M. S. Hacid, U. Sattler

Modeling Multidimensional Databases: A formal object-centered approach

Proc. of the Sixth European Conference on Information Systems 1998 (ECIS98).

MODELING MULTIDIMENSIONAL DATABASES: A FORMAL OBJECT-CENTERED APPROACH

Mohand-Said Hacid

LISI-INSA
20, av. A. Einstein
69621 Villeurbanne, France
msh@lisi.insa-lyon.fr

Ulrike Sattler

LuFg Theoretical Computer Science
RWTH Aachen, Ahornstrasse 55
52074 Aachen, Germany
uli@cantor.informatik.rwth-aachen.de

ABSTRACT

In this paper, we propose a formal framework based on description logics as a basis for both modeling multidimensional databases at the conceptual level and understanding related reasoning problems and services. Description logics are well-understood, object-centered formalisms which have already proved their suitability as a unifying framework for object-centered representation formalisms. We extend a description logic with new constructors which allow to define operators on cubes. The constructors we propose correspond in part to the operators on data cubes given in (Agrawal&al 1997).

1. INTRODUCTION

Recently, much attention has been focused on multidimensional databases. Many commercial systems based on their own data models have been developed. The main strength of multidimensional databases is their ability to view, analyze, and consolidate huge amounts of data. To do so, instead of presenting tables to the user, data is presented in the shape of so called *data-cubes* which can be manipulated by using operators to cut out pieces from large cubes, change the granularity of dimensions, turn cubes, etc. Because of these functionalities, multidimensional databases play an important role in Decision Support Systems, for On Line Analytical Processing, and in Data Warehousing. In general, multidimensional databases provide two categories of tools:

1) Tools for integration, efficient storage and retrieval of large volumes of data.

2) Tools for viewing and analyzing data from different perspectives. These tools allow interactive querying of data and their analysis, often referred to by "navigation" since this way of querying seems much more intuitive than classical ad-hoc queries.

For describing the interdependence of data in a multidimensional database, *data cubes* are regarded as an appropriate data model. In a data cube, each axis is associated with a dimension (e.g., time, space, or products) and its according values. Then, points in the cube (called cells) can be associated with values (called measures) of additional dimensions (like sales volumes). Displaying information to the user as well as navigation in this information are tasks that are heavily supported by this data model. In order to consolidate and analyse huge amounts of data, aggregation and roll-up play an important role. Data is often summarized at various levels of granularity and on various combinations of attributes. Therefore, queries are more complex and may take long time to complete. To face up to this problem, a great deal of effort has been invested in developing techniques for optimizing queries involving views and aggregate functions (Harinarayan&al 1996, Gray&al 1996, Ho&al 1997, Levy&al 1996, Ross&al 1994).

Unfortunately, a unifying framework for multidimensional databases is still missing. Hence, different multidimensional models, operators, techniques, etc. cannot be compared to each other or evaluated. Furthermore, there is a certain *lack of common vocabulary and common understanding of multidimensional data models*. In this paper, we propose an object-centered framework for multidimensional data models to overcome this lack. The language proposed allows to capture more of the semantics of multidimensional data models at the conceptual level. It facilitates the modeling and understanding of multidimensional databases and related operations, by using a vocabulary which is appropriate for the problem domain. It also allows the use of cube operations by making it easier for the user to interpret them. This work *integrates formalisms* that have been developed for *Data Warehouse* (Agrawal&al 1997, Gray&al 1996, Ho&al 1997) and in *knowledge representation* (Borgida 1992, Borgida 1995, Baader&Hanschke 1991, Brachman&Schmolze 1985, Schmidt-Schauss&Smolka 1991). It is a first step towards the development of a framework which

1) is equipped with well-defined semantics and

2) allows the precise definition of relevant reasoning services and problems. This is a prerequisite for the investigation of these problems with respect to their complexity and the comparison of different reasoning techniques.

