Proc. 21st ACM Symposium on Applied Computing (SAC 2006),
April 23-27, 2006, Dijon, France, 482-486

Grouping and Aggregation in the Concept-Oriented Data Model

Alexandr Savinov
Fraunhofer AIS
Schloss Birlinghoven
53757 Sankt Augustin, Germany
savinov@conceptoriented.com

ABSTRACT

In the paper we describe the problem of grouping and aggregation in the concept-oriented data model. The model is based on ordering its elements within a hierarchical multidimensional space. This order is then used to define all its main properties and mechanisms. In particular, it is assumed that elements positioned higher are interpreted as groups for their lower level elements. Two operations of projection and de-projection are defined for one-dimensional and multidimensional cases. It is demonstrated how these operations can be used for multidimensional analysis.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design – *Data models;*
H.2.3 [**Database Management**]: Languages – *Query languages;*

General Terms

Algorithms, Management, Theory.

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], entity and relationships in the ERM [4], facts and object roles in the ORM [12], subject-predicate-object triples in the RDF [2] and many others. In this paper we describe a new approach to data modelling proposed in [16,17,18] and called the concept-oriented data model (COM).

The COM belongs to a set of approaches based on using *dimension* (degree of freedom) as the main construct for data modelling. This direction has been developed in the area of multidimensional databases [1,11,14] and online analytical processing (OLAP) [3]. An important assumption underlying the COM 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) [6,13,15]. This assumption allows us to derive properties of elements from the properties of the whole model as well as automate many operations such logical navigation and query construction. Another important assumption is that the COM is based on ordering its elements which is analogous to concept-lattices, formal concept analysis (FCA) [8] and ontologies [7]. This means that elements do not possess any information except for their position among other elements. This relative position is precisely that determines the semantic properties of elements. In great extent everything in the COM is about order of elements and duality. 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) [9,10,19].

In section 2 we define the model and section 3 describes what is meant by dimensionality. In section 4 two operations of (one-dimensional) projection and de-projection are described while section 5 is devoted to multidimensional analysis.

2. MODEL DEFINITION

In the concept-oriented paradigm (not only data model) we assume that all things have two sides which are called *physical* and *logical*. In particular, in data modelling such a separation is used to distinguish *identity modelling* (how elements are represented and accessed) from *entity modelling* (how elements are characterized by other elements. Formally, two types of element composition are distinguished: *collection* and *combination*. An element is then represented as consisting of a collection of other elements and a combination of other elements from this model: $E = \{a,b,\ldots\}\langle c,d,\ldots\rangle$. Here $\{\}$ denotes a collection and $\langle\rangle$ denotes a combination. A collection can be viewed as a normal set with elements connected via logical OR and identified by means of *references* (for example, tables with rows). A combination is analogous to fields of an object or columns of a table which are identified by *positions* (offsets) and connected via logical AND.

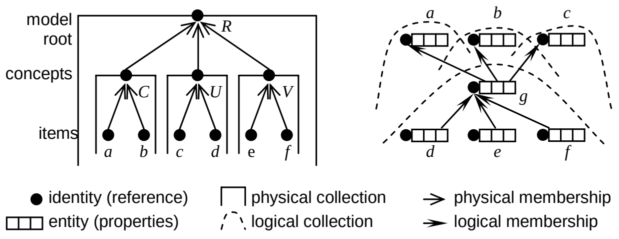

Figure 1. Physical (left) and logical (right) structures.

Physical structure (Fig. 1 left) has a hierarchical form where any element has a single parent which provides the means of representation and access (RA) for its members. For example, tables are physically living in a database while records are living in tables. Physical structure can be easily produced by removing all properties (fields, columns etc.) from elements: $E = \{a, b, \ldots\}\langle\rangle$. Physical inclusion can be thought of as inclusion by value, i.e., we assume that any element has some physical position by value within some other element (physical container). In the context of this paper it is important to understand that physical inclusion can be used for grouping. For example, if $E = \{a, b, \ldots\}\langle\rangle$ then elements *a* and *b* physically belong to one group *E*. However, one property of physical structure is that it is immutable because elements cannot change their parent group.

