On Schema Evolution in Multidimensional Databases[1]

Markus Blaschka, Carsten Sapia, Gabriele Höfling

FORWISS (Bavarian Research Center for Knowledge-Based Systems)
Orleansstr. 34, D-81667 Munich, Germany
Email: {blaschka, sapia, hoefling}@forwiss.tu-muenchen.de

Abstract. Database systems offering a multidimensional schema on a logical level (e.g. OLAP systems) are often used in data warehouse environments. The user requirements in these dynamic application areas are subject to frequent changes. This implies frequent structural changes of the database schema. In this paper, we present a formal framework to describe evolutions of multidimensional schemas and their effects on the schema and on the instances. The framework is based on a formal conceptual description of a multidimensional schema and a corresponding schema evolution algebra. Thus, the approach is independent of the actual implementation (e.g. MOLAP or ROLAP). We also describe how the algebra enables a tool supported environment for schema evolution.

1 Introduction

The main idea of a data warehouse architecture is the replication of large amounts of data gathered from different heterogeneous sources throughout an enterprise. This data is used by knowledge workers to drive their daily decisions. Consequently, easy-to-use interactive analysis facilities on top of the data warehouse are necessary. Most often multidimensional databases (specifically OLAP systems) are used for this purpose. These multidimensional information systems (MDIS) provide the user with a multidimensional view on the data and offer interactive multidimensional operations (e.g. slicing).

The user of an MDIS interactively formulates queries based on the structure of the multidimensional space (the MD schema of the database). This means that the schema of the MD database determines what types of queries the user can ask. Thus, the design of the schema in such an environment is a very important task. This has been recognized by the research community as several publications in the field of multidimensional schema design show (e.g. [7], [12]). Nevertheless, a complete methodology for designing and maintaining an MDIS must also take schema evolution into account which has so far received almost no attention. This paper provides a framework to formally approach the evolution issue for MDIS and shows how this formal frame-

[1] An extended version of this paper can be downloaded at http://www.forwiss.tu-muenchen.de/~system42/publications/

work can be used to implement a tool supported evolution process. In such an environment, the designer can specify the required schema evolution on a conceptual level and the corresponding implementation is adapted automatically.

To understand why schema evolution plays an important role especially in decision support environments (where data warehouse and OLAP applications are mostly found), let us first take a look at the typical design and maintenance process of such a system. Interactive data analysis applications are normally developed using an iterative approach. The main two reasons for this very dynamic behavior are:

- the interactive multidimensional analysis technology is new to the knowledge worker. This means that it is impossible for him to state his requirements in advance.
- the business processes in which the analyst is involved are subject to frequent changes. These changes in business processes are reflected in the analysis requirements [15]. New types of queries that require different data become necessary. Because the schema of an MDIS restricts the possible analysis capabilities, the new query requirements lead to changes in the MD database schema.

A single iteration of the design and maintenance cycle [17] consists of the phases **'Requirement Analysis'** (where the requirements of the users concerning data scope, granularity, structure and quality are collected), the **'Conceptual Design'** (where the required views of the users are consolidated into a single conceptual model and – during further iterations of the development cycle – the schema is adopted according to the changed requirements), the **'Physical (Technical) Design'** (where implementation decisions are taken), **'Implementation'** (rather mechanic realization of the specifications developed during the technical design phase), and the **'Operation'** phase (where new data is loaded to the database on a regular basis and the users analyze data). Typically, when a system is in operation, new requirements for different or differently structured data arise. If a certain amount of new requirements is reached, a new iteration is started.

The conceptual multidimensional data model is the central part of the design and maintenance cycle as it already contains a consolidation of all user requirements but does not yet contain implementation details. All data models that occur later in the design process are refinements of the conceptual model.

Thus, the starting point for our research of schema evolution operations and their effects is the conceptual level. The goal of our approach is to automatically propagate changes of the conceptual model to the other models along the design cycle. A prerequisite for this is a formal framework to describe the evolution operations and their effects, which we present in this paper.

The rest of the paper is structured as follows: In section 2 we discuss related work from the areas of data warehousing and object-oriented databases. Section 3 summarizes the objectives and benefits of our framework for multidimensional schema evolution. Section 4 develops a formal notion of multidimensional schemas and instances which serves as a basis for the description of schema evolution operations and their effects that is described in section 5. Section 6 sketches how the formal framework can be used to implement an interactive tool-supported schema evolution process. We conclude with directions for future work in section 7.

