Proc. the 3rd international conference on Concept Lattices and Their Applications (CLA 2005),
Olomouc, Czech Republic, September 7-9, 2005, 123-134.

Hierarchical Multidimensional Modelling in the Concept-Oriented Data Model

Alexandr Savinov

Fraunhofer Institute for Autonomous Intelligent Systems
Schloss Birlinghoven, Sankt Augustin, D-53754 Germany
savinov@ais.fraunhofer.de
http://www.ais.fraunhofer.de/~savinov
http://conceptoriented.com

Abstract. In the paper the concept-oriented data model (COM) is described from the point of view of its hierarchical and multidimensional properties. The model consists of two levels: syntactic and semantic. Concepts are combinations of superconcepts while items are combinations of superitems. It is described how this model can be interpreted as a hierarchical coordinate system. Two operations of projection and de-projection are used in the mechanism of access path. Grouping and aggregation with roll up and drill down are implemented via multidimensional de-projection. The described approach can be applied to very different problems for multidimensional modelling including database systems, online analytical processing, knowledge based systems, ontologies, complex categorizations, knowledge sharing and semantics web.

1 Introduction

Currently there exist several general approaches to data modelling based on different principles and main notions such as relations in the RM [5,9], entity and relationships in the ERM [4], facts and object roles in the ORM [13], subject-predicate-object triples in the RDF [2] and many others. One of the most interesting approaches is based on using *dimension* as the main notion and construct of the model and dimensionality (degrees of freedom) as one of its main characteristics. This direction has been developed in the area of multidimensional databases [1,12,15,17,22] and online analytical processing (OLAP) [3,17]. In this paper we describe an approach to dimensionality modelling based on the concept-oriented data model (COM).

The described model has been proposed in [19,20] and it takes its origin from the idea of multidimensional hierarchical space (analytical space) [18]. There are several general assumptions about the nature of data in the COM. One of them is that the whole model is viewed as one global construct with canonical syntax and semantics. Analogous assumption is used in the universal relation model (URM) [6,14,16]. Using this assumption we can derive properties of elements as properties of the whole model as well as automate many operations such logical navigation and query construction. Another important assumption is that the model is based on ordering its elements which is analogous to concept-lattices, formal concept analysis (FCA) [8]

and ontologies [7]. This means that the order of elements is of crucial importance for the model and element position with respect to other elements determines most of its properties. In great extent everything in the COM is about order of elements. In contrast to FCA where objects are grouped into concepts depending on their properties (represented in incidence relation), items in the COM belong to concepts in advance and then their properties depend on the relative position to other items. The third assumption is that the hierarchical multidimensional structure of the model can be used for automating data access and logical navigation. The mechanism of access paths and queries in the COM is very close to that used in the functional data model (FDM) [10,11,21].

In section 2 we formally define the COM including its physical and logical structure (section 2.1), syntactic dimensionality structure (section 2.2) and semantics (section 2.3). In section 3 we describe how this model can be interpreted as a hierarchical multidimensional coordinate system. In section 4 we introduce two operations of projection and de-projection and the mechanism of access path based on them. Section 4 describes multidimensional grouping and aggregation. We will follow a convention that concept names are capitalized and written in plural while dimension names are in singular and start from lower case letter, for example, `Products` is a concept while `product` is a dimension with the domain in `Products`. Formulas will be written as usual in italics while queries and examples in monotype font (Courier).

2 Model Definition

2.1 Physical and Logical Structure

In the concept-oriented model each element E is defined via other elements and this definition consists of two parts: (i) a *collection* of other elements $E = \{a, b, \ldots\}$, $a \in E, b \in E, \ldots$, and (ii) a *combination* of other elements: $E = \langle U, V, \ldots \rangle$, $U \lhd E, V \lhd E, \ldots$. Here $\in$ denotes membership in collection and $\lhd$ denotes membership in combination. This means that each element stores and knows directly two groups of elements which have different purpose and interpretation. Elements within a collection are identified by *reference* while elements within a combination are identified by *position*. Collectional parts of elements describe *physical structure* of the model which is assumed to be strictly hierarchical so that each element is a member of one parent collection (Fig. 1a). It is important that this structure is fixed in the sense that elements can be added or deleted but they cannot change their parent collection. For example, element *b* cannot be moved from *C* to *U* (Fig. 1a). Physical structure is used to implement references which are a mechanism of physical representation and access. In other words, each element in the model has a reference according to its position in the physical hierarchy of elements starting from the root.

