A Linguistic Framework for Querying Dimensional Data

Elisa Bertino[1] Tsz S. Cheng[2] Shashi K. Gadia[3] Giovanna Guerrini[4]

[1]DSI - Università di Milano
Milano - Italy
bertino@dsi.unimi.it

[2]IBM Global E-Business
Solution Center, Dallas - Texas
scheng@us.ibm.com

[3]Department of Computer Science
Iowa State University - Iowa
gadia@cs.iastate.edu

[4]DISI - Università di Genova
Genova - Italy
guerrini@disi.unige.it

Abstract

This paper deals with dimensional data. Examples of dimensions are space and time. Thus, temporal, spatial, spatiotemporal values are examples of dimensional data. We define the notion of dimensional object, extending an object-oriented ODMG-like type system to include dimensional types. We then address the problem of querying dimensional objects. Linguistic constructs are introduced that allow objects with different dimensions to be mixed in the same phrases. This allows the user to formulate both associative and navigational accesses seamlessly without having to worry about the dimensions of the various data elements involved.

1 Introduction

The notion of *parametric* or *dimensional* data has been proposed in [12] to model in a uniform way ordinary, temporal, spatial, spatiotemporal data. That notion has also been applied to multilevel security [10] and to model multiple beliefs. However, a formal treatment of dimensional data has not been given. The basic notion beyond dimensional data is that of *dimensional element*, that is, a finite union of points in the dimensional space; attribute values are functions with dimensional elements as domains. The parametric data model is based on dimensional relations, such that all values in the attributes of a tuple have the same dimensional domain and a key is declared for each relation, in such a way that no key attribute value of a tuple can change from one point in the dimensional space to another. An algebra for the parametric model has been defined, with the main goal of minimizing the "user complexity" of queries. A uniform handling of dimensional data is achieved through dimension alignment, that automatically allows lower dimensional data and queries to be used in higher dimensional contexts.

Object database systems (both the pure object-oriented systems and the object-relational ones) are currently replacing conventional relational database systems. The notion of dimensional data is orthogonal to the specific data model considered. In this paper, we present a rigorous treatment of dimensional objects, that is suitable for object-oriented databases. This formal treatment entails addressing several interesting issues. First, parametric non-1NF data need to be considered. In the object model attribute values can be of any type of the (complex) type system, including, for instance, set and tuple constructors. A dimensional object thus requires the introduction, in the type system, of the notion of dimensional type, in order to type in a uniform way dimensional and non-dimensional data. We consider a simple object model, that is a simplified version of the ODMG standard data model [3].

We then investigate how dimensional objects can be used together in queries. We do not develop a full query language, rather we focus on how objects with different dimensions can be mixed together in the same queries. Specifically, we consider both associative queries (based on classical comparison operators) and navigational queries (based on the notion of *path expression*). The basic goal of the constructs we define is to allow a user to query dimensional objects without having to worry about the dimensions of the various data elements involved in the query. The constructs we introduce in this paper will form the basis for query languages for dimensional object collections.

Note that, though several different temporal object-oriented data models ([13, 14]) and query languages (e.g.

[7, 8, 15]) have been proposed, the emphasis of this paper is on a uniform handling of several dimensions, and on dimension alignment at the data and language levels.

The remainder of the paper is structured as follows. Section 2 introduces the notion of dimensional object, whereas Section 3 is devoted to queries on dimensional objects. Section 4 concludes the work and discusses some issues that are currently under investigation.

2 Dimensional Object Model

In this section we briefly introduce the dimensional type system which constitutes the basis of the dimensional object model.

2.1 Dimensional Elements

We assume a set $\mathcal{S} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ of dimensions and an underlying universal dimensional domain $\mathcal{P}$ as the cartesian product $\mathcal{D}_1 \times \mathcal{D}_2 \times \ldots \times \mathcal{D}_n$ of these dimensions. The user can view it as a set of points in the dimensional space. Let $\mathcal{DN}$ denote the set of names of dimensions in $\mathcal{S}$.

We postulate that certain subsets of $\mathcal{P}$, called *dimensional elements*, are of interest to users, and they are closed under union ($\cup$), intersection ($\cap$), difference ($\setminus$), and complementation (^{c}). Dimensional values will be modeled as functions from dimensional elements, such that if a function maps a dimensional element μ to a given value v this means that it assigns v to every dimensional point in μ. We are indeed interested in modeling functions with sets of dimensional points as domains. The general principles for identifying legitimate domains for dimensional values are: (i) a real world object should correspond to a single object in the database; (ii) whatever domains users consider important, should be admitted as dimensional element; (iii) the domains should allow easy and flexible queries. Applying these principles to temporal databases means: (i) an interval is not enough, but a finite union of intervals is; (ii) users like to compute the state of an object at a point, thus a point should be a temporal element; (iii) queries will require dimensional elements to be closed under set-theoretic operations (since conjunction corresponds to intersection, disjunction to union, negation to complementation). Therefore, temporal elements are finite unions of intervals. Note that, by this definition, an instant is also a temporal element. Thus, if the dimensional domain is the set of time instants $TIME = [0, now] = \{0, 1, 2, \ldots, now\}$, then the dimensional elements are *temporal elements* [9]. For instance, $[\mathtt{11}, \mathtt{40}] \cup [\mathtt{57}, \mathtt{now}]$ is a temporal element. In spatial databases users will like to use certain elementary spatial domains. Then dimensional spatial elements will be finite unions of those domains. Thus, if the dimensional domain is the spatial domain $SPACE$, then the dimensional elements are *spatial elements* [5]. In a two-dimensional space, a dimensional element will be a union of cross-products of one-dimensional elements. Specifically, if the dimensional domain is $TIME \times SPACE$ we get *spatiotemporal elements*. An example of spatiotemporal element is $([\mathtt{11}, \mathtt{40}] \times \mathtt{reg}_1) \cup ([\mathtt{57}, \mathtt{62}] \times \mathtt{reg}_2) \cup ([\mathtt{75}, \mathtt{now}] \times \mathtt{reg}_2)$.