Logical structure (Fig. 1 right) appears when elements get some properties. The combinational part is not empty and elements are referencing other elements of the model. Such a referencing is a method of mutual characterization. For example, element $E = \{\ldots\}\langle c, d, \ldots\rangle$ is referencing elements *c* and *d* and we say that *E* is characterized by values *c* and *d*. Logical structure provides the second method for grouping using the following principle: *an element belongs to all elements it combines (references)*. In other words, object properties are groups for the object they characterize. On the other hand, an object is a group for all objects that reference it. In contrast to physical grouping, logical grouping has two advantages: an element may belong to many groups simultaneously and this structure is not constant so that element can change its parent groups by changing its field values.

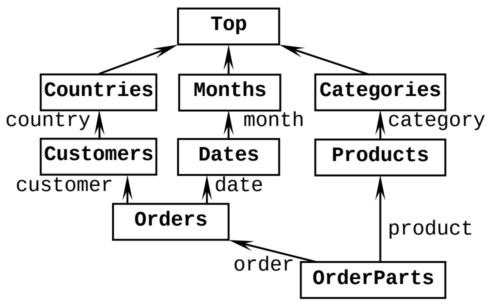

Figure 2. An example of the model syntax.

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 the 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*), and (ii) a physical collection of *data items* (or concept instances), $C = \langle C_1, C_2, \ldots, C_n\rangle\{i_1, i_2, \ldots\} \in R$;

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

[Special elements] If a concept does not have a superconcept then it is assumed to be one common *top concept*; with direct subconcepts called *primitive concepts*, 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.

Fig. 2 is an example of a logical concept structure (the root element and items are not shown) where each concept is a combination of its superconcepts. For example, concept `Orders` is a combination of superconcepts `Customers` and `Dates`. It has one subconcept `OrderParts` which is also the bottom concept of the model. In this case an order item (instance of `Orders`) is logically a member of one customer item and one date item. At the same time one order item logically includes a number of order parts which are its subitems. One order part is logically included into one order and one product.

3. MODEL DIMENSIONALITY

A named link from subconcept to a superconcept is referred to as *dimension*. A dimension can be also viewed as a unique position of a superconcept in the definition of subconcept: $C = \langle x_1 : C_1, x_2 : C_2, \ldots, x_n : C_n\rangle$. Superconcepts $C_1, C_2, \ldots, C_n$ are called *domains* or ranges for dimensions $x_1, x_2, \ldots, x_n$, $C_j = \mathrm{Dom}(x_j)$. The model syntactic structure can be represented by a directed acyclic graph where nodes are concepts and edges are dimensions (Fig. 2 and 3). A dimension $x = x^1.x^2.\cdots.x^k$ of rank *k* is a sequence of *k* dimensions where each next dimension belongs to the domain of the previous one. Dimensions will be frequently prefixed by the very first concept: $C.x^1.x^2.\cdots.x^k$. Each dimension is represented by one path in the concept graph. The number of edges in the path is the dimension rank. A dimension with the primitive domain is referred to as *primitive dimension*. For example, `Auctions.product.category` (Fig. 3) is a primitive dimension of rank 2 from the source concept `Auctions` to its superconcept `Categories`. There may be several different paths (dimensions) between a concept and its superconcept. The number of different primitive dimensions of a concept is referred to as the *concept primitive dimensionality*. The length of the longest dimension of a concept is referred to as *concept rank*. The dimensionality and rank of the whole model are equal to that of the bottom concept. Thus any concept-oriented model is characterized by two parameters: (i) model rank describing its hierarchy (depth), and (ii) model dimensionality describing its multidimensionality (width). The models in Fig. 2 and 3 are 3-dimensional and 6-dimensional, respectively, however, both have rank 3.

