TARS: An Array Model with Rich Semantics for Multidimensional Data

Hermano Lustosa[1], Noel Lemus[1], Fabio Porto[1], and Patrick Valduriez[2]

[1] National Laboratory for Scientific Computing, DEXL Lab, Petropolis, RJ, Brazil
[2] Inria and LIRMM, University of Montpellier, France

Abstract. Relational DBMSs have been shown to be inefficient for scientific data management. One main reason is the difficulty to represent arrays, which are frequently adopted as a data model for scientific datasets representation. Array DBMSs, e.g. SciDB, were proposed to bridge this gap, building on a native array representation. Unfortunately, important scientific applications, such as numerical simulation, have additional requirements, in particular to deal with mesh topology and geometry. First, transforming simulation results datasets into DBMS array format incurs in huge latency due to the fixed format of array DBMSs layouts and data transformations to adapt to mesh data characteristics. Second, simulation applications require data visualization or computing uncertainty quantification (UQ), both requiring metadata beyond the simulation output array. To address these problems, we propose a novel data model called TARS (Typed ARray Schema), which extends the basic array data model with typed arrays. In TARS, the support of application dependent data characteristics, such as data visualization and UQ computation, is provided through the definition of TAR objects, ready to be manipulated by TAR operators. This approach provides much flexibility for capturing internal data layouts through mapping functions, which makes data ingestion independent of how simulation data has been produced, thus minimizing ingestion time. In this paper, we present the TARS data model and illustrate its use in the context of numerical simulation application.

Keywords: Data Model, Multi-dimensional array, simulation

1 Introduction

The advent of exascale computing creates a major challenge for scientific data management, i.e. storing, processing and obtaining insights from big datasets. An important class of scientific applications is large-scale numerical simulation, which models natural phenomena in different domains such as oil and gas, medicine, meteorology, etc. These applications produce a high number of output datasets that are used in scientific visualization and data analysis. However, data management for simulation applications is difficult. Data produced by simulations include large datasets with simulation results and space discretization in the form of a mesh. Simulation applications have stringent requirements for efficient data ingestion which takes place during real-time visualization and simulation with interaction with scientists. Furthermore, simulation applications require support for complex analysis, as in uncertainty quantification (UQ).

As data can be generated faster and faster by large-scale numerical simulation applications, current data management technologies, e.g. relational DBMSs, become a bottleneck [1]. Scientific data produced by simulations or captured in observations naturally exhibit a multidimensional representation, in which attribute values vary in space-time. Thus, the array data model, as implemented in SciDB [11], has been proposed to represent scientific datasets [14]. Unfortunately, simulation application data have additional requirements that are not addressed by array DBMSs. Although most simulation data can be well represented by multidimensional arrays, the simulation mesh or grid is hard to represent in a pure array data model [9]. To address this problem, a few data models have been proposed to model mesh datasets generated by numerical simulations [7, 8, 13]. These data models are rather specific to mesh data, and less efficient for modeling the geometry aspects of simulation data. So far, none of these data models has been implemented in a complete system yet.

Besides, loading and indexing simulation data into a DBMS, irrespective of the data model, incur high overhead, thus preventing scientists from adopting DBMSs at all. Instead, scientists typically rely on I/O libraries that provide support for multidimensional arrays, e.g. HDF [6] and netCDF [15]. These libraries give the users more control over their data without incurring the performance penalties for data loading in a DBMS [3, 5]. They are also flexible, allowing the users to specify how their data (produced by numerical simulation) is laid out and avoid the expensive conversions performed by the DBMS during data ingestion.

However, I/O libraries do not necessarily offer all benefits that a full-fledged DBMS does, which calls for a compromise between approaches. NoDB [2] is a first attempt to bridge the gap between DBMS high ingestion costs and I/O libraries access efficiency for relational DBMSs. The approach advocates that the DBMS should be able to work with data as laid out by the data producer, with no overhead for data ingestion and indexing. Any subsequent data transformation or indexing performed by the DBMS in order to improve the performance of data analyses should be done adaptively as queries are submitted. We believe that even though NoDB is currently implemented on top of a RDBMS and lacks support for multidimensional data, its philosophy can be successfully applied for array databases as well.

Last but not least, scientific application requirements for dataset analysis go beyond basic array operations, such as slice, subarrays, joins, etc. [10]. Scientific visualization, for instance, is the standard procedure to assess and analyze simulation results. Currently the interface with a visualization tool is implemented externally to the DBMSs, requiring an extra and costly data transformation. Such transformation can be implemented as a DBMS operation provided additional visualization information is associated to the simulation output. Similarly, uncertainty quantification analysis relies on the interpretation of probability distribution functions (PDF) on output variable values to compute simulation result uncertainty. In this scenario, the data model would need to be extended with : trial -ids, time-steps and PDFs.