Let $\mathcal{DEL}$ denote the set of dimensional elements of the dimensional space $\mathcal{P}$. Moreover, given a set $\mathbf{D} = \{D_1, \ldots, D_k\}$, $k \leq n$, of dimension names, denoting a subset $\{\mathcal{D}_{i_1}, \ldots, \mathcal{D}_{i_k}\}$ of $\mathcal{S}$, let $\mathcal{DEL}_{D_1,\ldots,D_k}$ denote the set of dimensional elements with dimensions $\mathcal{D}_{i_1}, \ldots, \mathcal{D}_{i_k}$. Given a dimensional element $\mu \in \mathcal{DEL}$, and a set $\mathbf{D}$ of dimension names, let $\Pi_{\mathbf{D}}\mu$ denote the projection of μ on dimensions in $\mathbf{D}$. As a particular case, given a dimension name D, $\Pi_D\mu$ denotes the component of μ corresponding to dimension D.

Note that, whenever a dimension is missing in a dimensional element, the whole corresponding dimensional domain is assumed. In our model, indeed, each value is dimensional; if a value has no explicitly specified element for a given dimension it refers to every point of that dimension. Thus, given a dimensional element $\mu \in \mathcal{DEL}_{D_1,\ldots,D_k}$, and a set $\mathbf{D}' \subseteq \mathcal{DN}$, its completion to $\mathbf{D}'$ is a dimensional element $\bar{\mu}^{\mathbf{D}'} \in \mathcal{DEL}_{\mathbf{D}'}$, such that, $\forall D'_j \in \mathbf{D}'$:

$$\Pi_{D'_j}\bar{\mu} = \begin{cases} \Pi_{D'_j}\mu & \text{if } D'_j \in \{D_1, \ldots, D_k\} \\ \mathcal{D}_{i_j} & \text{otherwise} \end{cases}$$

As a particular case, the completion to the whole set of dimensions, namely, $\bar{\mu}^{\mathcal{DN}} \in \mathcal{DEL}$ is simply denoted as $\bar{\mu}$. For instance, if $\{TIME, SPACE\} \subseteq \mathcal{S}$, $\mathbf{D} = \{\mathtt{TIME}, \mathtt{SPACE}\}$,[1] $\mu \in \mathcal{DEL}_{\mathtt{SPACE}} = \mathtt{reg}_1 \cup \mathtt{reg}_2$, then $\bar{\mu}^{\mathbf{D}} = (\mathtt{reg}_1 \times [\mathtt{0}, \mathtt{now}]) \cup (\mathtt{reg}_2 \times [\mathtt{0}, \mathtt{now}])$.

2.2 Types

The types of the dimensional object model are obtained as an extension of the usual set of types of an object model. We postulate the existence of a set of predefined basic literal types $\mathcal{BLT}$ (containing the types $integer, real, boolean, character$ and $string$). Moreover, class names are also basic types, referred to as object types. We consider a set $\mathcal{CI}$ of class names. Finally, we consider *structured types* obtained by applying the set and record constructors to existing types. The following definition introduces non-dimensional types.

Definition 1 *(Non-dimensional Types).* The set of non-dimensional literal types $\mathcal{NLT}$ is recursively defined as follows:

[1] We denote the name of dimensions in typewriter font.

- the predefined basic literal types are literal types ($\mathcal{BLT} \subseteq \mathcal{NLT}$);
- if T is a literal type or an object type then $set\langle T\rangle$ is a non-dimensional literal type;
- if $a_1, ..., a_n \in \mathcal{AN}$ are distinct labels and $T_1, ...T_n$ are literal or object types, then $struct\langle T_1\ a_1, ..., T_n\ a_n\rangle$ is a non-dimensional literal type.

The set of non-dimensional types $\mathcal{NT}$ is defined as the union of literal types $\mathcal{NLT}$ and object types $\mathcal{OT} = \mathcal{CI}$.

This set of types is extended with a collection of *dimensional types*. Dimensional types are introduced to type in a uniform way dimensional (e.g., temporal) variables, and non-dimensional variables, that is, variables with which no dimension is associated.

Definition 2 *(Dimensional Types).* Let $T \in \mathcal{NT}$ be a non-dimensional type, $D_1, \ldots, D_k \in \mathcal{DN}$ be dimension names, then $T[D_1, \ldots, D_k]$ is a dimensional type.

Example 1 *If we consider $TIME$ and $SPACE$ as dimensions, and* `integer` *and* `Person` *as non-dimensional types,*
$\mathtt{integer[TIME]}, \mathtt{Person[SPACE]}$,
$\mathtt{set\langle Person\rangle[TIME, SPACE]}$
are examples of dimensional types.

Intuitively, instances of type $T[D]$ are partial functions from dimensional elements of dimension D to instances of type T, as discussed in the following section. Note that dimensional types cannot be nested; note moreover that the set of legal values for $T[D_1][D_2]$ would correspond to the set of legal values of $T[D_1, D_2]$.

Let $\mathcal{DT}$ be the set of dimensional types as defined by Definition 2. In our model, dimensional types can be used in the definition of set and struct literal types, as stated by the following definition.

Definition 3 *(Literal Types).* The set of literal types $\mathcal{LT}$ is recursively defined as follows:

- non-dimensional literal types are literal types ($\mathcal{NLT} \subseteq \mathcal{LT}$);
- if T is a literal type, or an object type, or a dimensional type then $set\langle T\rangle$ is a literal type;
- if $a_1, ..., a_n \in \mathcal{AN}$ are distinct labels and $T_1, ...T_n$ are literal, object, or dimensional types, then $struct\langle T_1\ a_1, ..., T_n\ a_n\rangle$ is a literal type.

