$\mathcal{B}ig\mathcal{C}ube$: A Metamodel for Managing Multidimensional Data

Ganesh Viswanathan & Markus Schneider
University of Florida, Gainesville, FL 32611, USA
{gv1, mschneid}@cise.ufl.edu

Abstract

New emerging scientific applications in geosciences, sensor and spatio-temporal domains require adaptive analysis frameworks that can handle large datasets with multiple dimensions. However, existing conceptual design strategies for multidimensional data using the data warehousing framework are not suitable for users, since they involve complex extensions of traditional database design frameworks like E/R and UML diagrams, or the relational star and snowflake schema. There is a lack of a generalized model that provides a user-centric design approach to let analysts abstractly design and query multidimensional data. In this paper, we propose a solution to this problem by presenting a generic metamodel *for multidimensional data that keeps the* user as the focal point *and achieves a* clear abstraction *for all users. Our model called the $\mathcal{B}ig\mathcal{C}ube$ provides users with a set of multidimensional abstract data types for data modeling and includes aggregate operations for performing analysis. Overall, we provide a formal and adaptable,* user-centric *data warehouse modeling approach to handle large multidimensional datasets.*

1 Introduction

For more than a decade, data warehouses have been at the forefront of information technology applications as a way for organizations to effectively use information for business planning and decision making. They contain large repositories of analytical and subject-oriented data integrated from several heterogeneous sources over a historical timeline. The technique of performing complex analysis over the information stored in the data warehouse is popularly called Online Analytical Processing (OLAP).

There have been several proposals of a conceptual design model for multidimensional data based on a data warehousing framework. These lie in the direction of *Entity Relationship (E/R) models*, *UML extensions* and other *ad-hoc systems* like *model driven architecture* based approaches. However, they are not extensively used for data warehouse design since most of them are *system-centric*, *unintuitive for the non-database user* or have *overly convoluted terminology.* In practice, most data warehousing models directly expose the logical design schemata to all users. This logical design often consists of developing relational tables through star, fact constellation or snowflake schema (called *Relational OLAP* or *ROLAP*) [5], or multidimensional cubes (called *Multidimensional OLAP* or *MOLAP*). As a result, the user requirements are often neglected and the approach becomes totally system-centric, instead of being focused on the user's need to understand and analyze the complex datasets. Moreover, such direct implementation strategies can lead to an inefficient translation of the data analysis requirements into the design schema. A better approach for multidimensional data modeling is a generic metamodel at an abstract level that captures the needs of data analysts and does not expose the logical implementation aspects and system features. Thus the user gets a platform for analyzing data without the need to handle implementation details such as the ROLAP or MOLAP constraints on the schema.

In this paper, we present a novel approach for conceptually analyzing multidimensional datasets using the data warehousing framework. Our model generalizes over existing design techniques and this metamodel is called as $\mathcal{B}ig\mathcal{C}ube$.As the name suggests, we use the cube metaphor to provide a complete set of abstract multidimensional data types to model complex structured data. This cube view corresponds well with the cognitive understanding of multidimensional data for the analysts. We also make a clear distinction in terminology from other logical design models to emphasize that $\mathcal{B}ig\mathcal{C}ube$ lies at the conceptual design level. Our model natively uses *abstract data types* for the cube components and thus it can be extended to support complex data such as geo-spatial and genomic data and to develop multiple layers of queries on them. This model can serve as a basis for *spatial* and next-gen *moving object data warehouses* capable of handling OLAP queries on complex multidimensional data.

The rest of this paper is organized as follows. In Section 2, we study existing conceptual modeling

approaches for multidimensional data and motivate the need for a new metamodel. An example case study is provided in Section 3 and is used to illustrate concepts in the rest of the paper. Section 4 formally introduces the *BigCube* model and its data types, and Section 5 specifies analysis operations. Section 6 concludes the paper and mentions future work.

