*) and a Hasse diagram for the columns (the *column graph*), as we have already seen in Figure 8.

The schema for the well table is defined by another table, the topological space part table, shown in Figure 9. This table specifies the schema for a simple object-

oriented type hierarchy consisting of two user defined types, *part* and *topological space part*. The part type specifies one attribute, *name*, of type string. Topological space part inherits part and defines an additional attribute, *cell type*, also of type string.

The order relation in a schema table represents subobject inclusion. The subobjects associated with a given type consist of any user-defined types it inherits and any data members it aggregates, which may be of a user-defined or primitive type. A type hierarchy thus defines a subobject inclusion hierarchy, which can be interpreted as a part space with the parts being data types and the part of relation meaning *subobject of*. In any part space, the parts that have no subparts are referred to as *atoms*. In a schema table, the atoms correspond to data members of primitive type such as integer, floating point, and string. A schema table thus describes the decomposition of a user-defined data type into primitives that can be directly represented on the computer. The decomposition only describes the data members of the data type. The operations associated with the type are only implied by the type attribute (the second column in Figure 9), which is assumed to be a reference to an external specification or implementation of the type.

As we said above, the schema for any table must be defined in another table. The recursion implied by this requirement terminates in a special table, the *schema part table*, a simple example of which is shown in Figure 10. The schema part table is its own schema, its column graph is its own row graph, thus terminating the recursion. The attributes of a schema part include *size* and *alignment*, information necessary for allocating memory for instances of the type defined by the schema part.

Data types for scientific computing

We have described in the preceding section the basic data representation mechanism of the sheaf data metamodel. In the context of geoscience applications, we are particularly interested in representing the types associated with fields: base space types, fiber space types, and section space types.

We have already seen how to represent base space types, as demonstrated by the tables for the well example. Fiber space types, such as scalars, vectors, and tensors, are also represented by tables using the methods we have already described, but we will need an explicit example to discuss section representation. Figure 11 shows the table representation for a simple 2D Euclidean vector type E2 and its schema. (A 3D vector type that we could think of as physical coordinates would provide a more realistic example, but as with the well, we have to keep the example as simple as possible to keep the diagrams manageable.) The table in Figure 11a shows three vector values, and, as is typical for fiber space types, the row graph has no edges. This is the empty cover relation, and it corresponds to the set of rows being unordered. The interesting part structure for fiber spaces is in their schema, but the E2 type hierarchy in Figure 11b is deliberately simplified; the full mathematical type hierarchy is much more complex.

Section types are more complicated than either the base space types or the fiber space types. The schema for a section type depends on the base space table and the fiber space schema table. Figure 12 shows the schema for a section space defined over the base space shown in Figure 8 and the fiber space schema shown in Figure 11b.

Two aspects of this schema need to be emphasized. First, the section type specified by this schema is entirely conceptual and second, the schema has a member for every pair (base space basic part and fiber schema

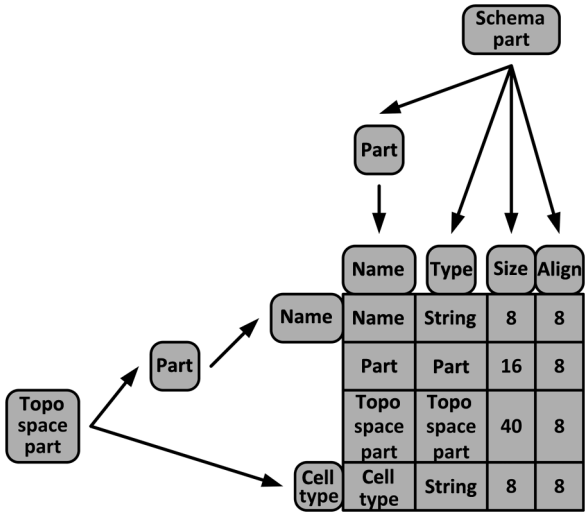

Figure 9. Topological space part schema. "Topological" is abbreviated as "topo."

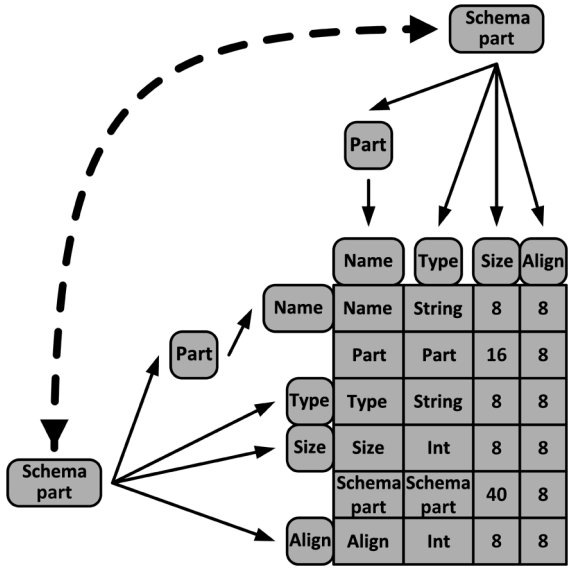

Figure 10. The schema part table. The column graph for schema part is its own row graph, as emphasized by the dashed line. This terminates the schema recursion.

basic part). The schema encodes the abstract mathematical properties of the section type: A section has parts associated with each part of the base space and each component of the (vector) fiber space.

This section schema does not describe a quantitative representation. As discussed above, to define a concrete representation for a field, we first need to discretize the base space. Figure 13a shows the geometry and Figure 13b shows the table for a simple discretization of our well example. As before, we have chosen this example to be very simple, just two segments *s1* and *s2* and one additional vertex *v2*. A realistic discretization would introduce many more segments. Note that the derrick floor and bottom hole parts have been approximated as vertices and the well has been approximated as the composite of the two segments.

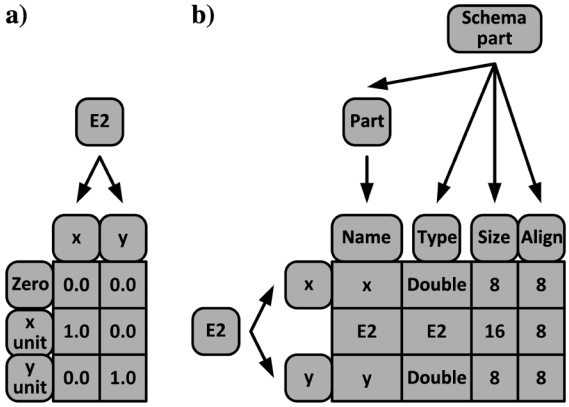

Figure 11. Fiber space example: 2D Euclidean vector E2: (a) table showing three values and (b) schema table.