The set of types $\mathcal{T}$ of our dimensional model is defined as the union of literal types $\mathcal{LT}$, object types $\mathcal{OT}$, and dimensional types $\mathcal{DT}$.

2.3 Values

We now introduce the set of legal values of our model. Oids in $\mathcal{OI}$ are handled as values. Thus, an object identifier i is a value of an object type in $\mathcal{OT}$. The set of objects instances of a class depends on the point of the dimensional space we refer to. Thus, to define the extension, that is, the set of legal values for each type, we introduce a function π: $\mathcal{CI} \times \mathcal{DEL} \rightarrow 2^{\mathcal{OI}}$, assigning an extent to each class, for each dimensional element μ. For each $c \in \mathcal{CI}$, for each $\mu \in \mathcal{DEL}$, $\pi(c, \mu)$ is the set of the identifiers of objects that, at the dimensional points in μ, belong to class c. By contrast, the set of instances of a literal type does not vary over dimensions (e.g, the set of instances of `integer` is always $\mathbf{Z}$, the set of instances of `boolean` is always $\{true, false\}$). Given a literal type T, let $ext(T)$ denote this invariant set of instances.

Definition 4 (*Non-dimensional Type Legal Values*). Let $T \in \mathcal{LT} \cup \mathcal{OT}$ be a non-dimensional type and $\mu \in \mathcal{DEL}$ be a dimensional element, then $\{\!\!\{ T \}\!\!\}_\mu$ denotes the extension (i.e., the set of legal values) of type T at μ:

$$\{\!\!\{ T \}\!\!\}_\mu = \begin{cases} ext(T) & \text{if } T \in \mathcal{BLT} \\ \pi(T, \mu) & \text{if } T \in \mathcal{OT} \\ 2^{\{\!\!\{ T' \}\!\!\}_\mu} & \text{if } T = set\langle T'\rangle \\ \{(a_1 : v_1, \ldots, a_n : v_n) \mid & \text{if } T = struct \\ v_i \in \{\!\!\{ T_i \}\!\!\}_\mu, 1 \leq i \leq n\} & \langle T_1\ a_1, \ldots, T_n\ a_n\rangle \end{cases}$$

The set of instances of a dimensional type $T[D]$, formally specified by the following definition, is the set of partial functions from dimensional elements of D to instances of type T. Note that this set does not vary over the dimensional space, that is, for a dimensional type $T[D_1, \ldots, D_k]$, $\{\!\!\{ T[D_1, \ldots, D_k] \}\!\!\}_\mu$ is the same for all $\mu \in \mathcal{DEL}_{D_1,\ldots,D_k}$, thus it will be denoted simply as $\{\!\!\{ T[D_1, \ldots, D_k] \}\!\!\}$.

Definition 5 (*Dimensional Type Legal Values*). Let $T[D_1, \ldots, D_k] \in \mathcal{DT}$ be a dimensional type, then $\{\!\!\{ T[D_1, \ldots, D_k] \}\!\!\}$ denotes the set of legal values of type $T[D_1, \ldots, D_k]$:

$$\begin{aligned} &\{\!\!\{ T\ [D_1, \ldots, D_k] \}\!\!\} = \\ &\quad \{f \mid f : \mathcal{DEL}_{D_1,\ldots,D_k} \rightarrow \textstyle\bigcup_{\mu \in \mathcal{DEL}} \{\!\!\{ T \}\!\!\}_\mu \\ &\quad \text{such that, for each } \mu \in \mathcal{DEL}_{D_1,\ldots,D_k}, \\ &\quad \text{if } f(\mu) \text{ is defined then } f(\mu) \in \{\!\!\{ T \}\!\!\}_{\bar{\mu}}\} \end{aligned}$$

Note that we will denote a function in this set as a set of pairs $\{\langle\mu_1, v_1\rangle, \ldots, \langle\mu_n, v_n\rangle\}$, where $v_1, \ldots, v_n$ are legal values for type T, and $\mu_1, \ldots, \mu_n$ are dimensional elements in $\mathcal{DEL}_{D_1,\ldots,D_k}$ such that the function assumes value v_i for dimensional element μ_i, $i = 1, \ldots, n$.

Given a dimensional value v, operator $[\![v]\!]$ denotes its dimensional domain. Thus, the dimensional domain of the spatial value $\{\langle \mathtt{reg_1}, \mathtt{wheat}\rangle, \langle \mathtt{reg_2}, \mathtt{corn}\rangle\}$, denoted as $[\![\{\langle \mathtt{reg}_1, \mathtt{wheat}\rangle, \langle \mathtt{reg}_2, \mathtt{corn}\rangle\}]\!]$ is $\mathtt{reg}_1 \cup \mathtt{reg}_2$, whereas the domain of the temporal value $\{\langle [60,65], 20\mathrm{K}\rangle, \langle [66,80], 35\mathrm{K}\rangle\}$ is $[60,80]$. Moreover, given a dimensional value v, let $\delta(v) \subseteq \mathcal{DN}$ be the set of dimensions of value v. That is, if v is a value of type $T[D_1,\ldots,D_k]$, $\delta(v) = \{D_1,\ldots,D_k\}$. We also define the notion of restriction of a dimensional value to a dimensional element, formalized as follows.