The framework presented in this paper is based on description logics, a family of formalisms which have already proved their usefulness as unifying framework for object-centered representation formalisms (Brachman&Schmolze 1985, Calvanese&al 1994). These formalisms are equipped with well-defined semantics and sound and complete reasoning algorithms. In fact, a main characteristic of these formalisms is that problems like satisfiability, containment or consistency are effectively decidable. Finally, they can be equipped with a kind of interface to specific "concrete domains" (e.g., integers, strings, reals), and built-in predicates (Baader&Hanschke 1991).

To serve as a framework for multidimensional data model, we will describe data cubes within a description logic. We build on works by Agrawal et al.

(Agrawal&al 1997) and Baader and Hanschke (Baader&Hanschke 1991) to develop a description logic that takes into account the basic operators on cubes. Like (Agrawal&al 1997), operators are defined on cubes and produce cubes. Thus, we can define classes of cubes and build new cubes from already existing ones using operators like join, restrict, aggregate, etc. Finally, we can ask whether a class of cubes is contained in another one or whether it can ever be instantiated.

Paper outline: Section 2 gives an example illustrating the features of a data cube. Section 3 summarizes description logics. In Section 4, we present the syntax and declarative semantics of our language based on ALC(D) (Baader&Hanschke 1991). We also give examples to illustrate the use of operators at the assertional level. We conclude in Section 5 by highlighting some perspectives.

2. EXAMPLE

Consider the following fragment (Figure 1) of a database that contains a report of car sales. This example is drawn from (Gray&al 1996) with some modifications. It presents the number of cars sold with respect to a car's model, (construction-)year, and color. We use this database extract as a running example throughout this paper.

Model	Year	Color	Units
M1	1995	White	70
M1	1995	Black	35
M1	1996	White	50
M1	1996	Black	62
M2	1995	White	55
M2	1995	Black	80
M2	1996	White	60
M2	1996	Black	85
M3	1995	White	35
M3	1995	Black	22
...	...	...	... .

Figure 1. A relational representation of car sales relation

Trying to compare the sales of white cars in 1996, it becomes obvious that it is difficult to extract and analyze information from this table. There is an alternative way of representing the same data which overcomes this problem of jumping between rows. That representation is commonly known as a data cube. It is a cube with three intersecting 2D cross tabs. In this representation, each axis is associated with a dimension, that is a column of a relational table.

Elements within a dimension are called *positions*. Points in a data cube are called *cells*, and each cell is associated with the corresponding element of the column *unit*. Then, the cube is said to have dimensions *model*, (construction-)*year*, and *color*, and to have the measure *unit*. The data in this representation is more organized, hence it is more easily accessible than the organization offered by a relational table. Note that the cube representation is only possible because the number of units sold is uniquely determined by a car's model, its construction year and its color.
Multidimensional databases are designed for ease and performance in manipulating and analyzing huge amounts of complex data, hence values of dimensions or measures can be aggregated, decomposed, or combined to new values. This corresponds to what is commonly known as data *consolidation*. (Agrawal&al 1997) defines a set of operators for data consolidation which work on data cubes. This set of operators is claimed to be *minimal* and powerful enough to be used for all interesting queries on data cubes. It contains the following operators:

1) *Push* allows to convert dimensions into measures,
2) *Pull* allows to create a new dimension from a specified measure,
3) *Destroy* allows to remove a dimension that has a single value in its domain,
4) *Restrict* removes from the cube those values of a given dimension that do not satisfy a stated condition,
5) *Join* allows to relate information in two cubes,
6) *Merge* allows to merge cubes by aggregating the values of some dimensions into new values of the same dimension and by aggregating the according measures.

3. DESCRIPTION LOGICS

The representation formalism presented in the following belongs to the family of description logics, hence we first briefly introduce these formalisms. Description logics (also called concept or terminological logics) (Brachman&Schmolze 1985) are a family of formalisms designed to represent the *taxonomic and conceptual knowledge of a particular application domain on an abstract, logical level*. So it is not surprising that they are particularly adept at representing the semantics of real world situations, including data semantics. They are equipped with well-defined, model-theoretic semantics and most of them have strong expressive power. Furthermore, the interesting reasoning problems such as containment and satisfiability are, for most description logics, effectively decidable.