Considering these aforementioned problems, a novel data model is needed to fulfill the requirements of current and future demanding scientific applications. First, this model should be based on multidimensional arrays, which are the natural model for scientific data. Second, the model should adhere to the NoDB philosophy, allowing users to append large datasets regardless of their memory layout, and adapting itself to encompass these same datasets without costly data conversions. Finally, the array model must be extended to conform with particular characteristics of numerical simulation. It should provide a mechanism for semantic annotations related to data elements. These annotations will help determining the semantics of every dimension and attribute of an array throughout array transformations during query execution. These annotations should classify arrays into types, involving special dimensions and attributes.

In this paper, we propose a novel data model called TARS (Typed ARray Schema), which extends the basic array data model with typed arrays (TARs). TARS can be used as the underlying data model for a simulation data management system, providing a powerful query language with array operators to allow users to express a variety of analytical queries and determine how results are to be visualized. To validate our approach, we show how a TARS schema could be used in the context of a numerical simulation application.

This paper is organized as follows. Section 2 gives a brief overview of the array data model and discusses its limitations. In Section 3, we provide a general overview of TARS. In Section 4, we give a formal definition of the data model. In Section 5, we illustrate TARS by defining a schema for a generic simulation application use case. Section 6 concludes.

2 Multidimensional Arrays

In our previous work [9], we considered the use of the array data model to manage simulation data. A simple definition of the array data model is presented by [10]. In short, an array is a regular structure formed by a set of dimensions. A set of integer indexes for all dimensions identify a cell or tuple containing values for a set of array attributes.

If carefully designed, arrays offer many advantages when compared to simple bidimensional tables. Cells in an array have an implicit order defined by how the array data is laid out in linear storage. We can have row-major, column-major or any other arbitrary dimension ordering. Array database management systems can quickly lookup data and carry out range queries by taking advantage of this implicit ordering. If the data follows a well behaved array-like pattern, using arrays saves a lot of storage space, since dense arrays indexes do not need to be explicitly stored. Furthermore, arrays can be split into subarrays, usually called tiles or chunks. These subarrays are used as processing and storage data units. They help answering queries rapidly and enforce a coherent multidimensional data representation in linear storage.

However, current array data model implementations, e.g. [11,12], have some limitations, preventing an efficient representation of simulation datasets. For instance, suppose that an application needs to condense many datasets into a single array, and that these datasets have different mappings of array cells into linear storage, i.e., one adopts a row major linearization while another follows a column major ordering. In this case, current array DBMSs offer a single mapping for arrays into memory and would need to reorder data as it is loaded, so that all subarrays obey the same array cell to memory layout, either row o column major.

In SciDB [11] for instance, it is sometimes necessary to preload multidimensional data into an unidimensional array and then rearrange it before querying. Rasdaman [12], another array database, requires either the creation of a script or the generation of compatible file formats for data ingestion. This may also require costly ASCII to binary conversion (since numerical data is likely to be created in binary format) for adjusting the data to the final representation on disk. In both cases, the amount of work for loading the dataset alone is proportional to its size, making it impractical for the multi-terabyte data output by complex modern simulation applications.

Furthermore, the array data model does not explicitly incorporate the existence of dimensions whose indexes are non-integer values. In some applications, e.g. simulations, the data follows an array-like pattern, but one of the identifiable dimensions can be actually a non-integer attribute. For instance, in 3D rectilinear regular meshes, we have points distributed in spatial dimensions whose indexes or coordinate values are usually floating point numbers. To address this issue, we need to map non-integer values into integer indexes that specify positions within the array. Array DBMSs like SciDB or Rasdman do not support this kind of functionality currently.

Arrays can also be sparse, meaning that there is no data values for every single array cell. Data may also have some variations in their sparsity from a portion of the array to another. This is the case for complex unstructured meshes geometry (with an irregular point distribution in space) when directly mapped to arrays. SciDB provides support to sparse arrays, but since it splits an array into chunks (equally sized subarrays), it is very hard to define a balanced partitioning scheme, because data can be distributed very irregularly. Rasdaman is more flexible in this regard, and allows arrays to be split into tiles or chunks with variable sizes.

Another characteristic of complex multidimensional data representation is the existence of partial functional dependencies with respect to the set of indexes. Consider a 3D array A with dimensions x, y and z and a set of attributes S. Consider the attribute $v \in S$. Suppose that logically, every cell in A has a well defined v value and that this values is potentially different for every combination of x and y index but remains the same with respect to the z dimension. This means that v functionally depends on x and y but not z, thus characterizing a partial functional dependency. The solution with the relational data model to avoid unnecessary data redundancy in this case is normalization, which would require removing v from A, adding it to another array, say B with only x and y and joining A and B to recreate the full dataset. However, since arrays are well structured and the repetition of values for different dimension indexes follows a regular pattern, normalization could be done transparently by the array DBMS.

Partial dependencies occur in constant or varying mesh geometries and topologies, or any other kind of data that does not necessarily varies along all array dimensions. For instance, when researchers create models for simulating transient problems, the time is a relevant dimension to all data. However, the mesh, which is the representation of the spatial domain, may not change in time, meaning that the coordinate values and topology incidence remain the same throughout the entire simulation. Another possibility is the usage of the same mesh for a range of trials, and another mesh for another range. In both cases, there is a mesh for every single time step (an index in the array time dimension) or trial, but actually only one mesh representation needs to stored for an entire range of indexes.