Combinational parts of elements are intended to describe *logical structure* of the model. Logical structure defines how elements are characterized by other elements. It is important that an element may be a member of several combinations and itself

combines several other elements (Fig. 1b). Another important property of logical structure is that it is not fixed and we can always change how any element is logically defined.

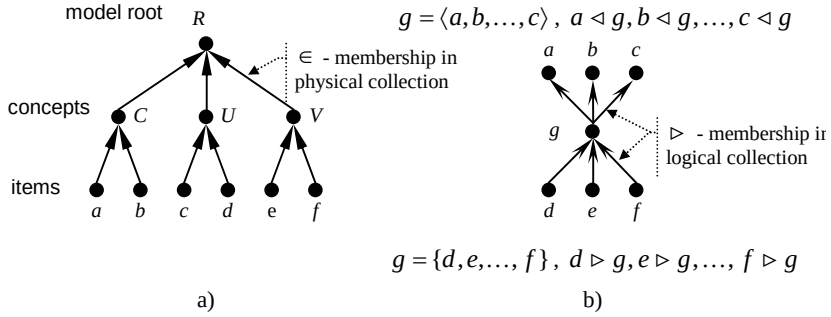

Fig. 1. a) Physical structure is defined by a hierarchical collection where each element has only one parent. In two-level model the root consists of concepts (syntax) and concepts consists of items or concept instances (semantics). b) Logical structure is defined by combinations with dual interpretation as a logical collection of elements. In two-level model each concept is a combination of superconcepts and each item is a combination of superitems. Here element *g* can be a concept (and then *a, b,…,c* are superconcepts and *d,e,…,f* are subconcepts) or an item (and then *a, b,…,c* are superitems and *d,e,…,f* are subitems).

Elements in combination are interpreted as object properties, i.e., an element is a combination of its properties. The dual interpretation is that an element is a *logical collection* consisting of all the elements which include it into their combinational part. Thus any element is a combination of other elements and, dually, a (logical) collection of other elements. Formally, if $E = \langle \ldots, U, \ldots \rangle$ ($U \lhd E$) then $U = \{\ldots, E, \ldots\}$ ($E \rhd U$) where $\rhd$ denotes membership in logical collection. In contrast to physical collections which are fixed and intended to implement references, logical collections (dual combinations) are intended to represent data itself. We will follow a convention that combination $E = \langle U, V, \ldots \rangle$ is drawn below its elements $U, V, \ldots$. In this case an upward arrow connects combination with its constituents and denotes membership in logical collection. Then each element can be represented as a combination of all elements above it and, dually, a (logical) collection of all elements below it (Fig. 1b). In conventional terms this means that an object/record is a combination of its properties but at the same time it is a collection/group of all objects/records where it is used as a property. (A property is a group/category for all objects that have it.)

From the point of view of physical structure we distinguish three types of the model: (i) *one-level model* has one root and a number of data items in it, (ii) *two-level model* has one root, a number of concepts in it, each of them having a number of data items, (iii) *multi-level model* has an arbitrary number of levels in the physical hierarchy. In this paper we consider only two-level model defined as consisting of the following elements:

[Root] One root element *R* is a physical collection of concepts, $R = \{C_1, C_2, \ldots, C_N\}$,

[Syntax] Each concept is (i) a combination of other concepts called *superconcepts* (while this concept is a *subconcept*), $C = \langle C_1, C_2, \ldots, C_n \rangle \in R$, and (ii) a physical collection of *data items* (or concept instances), $C = \{i_1, i_2, \ldots\} \in R$,

[Semantics] Each data item is (i) a combination of other data items called *superitems* (while this item is a *subitem*), $i = \langle i_1, i_2, \ldots, i_n \rangle \in C$, and (ii) empty physical collection, $i = \{\}$,

[Special elements] If a concept does not have a superconcept then it is referred to as *primitive* and its superconcept is one common *top concept*; and if a concept does not have a subconcept then it is assumed to be one common *bottom concept*, and an absence of superitem is denoted by one special *null item*.

[Cycles] Cycles in subconcept-superconcept relation and subitem-superitem relation are not allowed,

[Syntactic constraints] Each data item from a concept may combine only items from its superconcepts.