Definition 6 (*Restriction of a Dimensional Value to a Dimensional Element*). Given a dimensional value v, of type $T[D_1,\ldots,D_k]$, and a dimensional element $\mu \in \mathcal{DEL}_{D'_1,\ldots,D'_h}$, such that $\{D'_1,\ldots,D'_h\} \subseteq \{D_1,\ldots,D_k\}$, the restriction of v to μ, denoted as $v_{|_\mu}$, is a dimensional value of type $T[D_1,\ldots,D_k]$ such that $[\![v_{|_\mu}]\!] = [\![v]\!] \cap \bar{\mu}^{D_1,\ldots,D_k}$, and $\forall \nu \in [\![v]\!] \cap \bar{\mu}^{D_1,\ldots,D_k} : v_{|_\mu}(\nu) = v(\nu)$.

For instance, given the spatiotemporal value $v = \{\langle (\mathtt{reg}_1 \times [45,57]) \cup (\mathtt{reg}_1 \times [80,\mathtt{now}]), \mathtt{wheat}\rangle, \langle \mathtt{reg}_2 \times [40,\mathtt{now}], \mathtt{corn}\rangle\}$, and the temporal element $\mu = [50,60] \cup [80,90]$

$$v_{|_\mu} = \{\ \langle (\mathtt{reg}_1 \times [50,57]) \cup (\mathtt{reg}_1 \times [80,90]),\ \mathtt{wheat}\rangle, \langle (\mathtt{reg}_2 \times [50,60]) \cup (\mathtt{reg}_2 \times [80,90]),\ \mathtt{corn}\rangle\}.$$

2.4 Classes and Objects

Here we will focus only on the signature of a class, that contains all the information for the use of the class and its instances, and, specifically, on the information about the attributes contained in the signature of a class. Each attribute is characterized by its name and its type. Consider the following example. Note that, because of the homogeneity assumption, all the attributes of a class must have the same dimensions, thus, it is most convenient to specify the dimension in the class declaration. Thus, only non-dimensional types are used in the definition of attribute domains, and a dimensional attribute a of type $T[D_1,\ldots,D_k]$ is specified by means of the pair (a,T), once the corresponding class has been declared as dimensional in $D_1,\ldots,D_k$.

Example 2 *In what follows we sketch the definition of some dimensional classes, inspired from [12]. Non-relevant attributes are omitted and substituted by dots. These classes refer to: persons (the information about persons are time dependent); lands (owned by persons, with a soil with a certain texture and a specific crop grown, these information are space dependent); monitors (referring to the concentration of chemicals in up-gradient and down-gradient wells, these information are both time and space dependent). Moreover, there are non-dimensional classes related to soil textures, crops, and chemicals (with the environmentally acceptable range of chemicals in the soil).*

```
class Person (dimension TIME) {
  attribute string name;
  attribute struct ⟨ string street,
                     string city⟩ address;
  attribute integer income;
  attribute set⟨Person⟩ children;
  attribute set⟨Land⟩ owns; }

class Land (dimension SPACE) {
  attribute Person owner;
  attribute integer acres;
  attribute Texture texture;
  attribute Crop crop;
  attribute string tillage; }

class Texture {
  attribute string description;
  ...; }

class Crop {
  attribute string name;
  ...; }

class Monitor (dimension TIME,SPACE) {
  attribute Chemical chemical;
  attribute short U/G_well;
  attribute short D/G_well; }

class Chemical {
  attribute string name;
  attribute integer max_level;
  attribute integer min_level; }
```

A dimensional object is an object whose value (state) is a dimensional value. Thus, we adopt a rather standard notion of object, formalized as follows.

Definition 7 *(Object).* An object o is a triple (i,v,c), where

$i \in \mathcal{OI}$ is the oid of o;

$v \in \mathcal{V}$ is a struct value of the form: $(a_1 : v_1,\ldots,a_n : v_n)$, where $a_1,\ldots,a_n \in \mathcal{AN}$ are the names of the attributes of o, and $v_1,\ldots,v_n \in \mathcal{V}$ are their values;

c is the most specific class to which o belongs

such that $[\![v_1]\!] = \ldots = [\![v_n]\!]$, that is, all the object attributes have the same dimensional domain[2] (*homogeneity assumption* [9]), and v is in accordance with the attribute specification given in class c.

The dimensional domain of an object o, denoted $[\![o]\!]$, is simply the domain of any of its attributes.

[2]Note that, however, the attributes can assume different values in different dimensional elements of that domain, thus our model is based on attribute timestamping rather than on object timestamping.

Example 3 *The following are examples of objects, referring to the classes of Example 2*[3].

$i = \mathtt{m}_1$
$v = (\,\mathtt{chemical}: \{\langle([0,\mathtt{now}]\times \mathtt{p}_1)\cup([0,\mathtt{now}]\times\mathtt{p}_2),\mathtt{c}_1\rangle\},$
$\mathtt{U/G\_well}: \{\langle[0,\mathtt{now}]\times\mathtt{p}_1,1.0\rangle,\langle[0,5]\times\mathtt{p}_2,1.5\rangle, \langle[6,\mathtt{now}]\times\mathtt{p}_2,3.5\rangle\}$
$\mathtt{D/G\_well}: \{\langle[0,\mathtt{now}]\times\mathtt{p}_1,0.9\rangle,\langle[0,10]\times\mathtt{p}_2,1.4\rangle, \langle[11,\mathtt{now}]\times\mathtt{p}_2,2.9\rangle\})$
$c = \mathtt{Monitor}$

$i = \mathtt{m}_2$
$v = (\,\mathtt{chemical}: \{\langle([0,\mathtt{now}]\times\mathtt{p}_1),\mathtt{c}_2\rangle\},$
$\mathtt{U/G\_well}: \{\langle[0,9]\times\mathtt{p}_1,10\rangle,\langle[10,\mathtt{now}]\times\mathtt{p}_1,12.2\rangle\}$
$\mathtt{D/G\_well}: \{\langle[0,\mathtt{now}]\times\mathtt{p}_1,9.2\rangle\})$
$c = \mathtt{Monitor}$