2 Related Work

Existing conceptual modeling approaches for multidimensional data can be broadly classified into *Extensions of Entity Relationship (E/R)* models, *Extensions of Unified Modeling Language (UML)*, and *Ad-hoc design* models. We only provide a brief outline of these models here due to space limitations.

Extensions of E/R diagrammatic approaches include the *Generalising Conceptual Multidimensional Data Model* [2], the Multidimensional E/R or *ME/R diagram* model [9] and the *MultiDimER* model [7]. These provide complex extensions to handle aggregations and dimension relationships. However, the complexity of terminology and syntax renders them inapt for user-centric applications. There is also a lack of support for specifying multiple levels of aggregations.

Among the UML approaches, [1] and [6] present UML extensions for modeling multidimensional data. [1] presents YAM^2 as an extension of UML which emphasizes on part-whole relationships for aggregation, uses state diagrams to represent the system behavior and defines basic OLAP operations. [6] uses an *UML profile* defined by a set of stereotypes, constraints and tagged values to represent the main properties at the conceptual level. However, these models only present basic UML classes and lack a generic metamodeling strategy.

Among the ad-hoc design approaches, the Dimensional Fact Model (DFM) is constructed in [3] from an operational E/R schema based on requirement analysis. DFM is based on the relational star schema [5] and does not support generalization/specialization hierarchies and many-to-many relationships. [10] presents a *model driven architecture* (*MDA*) for producing candidate multidimensional schema from operational E/R schema based on requirement analysis. [4] presents another ad-hoc modeling approach based on an extension of the relational database normal forms. These are good examples of multidimensional modeling approaches but from a *system design* viewpoint. Pedersen et.al. present eleven requirements for a multidimensional model in [8] along with an extended conceptual multidimensional model which helps to handle several different types of hierarchies and aggregation semantics. However, the model lacks extensibility and does not support new operations and new types of complex data.

In our approach, the *BigCube* metamodel helps to achieve clear abstract modeling by using the multidimensional cube view and abstract data types. This helps users to easily model and develop analysis systems using the data warehousing framework.

3 Case Study: Product Management

Let us consider an example *product management* dataset from a *supplier* for illustrating the features of the conceptual data warehouse model. The dataset stores *inventory*, *sales orders* and *invoice* information for the *sales* of *products* to *customers*. Products are categorized by their *name* and *brand*. Each product also has its *type* and *color* information. The *cost price* for each product is maintained by the supplier, along with its available *quantity* and the *manufacturing location*, as part of the *inventory*. The *invoice* issued by the seller contains the *selling price* and *quantity of products ordered* by the customer. Once an order is placed, the *order commit time* is recorded using *day*, *month*, *week* and *year* fields. The order *location* is also recorded for shipping purposes, and includes *city*, *county*, *state*, *zone* and *country* information. Further, the supplier also records the *profit* resulting from the *sales* of a certain quantity of products to track their overall sales revenue.

We will now briefly illustrate how to perform multidimensional modeling of this dataset using the *BigCube* approach. The formalization of the *BigCube* data types and its operations are provided in the next section. From the example, we gather the *base data values* for the *BigCube* as *Gainesville* (for *City*), *2010* (for *year*), *100* (for *sales quantity*), etc. We then construct *categories* of values from the source data. Examples of these include *year, month, week* and *day* (for *time*). Similarly for *location*, we can construct categories such as *city, county, state, zone* and *country*. Internally these are represented as multiset data types (as explained in the Section 4). Next, we build *hierarchies* over the categories of data. By definition each of the categories constituting the hierarchy should be at a unique level. Let us construct two hierarchies for *time* as: *day, month, year* and *day, week, year*. Similarly, from the *location*, we can construct two hierarchies as: *city, state, country* and *city, county, zone, country*. At the next level, we are now ready to build *perspectives* for analysis. This is rather clear from the source data. For e.g., we can construct *time* and *location* perspectives using the two hierarchies we built at the previous step. Similarly, *subjects* such as *invoice, sales quantity, profit* and *inventory* can be constructed from their constituent hierarchies. Finally, we build the *BigCube* as the union of cells,