Finally, the context of numerical simulation and UQ involves very specific data semantics for various data attributes. In simulation data, we usually have the repetition of a similar structure of values for many points or mesh elements in space and time for various simulation trials. The diverse data structure of field data (actual output values for simulations), geometry (coordinate values) and topologies (adjacency relationships) should be explicitly represented in the data model, allowing the definition of special purpose algebraic operators that are useful for creating complex analysis. Therefore, in TARS, we devise a mechanism for allowing the creation of types that enable users to qualify their datasets in accordance to the application semantics.

3 TARS Overview

The TARS (Typed Array Schema) data model extends the basic array data model to cope with complex multidimensional data. A TARS contains a set of typed arrays (TAR). A TAR has a set of data elements: dimensions and attributes. A TAR cell is a tuple of attributes accessed by a set of indexes. These indexes define the cell location within the TAR. A TAR has a type, formed by a set of roles. A role in a type defines a special purpose data element with specific semantics. If a TAR is of a given type T, it is guaranteed to have a set of data elements that fulfill the roles defined in T. This facilitates the creation of operations that require additional semantics about the data. In TARS, we define mapping functions as a way to provide support for sparse arrays, non-integer dimensions, heterogeneous memory layouts and functional partial dependencies with respect to dimensions. Figure 1 gives a general view of the model.

A TAR region is instantiated with data as a subTAR. A subTAR encompasses an n-dimensional slice of a TAR. Every subTAR is defined by the TAR region it represents and two mapping functions: position mapping function and data mapping function. The former reflects the actual data layout in memory, since it defines where every TAR cell within a given subTAR ended in linear storage.

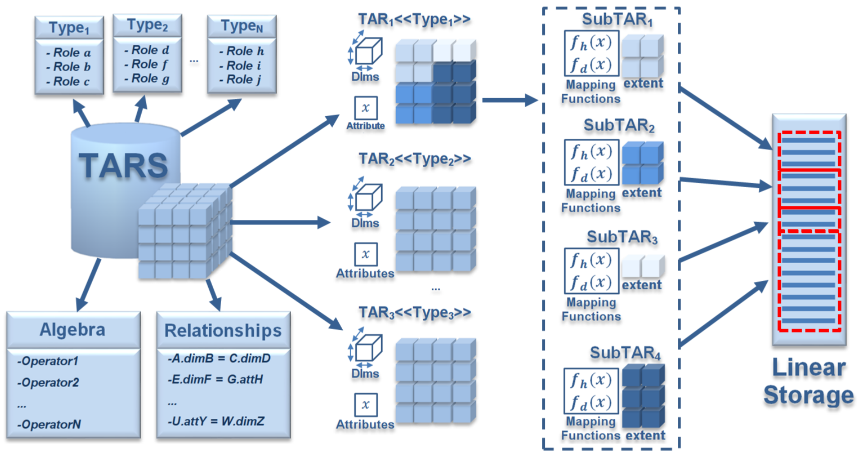


Fig. 1. Typed Array Schema with its main elements: types, operators, relationships, TARs and SubTARs

Therefore, the position mapping function should implement the multidimensional linearization technique used for the data. It can incorporate an n-dimensional to linear translation based on how dimensions are ordered (the case for a row-major or column-major scheme) or even consider a space filling curve approach (e.g. Z-order). The data mapping functions translate a linear address into actual data values. In a simple scenario, this function does basically a lookup into a linear array that stores the data. In a more complex scenario, it could compute a derived value from the actual subTAR data. The actual implementation of mapping functions is done in various forms. In some cases, it may be necessary to explicitly store the mapping for every value. In other situations, the mapping occurs in a regular pattern, so translation is done by a single parametrized procedure, in which case only the parameters need to be stored.

SubTARs not only define a partitioning scheme for a TAR, but also serve as a way to allow users to specify the details about how their data is laid out, avoiding costly data transformations and rearrangements during ingestion into the DBMS.

4 TARS Formalization

A TARS denoted by Γ is a quintuple $(\Omega_\Gamma, R_\Gamma, T_\Gamma, L_\Gamma, \Theta_\Gamma)$ where Ω_Γ is a set of typed arrays, R_Γ is a set of roles, T_Γ is a set of types, L_Γ is a set of links or relationships, and Θ_Γ is a set of operators forming a TAR algebra.

A Typed Array (TAR) $A \in \Omega_\Gamma$ is a septuple $(N_A, D_A, S_A, C_A, R_A, \Phi_A, \Upsilon_A)$, where N_A is a string containing the TAR name, D_A is a set of data elements, S_A is a set of subTARs, C_A is a set of locations forming the TAR location space, R_A is the subTARs location function, Φ_A is the dimension data mapping function, and Υ_A is the role mapping function.