$i = \mathtt{c}_1$
$v = (\,\mathtt{name}: '\mathtt{atrazine}', \mathtt{max\_level}: 3.0, \mathtt{min\_level}: 0.05)$
$c = \mathtt{Chemical}$

$i = \mathtt{c}_2$
$v = (\,\mathtt{name}: '\mathtt{simazine}', \mathtt{max\_level}: 35.0\ \mathtt{min\_level}: 0.05)$
$c = \mathtt{Chemical}$

$i = \mathtt{p}_1$
$v = (\,\mathtt{name}: \{\langle[5,45],'\mathtt{john}'\rangle\},$
$\mathtt{address}: \{\langle[5,30],(\mathtt{street}:'\mathtt{12th}',\mathtt{city}:'\mathtt{ames}')\rangle, \langle[31,45],(\mathtt{street}:'\mathtt{4th}', \mathtt{city}:'\mathtt{iowacity}')\rangle\}$
$\mathtt{income}: \{\langle[5,20],1000\rangle,\langle[21,40],1500\rangle, \langle[41,45],2000\rangle\}$
$\mathtt{children}: \{\langle[5,45],\{\}\rangle\}$
$\mathtt{owns}: \{\langle[5,40],\{\}\rangle,\langle[41,45],\{\mathtt{l}_1\}\rangle\})$
$c = \mathtt{Person}$

$i = \mathtt{l}_1$
$v = (\,\mathtt{owner}: \{\langle\mathtt{reg}_1\cup\mathtt{reg}_2\cup\mathtt{reg}_3,\mathtt{p}_1\rangle\},$
$\mathtt{acres}: \{\langle\mathtt{reg}_1,40\rangle,\langle\mathtt{reg}_2,80\rangle,\langle\mathtt{reg}_3,30\rangle\}$
$\ldots)$
$c = \mathtt{Land}$

As in any object-oriented data model, classes are related by a user-defined ISA hierarchy, that induces a subtype hierarchy on types of our type system. For a subclass both substitutability (the subclass must have all the attributes of the superclass, and optionally some additional ones) and extent inclusion (the instances of the subclass are also instances of the superclass) are required. Due to space limitations, we do not discuss issues related to inheritance in this paper. Moreover, in this preliminary work, we do not consider attribute refinement, nor any subtype relationship holding among dimensional types with different dimensions.

[3]Only the attributes of object $\mathtt{l}_1$ that are used in the following are reported, for the sake of conciseness.

3 Querying Dimensional Objects

In this section we address the problem of querying dimensional objects. The query language we refer to combines features typical of associative query languages, features typical of navigational query languages proposed for object models, and features typical of dimensional data. The definition of a full query language is beyond the scope of this paper, in what follows we introduce and illustrate through examples the most interesting/peculiar aspects of dimensional data handling.

Before introducing the various kinds of expressions that can be used to assemble a query, let us first briefly discuss how dimensional values can be used together, combined, and compared in expressions. Classical operators extends to a dimensional context in a straightforward way: they simply operate pointwise. For instance, given two expressions e_1 and e_2 of type $\mathtt{integer}[\mathtt{TIME}]$ their sum $e_1 + e_2$ has type $\mathtt{integer}[\mathtt{TIME}]$ and the corresponding value is a function s such that, if v_1 and v_2 are the values denoted by e_1 and e_2, respectively, $\forall\mu \in \mathcal{DEL}_{\mathtt{TIME}} : s(\mu) = v_1(\mu) + v_2(\mu)$. Similarly, their comparison $e_1 > e_2$ has type $\mathtt{boolean}[\mathtt{TIME}]$ and the corresponding value is a function b such that $\forall\mu \in \mathcal{DEL}_{\mathtt{TIME}} : b(\mu) = v_1(\mu) > v_2(\mu)$.

Example 4 *Referring to the database schema of Example 2, given variables* $\mathtt{p}$, $\mathtt{q}$ *of type* $\mathtt{Person}$ *and variable* $\mathtt{m}$ *of type* $\mathtt{Monitor}$, $\mathtt{p.income} + \mathtt{q.income}$ *is an expression of type* $\mathtt{integer}[\mathtt{TIME}]$, *whereas* $\mathtt{m.U/G\_well} - \mathtt{m.D/G\_well} > 1$ *is an expression of type* $\mathtt{boolean}[\mathtt{TIME,SPACE}]$. *Referring to the objects of Example 3, this expression evaluated on* $\mathtt{m}_1$ *returns* $[6,10]\times\mathtt{p}_2$, *whereas evaluated on* $\mathtt{m}_2$ *it returns* $[10,\mathtt{now}]\times\mathtt{p}_1$.

The only further point to consider is what happens when values with different dimensions are used together. Following [12] we take the approach that, when a dimension is missing in a value, the whole dimension domain is intended. Thus, two expressions e_1 and e_2 of types $T_1[D'_1,\ldots,D'_k]$ and $T_2[D''_1,\ldots,D''_h]$, $0 \le h,k \le n$, can be combined through operator *op* if operator *op* can be applied to non-dimensional values of types T_1, T_2, producing a result of type T. The resulting expression will have type $T[D_1,\ldots,D_w]$ with $\{D_1,\ldots,D_w\} = \{D'_1,\ldots,D'_k\} \cup \{D''_1,\ldots,D''_h\}$. e_1 are e_2 are extended to dimensions $D_1,\ldots,D_w$ simply associating with them the whole dimension domain for missing dimensions, as already discussed.

Example 5 *Referring to the database schema of Example 2, given variable* $\mathtt{l}$ *of type* $\mathtt{Land}$ *and variable* $\mathtt{p}$ *of type* $\mathtt{Person}$, $\mathtt{l.acres} > \mathtt{p.income}$ *is an expression of type* $\mathtt{boolean}[\mathtt{SPACE,TIME}]$.