Description logics are built around concepts, which are interpreted as classes of objects in the domain of interest, and roles, which are interpreted as binary relations on these objects. A description logic is mainly characterized by a set of constructors which allow to build complex concepts and roles from atomic ones. For example, from atomic concepts *Human* and *Female* and the atomic role *child* we can build the concept (*Human* **and forall** child.*Female*) which denotes the set of all *Human* whose children are all instances of *Female*. Here, the constructor **and** denotes conjunction between concepts, while **forall** denotes (universal) value restriction.

A knowledge base in a description logic system is made up of two components: (1) the Tbox is a general schema concerning the classes of individuals to be represented, their general properties and mutual relationships; (2) the Abox contains a partial description of a particular situation, possibly using the concepts defined in the Tbox. It contains descriptions of (some) individuals of the situation, their properties and their interrelationships.
Retrieving information in a knowledge representation system based on description logics is a deductive process involving the schema (Tbox) and possibly an instantiation (Abox). In fact, the Tbox is not just a set of constraints on possible Aboxes, but contains intensional information about classes. This information is taken into account when answering queries to the knowledge base. The following reasoning services are the most important ones provided by knowledge representation systems based on description logics (See (Donini&al 1995) for an overview):

1) *Concept satisfiability*: Given a knowledge base and a concept C, does there exists at least one model of the knowledge base assigning a non-empty extension to C?
2) *Subsumption*: Given a knowledge base and two concepts C and D, is D more general than C? That is, is in all models of the knowledge base each instance of C also an instance of D?
3) *Knowledge base satisfiability*: Are an Abox and a Tbox consistent with each other? That is, does the knowledge base admit a model?
4) *Instance checking*: Given a knowledge base, an object o, its possibly incomplete description, and a concept C, is o an instance of C in all models of the knowledge base?

4. THE LANGUAGE

In this section, the syntax and semantics of a description logic for the representation of multidimensional data is introduced. This language is based on the one presented in (Baader&Hanschke 1991) and extends it by a constructor that captures functional dependencies (Borgida&Weddell 97) and a set of operators for the handling of data cubes.

4.1 Basic Definitions

In order to define aggregation appropriately, we first introduce the notion of *multisets*: In contrast to simple sets, in a multiset an individual can occur more than once, this is to say that, for example, the multiset {1} is different from the multiset {1,1}.

Definition 1 (**Multisets**) A multiset (or bag) on a set **S** is a subset **M** of **S**×**N** such that for every **a** element of **S** there is a non-negative integer **n** (the number of occurrences of **a** in **M**) such that (**a**, **m**) is in **M** if and only if **m** < **n**. The set of all multisets on **S** is denoted by **M**(**S**).

Next, we define an extension of *concrete domains* (we use concrete domains as defined in (Baader&Hanschke 1991)) In addition to application-specific domains (i.e., strings, reals, non-negative integers, etc.), we incorporate built-in predicates like ≤, ..., functions, and aggregation functions into the abstract domain of objects.

Definition 2 (**Concrete Domains**) A concrete domain
D = (dom(D), pred(D), funct(D), agg(D)) consists of:
- the domain dom(D),
- a set of predicate symbols pred(D), where each predicate symbol **P** ∈ pred(D) is associated with an arity **n** and a n-ary relation $\mathbf{P}^D \subseteq \text{dom}(D)^\mathbf{n}$,
- a set of 2-ary function symbols funct(D), where each function symbol **f** ∈ funct(D) is associated with a function $\mathbf{f}^D : \text{dom}(D)^2 \rightarrow \text{dom}(D)$, and
- a set of aggregation symbols agg(D), where each aggregation symbol Σ ∈ agg(D) is associated with an aggregation function $\Sigma^D : \mathbf{M}(\text{dom}(D)) \rightarrow \text{dom}(D)$.