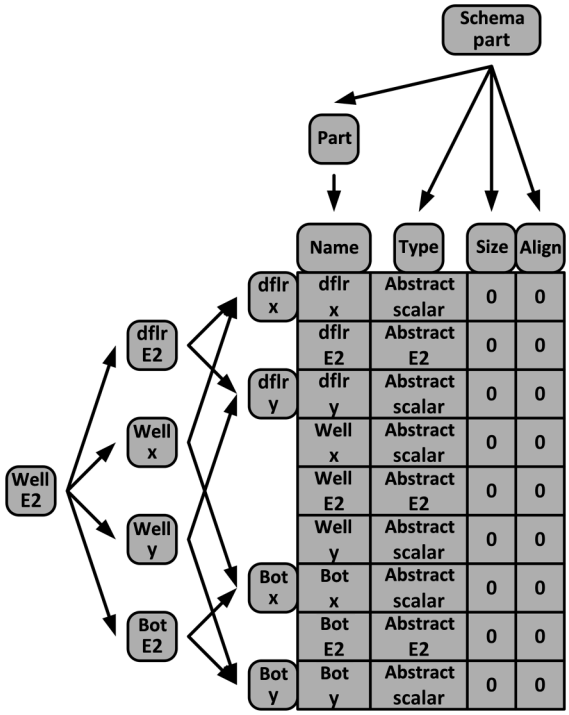

Figure 12. Section space schema example for the base space of Figure 8 and the fiber space schema of Figure 11b. "Derrick floor" and "bottom hole" are abbreviated as "dflr" and "bot," respectively.

Given the discretization, we can define a schema for a concrete representation of our field that stores the value of the field on the vertices and uses linear interpolation as the evaluation method on the segments. The schema for this representation is shown in Figure 14.

We need to emphasize two aspects of this schema as well. First, this schema explicitly associates evaluator types with each part of the discretization, thus satisfying metamodel requirement 3. Second, comparing the concrete schema in Figure 14 to the abstract schema in Figure 12, we see that the concrete schema repeats the abstract schema but contains additional members describing the numerical representation. This is a crucial capability of the sheaf model: The sheaf schema provides a rigorous bridge from the abstract specification of the field, represented by the leftmost members of the graph, to the numerical details, represented by the rightmost members of the graph.

It would be impractical, to say the least, to construct such schema tables manually, or even by direct programmer control, for any realistic example. However, it is important to remember that the objects we are describing here with diagrams and tables are in fact mathematical objects and can be created and manipulated algebraically. In particular, a section schema such as shown in Figure 12 or Figure 14 can be constructed using an algebraic operation called the *lattice tensor product* of the base space lattice and the fiber schema lattice. The mathematical structure allows us to construct the

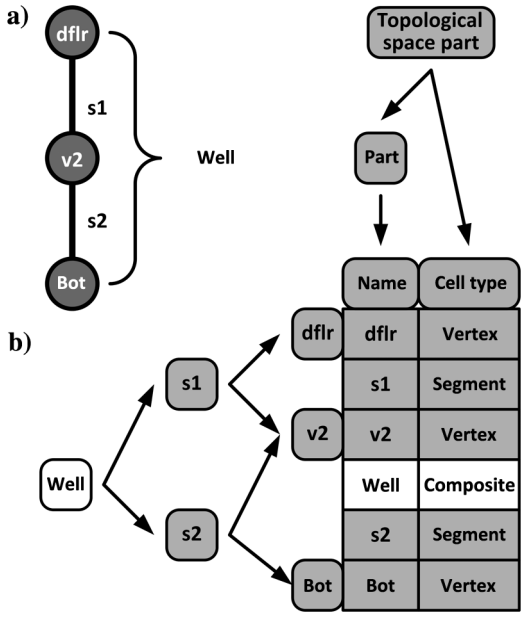

Figure 13. A simple discretization for the well example: (a) geometry and (b) table representation. "Derrick floor" and "bottom hole" are abbreviated as "dflr" and "bot," respectively.

table automatically and represent it efficiently. In fact, only the composite parts, if any, need be stored. Any basic part can be generated on demand using its algebraic properties.

Figure 15 shows a section space table based on the schema in Figure 14 and containing two sections. We have named the two sections "plan a" and "plan b," suggesting the well paths for two drilling plans. (As we mentioned above, a 3D vector fiber space would make these path coordinates more realistic, but then the figures would have been too large.)

Where's the sheaf?

The reader may well be asking by this point: "It's called the sheaf data model, so where's the sheaf?" The answer: it is hiding in plain sight, implied by the lattice schema of the tables. Given any table, Figure 15 for example, we can construct another table by picking a schema member, say s1 E2 and in each row selecting only the cells in columns that are reachable by following the edges down from the schema member. Discarding cells from the source rows may create duplicate result rows; if so, we only keep one copy. This process creates a new table and maps each row in the source to a row in the new table. The new table and the rows in it are referred to as the *restriction* to s1 E2 of the source table and source rows, respectively.

If we do this recursively for each member of the schema, we get a family of tables related by restriction. The full family of tables corresponding to Figure 15 is too large to show explicitly in this publication. Instead, we show in Figure 16 only the tables associated with schema members below s1 E2.

The figure can be interpreted as a map that assigns a restricted table to each member of the schema. A map like this from an FDL to a family of sets related by restriction is called a *sheaf*. In our case, the sets are lattice-ordered relations; hence, we have a *sheaf of lattice-ordered relations*. This object is the top level of the data model, the most aggregate structure, and hence the model is named after it.

The primary operation provided by the sheaf is row restriction. Looking at Figure 15 for instance and picking any row, say, the "plan a" row, we can restrict it to any member of the schema, producing a row containing only the data associated with that member of the schema. The lattice tensor product construction described above specifically creates a member of the section space schema for every combination of base space part and fiber schema part. Hence, we can find the data values associated with any part of the base space or fiber using restriction. Thus, the restriction operation satisfies metamodel requirement 4, the association of data with parts in a decomposition.

Effective glue for interpretation

Having described the sheaf data metamodel, we now turn to describe the use of the model for interpretation

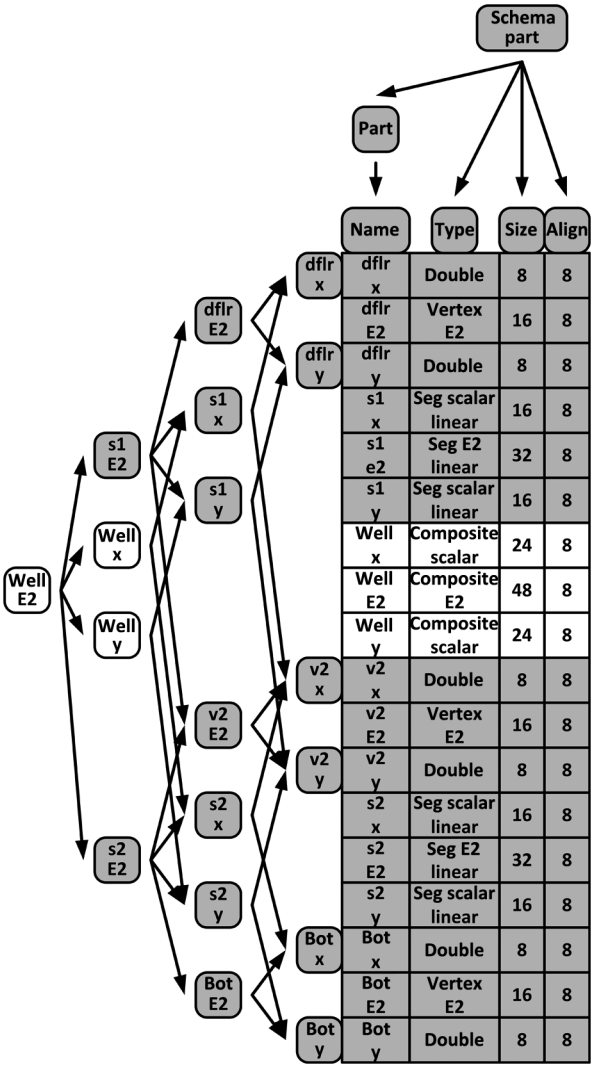

Figure 14. Schema for section representation using the discretization of Figure 13 and linear interpolation.