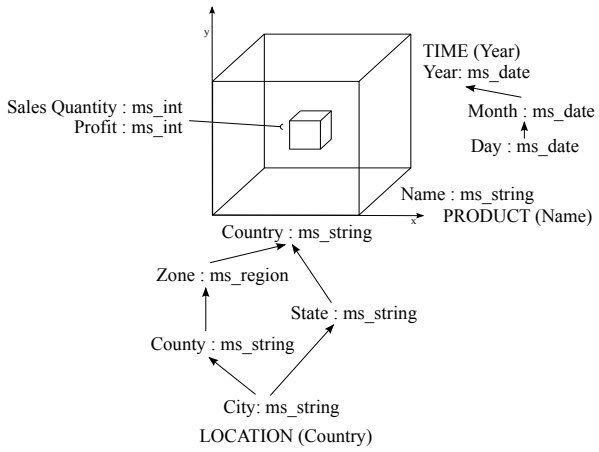

Figure 1: *Product Management* $\mathcal{BigCube}$ structure showing three perspectives: Time, Product and Location that define two subjects of interest: Sales Quantity and Profit.

each being defined by the combination of available perspectives. The subjects of the $\mathcal{BigCube}$ are placed in the *cells* of the $\mathcal{BigCube}$. In this manner the entire *product management* data can be modeled using a cube view. The resulting $\mathcal{BigCube}$ structure is illustrated in Figure 1.

4 The BigCube Model

In this section, we describe the $\mathcal{BigCube}$ metamodel for user-centric modeling of multidimensional data using a data warehouse framework. First, we introduce specialized *multidimensional abstract data types* over five levels, to develop and design the overall framework for the data warehouse. These multidimensional types are collectively called $\mathcal{BigCube}$ *Data Types* (*BDTs*) as shown in Table 1. Each level has three basic structural components, namely, *kinds* (containing a set of types), the *data types* and *elements* of the data types (*instances*). The $\mathcal{BigCube}$ structure is illustrated in Figure 1 and its instantiation is shown in Figure 2. At each level of BDTs, we introduce an example to illustrate the need for the structural component and then define it formally. These are essential user concepts to model multidimensional data.

To provide a consistent and independent user view for the data warehouse using $\mathcal{BigCube}$, we refrain from using common relational data warehouse terminology like *fact tables* and *dimension tables*. Instead we provide a new, formal definition for the basic components of the $\mathcal{BigCube}$.

Example: Consider the Product Management case study presented in Section 3. The basic data that needs to be stored (and later analyzed) in the data

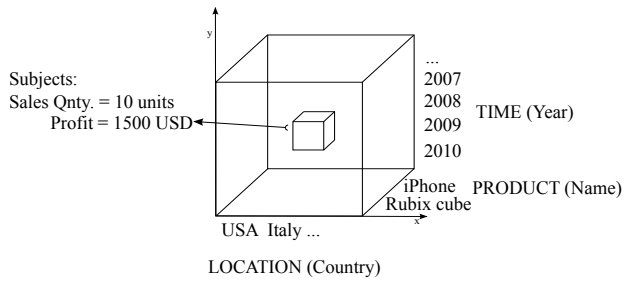

Figure 2: An instance of the *Product Management* $\mathcal{BigCube}$ with the three perspectives defining the two subjects of interest, namely Sales Quantity and Profit.

warehouse are values such as *1500* (of type *int*) for the profit in USD and "*Gainesville*" (of type *string*) for the City name. These are called the *base data values* of the dataset and are defined in Definition 1. The *base data type* for each value is indicated within parenthesis.