We do not detail the set of operators we consider to combine dimensional values; however, they surely include classical arithmetic, set, and comparison operators.

3.1 Dimensional Expressions

A dimensional expression is an expression that evaluates in a dimensional element. Dimensional expressions derive from temporal expressions proposed in [12] and are defined as follows.

Definition 8 *(Dimensional Expressions).* The set $\mathcal{DE}$ of dimensional expressions is defined as follows:

- a constant dimensional element in $\bigcup_{1\leq k\leq n} \mathcal{DEL}_{D_1,\ldots,D_k}$ is a dimensional expression;
- if e is an expression of a dimensional type $T[D_1,\ldots,D_k]$, $[\![\, e \,]\!]$ is a dimensional expression, denoting a dimensional element $\mu \in \mathcal{DEL}_{D_1,\ldots,D_k}$ corresponding to the points in which e is defined;
- if e is an expression of a boolean dimensional type $boolean[D_1,\ldots,D_k]$, $[\![\, e \,]\!]^{\,b}$ is a dimensional expression, denoting a dimensional element $\mu \in \mathcal{DEL}_{D_1,\ldots,D_k}$ corresponding to the points in which e has value $true$;
- if $\mu, \nu \in \mathcal{DE}$ are dimensional expressions, then so are $\mu \cup \nu$, $\mu \cap \nu$, $\mu \setminus \nu$, μ^c.

We extend function δ introduced in Section 2.1 to work on dimensional expressions, thus, given a dimensional expression e, $\delta(e) \subseteq \mathcal{DN}$ returns its dimensions. We remark that whenever two dimensional expressions e_1, e_2 with different dimensions are used together in a more complex expression, they are *aligned* to a common set of dimensions $\mathbf{D} = \delta(e_1) \cup \delta(e_2)$, by considering the whole dimensional domain for the missing dimensions.

Example 6 *Referring to the database schema of Example 2, given variable* `p` *of type* `Person` *and variable* `m` *of type* `Monitor`, $[\![\, \texttt{p.income} \,]\!] \cap [\![\, \texttt{m.U/G\_well} - \texttt{m.D/G\_well} > 1 \,]\!]^{\,\texttt{b}}$ *is a dimensional expression denoting a dimensional element with dimensions* `TIME,SPACE`*, corresponding to the spatiotemporal points in which the value of the U/G_well of* `m` *is greater then the value of its D/G_well plus one, and in whose time components the income of* `p` *is defined. Note that* $\delta([\![\, \texttt{p.income} \,]\!]) = \texttt{TIME}$*, whereas* $\delta([\![\, \texttt{m.U/G\_well} - \texttt{m.D/G\_well} > 1 \,]\!]^{\,\texttt{b}}) = \{\texttt{TIME}, \texttt{SPACE}\}$*. Referring to objects* $\texttt{m}_2$ *and* $\texttt{p}_1$ *of Example 3 the dimensional expression above denotes* $([5,45] \times \texttt{SPACE}) \cap ([10,\texttt{now}] \times \texttt{p}_1) = [10,45] \times \texttt{p}_1$.

3.2 Boolean Expressions

A boolean expression is an expression that evaluates in a non-dimensional boolean value. Boolean expressions are defined as follows.

Definition 9 *(Boolean Expressions).* The set $\mathcal{BE}$ of boolean expressions is defined as follows:

- if e is an expression of the boolean non-dimensional type $boolean$, e is a boolean expression;
- if μ, ν are dimensional expressions in $\mathcal{DE}$, such that $\delta(\mu) = \delta(\nu)$, $\mu \subseteq \nu$ is a boolean expression;
- if e is an expression of a boolean dimensional type $boolean[D_1,\ldots,D_k]$, e^S is a boolean expression, that is used as a shorthand for $[\![\, e \,]\!]^{\,b} \neq \emptyset$;[4]
- if e is an expression of a boolean dimensional type $boolean[D_1,\ldots,D_k]$, e^A is a boolean expression, that is used as a shorthand for $[\![\, e \,]\!]^{\,b} = [\![\, e \,]\!]$;[5]
- if $f, g \in \mathcal{BE}$ are boolean expressions, then so are $f \wedge g$, $f \vee g$, $\neg f$.

Example 7 *Referring to the database schema of Example 2, given variable* `c` *of type* `Chemical` *and variable* `m` *of type* `Monitor`*, the following are examples of boolean expressions:*

- $\texttt{c.max\_level} > \texttt{c.min\_level}$;
- $(\texttt{m.U/G\_well} - \texttt{m.D/G\_well} > 1)^{\texttt{A}}$;
- $(\texttt{m.U/G\_well} - \texttt{m.D/G\_well} > 1)^{\texttt{S}}$;
- $[\![\, \texttt{m.U/G\_well} > 1.0 \,]\!]^{\,\texttt{b}} \subseteq [\![\, \texttt{m.D/G\_well} > 1.5 \,]\!]^{\,\texttt{b}}$.

3.3 Path Expressions

A path expression is an expression that allows one to navigate through aggregation hierarchies on objects. Navigational access to temporal objects has been investigated in OOTempSQL [4], and revisited in [2]. The notion of path expression we propose here, besides being generalized to a multidimensional context, actually combines and extends the two notions. In particular, we distinguish among simple path expressions (inspired by [4]), that navigate through the object aggregation hierarchy, aligning the dimensions of the traversed objects; qualified path expressions (inspired by [2]), in which the portion of the dimensional space one is interested in navigating can be restricted; and single-valued

[4] S is a shorthand for *sometimes*, since it corresponds to the fact that the boolean expression is sometimes true.