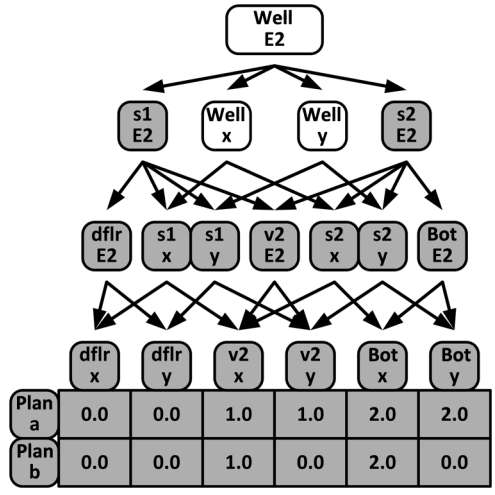

Figure 15. Section space example using the schema in Figure 14.

and modeling of the subsurface. We present a selection of topics intended to demonstrate the power and benefits of the approach.

Representation is refinement

We have seen above that constructing a field representation requires discretization of the base space. Discretization is commonly thought of as somehow sampling the base space, selecting a finite number of points from the infinite number of points in it. This description is inaccurate for two reasons. First, a typical discretization contains segments, faces, or other cells, not just points. Second, even when the discretization is formally just points, such as for some types of data acquisition or for finite difference simulation methods, common practice is to immediately restore the "space between the points" using interpolation.

We can provide a more accurate description using the lattice formalism presented here. Discretization decomposes some or all of the old conceptual parts into new basic parts, turning old basic parts into composite parts. For our well example, comparing Figures 8 and 13, we see that discretization introduces new basic parts s1, s2, and v2 and that the well part becomes composite. The well part is approximated as the composite of the two segments s1 and s2 and the derrick floor and bottom hole parts are approximated as vertices. Comparing Figures 12 and 14, we see that decomposition of the base space induces a decomposition of the section space schema as well.

We find it useful to think of this decomposition plus approximation process as *refinement*. The process satisfies the mathematical definition of refinement, which requires that each new part be part of exactly one old part in a certain sense. However, more importantly, it also matches the ordinary English language definition of *refine*: "Improve (something) by making small changes, in particular make (an idea, theory, or method) more subtle and accurate" (Oxford Dictionaries, 2015). Thus, refinement connotes increasing the level of detail and the resolution of a model.

Thus, we view the interpretation process as introducing abstract physical objects and property fields at the conceptual level and refining these concepts into approximate concrete representations.

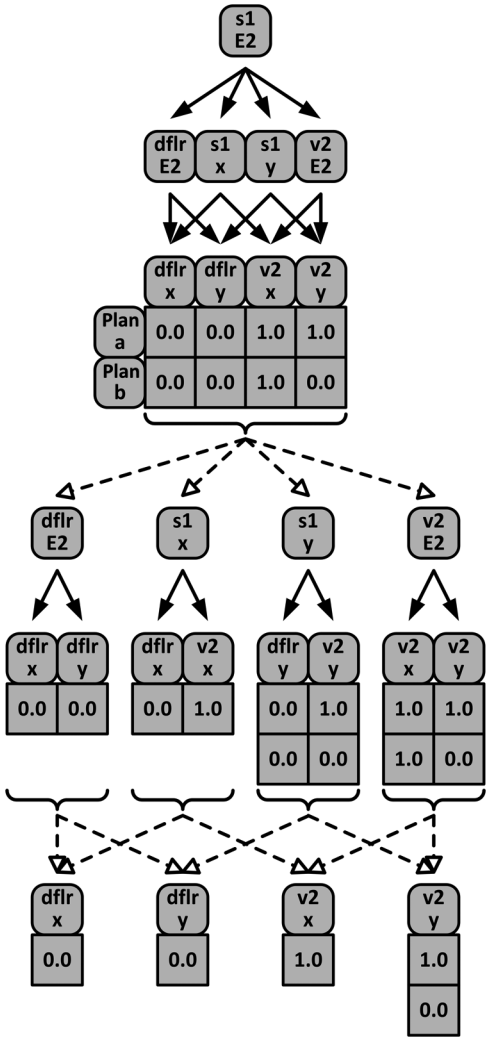

Figure 16. Sheaf of lattice-ordered relations corresponding to Figure 15. The restriction relation is represented by dashed arrows. Duplicate rows in the restricted tables have been removed.

Explicit link from conceptual model to representation

We can explicitly represent the arrows in Figure 1, that is, the links between the conceptual model and its representations, using *order-preserving maps*. An order-preserving map is a map from one part space to another that preserves the part relationships. If C is the part space describing the part structure of the conceptual model and R is the part space describing the refined part structure of a representation, then we can create a map φ from C to R that associates each part c in C with a part $r = \varphi(c)$ in R. Order-preserving means that for two conceptual parts c1 and c2 and the corresponding representation parts $r1 = \varphi(c1)$ and $r2 = \varphi(c2)$ that c1 is part of c2 implies r1 is part of r2. In other words, an order-preserving map explicitly encodes how the part structure of the representation represents the part structure of the conceptual model.

We can visualize an order-preserving map for the well example as shown in Figure 17. The map associates each part of the conceptual model on the left with the corresponding part in the representation on the right, as shown by the dashed arrows. The map has the property that for two conceptual parts c1 and c2 and corresponding representation parts r1 and r2, c1 part of c2 implies that r1 is part of r2. For instance, the derrick floor is part of the well in the conceptual model and in the representation.

Order-preserving maps are directly describable in the sheaf data model, so the model can describe the bubbles and the arrows of the abstract framework we presented.

Coordination and update of dependent representations

The various computer representations of an interpretation typically have some interdependencies. As long as we exclude circular dependencies, which is not a problem in practice, the dependency relation has the mathematical properties that define a partial order relation. Hence, we can describe the dependencies as a part space. In this case, the parts are models and the covers relation means *depends directly on*.

Figure 18 shows the Hasse diagram for the dependency relation of the salt structure example. The conceptual model depends directly on the meshed and gridded representations of the velocity, which both depend directly on the velocity between surfaces representation, which depends directly on the surface geometry representation.