A data element $e \in D_A$ is a triple (N_e, V_e, D_e), where N_e is a string value defining the data element name, V_e and D_e are sets of atomic values such that $V_e \subset D_e$. D_e is the domain of the data element, representing the set of all possible values a TAR can hold in that data element, and V_e is the data element image, containing the actual set of values for a data element at any given time.

D_A can be divided into two subsets, $Dim_A \subset D_A$ containing data elements that are dimensions and $Att_A \subset D_A$ containing data elements that are tuple attributes. We also have that $Dim_A \cap Att_A = \emptyset$, meaning that a data element is either a dimension or an attribute and never both.

A subTAR $s \in S_A$ is a triple (η_s, Fi_s, Fd_s) where η_s is the set of TAR locations that represents the extent of the subTAR, Fi_s is the subTAR position mapping function, and Fd_s is the subTAR data mapping function. The function Φ_A in the TAR definitions maps a set of dimension values V_{di} for every dimension $d_i \in Dim_A$ to a set of $x_i \in \mathbb{Z}$:

$$\Phi_A : (V_{d1} \times V_{d2} \times ... \times V_{dn}) \to \mathbb{Z}^n \tag{1}$$

A location L_A is a position in the A TAR defined as a set of integers coordinate values obtained through the application of Φ_A in a set of dimension values. An array location is nothing more than a multidimensional address formed by a set integer indexes that identifies a tuple or cell within the TAR. The definition of Φ_A is trivial when $d_{i_{D_i}} = \mathbb{Z} \ \forall \ d_i \in Dim_A$. Every value held in a TAR A can be specified by a location in A and the specification of the data element (either a dimension or attribute). The location space C_A is a set formed by all possible locations in an array given by the image of the function Φ_A.

A subTAR $s \in S_A$ holds the functions to translate a location in a TAR into an address in a linear storage scheme, and then to translate the linear address into a data value valid for an extent of the TAR. Different TAR locations may be encompassed by different subTARs, in which case the functions to carry out this translation will be different. The extent of a subTAR is the region of the TAR for which its translation functions are valid. The definition of which subTAR is responsible for which TAR locations is given by the function R_A.

$$R_A : C_A \to S_A \tag{2}$$

R_A for a TAR A maps every single TAR location to a subTAR in S_A that encompasses it.

Moreover, the following relation holds true:

$$\forall s \in S_A \ \forall \ l_i \in \eta_s (R_A(l_i) = s) \tag{3}$$

Every location in an subTAR s extent is a location that maps to s itself when applied to the subTARs location function R_A. As a consequence of this functional definition, we have the impossibility of a intersection of two subTARs extents for the same TAR. Any location in a TAR is associated to at most one subTAR.

SubTARs have an associated position mapping function:

$$Fi_s : \mathbb{Z}^n \rightarrow \mathbb{Z} \tag{4}$$

The function Fi_s for a subTAR s maps a n-dimensional TAR location to a single integer representing an index or offset for a data value into the linear storage scheme. This linear address is then used to access the actual data values in the TAR. The subTAR data mapping function is responsible for this translation:

$$Fd_s : \mathbb{Z} \times D_A \rightarrow D \tag{5}$$

Where D is the domain of a data element $e \in D_A$. The data mapping function maps a linear address along with a TAR data element to an atomic value in the domain of the respective data element given as the input for the function. Users may need to define data elements related to an entire TAR, or, more precisely, a data element that do not depend functionally to any dimension. This special type of data element is called a TAR property and Fd_s becomes a constant.

A TARS Γ also has the sets of roles R_Γ and types T_Γ. A role is a string value that represents a special purpose data element with an important meaning in the application context. The set R_Γ contains all defined roles within a TARS. Roles are part of a type. A type T is a triple (N_T, M, O) where N_T is a string containing the type's name, M is a set of mandatory roles, and O is a set of optional roles. Types allow users to give special meaning to every data element in a TAR. Special purpose operators in a TARS may take the TAR type into consideration. Some operators may only make sense for a TAR of a given type, since they depend on the existence of special purpose dimensions and attributes with well defined meaning in the application domain.

A TAR $A \in \Omega_\Gamma$ has an injective role mapping function defined as:

$$\Upsilon_A : D_A \rightarrow R_\Gamma \tag{6}$$

The function Υ_A maps a data element to a role in the TARS, which indicates the role within the application context that the given data element fulfills. The notation $Type(A)$ refers to the type of the TAR A. A TAR A is said to be of type T if all data elements in D_A are mapped to a role in T_M or T_O, meaning that all roles are defined in the same type. There must exist one element in D_A that fulfills every mandatory role in T_M. Thus, we say:

$$\begin{aligned} Type(A) = T \rightarrow \quad & \forall\ d_e \in D_A \quad (\Upsilon_A(d_e) \in T_M) \vee (\Upsilon_A(d_e) \in T_O) \\ \wedge \quad & \forall\ r \in T_M \ \ \exists\ d_e \in D_A \ \ \textit{such that} \ \ \Upsilon_A(d_e) = r \end{aligned} \tag{7}$$

Two different data elements in D_A cannot be mapped to the same role. This is guaranteed because Υ_A is an injective function.