[5] A is a shorthand for *always*, since it corresponds to the fact that the boolean expression is always true.

qualified path expression (also inspired by [2]) in which the dimensional value obtained through the navigation can be instantiated to a single dimensional point, thus obtaining a non-dimensional value. The two latter notions also allow one to specify a boolean condition to hold for the path expression to be defined, this is another extension with respect to the notions of path expressions previously proposed.

A path expression evaluates in a value v, that can be used in other expressions provided that type correctness is ensured. Path expressions are defined as follows.

Definition 10 *(Path Expressions).* If e is an expression of type $T[D_1,\ldots,D_k]$, $T \in \mathcal{OT}$, $1 \leq k \leq n$, a is an attribute of T with type $dom(a,T) = T'[D'_1,\ldots,D'_h]$, $1 \leq h \leq n$, $e.a$ is a path expression. This expression, whose type is $T'[D''_1,\ldots,D''_w]$, where $\{D''_1,\ldots,D''_w\} = \{D'_1,\ldots,D'_h\} \cup \{D_1,\ldots,D_k\}$, denotes the value such that $\forall\mu \in \mathcal{DEL}_{D''_1,\ldots,D''_w} : e.a(\mu) = e(\Pi_{D_1,\ldots,D_k}\mu).a(\Pi_{D'_1,\ldots,D'_h}\mu)$.

Example 8 *Consider an expression* `e` *of type* `C[TIME]`*, such that in class* `C` *attribute* `a` *has type* `integer[SPACE,TIME]`*. Suppose that* `e` *denotes the value* $\{\langle[10,20],\mathtt{c_1}\rangle,\langle[21,\mathtt{now}],\mathtt{c_2}\rangle\}$ *and suppose moreover that*

$$
\begin{aligned}
c_1.a = \{\ &\langle \mathtt{reg_1}\times[0,15],10\rangle,\langle \mathtt{reg_1}\times[15,40],20\rangle,\\
&\langle \mathtt{reg_2}\times[15,35],50\rangle\}\\
c_2.a = \{\ &\langle \mathtt{reg_1}\times[15,40],30\rangle,\langle \mathtt{reg_1}\times[41,\mathtt{now}],40\rangle,\\
&\langle \mathtt{reg_2}\times[40,50],70\rangle\}
\end{aligned}
$$

then `e.a` *denotes the following value of type* `integer[SPACE,TIME]`*:*

$$
\begin{aligned}
\{\ &\langle \mathtt{reg_1}\times[10,15],10\rangle,\langle \mathtt{reg_1}\times[15,20],20\rangle,\\
&\langle \mathtt{reg_2}\times[15,20],50\rangle,\langle \mathtt{reg_1}\times[21,40],30\rangle,\\
&\langle \mathtt{reg_1}\times[41,\mathtt{now}],40\rangle,\langle \mathtt{reg_2}\times[40,50],70\rangle\}
\end{aligned}
$$

Often, however, a user is not interested in navigating through the whole set of values taken by a dimensional value over the dimensional space, rather it is interested to restrict them to a specific dimensional element. This can be achieved through the notion of qualified path expression, formalized as follows.

Definition 11 *(Qualified Path Expressions).* If $e.a$ is a path expression of type $T[D_1,\ldots,D_k]$, f is a boolean expression, μ is a dimensional expression such that $\delta(\mu) \subseteq \{D_1,\ldots,D_k\}$, $e.a \downarrow (f,\mu)$ is a qualified path expression. This expression, whose type is $T[D'_1,\ldots,D'_h]$, where $\{D'_1,\ldots,D'_h\} = \delta(\mu) \setminus \{D_1,\ldots,D_k\}$, denotes the value denoted by $e.a_{|_\mu}$ if f holds, it is undefined otherwise.

When the boolean expression f is missing in a qualified path expression, the boolean constant $true$ is implicitly assumed, and when the dimensional expression μ is missing, the null dimensional element in $\mathcal{DEL}_\emptyset$ is implicitly assumed.

Example 9 *Referring to the database schema of Example 2, given variable* `l` *of type* `Land` *and variable* `m` *of type* `Monitor`*, the following are examples of qualified path expressions:*

- $\mathtt{m.D/G\_well} \downarrow ((\mathtt{m.chemical.name} = '\mathtt{atrazine}')^{\mathtt{A}}$, $[\![\,\mathtt{m.D/U\_well} > \mathtt{m.chemical.max\_level}\,]\!]^{\mathtt{b}})$,

  whose type is `short[TIME,SPACE]`*, and that, evaluated on object* $\mathtt{m_1}$ *of Example 3, denotes the value* $\{\langle[6,10]\times\mathtt{p_2},1.4\rangle,\langle[11,\mathtt{now}]\times\mathtt{p_2},2.9\rangle\}$*;*

- $\mathtt{l.owner} \downarrow (\,[\![\,\mathtt{l.acres} > 50\,]\!]^{\mathtt{b}}).\mathtt{income}$*, whose type is* `integer[SPACE,TIME]`*, and that, evaluated on object* $\mathtt{l_1}$ *of Example 3, denotes the value*

  $\{\langle \mathtt{reg_2}\times[5,20],1000\rangle,\ \langle \mathtt{reg_2}\times[21,40],1500\rangle,\ \langle \mathtt{reg_2}\times[41,45],2000\rangle\}$.

Finally, when a path expression evaluates to a value v and is qualified by a dimensional element μ, such that v assigns a single value to μ, the user can be interested in simply obtaining that (non-dimensional) value. This can be achieved through the notion of single-valued qualified path expression, formalized as follows.