2.2 Dimensions and Inverse Dimensions

Each superconcept has a uniquely identified position within a concept definition, $C = \langle x_1 : C_1, x_2 : C_2, \ldots, x_n : C_n \rangle$, which is called *dimension* (of rank 1 or local dimension). Here superconcepts $C_1, C_2, \ldots, C_n$ are called *domains* or ranges for dimensions $x_1, x_2, \ldots, x_n$, $C_j = \mathrm{Dom}(x_j)$. Normally dimensions (concept positions within a combination) are identified by names or integer values. The syntactic structure of concepts can be represented by a graph where nodes are concepts and edges are dimensions connecting subconcepts with their superconcepts. Cycles are not allowed in this graph so that concepts cannot be defined via themselves. Loops can be permitted for simplicity and performance reasons. If one of two concepts is direct or indirect parent of another then they are called syntactically *dependent* or parallel concepts, otherwise the two concepts are referred to as *independent* or orthogonal. In particular, all primitive concepts are orthogonal.

A dimension x of rank k is a sequence of k dimensions of rank 1 (separated by dots) where each next dimension in the sequence belongs to the domain of the previous one:

$x = x^1.x^2.\cdots.x^k$, where $\mathrm{Dom}(x^j) \lhd \mathrm{Dom}(x^{j-1})$, $j = 2,3,\ldots,k$

Dimensions will be frequently prefixed by the very first concept: $C.x^1.x^2.\cdots.x^k$. Also we frequently will use the terms dimension and its domain interchangeably. Each dimension is represented by one path in the concept graph and the number of edges in the path is its rank. A dimension with the primitive domain (direct superconcept of the top concept) is referred to as *primitive dimension*. For example, `Auctions.product.category` in Fig. 2 is a primitive dimension of rank 2 from the source concept `Auctions` to its superconcept `Categories` of rank 2. Note that there may be several different paths (dimensions) between a concept and its superconcept.

The number of different primitive dimensions (paths) of a concept is referred to as the *concept primitive dimensionality*. The model dimensionality is equal to that of the bottom concept, that is, the number of dimension paths from the bottom concept to primitive concepts (or, equivalently, to the top concept) is the model primitive dimensionality. The length of the longest dimension (path) of a concept is referred to as *concept rank*. The length of the longest dimension of the bottom concept is the *model*

rank. Thus each concept-oriented model is characterized by two properties: (i) model rank describing its hierarchy (depth), and (ii) model dimensionality describing its multidimensionality (width). Model structure can vary from the flat space with many dimensions to one-dimensional hierarchal space. The task of the model syntactic design consists in defining what concrete structure of dimensions is appropriate for one or another problem domain.

Dually, each concept is a logical collection of its subconcepts: $C = \{S_1, S_2, \ldots, S_n\}$, $C \lhd S_j$. *Inverse dimension* is a dimension with opposite direction, i.e., where the dimension starts the inverse dimension has its domain, and where the dimension ends the inverse dimension has its start. In concept graph inverse dimension is a path from superconcept to some its subconcept. Inverse dimension of rank 1 is an edge in concept graph with the opposite direction (from superconcept to a subconcept). Inverse dimensions do not have their own identifiers because they can be produced from the corresponding dimensions by inverting their direction. We will denote inverse dimensions by enclosing the corresponding dimension into curly brackets. If $C.x^1.x^2.\cdots.x^k$ is a dimension of concept C with rank k then $\{C.x^1.x^2.\cdots.x^k\}$ is inverse dimension of concept $\mathrm{Dom}(x^k)$ with the domain in C and the same rank k. Thus any concept is characterized by (i) a set of dimensions leading to superconcepts and (ii) a set of inverse dimensions leading to subconcepts. For example, `{Auctions.product.category}` is an inverse dimension of rank 2 from superconcept `Categories` to its subconcept `Auctions` (Fig. 1).

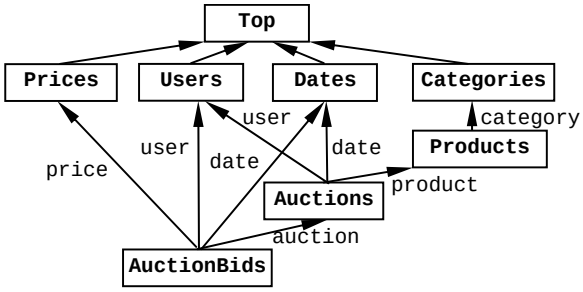

Fig. 2. An example of logical structure of concepts (syntax). Each concept is a combination of its superconcepts and a logical collection of its subconcepts. The model syntax can be represented as a concept graph where one path is one named dimension. Dimensionality and inverse dimensionality of this model is 6.