2 Related Work

A first approach to changing user requirements and their effects on the multidimensional schema is [11]. Kimball introduces the concept of "slowly changing dimensions" which encompasses so called structural changes, i.e. value changes of dimension attributes, like changing the address of a customer. Since data in an OLAP system is always time related, the change history has to be reflected. If a customer moves, both the old and the new address have to be stored. Solutions for this case (data evolution) are rather straightforward. The slowly changing dimensions approach is not complete and provides a rather informal basis for data and schema changes. Further, there are no clear decision criteria for the proposed implementation alternatives.

Golfarelli et al. [7], [8] proposed a methodological framework for data warehouse design based on a conceptual model called dimensional fact (DF) scheme. They introduce a graphical notation and a methodology to derive a DF model from E/R models of the data sources. Although the modeling technique supports semantically rich concepts it is not based on a formal data model. Furthermore, the framework does not concentrate on evolution issues which we believe is an important feature for the design and maintenance cycle.

Schema evolution has been thoroughly investigated in the area of object-oriented database systems (OODBMS) because - similar to the multidimensional case – there are conceptual relationships representing semantical information: the *isa* relationships representing inheritance between classes. Schema evolution in object-oriented database systems has been broadly discussed both in research prototypes and commercial products (e.g. [1], [20], [10]). We take these approaches as a foundation for our work and investigate how techniques and approaches from object-oriented schema evolution can be adopted to the case of multidimensional information systems.

Most research work in the area of OLAP and data warehousing concentrates on view management issues. These approaches see the warehouse database as a materialized view over the operational sources. The arising problems are how these views can be maintained efficiently (*view maintenance problem*, see e.g.[6]), which aggregations on which level improve performance with given space limitations (*view selection problem*, see e.g. [3]), and how the views can be adopted when changes in the view definition or view extent arise (*view adaptation and synchronization problem*, see e.g. [14], [16]). Our work supplements these approaches because we develop the warehouse schema from the user requirements and not from the schemas of the operational sources.

A recent approach to schema evolution is [9]. From the related work mentioned above this approach is the most closely related to our work. However, it differs in the following aspects. First, it only addresses changes in the dimensions. We provide a set of evolution operations covering also facts and attributes. Next, insertions of levels are limited to certain positions in a dimension. Our framework allows random insertions of dimension levels at any place of a given MD schema. Further, our approach is based on a conceptual level, thus not assuming any specific implementation details (e.g. a ROLAP implementation).

3 An Overview of the Schema Evolution Framework for MDIS

The objective of the work described in this paper is to propose a methodology that supports an automatic adaptation of the multidimensional schema and the instances, independent of a given implementation. We provide a conceptual multidimensional schema design methodology including a schema evolution algebra. Our vision of the warehouse design and maintenance cycle is that the whole system is specified and designed on a conceptual level. Changes that arise when the system is already in production would only be specified on the conceptual level. Our design and evolution environment cares for the necessary changes in the specific target system (i.e. database and query/management tools).

To this end, our general framework comprises

1. a data model (i.e. a formal description of multidimensional schemas and instances),
2. a set of formal evolution operations,
3. the descriptions of effects (extending to schema and instances) of the operations,
4. an execution model for evolution operations, and finally
5. a methodology how to use our framework.

This paper addresses points one, two and three. It further contains ideas for the methodology (point five). The main objectives of our framework for multidimensional schema design and evolution (see [2] for the complete list) are

- automatic adaptation of instances: existing instances should be adapted to the new schema automatically. Further, the adaptation of instances should be possible separately from the adaptation of the schema (in case there are no instances yet), physical and/or logical adaptation should be possible.
- support for atomic and complex operations: our methodology defines atomic evolution operations as well as complex operations.
- clear definition of semantics of evolution operations: the semantics of a given schema evolution operation may offer more alternatives and are not always clear. Our methodology fixes an alternative for execution.
- providing a mechanism for change notification (forward compatibility): we provide a change notification mechanism and guarantee that existing applications do not have to be adapted to the new schema. Further, there is no need for immediate adaptations of the tool configurations.
- concurrent operation and atomicity of evolution operations: the framework should allow concurrency of schema changes and regular queries. Further, schema evolution transactions shall be atomic.
- different strategies for the scheduling of effects: the framework should offer lazy strategies for the execution of effects of a schema change. Based on a cost model, the system may schedule the execution of effects for a later point in time. If the adapted instances are needed immediately, the system notifies the user about possible arising performance problems.
- support of the design and maintenance cycle: the framework supports all phases of the design and maintenance cycle. Thus, we cover not only the initial design (where the OLAP system is not populated with instances yet), but allow also adaptations of a populated system.