In the following, we define our extension of the description logic ALC(D) as presented in (Baader&Hanschke 1991). ALC(D) itself is an extension of ALC, introduced in (Schmidt-Schauss&Smolka 1991), a well-known description logic with high expressive power. In ALC, concepts can be built using boolean operators (i.e., **and**, **or**, **not**), and value restrictions on those individuals associated to an individual via a certain role (binary relation). These include *existential* restrictions like in (***exists*** *has_child.Girl*) as well as *universal* restrictions like (***forall*** *has_child.Human*). Additionally, in ALC(D), (abstract) individuals which are described using ALC can now be related to values in a *concrete domain*, like, for example, the integers or strings. This allows us to describe, for example, persons whose savings balance are higher than their yearly income, by *Human* ***and*** (*savings* > *y_income*).

In (Baader&Hanschke 1991), it is shown that all interesting inference problems for ALC(D) are decidable provided that D does not contain any aggregation function and n-ary functions, and satisfies some additional very weak conditions.

4.2 The Concept Language

In this subsection, we propose to extend ALC(D) with a constructor that captures functional dependencies (Borgida&Weddell 1997) and operators on cubes specified at the extensional level. The language resulting from the extension of ALC(D) with a functional dependency operator and cube operators at the extensional level will be called $ALC(D)^{fd}$.

We start with the definition of complex concepts, which are constructed using certain operators. These concepts can be used to specify the terminology in a so-called TBox. This TBox can be viewed as an encyclopedia in which the meaning of certain concepts is defined using other concepts (which are possibly defined themselves in this TBox). Then, a specific situation can be described in a so-called ABox, possibly using the concepts defined in the TBox. In the ABox, we introduce some individuals, describe their properties and their interrelationships.

Definition 3 (**Syntax of** $ALC(D)^{fd}$ **- concepts)** Let N_C, N_R, and N_F be disjoint sets of concept, role, and feature names, and let D be an admissible[1] concrete domain. Then each concept name is a concept, and complex concepts are defined inductively by the following rules, where C, D are concepts, R denotes a role name or feature name, **P** $\in$ pred(D) is a n-ary predicate name, u_1, ..., u_n are feature chains (A feature chain $u = f_1 \bullet ... \bullet f_n$ is a composition of features), f, f_1, ..., f_n are feature names. Then the following expressions are also concept terms:

1. C **or** D (disjunction), C **and** D (conjunction), and **not**(C) (negation),
2. **exists** R. C (exists-in restriction) and **forall** R.C (value restriction),
3. $P(u_1, ..., u_n)$ (predicate restriction),
4. [**fd** C {f_1 ... f_n} f] (functional dependency).

A terminology (or TBox) T is a (finite) set of concept definitions, each of the form A = C where A is a concept name and C a complex concept. We restrict our attention to those terminologies where each concept name A occurs at most once on the left hand side of a concept definition, and which does not contain definitional cycles.

An Abox A is a (finite) set of assertions. Given a set of individual names N_I, assertions are of the following forms:
a:C, (a, b):R, (a, b):f, (a, x):f, $(x_1, ..., x_n)$:P,
a=**rename**(b, f, g), a=**destroy**(b, f), a=**restrict**(b, C), a=**join**(b, c),
a=**Join**((b, c, <n_1, o_1, f_1, g_1>, ..., <n_m, o_m, f_m, g_m>), a= **aggr**(b, f, Σ, g).
for (possibly complex) concepts C, feature names f, f_i, g, g_i, n_i, role name R, function names $o_i \in$ funct(D), an aggregation function $\Sigma \in$ agg(D), (abstract) individuals a, b, c $\in N_I$, a n-ary predicate name P, and elements of concrete domain x, x_1, ..., x_n.

The semantics of these constructs is now defined in a model-theoretic way.

Definition 4 (**Semantics**) Let D be an admissible concrete domain. The semantics is then given by an interpretation $I = (\Delta^I, .^I)$, which consists of an

[1] A concrete domain D is admissible iff (1) **pred**(D) is closed under negation and contains a unary predicate name Top_D for **dom**(D), and (2) satisfiability of finite conjunction over **pred**(D) is decidable.