Inverse dimension is a dimension with the opposite direction, i.e., where a dimension starts the inverse dimension has its domain, and where a dimension ends the inverse dimension has its start. In concept graph inverse dimension is a path from superconcept to some its subconcept. Inverse dimensions do not have their own identifiers. Instead, we apply an *operator of inversion* 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*. An important thing is that 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, an order (Fig. 2) is characterized by two dimensions (`date` and `customer`), as well as one inverse dimension `{OrderParts.order}`. Such a duality is one of the distinguishing features of the COM because it allows us to characterize items as a combination of (more general) superitems and a collection of (more specific) subitems.

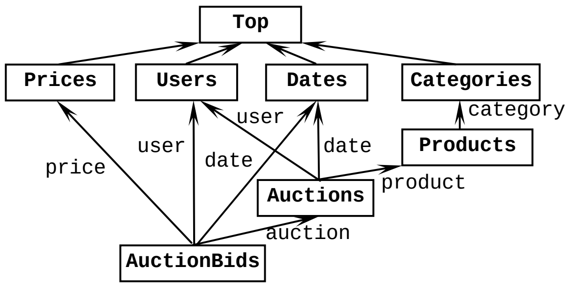

Figure 3. An example of logical structure of dimensions.

4. PROJECTION AND DE-PROJECTION

Given an item or a set of items it is possible to get related items by specifying some path in the concept graph. Informally, if we move up along a dimension to superitems then it is thought of as an operation of projection. If we move down along an inverse dimension to subitems then it is de-projection.

If d is a dimension of C with the domain in superconcept $U = \text{Dom}(d)$ then operation $I \to d$, $I \subseteq C$, is referred to as *projection* of items from I along dimension d. It returns a set of superitems referenced by items from I: $I \to d = \{u \in U \mid i \to d = u, i \in I \subseteq C\}$. Each item from U can be included into the result collection (projection) only one time. If we need to include superitems as many times as they are referenced then dot operation has to be used instead of arrow, i.e., $I.x$ includes *all* referenced superitems from U even if they occur more than once. The operation of projection (arrow) can be applied consecutively. For example, if A is a collection of today's auctions then `A->product->category` will return a set of today's categories while `A.product.category` will return categories for all auctions in A (as many categories as we have auctions). If `P` is a subset of order parts then projection `P->order->customer->country` is a set of countries.

If $\{d\}$ is an inverse dimension of C with the domain in subconcept $S = \text{Dom}(\{d\})$ then *de-projection* of I to S along $\{d\}$ consists of all subitems that reference items from I via dimension d: $I \to \{d\} = \{s \in S \mid s \to d = i, i \in I \subseteq C\}$. For example, if `C` is a set of auction product categories then `C->{Auctions ->product->category}` is a set of all auctions with the products having these categories. Given month `m` we can get all related orders by de-projecting it onto concept `Orders`: `m->{Orders->date->month}`.

Dimension d specifying a path from a subconcept to some its superconcept is referred to as *bounding* dimension. *Access path* is a 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. An 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 returned by de-projection operation by providing a condition that all items from the domain subconcept have to satisfy:

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

Here d is a bounding dimension from subconcept S to the source collection I; s is an 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). For example, access path