2.3 Semantics

Semantically each concept is a physical collection of its items, $C = \{i_1, i_2, \ldots\}$, where each item is a combination of items from its superconcepts:

$i = \langle i_1, i_2, \ldots, i_n \rangle$, where $i \in C = \langle C_1, C_2, \ldots, C_n \rangle$ and $i_j \in C_j$, $j = 1,2,\ldots,n$

Just like for the concept graph cycles are not permitted so that the items constitute an acyclic graph. In such a definition two levels of logical description (syntactic and

semantic) are isolated, i.e., we can represent things either by means of concepts or by means of items within their own acyclic graphs. A bridge between them is established via the physical structure where each item is physically a member of one concept. The logical structure of concepts is used to describe model dimensionality and impose syntactic constraints on possible items. In other words, without syntactic constraints an item can combine any items from any other concepts. In the presence of syntactic constraints an item can combine only items from its superconcepts.

An important property of the concept-oriented semantics is its *globality* which means that each item has its semantics distributed all over the model. Indeed, each individual item taken in isolation as an instance of its concept has no its own intrinsic properties or characteristics and the only thing that can be used to distinguish it from other items is its (physical) reference (location in the physical hierarchy). Items in the concept-oriented model can be only characterized by other items which in turn are characterized by other items and so on. If we want to get some property then we need to specify what other items we want to retrieve. However, an item representing a property may well be also semantically empty (if it is not a primitive item) so we need to retrieve some other item and so on. Thus any individual item semantics is distributed all over the model and finding item properties is reduced to specifying what part of the whole model semantics (associated with this item) we want to get.

Another property of this approach is that it does not have a dedicated mechanism of multi-valued properties. In order to model single-valued and multi-valued attributes the COM uses dimensions and inverse dimensions, respectively. In other words, dimensions are always single-valued and return one superitem as a value while inverse dimensions are always multi-valued and return a collection of subitems as its values. Single-valued attributes are upward directed while multi-valued attributes are downward directed in the concept graph. Such a mechanism has significant consequences. We cannot now simply change the multiplicity of an attribute because it entails changing the relative position of the domain. For single-valued attributes the domain is positioned above the source concept while for multi-valued attributes the domain is positioned below the source concept. Thus changing the multiplicity results in significant changes in the model semantics and syntax.

3 Hierarchical Coordinate System

Universe of discourse of a concept is the Cartesian product of its superconcepts:

$$\Omega = C_1 \times C_2 \times \ldots \times C_n = \{\omega = \langle i_1, i_2, \ldots, i_n \rangle \mid i_j \in C_j,\ j = 1,2,\ldots,n\}$$

Each syntactically possible combination $\omega \in \Omega$ of superitems is referred to as *point*. Only a subset of all points from Ω really exists and this subset is precisely the concept semantics (a set of its items). The superconcepts in the definition of a concept are treated as *axes* of the multidimensional space while items in these superconcepts are *coordinates* on these axes. The model can be then interpreted as a hierarchical multidimensional coordinate system where each item is characterized by its position along superconcepts as axes and, at the same time, is a coordinate taken by its subitems.

For example, in Fig. 3 the problem domain consists of 2-dimensional plane *XY* 4 points of which are combinations of 2 coordinates along two axes *X* and *Y*. From the

concept-oriented point of view both axes *X* and *Y* are superconcepts each with 2 superitems (black circles). Their common subconcept *XY* potentially can consist of 4 subitems because there exist only 4 combinations of superitems. However, subconcept *XY* has only 3 subitems in its semantics while the fourth potentially possible subitem does not exist (dashed circles). Each item from concept *XY* corresponds to one point in the 2-dimensional plane while its constituents are interpreted as coordinates taken along the superconcepts as axes.

Such a coordinate system is flexible because we can change the available coordinates for positioning objects by adding or removing items. This means that the coordinate system is an integral part of and is described by the model itself. New axes are introduced by defining new concepts and new possible coordinates can be added by defining new data items within existing concepts. For example, in Fig. 3 we might add a new coordinate *Z* and then data points from their subconcept *XYZ* could be positioned by assigning three coordinates rather than two.