(abstract) interpretation domain Δ^I disjoint from the concrete domain dom(D), and an interpretation function $.^I$. The evaluation function $.^I$ associates each concept C with a subset $C^I \subseteq \Delta^I$, each role R with a binary relation $R^I \subseteq \Delta^I \times \Delta^I$, and each feature name f with a partial function f^I from Δ^I into $(\Delta^I \cup \mathrm{dom}(D))$. Additionally, I has to satisfy the following equations:

(C **and** D)^I	=	$C^I \cap D^I$
(C **or** D)^I	=	$C^I \cup D^I$
not(C)	=	Δ^I - C^I
(**forall** R.C)^I	=	$\{x \in \Delta^I$ such that for all y, if $(x, y) \in R^I$ then $y \in C^I\}$
(**exists** R.C)^I	=	$\{x \in \Delta^I$ such that there exists y, with $(x, y) \in R^I$ and $y \in C^I\}$
$P(u_1, ..., u_n)^I$	=	$\{ x \in \Delta^I$ such that $((u_1)^I(x), ..., (u_n)^I(x)) \in P^D \}$
[**fd** C $\{f_1 ... f_n\}$ f]	=	$\{ x \in \Delta^I$ such that for all $y \in C^I$, if $(f^I_1(x) = f^I_1(y) \wedge ... \wedge f^I_n(x) = f^I_n(y))$ then $f^I(x) = f^I(y)\}$

A concept C is *satisfiable* iff there exists an interpretation *I* such that $C^I \neq \{\}$. Such an interpretation is called a *model* of C. A concept C is subsumed by a concept D (written C< D) iff $C^I \subseteq D^I$ holds for each interpretation *I* .
An interpretation *I satisfies* a TBox T iff *I* satisfies each concept definition in T, and it satisfies a concept definition A=C iff A^I=C^I.

If $(a, b) \in R^I$ or $f^I(a)$=b, we say that b is an R-filler (resp. f-filler) of a. Interpretations of ABoxes, additionally, associate each individual name *a* to an element of Δ^I, in such a way that $a^I \neq b^I$ holds for two different individual names *a*, *b*. Again *I* satisfies an Abox A iff *I* satisfies each assertion in A , that is to say

$a^I \in C^I$	for each $a{:}C \in A$
$(a^I, b^I) \in R^I$	for each $(a, b){:}R \in A$
$f^I(a^I) = b^I$	for each $(a, b){:}f \in A$
$f^I(a^I) = x^I$	for each $(a, x){:}f \in A$
$(x^I_1, ..., x^I_n) \in P^D$	for each $(x_1, ..., x_n){:}P \in A$

The semantics of assertions containing operators like **destroy** or **join** is more difficult and will first be illustrated by examples and then given formally.

An ABox A is said to be *consistent* with a TBox T iff there exists a model for both, that is an interpretation *I* satisfying A and T.

Note that the description of individuals in an ABox needs not to be complete: A model of an ABox might have more elements than those explicitly mentioned, and the individuals mentioned in the ABox might have more properties than those explicitly stated in the ABox. This is due to the so-called *Open World Assumption*.

Example (cont.) Using ALC(D)^{fd} cubes containing information on car sales can be described by the following concept definition:

carsales = **forall** *has_cell*.(**exists** model.Model **and exists** year.INTEGER **and exists** color.STRING **and exists** unit.INTEGER **and** [fd carsales {model, year, color} unit])

where *has_cell* is a role name and model, year, color, and unit are feature names. Model can be seen as a primitive concept containing the different models of cars. Concrete domains needed in this example are INTEGER and STRING.