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

will return all *today's* auctions for the 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 or more items 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 of the concept.) Derived properties can use other properties:

```
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 ANALYSIS

The mechanism of access path is based on the assumption that there is only one bounding dimension between source superconcept and target subconcept. If target subitems can be bound to source superitems along several dimensions simultaneously then we get the case of multidimensional grouping and aggregation (Fig. 4). For example, order parts can be grouped using two dimensions `country` and `category`. One group is a combination of one county item and one category item and consists of a collection of associated order parts (Fig. 2).

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, $\text{Dom}(d_j) = C$, $j = 1,2,\ldots,n$, 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: (Fig. 4)

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

Grouping and aggregation by means of the operation of multidimensional de-projection can be used for online analytical processing (OLAP). This approach consists in choosing some target subconcept the items of which we want to group and aggregate. Then it is necessary to specify several dimension paths from this concept along which we want to analyze data. The level of details can be varied by choosing source superconcepts along each of these dimension paths. The source multidimensional concept is the Cartesian product of all the domain concepts chosen along dimension paths. Finally, the source items are de-projected onto the target concept by producing groups of items that can be aggregated.

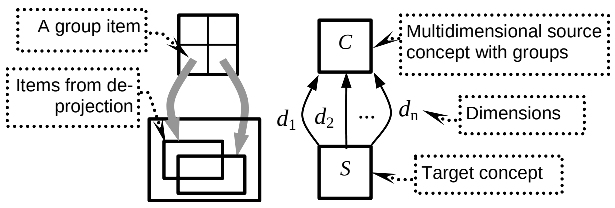

Figure 4. Multidimensional de-projection.

Let us assume that S is the target concept with items to be grouped and aggregated. In concept S we select a number of *dimension paths* $p_1, p_2, \dots, p_n$ which will be used for analysis. A *level* $L = \langle l_1, l_2, \dots, l_n \rangle$ is a set of integers specifying ranks along these dimension paths. Then $d_1, d_2, \dots, d_n$ are dimensions of concept S with ranks $l_1, l_2, \dots, l_n$ and domains $D_1, D_2, \dots, D_n$, $D_j = \mathrm{Dom}(d_j)$, $j = 1, 2, \dots, n$. For example, `order.customer.country` and `product.category` are two dimension paths of the target concept `OrderParts` (Fig. 2). We might choose level $L = \langle 2,1 \rangle$ for initial analysis, which produces dimensions `OrderParts.order.customer` and `OrderParts.product` with domains `Customers` and `Products`, respectively.

For each level the universe of discourse (called multidimensional cube in OLAP) is defined as a set of all possible items produced from the corresponding domains:

$$\Omega_L = D_1 \times D_2 \times \ldots \times D_n = \{\omega = \langle \omega_1, \omega_2, \dots, \omega_n \rangle \mid \omega_j \in D_j\}$$

Multidimensional projection of a set of items $I \subseteq S$ to level L is a set of points from the cube Ω_L referenced by items from I via dimensions $d_1, d_2, \dots, d_n$ of level L:

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

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

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

In the concept-oriented query language the source domains are listed after the keyword **`FORALL`**, for example:

```
N = FORALL(c Customers, p Products) { ... }
```

It can be thought of as iteration over all combinations of customers and products although the order is not dictated and the procedure can be optimized. A query returns a new collection of items which can be stored in a variable. Items returned from such a query are specified via keyword **`RETURN`** normally using some conditions specified via keyword **`IF`**, for example:

```
FORALL(c Customers, p Products) {
  IF( count( c->{Orders->customer} ) > 5
     && p.category='cars') RETURN(c, p);
}
```

This query selects only items from the source 2-dimensional space if the customer has more than 5 orders and the product is a car. Note that in order to compute the number of orders for the current customer we de-project it to the subconcept `Orders` and then apply aggregation function `count` to a set of orders.

Suppose that S=`OrderParts` is the target concept which is projected to the source concept along two dimension paths p_1=`OrderParts.order.customer.country` (3 levels) and p_2=`OrderParts.product.category` (2 leves). If it is necessary to analyze dependencies between customers and products then each their combination `<c,p>` (or keyword **`this`**) is de-projected into concept `OrderParts` along two dimensions and the result collection stored in the local variable `tmp`:

```
FORALL(c Customers, p Products) {
  IF( count(c->{Orders->customer}) > 5) ) {
    tmp=<c,p>->{ OrderParts.order.customer,
                 OrderParts.product };
    RETURN <c, p, avg(tmp.price) >;
  }
}
```

The intermediate local variable `tmp` stores a collection of order parts associated with the current customer and the current product (and element of 2-dimensional cube). For each such combination the query returns an average price in addition to the customer and product.

An operation of increasing rank l_j (one constituent of level) of one dimension d_j is referred to as *roll up*. An 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 direct superconcepts of S as domains.

For example, we can get more general distribution of average price along countries (instead of individual customers) and categories (instead of individual products) by rolling up (increasing dimensions rank) along both dimensions:

```
FORALL(c Countries, p Categories) {
  tmp =
  <c,p>->{OrderParts.order.customer.country,
          OrderParts.product.category };
  IF( count(tmp) > 5 )
    RETURN <c, p, avg(tmp.price) >;
}
```

This query computes 2-dimensional de-aggregation for each source point `<c,p>` and then returns average price for only those having more than 5 order parts.

An interesting feature of the COM is that in many cases the access path can be computed automatically. In order to retrieve the necessary data it is enough only to impose constraints and to indicate the target concept. The idea is that the constraints are propagated automatically downward in the concept graph till the very bottom. After that this (constrained) information from the most specific level is used to retreive items from the target concept. For example (Fig. 2), let us assume that we want to get all categories for some country and month. Instead of specify a concrete query we can simply impose these constraints while the model will do all the rest itself: This could be written as the following query:

```
Months = {m Months | m == 'June' }
Countries = {c Countries | c == 'Germany' }
N = FORALL(c Categories) {
  tmp = c->{OrderParts->product->category}
         ->order
  RETURN <c, sum( tmp.price ) >;
}
```

In the first two lines we simply redefine the two existing concepts by restricting their items. (These constraints are visible only from the current and all internal contexts but not from outside.) They are propagated downward by de-projecting till the concept `OrderParts`, which will contain only order parts for the specified country and month. The restricted order parts are then propagated upward to the target concept `Categories` by means of projection. This means that only categories for the selected order parts will satisfy the initial constraints. The query returns a category item as well as the total price of its orders. In order to compute this price we de-project the current category to `OrderParts` and then project it to `Orders`. A collection of all orders related to the current category is stored in a local variable and then the order price is summed up in return statement.

In more complex cases the constraint propagation path can be ambiguous and needs to be specified explicitly, say, by indicating some intermediate concept. For example, if we want to get all auction product categories related to some user then there exist two paths for constraint propagation: through the concept `AuctionBids` and through the concept `Auctions`.

6. CONCLUSIONS

In comparison to existing data models the proposed approach has a number of advantages and distinguishing features. It is an integrated full-featured model rather than a specific mechanism or auxiliary technology. This means that all necessary mechanisms exist within this very model. Another distinguishing feature of the concept-oriented model is its simplicity. By using only a few main constructs it is possible to implement all the most important data modelling mechanisms and manipulation techniques. Indeed, ordered sets of concepts and items as well as dimensions is enough to derive such data modelling constructs as multi-valued attributes, multidimensional cubes, measures, joins etc. It can be said that everything in the concept-oriented model is about order and duality because these two phenomena are of crucial importance for defining its properties. In particular, the relative order of an element defines its semantics. Data access and analysis are based on operations of projection and de-projection, which also reflect the order of elements and duality. As a result this model solves the problem of logical navigation by avoiding complex joins. The order of elements and these two operations also allow us to integrate the mechanism of grouping and aggregation into the model as its natural part rather than an additional (analytical) layer. All items are naturally grouped in the model while cubes, dimensions, measures are roles assigned to elements of the model for the purpose of concrete analysis task.

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

[10] 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.

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

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

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

[14] 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.

[15] 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.

[16] A. Savinov, Principles of the Concept-Oriented Data Model, Technical Report, Institute of Mathematics and Informatics.

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

[18] A. Savinov, Hierarchical Multidimensional Modelling in the Concept-Oriented Data Model, 3rd International Conference on Concept Lattices and Their Applications (CLA'05), Olomouc, Czech Republic, September 7-9, 2005, 123-134.

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