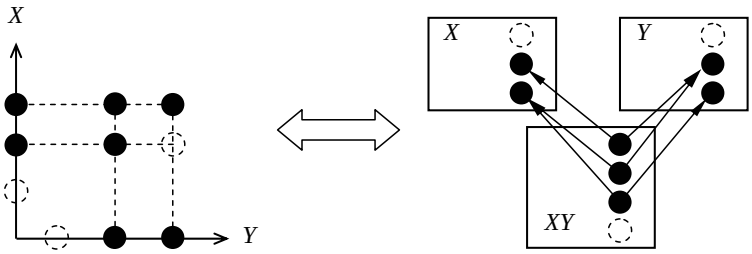

Fig. 3. Interpretation the model as a coordinate system. Plane with two axes is a subconcept with two superconcepts.

Another advantage is that it allows the modeller to create hierarchical coordinate systems. The thing is that all concepts have equal rights and can be treated as an axis with coordinates for its subconcepts and as a space with objects for its superconcepts. Thus any new subconcept defined via its superconcepts can be treated as a separate individual axis for future subconcepts. The coordinates in such a complex axis are complex multidimensional objects having many their own coordinates. For example, in Fig. 3 each point in *XY* is a combination of two coordinates from *X* and *Y* but in the hierarchical system each coordinate from *X* and *Y* can be itself a point with its own coordinates taken from their superconcepts. On the other hand, each item from *XY* can be used as a possible coordinate for new subconcepts added later. For example, we might add a new subconcept *A* which has two dimensions in *XY* and *Z*. The primitive dimensionality of *A* is 3 but we specify only two coordinates for each its item.

The problem of modelling in such a setting is formulated as designing a hierarchical coordinate system (syntax) and positioning objects in it via other objects treated as coordinates (semantics). After defining the model syntactic structure items are added in superconcepts and serve as coordinates for adding items in subconcepts. For example, in Fig. 2 each auction bid from concept `AuctionBids` has four coordinates along axes `Prices`, `Users`, `Dates` and `Auctions`. However, the last coordinate has its own three coordinates, i.e., each auction along axes `Users`, `Dates` and `Products`. Finally each auction bid has 6 primitive coordinates because there are 6 different

paths from the concept `AuctionBids` to the top concept. We might equivalently describe this model as the flat multidimensional space consisting of 6 primitive concepts and one 6-dimensional concept `AuctionBids`. Obviously, such a model would have very low level and too inflexible. Having a hierarchical coordinate system allows us to introduce intermediate levels, which impose syntactic constraints. The concept `AuctionBids` in this case has only 4 its own dimensions and describing its items in this case is much more convenient.

4 Projection and De-projection

Let us assume that I is a subset of items from concept C, $I \subseteq C$. In order to get related items from some target domain concept we need to specify a path by means of either dimension (with the domain in a superconcept of C) or inverse dimension (with the domain in a subconcept of C). This path (dimension or inverse dimension) is applied to the source subset of items and returns a subset of related items from the domain concept. If $d = d^1.d^2.\cdots.d^k$ is a dimension of C with domain in $U = \mathrm{Dom}(d)$ then operation $I \to d$ is referred to as *projection* of I to U and returns a set of superitems referenced by items from I:

$$I \to d = \{u \in U \mid i.d = u, i \in I \subseteq C\}$$

It is important that each item from U can be included into the result collection (projection) only one time. If we want to include superitems as many times as they are referenced then we can use dot instead of arrow, i.e., $I.x$ includes *all* referenced superitems of U even if they occur more than once. For example, if A is a collection of today's auctions then `A->product.category` will return all today's categories (each category only once) while `A.product.category` will return categories for each auction in A (as many categories as we have auctions).

Dual operation to projection returns a set of related items from a subconcept. If $\{d\} = \{d^1.d^2.\cdots.d^k\}$ is an inverse dimension of C with domain in $S = \mathrm{Dom}(\{d\})$ then *de-projection* of I to S is defined as follows:

$$I \to \{d\} = \{s \in S \mid s.d = i, i \in I \subseteq C\}$$

The result collection of de-projection $I \to \{d\}$ consists of all items from subconcept S which reference items from I via dimension d. For example, if C is a set of categories then `C->{Auctions.product.category}` returns a set of all auctions with the products having these categories. Dimension d specifying path between the source concept and the target concept is referred to as *bounding* dimension.