4.3. Semantics of Cubes Operators

The operators for which we are going to define the semantics make sense only for "cubes", which have been introduced only informally so far. In general, a cube is an object which is associated to cells which are all of similar form. This fact of being of the same form is described by the following concept cube. It is defined in such a way that, if a cell of an instance of cube has an f-filler for some feature name f, then all other cells have also f-fillers:

cube = **and**$_{f \in N_F}$((**exists** has_cell.(**exists** f.ALL **or** *Top*$_D$(f))) $\Rightarrow$ (**forall** has_cell.**exists** f.ALL))

where C $\Rightarrow$ D is used as shorthand for (**not**(C) **or** D) and ALL is a shorthand for the "universal" concept (A **or not**(A)) (A is a concept name). Given that-for each specific application-the set of feature names (N_F) is finite, *cube* is a concept and describes cubes according to our intuition. For the formal definition of the operators, further abbreviations are useful:

Definition 5 Let x,y be cubes in I. Then cells(x) := {y $\in \Delta^I$ such that (x, y) $\in$ has_cell^I }. We say x, y are disjoint if x $\neq$ y and they do not share cells. More formally disj(x, y) iff x $\neq$ y and cells(x) $\cap$ cells(y) = {}. In the sequel, we will use R for a role name or a feature name, and
R^I (a) as shorthand for all R-fillers of a, i.e. R^I(a)={b $\in \Delta^I \cup$ dom(D) such that (a,b) $\in R^I$ }.

Now we are ready to define the semantics of the operators. The intuition behind each operator is first given by an example and then defined formally. In general, so that *I* satisfies an assertion of the form x = **op**(y, ...), there has to exist a mapping π from the cells of x into the cells of y. Further properties of the mapping π depend on the kind of operator and its parameters.

Example 1 We want to restrict the cube car_sales (Figure 3) to those cells containing information about cars built in 1996 and for which at least 50 units were sold. The corresponding assertion is:

car_sales_yur = **restrict**(car_sales, $=_{1996}$(year) **and** $>_{50}$(unit)).

Here $=_n$ stands for the unary predicate which tests for equality with n and $>_n$ stands for the unary predicate which tests for being greater than n.

1. Definition 4 (cont. 1) An interpretation *I* satisfies an assertion x = **restrict**(y,C) iff *disj*(x, y) and there exists a mapping π such that

π : cells(y) $\cap$ C^I $\rightarrow$ cells(x) is bijective and
$\forall c \in$ cells(y) $\cap$ C^I, $\forall R$: $R^I(c)=R^I(\pi(c))$.

Example 2 The dimension year in the cube car_sales_yur has a single value in its domain. So, we can remove this dimension without loosing any information or destroying functional dependencies. The corresponding assertion is: car_sales_yur_96 = **destroy**(car_sales_yur, year)

2. Definition 4 (cont. 2) An interpretation *I* satisfies an assertion x = **destroy**(y, f) iff *disj*(x, y), and for all z, z' $\in$ cells(y) it holds $f^I(z) = f^I(z')$, and there exists a mapping π such that

π : cells(y) $\rightarrow$ cells(x) is bijective and
$\forall c \in$ cells(y): $f^I(\pi(c))$ is undefined and $\forall R$ if $R \neq f$ then $R^I(c) = R^I(\pi(c))$.

Note that an arbitrary feature f can be destroyed from a cube x even if not all cells in x have the same f-fillers. In this case, by applying **destroy**, (1) we really loose information and (2) might have different cells c_i whose remaining f_j-fillers coincide.

Example 3 Suppose we have given the cube car_sales, and another cube car_prices containing information about a car's price with respect to its model and its construction year. If we want to enhance the car_sales by the prices of the cars, we can do this by: car_sales_prices = **join**(car_sales, car_prices).

3. Definition 4 (cont. 3) An interpretation *I* satisfies an assertion x = **join**(y, y') iff *disj*(y, y'), *disj*(x, y), *disj*(x, y'), and there exists a mapping π such that

π : **common**(y, y') $\rightarrow$ cells(x) is bijective and
$\forall$(c, d) $\in$ **common**(y, y') $\forall R$: $R^I(c) \cup R^I(d) = R^I(\pi(c,d))$.
where **common** contains all pairs of cells of y and y' agreeing on all common features in y and y', i.e.,
common (y, y') := {(c, d) $\in$ cells(y)$\times$cells(y') such that $\forall f \in N_F$: $c \notin$ dom(f^I) or
d $\notin$ dom(f^I) or $f^I(c) = f^I(d)$}.
where dom(f^I) is the domain of f^I, i.e. dom(f^I) = {$x \in \Delta^I$ such that $f^I(x)$ is defined}.