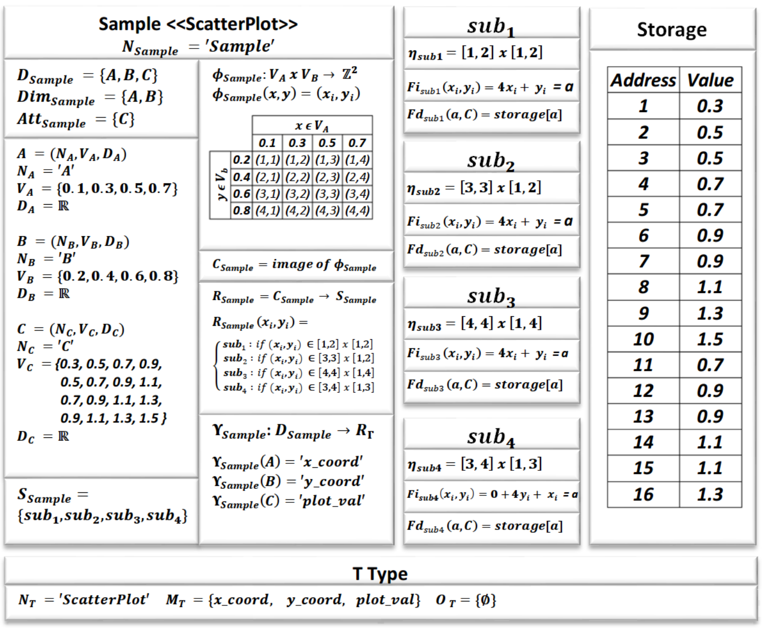
Relationships or links indicate that different data elements in the array schema correspond to the same entity. In a TARS, the L_Γ is a set of relationships present in the schema. A relationship R between TAR A and TAR B is a pair (d_a, d_b) where $d_a \in A_{D_A}$ is a data element of A and $d_b \in B_{D_B}$ is data element of B. Relationships are constraints that limit the domain of values that are valid in a data element given the current set of values held in another data element. This constraint can be expressed as:

$$d_{aV_e} \subseteq d_{bV_e} \tag{8}$$

The set of every value held in the data element d_a of A is a subset of the data values held in the data element d_b of B.

Figure 2 depicts an example for TARS, illustrating all the definitions given so far. In the example, we have a TARS with a single TAR named *Sample* of the type *ScatterPlot* plot. It represents a bi-dimensional scatter plot dataset with two real dimensions (A and B) that fulfill the roles $x\_coordinate$ and $y\_coordinate$ respectively. The real attribute C fulfills the role $plot_vat$.

The TAR *Sample* has 4 subTARs ($sub_1, sub_2, sub_3, sub_4$), the TAR regions they encompass being given by R_{Sample}. Every subTAR has its own mapping functions. The position mapping function is either a row major or column major conversion to a linear address space. The data mapping function is implemented as a simple access to a position (specified between squared brackets) in a linear storage working as a large linear unidimensional array for data.


Sample <<ScatterPlot>>
N_{Sample} = 'Sample'
$D_{Sample} = \{A,B,C\}$
$Dim_{Sample} = \{A,B\}$
$Att_{Sample} = \{C\}$
$A = (N_A, V_A, D_A)$
N_A = 'A'
$V_A = \{0.1, 0.3, 0.5, 0.7\}$
$D_A = \mathbb{R}$
$B = (N_B, V_B, D_B)$
N_B = 'B'
$V_B = \{0.2, 0.4, 0.6, 0.8\}$
$D_B = \mathbb{R}$
$C = (N_C, V_C, D_C)$
N_C = 'C'
$V_C = \{0.3, 0.5, 0.7, 0.9, 0.5, 0.7, 0.9, 1.1, 0.7, 0.9, 1.1, 1.3, 0.9, 1.1, 1.3, 1.5\}$
$D_C = \mathbb{R}$
$S_{Sample} = \{sub_1, sub_2, sub_3, sub_4\}$
$\phi_{Sample}: V_A \times V_B \to \mathbb{Z}^2$
$\phi_{Sample}(x,y) = (x_i, y_i)$
$x \in V_A$
$y \in V_b$
0.1 0.3 0.5 0.7
0.2 (1,1) (1,2) (1,3) (1,4)
0.4 (2,1) (2,2) (2,3) (2,4)
0.6 (3,1) (3,2) (3,3) (3,4)
0.8 (4,1) (4,2) (4,3) (4,4)
C_{Sample} = image of ϕ_{Sample}
$R_{Sample} = C_{Sample} \to S_{Sample}$
$R_{Sample}(x_i, y_i) =$
sub_1 : if $(x_i,y_i) \in [1,2] \times [1,2]$
sub_2 : if $(x_i,y_i) \in [3,3] \times [1,2]$
sub_3 : if $(x_i,y_i) \in [4,4] \times [1,4]$
sub_4 : if $(x_i,y_i) \in [3,4] \times [1,3]$
$\Upsilon_{Sample}: D_{Sample} \to R_\Gamma$
$\Upsilon_{Sample}(A)$ = 'x_coord'
$\Upsilon_{Sample}(B)$ = 'y_coord'
$\Upsilon_{Sample}(C)$ = 'plot_val'
sub_1
$\eta_{sub1} = [1,2] \times [1,2]$
$Fi_{sub1}(x_i,y_i) = 4x_i + y_i = a$
$Fd_{sub1}(a,C) = storage[a]$
sub_2
$\eta_{sub2} = [3,3] \times [1,2]$
$Fi_{sub2}(x_i,y_i) = 4x_i + y_i = a$
$Fd_{sub2}(a,C) = storage[a]$
sub_3
$\eta_{sub3} = [4,4] \times [1,4]$
$Fi_{sub3}(x_i,y_i) = 4x_i + y_i = a$
$Fd_{sub3}(a,C) = storage[a]$
sub_4
$\eta_{sub4} = [3,4] \times [1,3]$
$Fi_{sub4}(x_i,y_i) = 0 + 4y_i + x_i = a$
$Fd_{sub4}(a,C) = storage[a]$
Storage
T Type
N_T = 'ScatterPlot' $M_T = \{x\_coord, y\_coord, plot\_val\}$ $O_T = \{\emptyset\}$