Access path is sequence of dimensions or inverse dimensions separated either by dot or by arrow where each next operation is applied to the result collection returned by the previous operation. Each access path has a zigzag form in the concept graph where dimensions move up to a superconcept while inverse dimensions move down to a subconcept in the concept graph.

It is possible to restrict items that are included into de-projection by providing a condition that all items from the domain subconcept have to satisfy:

$$I \to \{s : S.d \mid f(s)\} = \{s \in S \mid s.d = i \wedge f(s) = \mathrm{true}, i \in I \subseteq C\}$$

Here $S.d$ is bounding dimension from subconcept *S* to the source collection *I*, *s* is instance variable that takes all values from set *S* and the predicate *f* (separated by bar) must be true for all items *s* included into the result collection (de-projection). After the query in angle brackets we can specify a list of values returned with each item of the new result collection. For example, access path

```
C->{a : Auction.product.category | a.date==today}<user,date>
```

will return user and date for all today's auctions for a subset of categories from *C*.

Frequently we need to have aggregated characteristics of items computed from related items. This can be done by defining a *derived property* of concept which is a named definition of a query returning one item or a collection for each item from this concept. For example, we could define a derived property `allBids` of concept `Auctions` returning a collection of all bids for one auction:

```
Auctions.allBids = this->{ AuctionBids.auction }
```

(Keyword `this` is an instance variable referencing the current item.) Derived properties can use other properties. For example, the maximum bid for an auction could be defined as the following derived property:

```
Auctions.maxBid = max( this.allBids.price )
```

Here we get a set of all bids by applying existing property `allBids` to the current item, then get their prices via dot operation and then find the maximum price. In the same way we might compute the mean price for ten days for one category:

```
Category.meanPriceForTenDays =
  avg( {ab in AuctionBids.auction.product.category |
    ab.auction.date > today-10 }.price );
```

5 Multidimensional Grouping and Aggregation

In the previous section we assumed that there is only one path between the source concept and the target concept with related items, which is specified by means of one bounding dimension. However, in multidimensional case there can be more than one path which is specified by means of several bounding dimensions. If *I* is a subset of items from the source concept *C*, $I \subseteq C$, *S* is some subconcept of *C*, and $d_1, d_2, \ldots, d_n$ are different dimensions of *S* with the domain in *C*, $\mathrm{Dom}(d_j) = C$, $j = 1,2,\ldots,n$, then *multidimensional de-projection* of *I* to *S* is defined as a set of subitems $s \in S$ that reference source items $i \in I$ along all dimensions:

$$I \to \{d_1, d_2, \ldots, d_n\} = \{s \in S \mid s.d_1 = i \wedge s.d_2 = i \wedge \ldots \wedge s.d_n = i, i \in I \subseteq C\}$$

If we use only one dimension then we get 1-dimensional de-projection defined in the previous section. The idea of multidimensional de-projection is that we want to get subitems *s* belonging to one group item *i* along *all* specified dimensions. For example, if we select a set of points interpreted as groups/categories then multidimensional de-projection will return a set of all objects with coordinates in these points (projected to these points).

The mechanism of multidimensional de-projection can be used for online analytical processing (OLAP) where the task consists in finding aggregated characteristics of

objects with the possibility to vary level of details. Let us assume that *S* is some concept with items to be aggregated. In concept *S* we select a number of *dimension paths* $p_1, p_2, \ldots, p_n$ which will be used for analysis. A *level* $L = \langle l_1, l_2, \ldots, l_n \rangle$ is a set of numbers specifying a set of positions along dimension paths, a set of dimensions $d_1, d_2, \ldots, d_n$ of concept *S* with ranks $l_1, l_2, \ldots, l_n$ with their domains $D_1, D_2, \ldots, D_n$, $D_j = \mathrm{Dom}(d_j)$, $j = 1,2,\ldots,n$. One constituent of level l_j is a rank of dimension d_j taken along path p_j. Operation of increasing rank l_j (one constituent of level) of one dimension d_j is referred to as *roll up*. Operation of decreasing rank l_j of one dimension d_j is referred to as *drill down*. If all level constituents are 0s then we get concept *S* which contains the most detailed information used for analysis. If all level constituents are 1s then we get a set of dimensions with rank 1 and their corresponding direct superconcepts of *S* as domains.