Explicit computer representation of the dependencies enables automatic coordination and update of dependent representations whenever a predecessor changes, relieving the interpreter of the burden of managing these dependencies manually.

Conversion between representations

The sheaf schema for a field type provides a complete description of a field representation, from the abstract mathematical specification down to the bits and bytes of the data layout. This description enables generalized, schema-driven field "remapping," that is, conversion from one mesh and/or numerical representation to another. The SheafSystem implementation provides a universal *push* operator for converting any representation of a field to any other representation of the same field, on the same or another mesh, provided they are defined on the same object in the conceptual model.

For example, we can invoke the push operation to convert the "velocity between surfaces" in the salt structure example to either the "velocity on tet mesh" or the "velocity on ijk grid."

Interoperation between modeling systems

We do not always want to spend the time and storage space cost of actually converting from one representation to another. Fortunately, we do not have to. The operations of the sheaf model imply an abstract interface for data access. Instead of converting, we can leave the data of a representation in their original format and write a sheaf interface adapter that allows us to access the data as if it were stored in the sheaf native form. We can then interoperate between any model formats for which we have a sheaf adapter.

An obvious candidate for encapsulation with a sheaf adapter is the seismic data that most interpretation projects depend on.

Interpretation persistence

The original motivation for developing the sheaf model was to provide application independent storage of technical data, and the SheafSystem implementation does indeed provide this capability. An important characteristic of the model and its implementation is that the persistent version of an interpretation contains complete information: the conceptual model, all the representations, all dependency, and relations between the representations, all so-called metadata, and the schema of all the types. This has particularly important implications for long-term archiving. Because the conceptual model and all the abstract specifications are present in the database, future use is not dependent on availability of the application that created the data. The data are meaningful and usable long after the original application is gone.

Comparison to other modeling frameworks

The potential contribution of the modeling framework we are proposing can perhaps be made clearer by com-

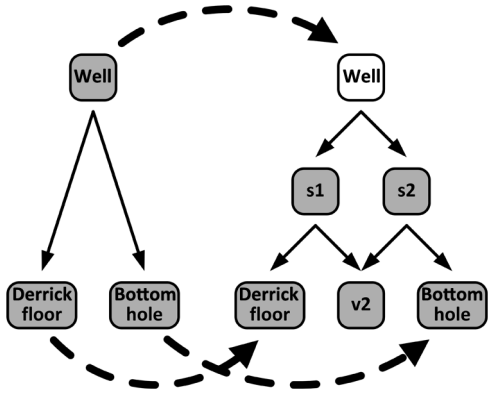

Figure 17. Order-preserving map for the well example. The map associates each part of the conceptual model on the left with the corresponding part in the representation on the right, as shown by the dashed arrows. The map has the property that for two conceptual parts c1 and c2 and corresponding representation parts r1 and r2, c1 part of c2 implies r1 part of r2.

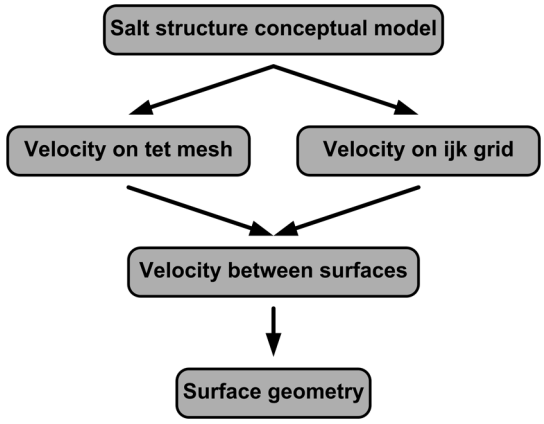

Figure 18. Hasse diagram for the dependency relation of the salt structure example. The conceptual model depends directly on the meshed and gridded representations of the velocity, which both depend directly on the velocity between surfaces representation, which depends directly on the surface geometry representation.

paring it to two existing approaches: the knowledge-driven approach of Perrin and Rainaud (2013), which we view as being more abstract than our framework; and the data-driven approach of the RESQML version 2 data exchange standard (Energistics and the RESQML SIG, 2014a, 2014b), which we view as being more concrete.

Comparison to Perrin and Rainaud

Perrin and Rainaud present a broad survey of the state of the art in shared earth modeling approximately 2013. They identify numerous issues and advocate the use of, as the subtitle of the book states, "knowledge-driven solutions for building and managing subsurface 3D geological models." They state: "The view of the authors is that no actual *shared earth modeling* can be efficiently undertaken unless it integrates, along with reservoir information, the knowledge used to produce that information. [. . .] In contrast to the existing *data-driven* approach, ours is a *knowledge-driven* approach that will facilitate the comparison and evaluation of different modeling hypotheses and allow partial model rebuilding and cooperation" (Perrin and Rainaud, 2013, pp. 65–66, authors' italics).

The primary tool in their approach is the notion of an *ontology*. The authors describe an ontology as a formalized representation of knowledge that enumerates the kind of objects that exist in a domain, gives human readable definitions of these object types, and formal axioms that constrain the interpretation and well-formed use of the object types (Perrin and Rainaud, 2013, pp. 189–190). They refer to a spectrum of ontology types, ranging from glossaries and data dictionaries at the unformalized extreme to logic-based ontologies for automated reasoning engines at the formalized extreme. Data modeling representations are placed somewhere in the middle. They also distinguish "*lightweight* ontologies, which are mainly taxonomies, from *heavyweight* ontologies, which model domains in a deeper way and provide restrictions on domain semantics" (Perrin and Rainaud, 2013, pp. 192–193, authors' italics). They specifically identify the need for "ontologies for data-intensive applications" (Perrin and Rainaud, 2013, p. 202). Finally, they identify the need for ontologies associated with geology itself and a number of related fields such as geophysics, solid modeling, and applied mathematics.

Although we have no background in knowledge engineering, it seems likely to us that the sheaf data metamodel defines, at least to within transliteration into an appropriate ontological specification language, an ontology that has the potential to make a major contribution to the Perrin and Rainaud program. We expect such a sheaf ontology would fit into their ontology spectrum somewhere toward the formalized side of the middle, toward the heavyweight ontologies, but probably short of automated reasoning engines. The sheaf model captures the semantics of mathematical abstractions that play a fundamental role in all the specific domains identified by Perrin and Rainaud, and hence the associated ontology would be useful in all these domains. A particularly valuable aspect of the sheaf model is its ability to span the entire scope from conceptual models to concrete representations. This scope enables automated processing, as we described above, which is exactly the kind of benefit Perrin and Rainaud envision their approach delivering.