Hence for this **join** operator the features on which the cubes are joined are those which occur in both input cubes. If one wants to use **join** in a different way, one may use the operator **rename** on the input cubes to change the feature names:

An interpretation *I* satisfies an assertion x = **rename**(y, f, g) iff *disj*(x, y), and there exists a mapping π such that

π : cells(y) $\rightarrow$ cells(x) is bijective and
$\forall c \in$ cells(y) $\forall R$: $R^I(c) = R[f \rightarrow g]^I(\pi(c))$,
where $R[f \rightarrow g] = R$ if $R \neq f$ and $R[f \rightarrow g] = g$ if $R = f$.

Example 4 Suppose we have the same cube car_prices as in the previous example, but instead of just adding the prices of the cars to car_sales, we want to see the sales volumes for each model, color and year besides the number of units sold. To enhance car_sales by the sales volumes of the cars, we need a more powerful form of the join operator, namely one which takes additional functions. The above example can then be obtained by:
car_sales_volumes = **Join** (car_sales, car_prices, < volumes, mult, units, price>), where volumes is a feature name that does not occur neither in car_sales nor in car_prices. For each cell in car_sales_volumes, its volumes-filler is the product of the unit- and price-filler of the according cells in car_sales and car_prices.

4. Definition 4 (cont. 4) An interpretation *I* satisfies an assertion x = **Join**(y, y', < n_1,o_1,f_1,g_1>, ..., <n_m, o_m, f_m, g_m>) iff *disj*(y, y'), *disj*(x, y), *disj*(x, y'), and there exists a mapping π such that

π : **common**(y, y') $\rightarrow$ cells(x) is bijective and
$\forall(c, d) \in$ common(y, y')
$\forall 1 \leq i \leq m$; $n^I_i(\pi(c,d)) = o_i(f^I_i(c), g^I_i(d))$ and
$\forall R$ if $R \neq \{f_1, ..., f_m, g_1, ..., g_m\}$ then $R^I(c) \cup R^I(d) = R^I(\pi(c,d))$.

Example 5 Finally, suppose we want to compute the number of units sold for each model and each year. This can be done using the operator **aggr** in the following way: car_sales_total_year = **aggr**(car_sales, unit , sum, color)

The operator **aggr** takes, besides the name of the cube, three parameters: The feature *unit* whose fillers are aggregated using the aggregation function *sum* $\in$ agg(D), and the feature color which does no longer occur in the output cube since we have summarized all cells regardless of their color-fillers.

5. Definition 4 (cont. 5) An interpretation *I* satisfies an assertion x = **aggr**(y, f, Σ, g) iff **disj**(x, y), and there exists a mapping π such that

π: **maxss**(y, f, g) $\rightarrow$ cells(x) is bijective and $\forall c \in$ cells(x): $c \notin$ dom(g^I) and
$f^I(c) = \Sigma f^I(c')$ (for $c' \in \pi^{-1}(c)$) and $\forall R$: if $R \notin \{f, g\}$ then
$R^I(c) = R^I(c')$ for some $c' \in \pi^{--1}(c)$,

where **maxss** contains all maximal subsets of cells of y which agree on all features but f and g, i.e.,
maxss(y, f, g):= $\{ S \subseteq$ cells(y) such that $\forall c, c' \in S\ \forall R$ if $R \notin \{f, g\}$ then $R^I(c) = R^I(c')$ and $\forall d \notin S\ \exists c \in S\ \exists R \notin \{f, g\}$ with $R^I(d) \neq R^I(c)\}$

5. SUMMARY

We presented a new approach for modeling multidimensional databases. We provided in particular the syntax and the declarative semantics of an object-centered formalism. The description logic ALC(D) is extended by a set of operators that work on cubes, for example for aggregating information and functional dependencies. Aggregation functions are not restricted to a fixed set, we allow for arbitrary ones. We want to distinguish between the internal representation of a cube and its visualization. Given an instance c of cube as described above, c can contain much more information than what can be shown in one single cube. In order to visualize the information contained in c, one has to choose at most 3 dimensions d_1, d_2, d_3 plus n measures f_i. Then, this part of the information contained in c can be visualized as cube representation provided that $<f_1, ..., f_n>$ is functionally dependent on d_1, d_2, d_3. The push and pull operators take a cube and transform dimensions into measures and vice versa. In our approach, this can be reduced to asking for a different visualization.