Definition 1 Base data value, base data types and BASE. *A* base data value v *is defined as any indivisible, low-level value in the data warehouse. A* base data type *is the data type of a base data value and includes all alphanumeric types (such as* int*,* real*,* char *confined to finite domains in* $\mathbb{Z}$*,* $\mathbb{R}$*,* $\mathbb{A}$ *respectively), all temporal types (such as* date*,* time*,* interval*) and all geo-spatial data types (such as* point*,* line*,* region *for complex spatial objects, confined to finite domain in* $\mathbb{R}^2$*). The set of all* base data types *is defined as a kind* BASE*.*

$$\begin{aligned} BASE &= ALPHA \cup NUM \cup TIME \cup GEO \\ ALPHA &= \{char, string, \ldots\} \\ NUM &= \{integer, real, \ldots\} \\ TIME &= \{date, time, interval, \ldots\} \\ GEO &= \{point, line, region, \ldots\} \end{aligned}$$

According to their functionality, base data values ($v \in \alpha \in BASE$) can be either *members* when used for analysis along data dimensions, or *measures* when used to quantify factual data.

The second step in modeling multidimensional data for analysis involves groupings or categorization of the base data values. However, before we can define a subjective grouping of base values as a *category*, we first construct a *multiset of valid types* from the available base data types.

The powerset operator yields the set of all subsets including the empty set and the set itself, i.e., $\mathcal{P}(A) = \{B \mid B \subseteq A\}$. Using powersets, we can now define the Multiset Type Constructor as shown in Definition 2.

Definition 2 Multiset type constructor (μ). μ *is a multiset type constructor that creates multiset types from the base data types as follows:*

$$\mu(\alpha) \quad = \mathcal{P}(\alpha \times \mathbb{N}), \alpha \in BASE$$

This means that, for example, $\mu(string) = \mathcal{P}(string \times \mathbb{N})$ holds. This *multiset string type* is the set of all multisets based on the data type *string*. The *number of occurrences* of each of the types is shown by the second element of the pair. For e.g.,

City = {("Gainesville", 3), ("Orlando", 2), ("Miami", 1)} $\in \mu(string)$.

We can now define the *multiset string type* as $ms\_string := \mu(string)$. Therefore, the City is a category of city measure values of the type ms_string and can be represented as $City : ms\_string$. Similarly, we can also define corresponding multiset types for all available base data types.

Categorization Relationships and Containment Relationships. Real-world data always has some form of symmetricity and assymetricity associated with its base data values. For e.g., all persons working in a University can be employees (symmetric relationship). Employees could be students, faculty or administrators (asymmetric relationship).

A *categorization relationship* defines groupings of base data values based on the similarity of data. The semantics of *categorization relationships* are defined in one of three ways: *arbitrary* (for e.g., split 100 base values into 10 categories equally according to some criteria), *user-defined* (for e.g., *Gainesville, Chapel Hill* and *Madison* can be categorized as *College Towns*), or *according to real-world behavior* (such as spatial grouping, for e.g., *New Delhi, Berlin* and *Miami* can be categorized as *Cities*).

A *containment relationship* defines the ordering of the levels in the hierarchy and is represented by the symbol $\prec$. This ordering can be defined by: (i) any arbitrary containment, for e.g., fifteen base data values can be ordered into a four- level hierarchy using the structure of a balanced binary tree, (ii) user-defined containment : for e.g., products can be ordered into a hierarchy based on their selling price, (iii) according to real-world behavior: these reflect the fact that a higher level element *is a context of* the elements of the lower level, it *offers constraint to* the lower level values, it *evolves at a lower frequency than* the lower level elements, or that *it contains the* lower level elements.

Example: In our case study, two examples of categories are City={("Gainesville", 3), ("Orlando", 2), ("Miami", 1)} and Profit={ ("1500, 3"), ("10000, 2"), ("45000, 1")} for the profit in USD. These are of the multiset types *ms_string* and *ms_int* respectively.