Summing up, our approach shall be used as a basis for tool-supported warehouse schema changes. The framework provides an easy-to-use tool allowing to perform schema modifications without detailed knowledge about the specific implementation and tools. The schema designer does not have to adapt different configurations of a tool and a database schema which must be consistent for a given implementation, but the tool is responsible for performing the necessary steps in a consistent and semantically correct way providing a single point of control accessible via a graphical formalism.

The contributions and scope of this paper are a formal model for MD schemas and instances together with a set of evolution operations. We describe the effects of these evolution operations on the MD schema and the instances, thus providing a formal algebra for MD schema evolution.

4 Multidimensional Schema and Instances

Several interpretations of the multidimensional paradigm can be found both in the literature (e.g. [4], [5], [13], [21]) and in product implementations. A comparison of the formal approaches shows that most of them do not formally distinguish between schema and instances ([19]) as their main goal is a formal treatment of queries using algebras and calculi. For our research work, we need a formalism that can serve as a basis for defining the schema evolution operations and their effects (see section 5). Therefore, this section contains a formal definition of a multidimensional schema and its instances (which was inspired by the formal multidimensional models mentioned above, esp. [4], [5], [21]).

The schema (or MD model) of an MDIS contains the structure of the facts (with their attributes) and their dimension levels (with their attributes) including different classification pathes [19]. We assume a finite alphabet Z and denote the set of all finite sequences over Z as Z*.

Definition 4.1 (MD model, MD schema): An MD model $\mathcal{M}$ is a 6–tuple <F, L, A, *gran*, *class, attr*> where

- **F** $\subset Z^*$ is a finite set of fact names $\{f_1,\ldots,f_m\}$ where $f_i \in Z^*$ for $1 \le i \le m$
- **L** $\subset Z^*$ is a finite set of dimension level names $\{l_1,\ldots,l_k\}$where $l_i \in Z^*$ for $1 \le i \le k$.
- **A** $\subset Z^*$ is a finite set of attribute names $\{a_1,\ldots,a_p\}$ where $a_i \in Z^*$ for $1 \le i \le p$. Each attribute name a_i has a domain *dom*(a_i) attached.
- The names of facts, levels and attributes are all different, i.e. $L \cap F \cap A = \emptyset$
- *gran*: $F \rightarrow 2^L$ is a function that associates a fact with a set of dimension level names. These dimension levels *gran*(f) are called the base levels of fact f.
- *class* $\subseteq L \times L$ is a relation defined on the level name. The transitive, reflexive closure *class** of class must fulfill the following property: $(l_1,l_2) \in class^* \Rightarrow (l_2,l_1) \notin class^*$. That means that *class** defines a partial order on L. $(l_1,l_2) \in class^*$ reads "l_1 can be classified according to l_2."
- *attr: A* $\rightarrow F \cup L \cup \{\perp\}$ is a function mapping an attribute either to a fact (in this case the attribute is called a measure), to a dimension level (in this case it is called

dimension level attribute) or to the special $\perp$-symbol which means that this attribute is not connected at all.

The MD model formalizes the schema of a multidimensional database. We use this formalism in section 5 to define a set of schema evolution operations. As we also want to analyze the effects of schema evolution operations on the instances of the schema, the remainder of this section presents a formal model for instances.

Definition 4.2 (Domain of a Dimension Level): The domain of a dimension level $l \in L$ is a finite set $dom(l) = \{m_1, \ldots, m_q\}$ of dimension member names.

Definition 4.3 (Domain and Co-domain of a fact): For a fact f the domain *dom*(f) and co-domain *codom*(f) are defined as follows:

$$dom(f) := \mathop{X}_{l \in gran(f)} dom(l)$$

$$codom(f) := \mathop{X}_{\{a | attr(a) = f\}} dom(a)$$

Definition 4.4 (Instance of MD model): The instance of an MD model $\mathcal{M}$ = <F, L, A, *gran*, *class, attr*> is a triple $\mathfrak{I}_{\mathcal{M}}$ = <R-UP, C, AV> where

- R-UP = { $r-up_{lev1}^{lev2}$ }is a finite set of functions with

  $r-up_{lev1}^{lev2}: dom(lev1) \to dom(lev2)$ for all $(lev1, lev2) \in class$
- $C = \{c_{f_1}, \ldots, c_{f_m}\}$; $f_i \in F \ \forall 1 \leq i \leq m$ is a finite set of functions

  c_f: dom(f) $\to$ codom(f); f $\in$ F. C maps coordinates of the cube to measures, thus defining the contents of the data cube.
- AV = { $av_1, \ldots, av_r$ } is a finite set of functions which contains a function av_a for each attribute a that is a dimension level attribute, i.e. *attr*(a) $\in$ L. The function av_a: *dom*(*attr*(a))$\to$ *dom*(a) assigns an attribute value (for attribute a) to each member of the corresponding level.