Comparison to RESQMLv2

RESQMLv2 is a "data-exchange standard that facilitates reliable, automated exchange of data among software packages used along the subsurface workflow" (Energistics and the RESQML SIG, 2014a, 12). The standard uses the Unified Modeling Language (UML; Object Management Group Inc., 2015) to define a data model for the subsurface workflow. The data types (classes and other elements) identified in the model are represented as XML schemas (XSD files) (World Wide Web Constortium, 2015). The data types are organized into a *knowledge hierarchy* containing four categories (Energistics and the RESQML SIG, 2014b, p. 26):

1) *feature*: something that has physical existence at some point during development, production, or abandonment of a reservoir;
2) *interpretation*: a single consistent description of a feature;
3) *representation*: a digital description of a feature or interpretation; and
4) *property*: semantic variables (e.g., porosity and permeability) and the corresponding data values, which are recorded in arrays.

This knowledge organization is hierarchical in that a property is associated with a specific representation is associated with a specific interpretation is associated with a specific feature.

To compare the sheaf data metamodel with RESQML, we first observe that the RESQML standard is a data model in the sense of definition 2 above; it specifies explicit data types for a specific activity. Because a typical use for a data metamodel is to use the language it provides to specify a data model, it is interesting to ask: Can we describe the RESQML data model using the sheaf metamodel? A full answer to this question would require a substantial investigation, but at first glance, the answer would appear to be yes, at least partially. It appears that it would be possible to replace the XML schemas describing the model with equivalent sheaf schemas.

Thus, there is a sort of rough syntactic compatibility between the sheaf data metamodel and the RESQML data model. The semantics provided by the sheaf model is, however, dramatically different than that of the RESQML knowledge hierarchy. The approach advocated in this paper recognizes physical objects and properties (fields) as entities that appear at the conceptual level and are refined into concrete representations

and data. The RESQML model, by contrast, recognizes only physical objects at the conceptual level (features). Hence, only physical objects can be interpreted and represented. Properties then occur only as data attached to a specific representation.

The root cause of this semantic difference is that the RESQML model does not recognize the existence of field as a type. Thus, the RESQML model is more or less forced into the property-is-data organization because it lacks the notion of field as a type and because it lacks the unique ability of the sheaf model to describe the connection between abstract field types and their representation as data. Conversion or interoperation between different property data in RESQML can only be described on a case-by-case basis, and the actual tasks of conversion or interoperation must be implemented by application software. By contrast, data in the sheaf model can be seen as instances of deeper mathematical abstractions with semantics that enable automatic conversion and interoperation.

Future work

We foresee two main categories of additional work. The first category is further development of the method we are advocating. There are numerous interesting topics that merit further investigation. A critical topic for data integration is the definition and implementation of a query language for the sheaf model, analogous to SQL for the relational model. Two other important ones are (1) versioning can be treated as refinement and (2) data and task decomposition for parallel and distributed computing can be described within the sheaf model.

The second category is much larger: application of the method to specific cases, tasks, and integration issues within subsurface workflows. Only by using the sheaf data metamodel to create actual explicit data models and applications can the approach be reduced to practice and hence actually contribute to shared earth modeling. We hope that this paper will stimulate interest within the research and development community to contribute to this undertaking.

Conclusion

In this paper, we have introduced a novel mathematical data metamodel, the sheaf model. We showed that the model encourages a mathematical approach to data modeling, emphasizes the importance of the field data type, and provides an integrated framework for describing conceptual and concrete models. We demonstrated the potential of the model to facilitate several tasks within the scope of shared earth modeling and placed the model in context between a more abstract knowledge-driven approach and a more explicit data-driven approach. Finally, we outlined future work suggested by our approach.

Acknowledgments

The authors thank Shell for permission to publish this paper, and we especially thank Shell GameChanger for their unwavering support through many proofs of concept. Comments from five anonymous reviewers, K. Osypov, S. Tockey, and C. Ramshorn are greatly appreciated and led to major improvements of this paper.

Appendix A

Summary of mathematical definitions for sets

A *set* is a collection of arbitrary objects referred to as *members* of the set. If S is a set, then $s \in S$ denotes that s is a member of S. A set is denoted by surrounding an explicit list of its members, or a specification of its members, in braces. For instance, {0, 1} or {i|i is an odd integer}. The *cardinality* of a set, denoted |S|, is the number of members of the set. If the cardinality is a nonnegative integer, we say the set is *finite*; otherwise, it is *infinite*. The *empty set* is the set with no members, denoted as ∅ or {}.

In the remainder of this appendix, let $S1, S2, \ldots, Sn$ be sets.

S1 *equals* S2, denoted $S1 = S2$, if and only if S1 has exactly the same members as S2.

S2 is a *subset* of S1, denoted $S2 \subseteq S1$, if and only if all the members of S2 are also members of S1. We also say that S2 is *included* in S1 or S1 includes S2.

The *union* of S1 and S2, denoted $S1 \cup S2$, is the set that contains those members that are either in S1 or in S2 or in both. The *intersection* of S1 and S2, denoted $S1 \cap S2$, is the set that contains those members that are in S1 and S2. The *difference* of S1 and S2, denoted S1 – S2, is the set containing those members in S1 that are not in S2.

The *binary Cartesian product* of S1 and S2, denoted $S1 \times S2$, is the set of all pairs $\{(s1 \in S1, s2 \in S2)\}$. The binary Cartesian product can be iterated to form the n-ary Cartesian product, denoted $S1 \times S2 \times, \ldots, \times Sn$ or $\prod_i Si$, which is the set of all *n*-tuples $\{(s1 \in S1, s2 \in S2, \ldots, sn \in Sn)\}$.

The *Cartesian power*, or *exponential*, denoted $S2^{S1}$, is the n-ary Cartesian product $S2 \times S2 \times, \ldots, \times S2$, in which the number of factors $n = |S1|$. In other words, there is one factor for each member of S1.

For further information on sets see Rosen (1995) and Halmos (1974).

Appendix B

Summary of mathematical definitions for relations and maps

A *binary relation* on S1 and S2 is any subset of $S1 \times S2$. An n-ary relation on a family of sets S1, S2, ..., Sn is a subset of the n-ary Cartesian product $\Pi_i Si$.

A *map* or *function* φ from S1 to S2 denoted $\varphi: S1 \rightarrow S2$, can be defined in two equivalent ways:

1) a binary relation on S1 and S2 such that there is exactly one pair in the relation for each member of S1
2) a member of the Cartesian power $S2^{S1}$.

A map is called *one to one* or *injective* if $\varphi(s) = \varphi(s')$ implies $s = s'$, for all $s, s' \in S1$. A map is called *onto* or *surjective* if every member of S2 is the image of some member of S1. A map is called a *one-to-one correspondence* or a *bijection* if it is injective and surjective.

A map φ: $S \to S$ such that $\varphi(s) = s$ for all $s \in S$ is called the *identity map*.

Let φ: $S1 \to S2$ be a bijection. The *inverse* of φ denoted φ^{-1}: $S2 \to S1$ is defined by $\varphi^{-1}(s2) = s1$ if and only if $\varphi(s1) = s2$, for all $s2 \in S2$.