For each level we can define its universe of discourse or *multidimensional cube* as a set of all possible points produced from the corresponding domains:

$$\Omega_L = D_1 \times D_2 \times \ldots \times D_n = \{\omega = \langle \omega_1, \omega_2, \ldots, \omega_n \rangle \mid \omega_j \in D_j\},\ D_j = \mathrm{Dom}(d_j)$$

Projection of a set of items $I \subseteq S$ to level *L* is a set of points from Ω_L referenced by items from *I* via dimensions $d_1, d_2, \ldots, d_n$ of level *L*:

$$I \to L = \{\omega \in \Omega_L \mid i.d_1 = \omega \wedge i.d_2 = \omega \wedge \ldots \wedge i.d_n = \omega, i \in I \subseteq S\}$$

Multidimensional de-projection of a subset of items $I \subseteq \Omega_L$ (where Ω_L is determined by level *L*) is a set of items from *S* with projection in *I*:

$$I \to \{L\} = \{s \in S \mid s.d_1 = \omega \wedge s.d_2 = \omega \wedge \ldots \wedge s.d_n = \omega, \omega \in I \subseteq \Omega_L\}$$

Multidimensional de-projection allows us to find items from a subconcept that are related to each point from one level. Thus items from the subconcept are broken into groups depending on the point from the level they are projected to. For each such group we can then find some aggregated property.

For example, let us suppose that *S*=`Auctions` is the concept we want to analyse. The first dimension path is p_1=`Auctions.date` has only one level of details. The second dimension path p_2=`Auctions.products.categories` has two levels of details so that we can consider either `Products` (more details) or `Categories` (less details) as domains for our analysis. The goal is to understand how properties of auctions are distributed over these two dimensions. The universe of discourse is defined as follows: `{Dates, Categories | having(this)}`. Here we enumerate domains from the level D_1=`Dates`, D_2=`Categories` and provide also restriction on the points we are interested in the predicate `having(this)` (analogous to HAVING clause in SQL). Then each point from this space has to be de-projected to the target concept *S*=`Auctions` and this group of items can be used to find some aggregated property. The following query will find average maximum bid for last week:

```
{d : Dates, c : Categories | isLastWeek(d) }<
  avg(this->{Auctions.date,Auctions.product.category}.maxBid)
  as averagePrice >
```

Here in angle brackets after the source collection we specify values computed and returned for each point from the universe of discourse. In this example, we select only

points for last week by imposing constraint on the source 2-dimensional space `isLastWeek(d)`. Each point from this space is de-projected to concept `Auctions` using query `this->{Auctions.date, Auctions.product.category}` with two bounding dimensions. Then we add property `maxBid` (defined in the previous section) to this access path because we want to find average value for all items of this de-projection. And finally this group of auction prices is passed as a parameter to the aggregation function which returns one value as a computed characteristic of one source point with the new name `averagePrice`. The result is that we have a two-dimensional space with one aggregated property (called *measure* in OLAP).

It is important that we can vary the level of details. For example, to get more details we can drill down along the second dimension path and consider `Products` instead of `Categories`. Additionally, we might want to consider only auctions with price higher than some threshold:

```
{d : Dates, c : Products | isLastWeek(d) }<
  avg(this->{Auctions.date,
             Auctions.product | this.maxBid > 10 }.maxBid)
  as averagePrice >
```

This query will produce all combinations of dates and products for last week and for each of them find a group of auctions with price higher than 10. Then this group of prices is averaged and returned as a new aggregated property. Note that grouping is carried out independent of any measure because we simply de-project a point and find a set of related subitems. Measure in this case is any property of items in de-projection (concept *S*). In particular, it is possible to define several measures in one or more different subconcepts with their own bounding dimensions. It is important also that dimension paths, measure and cubes are roles which are assigned only for certain type of analysis while the model itself is defined only by its concept graph.