The framework presented here is sufficiently flexible to admit with relative ease the introduction of new description constructors, which can be application specific.

Since this work is a first approach to an object-centered representation of multidimensional data models and the according operators, there remain some open questions and further work to be done: Until now, we have only defined the interesting reasoning problems like satisfiability, containment, consistency. This is a first step towards the investigation of the complexity of these problems and the design of suitable algorithms for them. We are currently working on a mechanism for the declarative specification of hierarchically structured dimensions like time, region, or product groups.

6. REFERENCES

Agrawal R. and Gupta A. and Sarawagi S. (1997) Modeling Multidimensional Databases. Proceedings of the International Conference on Data Engineering (ICDE'97), Birmingham, UK. Also available as Research Report via WWW at http: //www.almaden.ibm.com/cs/people/ragrawal/pubs.html/#olap.

Baader F. and Hanschke P. (1991). A Scheme for Integrating Concrete Domains into Concept Languages. Proceedings of the 12th International Joint Conference on Artificial Intelligent (IJCAI'91), Sydney, Australia, pp. 452-457.

Borgida A. (1992). From Type Systems to Knowledge Representation: Natural Semantics Specifications for Description Logics. Intelligent and Cooperative Information Systems, V. 1, N. 1, pp. 93-126.

Borgida A. (1995). Description Logics in Data Management. IEEE Transactions on Knowledge and Data Engineering (TKDE).

Borgida A. and Weddell G. (1997). Adding Functional Dependencies to Description Logics. Proceedings of the fifth International Conference on Deductive and Object-Oriented Databases (DOOD'97), Montreux, Switzerland.

Brachman R.J. and James G. Schmolze J.G. (1985). An Overview of the KL-ONE Knowledge Representation System. Cognitive Science, V. 9, N. 2, pp. 171-216.

Calvanese D. and Lanzerini M. and Nardi D. (1994). A Unified Framework for Class Based Representation Formalisms. Proceedings of the fourth International Conference on Principles of Knowledge Representation and Reasoning (KR’94), Bonn, pp. 109-120.

Donini F. M. and Lenzerini M. and Nardi D. and Schaerf A. (1995). Reasoning in Description Logics. Foundation of Knowledge Representation, Cambridge University Press.

Gray J. and Bosworth A. and Layman A. and Pirahesh H. (1996). Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. Proceedings of the 12th International Conference on Data Engineering (ICDE'96), New Orleans, USA, pp. 152-159.

Harinarayan V. and Rajaraman A. and Ullman J.D. (1996). Implementing Data Cubes Efficiently. Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD'96), Montreal, Canada, pp. 25.

Ho C.-T. and Agrawal R. and Srikant R. (1997). Range-Sum Queries in Data Cubes. Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data (SIGMOD'97), Tucson, Arizona, USA.

Levy A.Y. and Mumick I.S. and Sagiv Y. (1996). Range-Sum Queries in Data Cubes. in Proceedings of the International Conference on Extending Database Technology (EDBT'96), Avignon, France..

Schmidt-Schauss M. and Gert Smolka G. (1991). Attributive Concept Descriptions with Complements. Artificial Intelligence, V. 48, N. 1, pp. 1-26.

Ross K.A. and Srivastava D. and Stuckey P.J. and Sudarshan S. (1994). Foundations of Aggregation Constraints. Proceedings of the Second International Workshop on Principles of Constraint Programming (PPCP'94), Orcas Island, WA, pp. 193-204, Springer-Verlag, LNCS 874.