If $\varphi1$: $S1 \to S2$ and $\varphi2$: $S2 \to S3$, the *composition* of $\varphi1$ and $\varphi2$ denoted $\varphi2 \circ \varphi1$ is defined by $(\varphi2 \circ \varphi1)(s) = \varphi2(\varphi1(s))$, for all $s \in S1$.

For further information on relations and maps, see Rosen (1995) and Halmos (1974).

Appendix C

Summary of mathematical definitions for topological spaces

A *topological space* $\mathcal{X}$ is a pair (X, τ) where X is a set and τ is a set of subsets of X called the *open sets*. The open sets must satisfy:

1) Any union of open sets is open.
2) The intersection of any two open sets is open.
3) $\varnothing$ and X are open.

The set of open sets τ is called the *topology* of the topological space (X, τ). If τ is the set of all subsets of X, it is called the *discrete topology*.

If $\mathcal{X}$ and $\mathcal{Y}$ are topological spaces, a map φ: $\mathcal{X} \to \mathcal{Y}$ is called *continuous* if the inverse image of any open set in $\mathcal{Y}$ is open in $\mathcal{X}$. If φ is a bijection, it is called a *homeomorphism* if φ and its inverse are continuous. If $\mathcal{X}$ and $\mathcal{Y}$ are related by a homeomorphism, they are said to be *homeomorphic* or *topologically equivalent*.

For further information on topological spaces, see Janich (1984).

Appendix D

Summary of mathematical definitions for groups

A *group* is a set G together with a map:

$$+ : G \times G \to G : (g1, g2) \mapsto g1 + g2.$$

The map is called *addition*, and it must satisfy the following axioms:

1) $(g1 + g2) + g3 = g1 + (g2 + g3)$ for all g1, g2, $g3 \in G$.
2) There exists $0 \in G$ such that $g + 0 = g$ for all $g \in G$.
3) For each $g \in G$, there exists an element $-g \in G$ such that $g + (-g) = 0$.

A group is called *abelian* or *commutative* if it satisfies the additional axiom:

4) $g1 + g2 = g2 + g1$ for all g1, $g2 \in G$.

For more information on groups, see Gilmore (2005).

Appendix E

Summary of mathematical definitions for vector spaces

A *vector space* over the real numbers R is a set V together with two maps:

$$+ : V \times V \to V : (v1, v2) \mapsto v1 + v2 \text{ and}$$
$$* : R \times V \to V : (r, v) \mapsto r * v.$$

The first map is called *addition* and the second map is called *scalar multiplication*. The maps must satisfy the following axioms:

1) V is an abelian group under addition;
2) $r1 * (r2 * v) = (r1 * r2) * v$ for all r1, $r2 \in R$ and all $v \in V$.
3) $1 * v = v$ for all $v \in V$.
4) $r * (v1 + v2) = r * v1 + r * v2$ for all $r \in R$ and all v1, $v2 \in V$.
5) $(r1 + r2) * v = r1 * v + r2 * v$ for all r1, $r2 \in R$ and all $v \in V$.

The set of real numbers R is a vector space over itself.

A function f: $V1 \to V2$, in which V1 and V2 are vector spaces is called a *linear function* if it satisfies the following:

$f(v1 + v2) = f(v1) + f(v2)$ for all v1, $v2 \in V1$ and
$f(r * v) = r * f(v)$ for all $r \in R$ and $v \in V1$.

The set of all linear functions from V1 to V2 forms a vector space, denoted L(V1, V2), with addition and scalar multiplication defined as follows:

$(f + g)(v) = f(v) + g(v)$ for all f, $g \in L$ and $v \in V1$ and
$(r * f)(v) = r * f(v)$ for all $f \in L$ and $v \in V1$.

The vector space $L(V, R)$ is called the *dual space* of V and denoted V*.

If V1, V2, and V3 are vector spaces, a function f: $V1 \times V2 \to V3$ is called *bilinear* if it is linear in each variable, that is, if it satisfies the following:

$$f(r * v1 + r' * v1', v2) = r * f(v1, v2) + r' * f(v1', v2) \text{ and}$$
$$f(v1, r * v2 + r' * v2') = r * f(v1, v2) + r' * f(v1, v2')$$

for all r, $r' \in R$, v1, $v1' \in V1$, and v2, $v2' \in V2$.

Functions with n arguments, linear in each argument, are called *multilinear* functions.

Given a vector space V, a scalar valued multilinear function with all arguments in either V or V* is called a *tensor* over V.

For more information on vector spaces, see Janich (1994) and Bishop and Goldberg (1980).

Appendix F

Summary of mathematical definitions for fiber bundles

A *fiber bundle* consists of five parts (E, B, F, G, and π) satisfying the following requirements:

1) E is a topological space called the *bundle space*.
2) B is a topological space called the *base space*.
3) F is a topological space called the *fiber space*.
4) G is a group of homeomorphisms of the fiber F called the *structure group*.
5) π: E $\rightarrow$ B is a continuous surjective map called the *projection*.
6) For every point p $\in$ B, there is a *chart* (U, η) consisting of an open set U containing p and a homeomorphism $\eta : \pi^{-1}(\mathrm{U}) \rightarrow \mathrm{U} \times \mathrm{F}$.
7) Wherever two charts ($\mathrm{U_i}$, η_i), ($\mathrm{U_j}$, η_j) overlap, the *transition function* $\mathrm{g_{ij}(p)} : \mathrm{F} \rightarrow \mathrm{F}$ defined by $\mathrm{g_{ij}} = \eta_\mathrm{i}(\eta_\mathrm{j}^{-1}(\mathrm{p}, \mathrm{f}))$ for fixed p $\in \mathrm{U_i} \cap \mathrm{U_j}$ and any f $\in$ F is a member of G.

A continuous map $\sigma : \mathrm{B} \rightarrow \mathrm{E}$ such that $\pi(\sigma(\mathrm{p})) = \mathrm{p}$ for all p $\in$ B is called a *cross section*, or just *section*, of the bundle E.

For more information on fiber bundles, see Crampin and Pirani (1986) and Nash and Sen (1983).

Appendix G

Summary of mathematical definitions for partially ordered sets and lattices

A *preorder relation*, denoted $\leq$ and read "less than or equal to," on a set S is any relation with the following properties:

1) $\mathrm{s} \leq \mathrm{s}$ (*reflexivity*) and
2) $\mathrm{s1} \leq \mathrm{s2}$ and $\mathrm{s2} \leq \mathrm{s3}$ implies $\mathrm{s1} \leq \mathrm{s3}$ (*transitivity*),

where s, s1, s2, and s3 are arbitrary members of S.

A *partial order relation*, also denoted $\leq$, is a preorder relation that additionally satisfies

3) $\mathrm{s1} \leq \mathrm{s2}$ and $\mathrm{s2} \leq \mathrm{s1}$ implies $\mathrm{s1} = \mathrm{s2}$ (*antisymmetry*).

A *preordered set* $\mathcal{S}$ is a pair (S, $\leq$) in which S is any set, referred to as the *base set*, together with a preorder relation $\leq$. A *partially ordered set* (poset) is a preordered set in which the order relation is a partial order.