Address	Value
1	0.3
2	0.5
3	0.5
4	0.7
5	0.7
6	0.9
7	0.9
8	1.1
9	1.3
10	1.5
11	0.7
12	0.9
13	0.9
14	1.1
15	1.1
16	1.3

Fig. 2. TARS with a single Type and a single TAR composed of 4 subTARs.

Finally, a TARS has a set Θ_Γ of TAR operators that forms a TARS algebra. This algebra contains functions that allow users to create new TARs derived from the ones already defined. By combining the algebraic operators, users can express the most varied queries and analyses over data held in a one or more TARs. These operators can be type dependent, meaning that they work only with TARs of a given type. As stated before, a TAR of type T is guaranteed to have all mandatory roles of T, therefore an operators that relies on a TAR A with type T assumes the values in A to be compliant with type T.

An operator Op is defined as a triple (N_O, T_O, P_O) where N_O is a string containing the operator name, T_O is a list of types $< t_1, t_2, ...t_n > \in T_\Gamma$ defining the expected types for input TARS, and P_O is a list $< p_1, p_2, ..., p_m >$ of atomic values forming the operator parameters set identifiers. New derived TARs can be expressed by using operators in a database query. A TAR T_r resulting from the application of an operator O_1 on another TAR T_o can be input into another operator O_2. Therefore, complex derived TARs can be create as the result of nested operator calls. For instance, a derived TAR A can be produced by the query (where base_tar is a TAR, and v_a, v_b, v_c are parameters):

$$A = O_1(O_2(O_3(base\_tar, v_c), v_b), v_a) \tag{9}$$

This definition is the base for a functional query language, in which the functions are operations defined in TARS for a set of given types.

5 Scientific Application Use Case

In this section, we validate our approach, showing how a TARS schema is used in the context of a numerical simulation application. First, we introduce the characteristics of a simulation datasets. Then, we show how to support these data using TARS.

5.1 Simulation Datasets Overview

Numerical simulation is the process of designing a computational model of a system to understand and predict its behavior [4]. Simulations are particularly useful in situations where it is hard or even impossible to execute real tests to acquire data. They depend on the creation of mathematical models describing the relation between physical quantities. A mathematical model captures the behavior of a phenomenon, with equations, usually solved by numerical methods. Some methods require the discretization of the domain in a form of a grid or mesh. Depending on the domain, modelers can adopt either a structured or unstructured meshes, divided into cells or elements. Meshes have topological and geometrical representations. Geometrical aspects are related to shapes, sizes and absolute positions of their elements, such as points. Topology representation captures the relations between elements, like their neighborhoods or adjacency, without considering their position in time and space.

5.2 TARS Implementation

A TARS for a simulation dataset has a set of type roles that define special attribute types, such as *id*, depicted in Figure 3. Simulation data comprises a series of fields of physical quantities defined for positions in time and space. Different values exist for every point or mesh element (lines, polygons, solid, etc...). Therefore, we define the Field type to represent the core of numerical simulation data. The Field type has some mandatory roles such as the *id*, i.e. a data element that associates a physical quantity with a mesh element at which it has been computed. There is a set of roles for *field_values* to represent the actual physical quantities. The *element_type* is a string indicating if the element is a point, a line, a triangle, a tetrahedron, etc. The *geometry_tar* string property contains the name of another TAR in the TARS defining the mesh geometry. Optionally, a TAR could have a dimension or attribute to specify different time steps (for transient problems) and trials, for cases when data for various simulation runs are stored in the same structure.