Definition 12 *(Single-Valued Qualified Path Expressions).* If $e.a$ is a path expression of type $T[D_1,\ldots,D_k]$, f is a boolean expression, μ is a dimensional expression such that $\delta(\mu) = \{D_1,\ldots,D_k\}$, $e.a \downdownarrows (f,\mu)$ is a single-valued qualified path expression. This expression, whose type is T, denotes the value denoted by $e.a(\mu)$ if f holds and $e.a$ assumes a single value over μ, it is undefined otherwise.

Example 10 *Referring to the database schema of Example 2, given variable* `p` *of type* `Person` *the following is an example of single-valued qualified path expression:*

$$
\begin{aligned}
\mathtt{p.address} \downdownarrows (\ &(\mathtt{count(p.owns)} \neq 0)^{\mathtt{S}},\\
&[\![\,\mathtt{p.income} \geq 2000\,]\!]^{\mathtt{b}}).\mathtt{city}
\end{aligned}
$$

whose type is `string`*, and that, evaluated on object* $\mathtt{p_1}$ *of Example 3, denotes the value* `'iowacity'`.

4 Conclusions

In this paper we have proposed a notion of dimensional object, and we have investigated some issues at the basis for queries on dimensional objects. This work can be regarded as an important stepping stone for a dimensional object data model and query language.

In extending the proposed framework to model and query collections several issues will have to be revisited. We will investigate the implications of removing the homogeneity assumption, that is, the requirement that all the attributes of an object must have the same dimensional domain. Removing this assumption will require to deal with null values. Moreover, we assume that if a dimension is missing in a value, then that value is intended to be valid for all the elements in that dimension. Thus, there is no equivalent of the notion of static attribute nor of that of static object with the meaning in [1]: each value and each object are dimensional; if no dimension is specified, the whole dimensional space is assumed. Note that static values in a temporal context naturally refer to *now*; for other dimensions, however, it could not be possible to identify such a *default point* to associate with (static) data for which the dimension is missing. Finally, we are also interested in extending our model to a multigranularity context, in which granularities are seen as dimensions, and dimension alignment as a way to reveal granularities.

For what concerns the query language, the basic idea is that of extending with the essential constructs of the parametric relational algebra at the basis of TempSQL [12] (i.e. dimension alignment and restructuring) the object query language of the ODMG standard, OQL [3], and to investigate how that language can be employed to query dimensional data. One of the main consequences of an OQL-like query language is that the result of a query is a set of either values or objects of the appropriate type; in particular, the result of a projection can be a set of record values (tuples). Since, however, algebraic optimization is crucial also for object databases, a dimensional object algebra will be defined. This algebra is obtained as a dimensional extension of object algebras that are the formalisms under OQL [6]. "Classical" algebraic identities will be revisited, to obtain a set of identities still holding between dimensional object algebra expressions, and to clarify how the new constructs dealing with data dimensionality interact with existing ones [11].

References

[1] E. Bertino, E. Ferrari, and G. Guerrini. T_Chimera: A Temporal Object-Oriented Data Model. *Theory and Practice of Object Systems*, 3(2):103–125, 1997.

[2] E. Bertino, E. Ferrari, and G. Guerrini. Navigational Accesses in a Temporal Object Model. *IEEE Transactions on Knowledge and Data Engineering*, 10(4):656–665, 1998.

[3] R. Cattel et al. *The Object Database Standard: ODMG 3.0*. Morgan-Kaufmann, 1999.

[4] T. S. Cheng and S. K. Gadia. An Object-Oriented Model for Temporal Databases. In *Proc. of the Int'l Workshop on an Infrastructure for Temporal Databases*, 1993.

[5] T. S. Cheng and S. K. Gadia. A Pattern Matching Language for Spatio-Temporal Databases. In *Proc. of the Third Int'l Conf. on Information and Knowledge Management*, pages 288–295, 1994.

[6] S. Cluet and C. Delobel. A General Framework for the Optimization of Object-Oriented Queries. In *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, pages 383–392, 1992.

[7] C. Combi and G. Cucchi. GCH-OSQL: A Temporally-Oriented Object-Oriented Query Language Based on a Three-Valued Logic. In *Proc. of the Fourth Int'l Workshop on Temporal Representation and Reasoning*, pages 119–127, 1997.

[8] L. Fegaras and R. Elmasri. A Temporal Object Query Language. In *Proc. of the Fifth Int'l Workshop on Temporal Representation and Reasoning*, pages 51–59, 1998.

[9] S. K. Gadia. A Homogeneous Relational Model and Query Language for Temporal Databases. *ACM Transactions on Database Systems*, 13(4):418–448, 1988.

[10] S. K. Gadia. Applicability of Temporal Data Models to Query Multilevel Security Databases: A Case Study. In *Temporal Databases: Research and Practice*, LNCS 1399, pages 238–256, 1997.

[11] S.K. Gadia and S.S. Nair. Algebraic Identities and Query Optimization in a Parametric Model for Relational Temporal Databases. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):793–807, 1998.

[12] S.K. Gadia and S.S. Nair. Temporal Databases: A Prelude to Parametric Data. In A. Tansel et al., editors, *Temporal Databases: Theory, Design, and Implementation*, pages 28–66. Benjamin/Cummings, 1993.

[13] I. Goralwalla, M. Özsu, and D. Szafron. An Object-Oriented Framework for Temporal Data Models. In *Temporal Databases: Research and Practice*, LNCS 1399, pages 1–35, 1997.

[14] R. T. Snodgrass. Temporal Object-Oriented Databases: A Critical Comparison. In W. Kim, editor, *Modern Database Systems: The Object Model, Interoperability and Beyond*, pages 386–408. Addison-Wesley/ACM Press, 1995.

[15] S. Su, S. Hyun, and H. Chen. Temporal Association Algebra: A Mathematical Foundation for Processing Object-Oriented Temporal Databases. *IEEE Transactions on Knowledge and Data Engineering*, 10(3):389–408, 1998.