The relation *greater than or equal*, denoted $\geq$, is defined by $\mathrm{s1} \geq \mathrm{s2}$ if and only if $\mathrm{s2} \leq \mathrm{s1}$.

The relation *strict inequality*, denoted $<$, is defined by $\mathrm{s1} < \mathrm{s2}$ if and only if $\mathrm{s1} \leq \mathrm{s2}$ and $\mathrm{s1} \neq \mathrm{s2}$.

The *covers* relation, denoted $\rightarrowtail$, is defined by $\mathrm{s1} \rightarrowtail \mathrm{s2}$ if and only if $\mathrm{s2} \leq \mathrm{s1}$ and there is no s3 such $\mathrm{s2} \leq \mathrm{s3} \leq \mathrm{s1}$.

Any subset $\mathcal{S}'$ of a poset $\mathcal{S}$ is also a poset with the order inherited from $\mathcal{S}$.

Let $\mathcal{S}$ and $\mathcal{T}$ be posets and let $\varphi : \mathcal{S} \rightarrow \mathcal{T}$ be a map. Then, φ is called

- *order preserving* if $\mathrm{s1} \leq \mathrm{s2}$ in $\mathcal{S}$ implies $\varphi(\mathrm{s1}) \leq \varphi(\mathrm{s2})$ in $\mathcal{T}$,
- an *order embedding* if $\mathrm{s1} \leq \mathrm{s2}$ in $\mathcal{S}$ if and only if $\varphi(\mathrm{s1}) \leq \varphi(\mathrm{s2})$ in $\mathcal{T}$, and
- an *order isomorphism* if it is an order-embedding and surjective.

If there exists $\mathrm{s} \in \mathcal{S}$ such that $\mathrm{s}' \leq \mathrm{s}$ for all $\mathrm{s}' \in \mathcal{S}$, then s is called the *top* of $\mathcal{S}$, denoted $\top$. If there exists $\mathrm{s} \in \mathcal{S}$ such that $\mathrm{s} \leq \mathrm{s}'$ for all $\mathrm{s}' \in \mathcal{S}$, then s is called the *bottom* of $\mathcal{S}$, denoted $\bot$.

A subset $\mathcal{S}' \subseteq \mathcal{S}$ is called a *down set* if whenever $\mathrm{s1} \in \mathcal{S}'$, $\mathrm{s2} \in \mathcal{S}$ and $\mathrm{s2} \leq \mathrm{s1}$, then $\mathrm{s2} \in \mathcal{S}'$. Given $\mathrm{s} \in \mathcal{S}$, the down set *down s*, denoted $\downarrow$s, is defined by $\downarrow \mathrm{s} = \{\mathrm{s}' \in \mathcal{S} \mid \mathrm{s}' \leq \mathrm{s}\}$. Given any subset $\mathcal{S}' \subseteq \mathcal{S}$, the down set *down* $\mathcal{S}'$, denoted $\downarrow \mathcal{S}'$, is defined by $\downarrow \mathcal{S}' = \{\mathrm{s}' \in \mathcal{S} \mid \mathrm{s}' \leq \mathrm{s}$ for some $\mathrm{s} \in \mathcal{S}'\}$. *Up sets* $\uparrow$s and $\uparrow \mathcal{S}'$ are defined by replacing $\leq$ with $\geq$ in the corresponding down definitions.

The *set of all down sets* of $\mathcal{S}$ is denoted $\mathcal{O}(\mathcal{S})$ and is itself a poset, ordered by subset inclusion.

Given $\mathcal{S}' \subseteq \mathcal{S}$, $\mathrm{s} \in \mathcal{S}$ is called a *lower bound* of $\mathcal{S}'$ if $\mathrm{s} \leq \mathrm{s}'$ for all $\mathrm{s}' \in \mathcal{S}'$. If the set of all lower bounds of $\mathcal{S}'$ has a greatest member, it is called the *greatest lower bound* or *meet* of $\mathcal{S}'$, denoted $\bigwedge \mathcal{S}'$. If $\mathcal{S}'$ has only two members s1 and s2, the meet is denoted $\mathrm{s1} \wedge \mathrm{s2}$.

Similarly, $\mathrm{s} \in \mathcal{S}$ is called an *upper bound* of $\mathcal{S}'$ if $\mathrm{s}' \leq \mathrm{s}$ for all $\mathrm{s}' \in \mathcal{S}'$. If the set of all upper bounds of $\mathcal{S}'$ has a least member, it is called the *least upper bound* or *join* of $\mathcal{S}'$, denoted $\bigvee \mathcal{S}'$. If $\mathcal{S}'$ has only two members s1 and s2, the join is denoted $\mathrm{s1} \vee \mathrm{s2}$.

A poset $\mathcal{S}$ for which $\mathrm{s1} \wedge \mathrm{s2}$ and $\mathrm{s1} \vee \mathrm{s2}$ exist for all s1 and s2 in $\mathcal{S}$ is called a *lattice*.

A lattice $\mathcal{S}$ is called *distributive* if it satisfies $\mathrm{s1} \wedge (\mathrm{s2} \vee \mathrm{s3}) = (\mathrm{s1} \wedge \mathrm{s2}) \vee (\mathrm{s1} \wedge \mathrm{s3})$ for all s1, s2, and s3 in $\mathcal{S}$.

A member s of a lattice $\mathcal{S}$ is called *join irreducible* if $\mathrm{s} \neq \bot$ and $\mathrm{s} = \mathrm{s1} \vee \mathrm{s2}$ implies $\mathrm{s} = \mathrm{s1}$ or $\mathrm{s} = \mathrm{s2}$. The set of all join-irreducible members of $\mathcal{S}$ is denoted $\mathcal{J}(\mathcal{S})$.

The *Birkhoff representation theorem* states that any finite distributive lattice $\mathcal{S}$ is isomorphic to $\mathcal{O}(\mathcal{J}(\mathcal{S}))$.

The *lattice tensor product* of lattice $\mathcal{S}1$ and $\mathcal{S}2$, denoted $\mathcal{S}1 \otimes \mathcal{S}2$, is defined by $\mathcal{S}1 \otimes \mathcal{S}2 = \mathcal{O}(\mathcal{J}(\mathcal{S}1) \times \mathcal{J}(\mathcal{S}2))$.

The part space metaphor interprets join-irreducible members as basic parts and all other members as composite parts.

For more information on posets and lattices see Davey and Priestley (2002).

Appendix H

Summary of mathematical definitions for sheaves

The general definition of a sheaf is usually given in the context of category theory. We give a more specific definition here that is equivalent in the context of finite distributive lattices.

Let $\mathcal{S} = (\mathrm{S}, \leq)$ be an finite distributive lattice. A *presheaf* $\mathbf{\Phi}$ of sets on $\mathcal{S}$ consists of

1) a family of sets $\mathrm{F} = \{\mathrm{F(s)} | \mathrm{s} \in \mathcal{S}\}$
2) a family of maps $\{\rho_{\mathrm{s's}} : \mathrm{F(s)} \rightarrow \mathrm{F(s')}$ for all s, $\mathrm{s}' \in \mathcal{S}$ with $\mathrm{s}' \leq \mathrm{s}\}$.