5 Evolution Operations

After having formally defined the notion of a multidimensional schema and its instances, we present a set of formal evolution operations for MD models. For each operation, we introduce the operation with its parameters and describe the effect on the MD schema. Since the operations usually do not work on an database without instances, the modification of existing instances is also given.

Together with the exact definition of an MD model and an instance of such an MD model (in chapter 4), the evolution operations listed here provide a formal schema evolution algebra. A formal property of this schema evolution algebra is its closure, i.e. an algebra or language is closed if the result of any allowable operation is a legal construct in the language. The closure of our schema evolution algebra can be formally proved because the algebra recognizes only one underlying construct, an MD model. Every operation is defined to take an MD model as input argument and produces as output a new MD model. Therefore, by definition, the algebra is closed.

We provide a minimal set of atomic operations. Atomic refers to the property that every operation only 'tackles' (i.e. changes) exactly one set or function of the given MD model. Of course, for notational convenience, complex operations can be defined. These complex operations consist then of a sequence of atomic operations. A formal proof of the minimality and completeness of this set of operations is ongoing work.

Formally, a schema evolution operation *op* transforms an MD model $\mathcal{M}$ =<F, L, A, *gran*, *class*, attr> to an MD model $\mathcal{M}'$. Some operations also require an adaptation of the instances $\Im_{\mathcal{M}}$ to $\Im'_{\mathcal{M}'}$. We always denote elements *before* the evolution with the regular letter (e.g. L), whereas a letter with an apostrophe (e.g. L') denotes the corresponding element *after* the evolution. For a function f:*dom→codom* let $f|_{dom'}$ denote the restriction of f to *dom'* $\subseteq$ *dom*

1. **insert level**: this operation extends an existing MD model by a new dimension level. The operation extends the set of levels without changing the classification relationships, thus creating an isolated element. Classifications Relationships have to be defined separately.
   Parameter: new level name $l_{new} \notin L$.
   $\mathcal{M}'$=<F, L' := L $\cup$ { l_{new} }, A, *gran'*, *class'*, *attr'*>
   gran': $F \rightarrow 2^{L'}$; gran'(f) := gran(f)
   class' $\subseteq L' \times L'$; $(l_1,l_2) \in$ *class'* :$\Leftrightarrow$ $(l_1,l_2) \in$ *class* $\forall l_1,l_2 \in L'$
   attr': A $\rightarrow F \cup L' \cup \{\perp\}$; *attr'*(a) := *attr*(a)
   No effects on instances because informally the operation introduces a new and therefore empty dimension level. $\Im'_{\mathcal{M}'}$ = <R-UP, C, AV>

2. **delete level**: deletes a dimension level l_{del} from an MD model. The level must not be connected to a fact ($l_{del} \notin$ gran(f) $\forall f \in F$) or via classification relationships ((l_{del}, l) $\notin$ class $\wedge$(l, l_{del}) $\notin$ class $\forall l \in L'$). Further, the level must not have any attributes attached (attr(a) $\neq l_{del}$ $\forall a \in A$). Instances are deleted automatically together with the dimension level. Parameter: level name $l_{del} \in L$.
   $\mathcal{M}'$=<F, L' := L \ { l_{del} }, A, *gran'*, *class'*, *attr'*>.
   gran': $F \rightarrow 2^{L'}$; gran'(f) := gran(f)
   class' $\subseteq L' \times L'$; $(l_1,l_2) \in$ *class'* :$\Leftrightarrow$ $(l_1,l_2) \in$ *class* $\forall l_1,l_2 \in L'$
   attr': A $\rightarrow F \cup L' \cup \{\perp\}$; *attr'*(a) := *attr*(a)
   Instances: no effect because dimension members are deleted automatically. $\Im'_{\mathcal{M}'}$ = <R-UP, C, AV>

3. **insert attribute**: creates a new attribute without attaching it to a dimension level or fact. Assigning an existing attribute to a dimension level or fact is a separate operation (connect attribute). Parameter: attribute name $a_{new} \notin A$ with dom(a_{new}).