The mesh modelling for the discretization of the domain is represented by its geometry and topology. We have a series of types for capturing variations in representation for these two aspects. We have Cartesian geometries, with a mandatory role *id* for identifying points, and other roles for x, y and z coordinates. There is also the same optional roles *time step* and *trial*. A mesh geometry can change or evolve during the same trial, in a scenario in which the application domain is deformable, or a different mesh can be used for every simulation run. In Figure 3 we show the definition of other types of geometries, with spherical or cylindrical coordinate systems, each one with their specific roles.

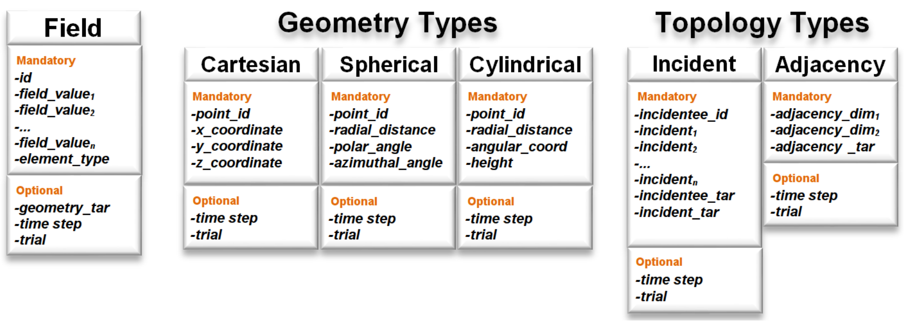


Fig. 3. Types for a Numerical Simulation TARS

Data related to the mesh topology requires special types, such as the Incident Topology and Adjacency Topology. The Incident Topology type captures the semantics of topologies specified as a series of incident relationships. In a mesh, we have lower order elements that are incident in higher order elements. For instance, a point is a zero-order element and a line is a first-order element. Two points are incident in a line, and this information are captured in an incidence matrix where each row represents a line and every column contains an id of a point that is incident in the line. We can have a different layout with points being incident in even higher order elements, like triangles and tetrahedra. Therefore, an incident Topology type captures the relationships between a series of incident elements into an incidentee element with their respective roles. There are two special roles for properties $incidentee\_field\_tar$ and $incident\_field\_tar$ that point to field TARS containing data associated with both the incidentee and the incident elements. Since the mesh can be different in every time step for the same run, or for different runs of the same model, the optional roles *time step* and *trial* are also present.

Alternatively, a topology can be specified through an adjacency or neighborhood relationship between mesh elements of the same order. We can use an adjacency matrix where every line and column represents a mesh element, for instance, a point. The existence of a connection between two points is given by a value in the array cell. In a TAR type, we have roles to identify both adjacency matrix dimensions along with a role for pointing to the field data TAR containing values for the points $adjacency\_field\_tar$. We keep roles for *time step* and *trial* as well for this representation.

A TARS implementation in this context contains a TAR definition for each type (Field, Geometry and Topology). Some relationships between TARs are expected to exist also. For instance, if the field TAR contains data for points, and there is a Geometry TAR holding coordinate values, the data element fulfilling the role of *id* in the file TAR must have a relationship with the data element fulfilling the *pointid* role in the geometry TAR, since logically they represent the same entity. Another example occurs between a Topology TARs and Field TARs, incidentee and incident data elements in Topology contains identifiers for mesh elements in Field TARs, and thus a relationship must exists between them and the respective data elements fulfilling the id roles in the Field TARs.

Along with the types, we added a series of special purpose array operators in the TARS. Many operations are common array operations that can be executed upon any TAR. They are: $filter$, *projection*, *join*, *apply*, *union*, *slice*, *group* and *window_group* as defined in [10]. Beyond that, we also might think of an extended set of operations, taking the semantics given by the types and roles into consideration. For instance, we can define a set of operators for the following tasks: spatial range query, comparison between trials, uncertainty quantification, finding closest elements to a point or that bound another elements. These tasks are implemented by operators depending on Geometry and Field TARs.

Additionally, some tasks require Topology TARs, in combination with field typed TARs, and give origin to operators depending on these types. For instance: conversion from incidence to adjacency topologies, definition of the amount of neighbors a mesh element has, obtaining a set of incident elements for an incidentee element or vice-versa, getting a path between points in a mesh, following a path from an initial mesh element following other mesh elements whose data values satisfy a predicate, calculate aggregate values (average, min, max, etc) of mesh elements considering their neighborhoods or adjacency.

Finally, an important family of special operators are responsible for creating data visualizations. These operators rely on the semantics given by the types and roles to create complete visualizations. They take into consideration not only field data, but also its geometry and topology. Instead of outputting a TAR, visualization operators implemented in a TARS DBMS are placed at the top of a query plan, and produce a graphical representation for the output of an analysis.

5.3 Concluding Remarks

The TAR implementation we proposed above fulfills the requirements of simulation applications. First, it enables the representation of the main simulation data structures, including data fields, mesh geometry and topology. Thus, applications can query any of these structures individually or in composition, with specific algebraic operators. Second, it provides support for online management of simulation data, as the outcome of the simulation does not need to go through costly data transformation to be used as a TAR by a analytical application.