The family F is called the *target* of the presheaf and each member F(s) is called the *set of sections* over s. The maps are called *restriction maps* and must satisfy the further conditions:

3) ρ_{ss} = identity map on F(s)
4) whenever $s'' \le s' \le s$, $\rho_{s''s} = \rho_{s''s'} \circ \rho_{s's}$.

We define a relation on F, denoted $\le_F$, by $F(s') \le_F F(s)$ if and only if there exists a restriction map $\rho_{s's}$: $F(s) \to F(s')$. Then, $\le_F$ is reflexive and transitive by conditions (3) and (4), respectively. Hence, $\le_F$ is a preorder relation and $\mathcal{F} = (F, \le_F)$ is a preordered set.

We can view a presheaf $\mathbf{\Phi}$ as an order-preserving map $\mathbf{\Phi}$: $\mathcal{S} \to \mathcal{F}$.

Let $s \in \mathcal{S}$ and let $C = \{s_c\} \subseteq S$ such that $s = \bigvee_C s_c$; then C is called a *covering* of s.

Let $\mathcal{C} = \{f_{s'} | f_{s'} \in F(s'), s' \in C\}$ be a family of sections such that $\rho_{s'\cap s''s''}(f_s) = \rho_{s'\cap s''s'}(f_{s'})$ for all s', $s'' \in C$; then $\mathcal{C}$ is called a *compatible family* for the covering C.

A presheaf $\mathbf{\Phi}$ is a *sheaf* if it satisfies, for all members s and coverings C,

5) $\rho_{s's}(f) = \rho_{s's}(f')$ for all $s' \in C$ implies $f = f'$, where $f, f' \in F(s)$; and
6) for every compatible family $\mathcal{C}$ on C, there exists $f \in F(s)$ such that $f_{s'} = \rho_{s's}(f)$ for all $f_{s'} \in \mathcal{C}$.

Condition 5 states that a section is uniquely determined by its restrictions to a covering. Condition 6 states that every compatible family on a covering of s comes from restriction of a section over s.

For more information on sheaves, see Miraglia (2006).

References

Benger, W., 2009, On safari in the file format jungle — Why can't you visualize my data?: Computing in Science and Engineering, **11**, 98–102, doi: 10.1109/MCSE.2009.202.

Benger, W., 2011, The fiber bundle HDF5 library, http://sciviz.cct.lsu.edu/projects/F5, accessed 16 May 2015.

Bishop, R. L., and S. I. Goldberg, 1980, Tensor analysis on manifolds: Dover Publications, Inc.

Butler, D. M., 2013a, Part space, http://www.limitpoint.com/Publications/PartSpace.pdf, accessed 18 May 2015.

Butler, D. M., 2013b, Part spaces for scientific computing, http://www.limitpoint.com/Publications/PartSpacesForScientificComputing.pdf, accessed 18 May 2015.

Butler, D. M., and S. Bryson, 1992, Vector bundle classes form powerful tool for scientific visualization: Computers in Physics, **6**, 576–584, doi: 10.1063/1.4823118.

Butler, D. M., and M. H. Pendley, 1989a, The visualization management system approach to visualization in scientific computing: Computers in Physics, **3**, 40–44, doi: 10.1063/1.168344.

Butler, D. M., and M. H. Pendley, 1989b, A visualization model based on the mathematics of fiber bundles: Computers in Physics, **3**, 45–51, doi: 10.1063/1.168345.

Crampin, M., and F. A. E. Pirani, 1986, Applicable differential geometry: Cambridge University Press.

Davey, B. A., and H. A. Priestley, 2002, Introduction to lattices and order: 2nd ed.: Cambridge University Press.

Energistics and the RESQML SIG, 2014a, RESQML technical reference guide, http://w3.energistics.org/energyML/data/resqmlv2/v2.0/doc/RESQML%20Technical%20Reference%20Guide.pdf, accessed 16 May 2015.

Energistics and the RESQML SIG, 2014b, RESQML technical usage guide, %%%http://w3.energistics.org/energyML/data/resqmlv2/v2.0/doc/RESQML%20Technical%20Usage%20Guide.pdf, accessed 28 April 2015.

Gilmore, R., 2005, Lie groups, lie algebras, and some of their applications: Dover Publications, Inc.

Haber, R. B., B. Lucas, and N. S. Collins, 1991, Data model for scientific visualization with provisions for regular and irregular grids: Proceedings IEEE Visualization.

Halmos, P. R., 1974, Naive set theory: Springer-Verlag.

Janich, K., 1984, Topology: Springer-Verlag.

Janich, K., 1994, Linear algebra: Springer-Verlag.

Limit Point Systems, Inc., 2015, SheafSystem.org, http://www.sheafsystem.org, accessed 5 May 2015.

Miraglia, F., 2006, An introduction to partially ordered structures and sheaves: Polimetrica.

Nash, C., and S. Sen, 1983, Topology and geometry for physicists: Academic Press.

Object Management Group, Inc., 2015, "Unified Modeling Language™ (UML®)" Resource Page, http://www.uml.org, accessed 16 May 2015.

Oxford Dictionaries, 2015, Refine, http://www.oxforddictionaries.com/us/definition/american_english/refine, accessed 18 May 2015.

Perrin, M., and J. F. Rainaud, 2013, Shared earth modeling: Editions Technip.

Rosen, K. H., 1995, Discrete mathematics and its applications: McGraw-Hill.

Varzi, A., 2015, Mereology, http://plato.stanford.edu/archives/spr2015/entries/mereology, accessed 5 May 2015.

Visualization and Imagery Solutions, Inc., 2006, OpenDX, http://www.opendx.org, accessed 5 May 2015.

World Wide Web Constortium, 2015, Schema, http://www.w3.org/standards/xml/schema, accessed 16 May 2015.

Mark Verschuren received a Ph.D. in geology from Ghent University, Belgium. He started designing geoscience software technology approximately 30 years ago and has continued to do so at Shell. Making complex geoscience easy to learn and do with a computer required deeper understanding of the science as well as the data models to integrate it all, to make it interactive, and to learn more from the data faster. With this approach, he has dealt so far with interpretive modeling of complex fault and salt structural frameworks, faulted stacked reservoirs and stratigraphic age, interpretive seismic imaging

with Kirchhoff and reverse time migration, velocity scenario testing, and interpreter-guided seismic velocity inversion, all on previously disintegrated interfaces between geophysics and geology.

David M. Butler received a Ph.D. in physics from the University of Pittsburgh and has more than 30 years of experience in developing systems for processing scientific data in a wide range of application areas. He is the founder and president of Limit Point Systems Inc. His work has emphasized research and development in data models for scientific computing, software architecture, and programming in C++ and other languages. He is a member of ACM, IEEE Computer Society, SIAM, APS, PATCA, and LinkedIn.