Definition 3 Category, Category Type and CATEGORY. *A* category *of elements $c \in \mu(\alpha)$, is a grouping of base data values such that a valid categorization relationship exists among the set of elements. A* category type *$\mu(\alpha)$, provides the multiset data types for each category. The set of all available category types is defined as a kind* CATEGORY.

For n base data types, we have n corresponding multisets in CATEGORY. In our example, the categories *cities* and *profit* are multisets and there is *no order* among the constituent pairs of elements. We only store city names along with the number of occurrences of each city in the cube. The same applies for profit and all other categories of members/ measures in the cube. Categories help us to construct hierarchies of data as shown by the next level of $\mathcal{B}ig\mathcal{C}ube$ structure types in Definition 4.

Example: In our sample dataset, we notice that there are data hierarchies such as $City \prec State \prec Country$ for location, and $Day \prec Month \prec Year$ for time. These are defined as *hierarchies* in Definition 4.

Definition 4 Hierarchy, hierarchy type and HIERARCHY. *Given a dataset with n categories of base data values, a hierarchy is defined as a total ordering of the categories ($c_1 \ldots c_n$), such that a full logical containment relationship ($\prec$) exists between the constituent base data values. The* hierarchy type *represents the type of hierarchy and the kind HIERARCHY is defined as a set of all hierarchy types in the data warehouse.*

$$
\begin{aligned}
&Hierarchy = \langle c_1, c_2, \ldots, c_n \rangle \text{ such that} \\
&\quad (i)\, \forall 1 \leq i \leq n : c_i \in \alpha \in CATEGORY \\
&\quad (ii)\, c_0 = \{(\bot, 1)\}, c_{n+1} = \{(\top, 1)\} \\
&\quad (iii)\, \forall 1 \leq i \leq n : \prec_i \subseteq c_i \times c_{i+1}
\end{aligned}
$$

With n categories, each hierarchy contains upto 2^n *levels* in total linear order. Each logical containment relation between two categories is called a *path* in the hierarchy. Each hierarchy consists of two special levels called Apex or $\top$ and Base or $\bot$ that denote the topmost level (smallest set of instances) and the lowermost level (largest set of instances) respectively.

Thus at the instance level, specific sets of measures always exist at unique levels in the hierarchy and are comparable over the structure of the ordering. The use of unique $\bot$ and $\top$ levels helps to uniquely identify each (aggregation valid) hierarchy even after they are combined in later levels to form a lattice. It also helps to keep the hierarchies balanced (onto). The $\top$ level always points to the root of a hierarchy while the $\bot$ points to the leaf nodes.

By making use of hierarchies we can construct the next tier of $\mathcal{B}ig\mathcal{C}ube$ structure types. This next level helps to create and distinguish the *role* of the data dimension for analysis. Perspective types help to create data dimensions for analysis, while subject types help to create the measurements of data analysis.

Table 1: Five levels of $\mathcal{BigCube}$ data types

Kinds:	*Types*:	*Elements*:
BASE	base data type	base data value
CATEGORY	category type	category
HIERARCHY	hierarchy type	hierarchy
PERSPECTIVE	perspective type	perspective
SUBJECT	subject type	subject
$\mathcal{BigCube}$		

Example: In our sample dataset, we notice that there are three data dimensions, namely *Location*, *Time* and *Product*, each of which can be used for independently analyzing the dataset. Each of these is called a *perspective* in the $\mathcal{BigCube}$ model and is defined in Definition 5.

Definition 5 Perspective, perspective type and PERSPECTIVE. *A* perspective *P is a special union of one or more hierarchies (over members) that creates a* strict partial ordering *between their categories, such that meaningful querying and analysis of the multidimensional data is possible. Perspective types are defined over the constituent hierarchy types. The* PERSPECTIVE *is defined as a kind that contains a multiset of perspectives types.*

The perspective creates a special union of hierarchies such that a lattice structure is created in which categories from different hierarchies are *merged* according to user-defined semantics. While total order is maintained in the hierarchies, the perspective creates a strict partially ordered set (*poset*) of categories. The strict nature of the poset implies that each category exists at a unique level in the perspective, which helps to define and valid aggregation operations easily.