Third, the model provides support for extended application semantics. UQ and data visualization applications can be supported by operators that take advantage of the simulation data structures provided by the model offering analysis and queries beyond the basic array operations. Furthermore, by integrating application operations to the system as algebraic operators, a costly file transformation from DBMS output to visualization file format is saved, leading to faster data to visualization procedure.

6 Conclusion

Important scientific applications, such as numerical simulation, have requirements that are not supported by array DBMSs, in particular, to deal with mesh topology and geometry. In this paper, we propose a novel data model called TARS (Typed ARray Schema), which extends the basic array data model with typed arrays (TARs). In TARS, the support of application dependent data characteristics, such as data visualization and UQ computation, is provided through the definition of TAR objects, ready to be manipulated by TAR operators. This approach provides much flexibility for capturing internal data layouts through mapping functions, which makes data ingestion independent of how simulation data has been produced, thus minimizing ingestion time. A TAR has a type that defines the semantics of its dimensions and attributes, and eases the creation of complex algebraic operators that depend on the application semantics.

TARS can be used as the underlying data model for a simulation data management system, providing a powerful query language with array operators to allow users to express a variety of analytical queries and determine how results are to be visualized.

To validate our approach, we showed how a TARS schema can be used in the context of a numerical simulation application. First, it enables the representation of the main simulation data structures, including data fields, mesh geometry and topology, and their querying with specific algebraic operators. Second, it provides support for online management of simulation data, thus reducing the time required to deal with data layout transformations. Third, the model provides support for extended application semantics.

Acknowledgments This work has been funded by CNPq, CAPES, FAPERJ, Inria (SciDISC project) and the European Commission (HPC4E H2020 project) and performed (for P. Valduriez) in the context of the Computational Biology Institute (www.ibc-montpellier.fr) and for (F. Porto, H. Lustosa and N. Lemus) in the context of the DEXL Laboratory (dexl.lncc.br) at LNCC.

References

1. Ahrens, J.: Increasing scientific data insights about exascale class simulations under power and storage constraints. IEEE Computer Graphics and Applications 35(2), 8–11 (Mar 2015)
2. Alagiannis, I., Borovica, R., Branco, M., Idreos, S., Ailamaki, A.: Nodb: Efficient query execution on raw data files. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. pp. 241–252. SIGMOD '12, ACM, New York, NY, USA (2012)
3. Blanas, S., Wu, K., Byna, S., Dong, B., Shoshani, A.: Parallel data analysis directly on scientific file formats. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. pp. 385–396. SIGMOD '14, ACM, New York, NY, USA (2014), http://doi.acm.org/10.1145/2588555.2612185
4. Dym, C.: Principles of Mathematical Modeling. Elsevier Science (2004)
5. Gosink, L., Shalf, J., Stockinger, K., Wu, K., Bethel, W.: Hdf5-fastquery: Accelerating complex queries on hdf datasets using fast bitmap indices. pp. 149–158. SSDBM '06, IEEE Computer Society, Washington, DC, USA (2006), http://dx.doi.org/10.1109/SSDBM.2006.27
6. Group, T.H.: Hdf5 - the hdf group (2017), https://www.hdfgroup.org/HDF5, [Online; accessed 19-Mar-2017]
7. Howe, B.: Gridfields: Model-driven Data Transformation in the Physical Sciences. Ph.D. thesis, Portland, OR, USA (2007), aAI3255425
8. Lee, B.S., Chen, L., yeol Song, I.: I.l.: Modeling and querying scientific simulation mesh data. Tech. rep., International Electrotechnical Commision (1999)
9. Lustosa, H., Porto, F., Valduriez, P., Blanco, P.: Database system support of simulation data. Proc. VLDB Endow. 9(13), 1329–1340 (Sep 2016)
10. Marathe, A.P., Salem, K.: Query processing techniques for arrays. In: ACM SIGMOD Record. vol. 28, pp. 323–334. ACM (1999)
11. Paradigm4: Scidb (2017), http://www.paradigm4.com/, [Online; accessed 19-Mar-2017]
12. Rasdaman: Rasdaman - raster data manager (2017), http://www.rasdaman.org/, [Online; accessed 19-Mar-2017]
13. Rezaei Mahdiraji, A., Baumann, P., Berti, G.: Img-complex: graph data model for topology of unstructured meshes. In: Proceedings of the 22nd ACM international conference on Conference on information &#38; knowledge management. pp. 1619–1624. CIKM '13 (2013)
14. Stonebraker, M., Becla, J., Dewitt, D., Lim, K.T., Maier, D., Ratzesberger, O., Zdonik, S.: Requirements for science data bases and scidb. In: Conference on Innovative Data Systems Research (CIDR). Asilomar, USA (january 2009)
15. Unidata: netcdf (2017), https://www.unidata.ucar.edu/software/netcdf/, [Online; accessed 19-Mar-2017]