6 Conclusion

In the paper we described an approach to hierarchical multidimensional modelling based on the concept-oriented data model. This approach separates physical structure and logical structure by defining each element as a collection of other elements and as a combination of other elements. The logical structure is then used to represent data. For modelling (syntactic) dimensionality structure we use concepts which are combinations of their superconcepts. A distinguishing feature is that we define the model structure as an acyclic graph with edges as dimensions and after that this graph is used for analysis by assigning roles to its elements such as levels, cubes, dimensions and measures. In contrast, the conventional approaches are based on defining dimensions, cubes and other elements needed for analysis as initial elements of the model. Two important operations used for analysis are projection and de-projection which use the mechanism of dimensions and inverse dimensions, respectively. Taking into account simplicity and rich set of features and mechanisms this model can be applied not only to dimensionality modelling but used in many other application areas for modelling a wide range of practical situations and use cases.

References

1 R. Agrawal, A. Gupta and S. Sarawagi, Modeling multidimensional databases, Proc. 13th International Conference on Data Engineering (ICDE'97), 232-243, 1997.

2 T. Berners-Lee, J. Hendler and O. Lassila, The Semantic Web, Scientific American, May 2001.

3 A. Berson and S.J. Smith, Data warehousing, data mining, and OLAP, New York, McGraw-Hill, 1997.

4 Peter Pi-Shan Chen: *The Entity-Relationship Model. Toward a Unified View of Data*. In: ACM Transactions on Database Systems 1/1/1976 ACM-Press ISSN 0362-5915, S. 9-36

5 E.F. Codd, A relational model of data for large shared data banks, Communications of the ACM **13**(6), 377-387, 1970.

6 R. Fagin, A.O. Mendelzon, J.D. Ullman, A Simplified Universal Relation Assumption and Its Properties. ACM Trans. Database Syst. **7**(3), 343-360, 1982.

7 D. Fensel, Ontologies: a silver bullet for knowledge management and electronic commerce. Springer, 2004.

8 B. Ganter and R. Wille, Formal Concept Analysis: Mathematical Foundations, Springer, 1999.

9 H. Garcia-Molina, J. Ullman and J. Widom, Database Systems: the Complete Book, Prentice Hall, 2003.

10 P.M.D. Gray, P.J.H. King and L. Kerschberg (eds.), Functional Approach to Intelligent Information Systems (Special issue). Journal of Intelligent Information Systems **12**, 107–111, 1999.

11 P.M.D. Gray, L. Kerschberg, P. King, and A. Poulovassilis (eds.), The Functional Approach to Data Management: Modeling, Analyzing, and Integrating Heterogeneous Data, Heidelberg, Germany, Springer, 2004.

12 M. Gyssens and L.V.S. Lakshmanan, A foundation for multi-dimensional databases, Proc. 23th VLDB '97, Athens, Creece, 106-115, 1997.

13 T.A. Halpin, Entity Relationship modeling from ORM perspective. Journal of Conceptual Modeling (www.inconcept.com/jcm), **11**, 1999.

14 W. Kent, Consequences of assuming a universal relation, ACM Trans. Database Syst., **6**(4), 539-556, 1981.

15 C. Li and X.S. Wang, A data model for supporting on-line analytical processing, Proc. Conference on Information and Knowledge Management, Baltimore, MD, 81-88, 1996.

16 D. Maier, J. D. Ullman, and M. Y. Vardi, On the foundation of the universal relation model, ACM Trans. on Database System (TODS), **9**(2), 283-308, 1984.

17 T.B. Nguyen, A.M. Tjoa and R.R. Wagner, An Object Oriented Multidimensional Data Model for OLAP, Proc. 1st International Conference on Web-Age Information Management (WAIM'00), Shanghai, China, June 2000. LNCS, Springer, 2000.

18 A. Savinov, Analytical space for data representation and interactive analysis, Computer Science Journal of Moldova, **10**(1), 59-80, 2002.

19 A. Savinov, Principles of the Concept-Oriented Data Model, Technical Report, Institute of Mathematics and Informatics, http://conceptoriented.com/savinov/publicat/imi-report'04.pdf 2004.

20 A. Savinov, Logical Navigation in the Concept-Oriented Data Model. Journal of Conceptual Modeling, http://www.inconcept.com/jcm, 2005.

21 D.W. Shipman, The Functional Data Model and the Data Language DAPLEX. ACM Transactions on Database Systems, **6**(1), 140–173, 1981.

22 R. Torlone, Conceptual multidimensional models, In: Multidimensional Databases: Problems and Solutions, Idea Group, 69–90, 2003.