Perspectives have *hierarchies* built on member (or measure) categories along with an *Apex* $\top$ to denote the highest aggregated level of the ordering (analogous to the ALL construct) and *Base* $\perp$ to denote the lowest aggregated level of the ordering of the categories constituting the bottom-most level of the perspective.

Example: In our case study, we notice that there are three sets of measurable facts that can serve as the basis for analysis of the entire dataset: *Sales*, *Invoice* and *Inventory*. Each of these is defined as a *subject of analysis* in Definition 6.

Definition 6 Subject, subject type and SUBJECT. *A* subject *S is a special union of hierarchies over measures that help to signify measured facts, and helps to perform analysis of the multidimensional data. Each subject is a value of the subject type, and the* SUBJECT *is defined as a kind that contains a multiset of subject types.*

Subject are similar to perspectives, except that they help to quantify facts of analysis and thus are measurements, while perspectives form data dimensions for performing analysis.

Finally, having defined these four levels of $\mathcal{BigCube}$ types, we can now construct the complete $\mathcal{BigCube}$ as shown in Definition 7.

Definition 7 $\mathcal{BigCube}$**.** *Given a multidimensional dataset, the* $\mathcal{BigCube}$ cell structure *is defined as an injective function from the n-dimensional space defined by the Cartesian product of n functionally independent perspectives P (identified by its members) to a set of r subjects (identified by its measures) S and quantifying the data for analysis as:*

$$f_{\mathcal{B}}\colon (P_1 \otimes P_2 \otimes \ldots \otimes P_n) \longrightarrow S_i \quad where\, i \in \{1, \ldots r\} \wedge (S_i, P) \in BASE$$

The complete $\mathcal{BigCube}$ structure is now defined as a union of all its cells, given as:

$$\mathcal{BigCube}\,(\mathcal{B}) = \bigcup_{i \in \{1,\ldots r\},\, f_{\mathcal{B}}} S_i$$

The *combination* notation $\otimes$ means that any order of the two argument types is valid but it identifies one unique *cell* in the structure of the $\mathcal{BigCube}$. The $\mathcal{BigCube}$ cell stores the subjects consisting of one or more measure values. The base cube $\perp_B$ is a special state of the $\mathcal{BigCube}$ that defines the cube view in the least aggregated form (at finest granularity). The most aggregated state of the $\mathcal{BigCube}$ is called as the *apex cube* or $\top_B$. Thus the entire multidimensional dataset can be modeled using the data warehouse framework as a *constellation* of $\mathcal{BigCubes}$. The example dataset is represented in the $\mathcal{BigCube}$ model as illustrated in Figure 1 (structure) and Figure 2 (instance). All five levels of $\mathcal{BigCube}$ *data types* thus constructed are shown in Table 1.

5 $\mathcal{BigCube}$ Analysis Operators

We now provide examples of analysis operations for querying and aggregating over the multidimensional dataset. Due to space constraints, we only describe the basic strategy for incorporating analysis operations in $\mathcal{BigCube}$. All the standard multidimensional querying operations available in a data warehouse context can be incorporated into $\mathcal{BigCube}$ by specifying transformations on the elements of $\mathcal{BigCube}$ Data Types (BDTs), as shown below.

Table 2: $\mathcal{BigCube}$ aggregation operations on BDTs

Type	$\mathcal{BigCube}$ Aggregation Operator
Additive	*Sum, Count, Max or Apex, Min or Base, Concatenate, Convex Hull, Spatial Union, Spatial Intersection*
Semi-Additive	*Average, Variance, Standard Deviation, MaxN, MinN, Centroid, Center of Gravity, Center of Mass*
Non-Additive	*Median, MostFrequent, Rank, LastNonNullValue, FirstNonNullValue, Minimum Bounding Box, Nearest Neighbor, Equi-Partition*

The *slice* operation removes one perspective and returns the resulting $\mathcal{BigCube}$; *dice* performs slice across two or more perspectives. These operations change the *state* of the $\mathcal{BigCube}$, because any change in perspectives redefines the cells (measures) in it. *Pivot* rotates the perspectives for analysis across axes and returns a $\mathcal{BigCube}$ with a different ordering of subjects. *Roll-up* performs specialization transformation over one or more constituent hierarchical levels and *drill-down* applies the generalization transformation over one or more hierarchical levels. *Drill-through* obtains the base data values with highest granularity. *Drill-across* combines several $\mathcal{BigCubes}$ in order to obtain aggregated data across the common perspectives.

Further, we can also include *additive*, *semi-additive* and *non-additive* aggregate operations for different base data types as shown in Table 3. The additive operators can be meaningfully aggregated by addition over specified perspectives. The semi-additive operators can be summarized using addition only along some perspectives, and need to be recomputed from base data for others. Non-additive operators cannot be used additively (one aggregation following another) over any of the perspectives of the $\mathcal{BigCube}$.

6 Conclusions and Future Work

Existing multidimensional modeling techniques present a system-centric and logical view that is based on relational database design tools. In this paper, we present a *generic metamodel* for conceptual data warehouse design that helps to provide users with an *abstract view* of data suitable for multidimensional modeling and analysis. The $\mathcal{BigCube}$ model is simple and concise, and provides an extensible set of abstract multidimensional data types for data warehouse modeling. The *cube view* is intuitive to the cognitive understanding of multidimensional hierarchical data for analysts. Further, we also show some analysis operations for performing summarizations on the data.

As future work, we plan to include new spatio-temporal data types and operations in the $\mathcal{BigCube}$, handle data uncertainty and imprecision, and develop transformations from $\mathcal{BigCube}$ to multiple logical models for efficient implementation.

References

[1] A. Abelló, J. Samos, and F. Saltor. YAM^2: a multidimensional conceptual model extending UML. *Information Systems*, 31(6):541–567, 2006.

[2] E. Franconi and A. Kamble. A data warehouse conceptual data model. In *Proc. of Scientific and Statistical Database Management*, pages 435–436, 2004.

[3] M. Golfarelli, D. Maio, and S. Rizzi. The Dimensional Fact Model: a Conceptual Model for Data Warehouses. *Int. Journal of Cooperative Information Systems*, 7:215–247, 1998.

[4] B. Hüsemann, J.Lechtenbörger, and G.Vossen. Conceptual Data Warehouse Design. In *Workshop on Design and Management of Data Warehouses*, pages 3–9, 2000.

[5] R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling (Second Edition)*. Wiley, April 2002.

[6] S. Luján-Mora, J. Trujillo, and I. Song. A UML profile for multidimensional modeling in data warehouses. *Data Knowledge Engineering*, 59(3):725–769, 2006.

[7] E. Malinowski and E. Zimányi. Hierarchies in a multidimensional model: from conceptual modeling to logical representation. *Data Knowledge Engineering*, 59(2):348–377, 2006.

[8] T. Pedersen, C. Jensen, and C. Dyreson. A foundation for capturing and querying complex multidimensional data. *Information Systems*, 26(5):383–423, 2001.

[9] C. Sapia, M. Blaschka, G. Höfling, and B. Dinter. Extending the E/R Model for the Multidimensional Paradigm. In *ER '98: Workshops on Data Warehousing and Data Mining*, pages 105–116. Springer-Verlag, 1999.

[10] L. Zepeda, M. Celma, and R. Zatarain. A Mixed Approach for Data Warehouse Conceptual Design with MDA. In *Int. Conf. on Computational Science and Its Applications*, pages 1204–1217, 2008.