$\mathcal{M}'$=<F, L, A' := A ∪ { a_{new} }, *gran*, *class*, *attr'*>
attr': A' → F ∪ L' ∪ {⊥}; *attr'*(a) := *attr*(a)
Instances: no effect, $\mathfrak{I}'_{\mathcal{M}'}$ = <R-UP, C, AV>

4. **delete attribute**: deletes an existing, but disconnected attribute (i.e. the attribute is not attached to a dimension level or fact).
   Parameter: attribute name $a_{del} \in A$
   $\mathcal{M}'$=<F, L, A' = A - { a_{del} }, *gran*, *class*, *attr'*>
   attr': A' → F ∪ L' ∪ {⊥}; *attr'*(a) := *attr*(a)
   Instances: no effect, $\mathfrak{I}'_{\mathcal{M}'}$ = <R-UP, C, AV>

5. **connect attribute to dimension level**: connects an existing attribute a_{new} to a dimension level $l \in$ L. Parameters: attribute name $a_{new} \in$ A; dimension level $l \in$ L, a function g for computing a_{new}. This function can also assign a default value.
   $\mathcal{M}'$=<F, L, A, *gran*, *class*, *attr'*>
   $$attr' : A \to F \cup L \cup \{\perp\} \qquad attr'(a) := \begin{cases} l & if\ a = a_{new} \\ attr(a) & else \end{cases}$$
   Instances: $\mathfrak{I}'_{\mathcal{M}'}$ = <R-UP, C, AV'>, R-UP not changed, C not changed.
   AV': define av_{anew}: dom (l) → dom(a_{new}), AV' := AV ∪ {av_{anew}}
   $$\forall m \in dom(l) : av_{a\,new}(m) := g(m)$$

6. **disconnect attribute from dimension level**: disconnects an attribute a_{del} from a dimension level $l \in$ L. Parameters: attribute name $a_{del} \in$ A; dimension level $l \in$ L.
   $\mathcal{M}'$=<F, L, A, *gran*, *class*, attr'>
   $$attr' : A \to F \cup L \cup \{\perp\} \qquad attr'(a) := \begin{cases} \perp & if\ a = a_{del} \\ attr(a) & else \end{cases}$$
   Instances: $\mathfrak{I}'_{\mathcal{M}'}$ = <R-UP, C, AV'>, R-UP not changed, C not changed.

   AV': let av_{adel} be the corresponding attribute value function for a_{del}, AV' := AV - { av_{del} }.

7. **connect attribute to fact:** connects an existing attribute a_{new} to a fact $f \in$ F. Parameters: attribute name $a_{new} \in$ A; fact $f \in$ F, a function g for computing a_{new}.
   $\mathcal{M}'$=<F, L, A, *gran*, *class*, *attr'*>
   $$attr' : A \to F \cup L \cup \{\perp\} \qquad attr'(a) := \begin{cases} f & if\ a = a_{new} \\ attr(a) & else \end{cases}$$
   Instances: $\mathfrak{I}'_{\mathcal{M}'}$ = <R-UP, C', AV>, R-UP not changed, AV not changed.
   C := C – {c_f} ∪ {c_f'}; c_f': dom(f) → codom(f) with
   $$c'_f\,(x) := (z_1, \ldots, z_n, z_{n+1})\ with\ (z_1, \ldots, z_n) = c_f\,(x)\ and\ z_{n+1} = g(x)$$

8. **disconnect attribute from fact:** disconnects an existing attribute a_{dis} from a fact $f \in$ F. Parameters: attribute name $a_{dis} \in$ A; fact $f \in$ F.

$\mathcal{M}'$=<F, L, A, *gran*, *class*, *attr'*>

$$attr' : A \rightarrow F \cup L \cup \{\perp\} \qquad attr'(a) := \begin{cases} \perp & \text{if } a = a_{dis} \\ attr(a) & \text{else} \end{cases}$$

Instances: $\mathfrak{I}'_{\mathcal{M}'}$ = <R-UP, C', AV>, R-UP not changed, AV not changed.
C := C – {c_f} ∪ {c_f'}; c_f': dom(f) → codom(f) with

$$c'_f(x) := (z_1, \ldots, z_{n-1}) \text{ with } (z_1, \ldots, z_{n-1}, z_n) = c_f(x)$$

9. **insert classification relationship**: this operations defines a classification relationship between two existing dimension levels. Parameters: levels $l_1, l_2 \in L$
$\mathcal{M}'$=<F, L, A, *gran*, *class'*, *attr*>
class' = *class* ∪ {(l_1,l_2)}
Instances: $\mathfrak{I}'_{\mathcal{M}'}$ = <R-UP', C, AV>, C not changed, AV not changed.
R-UP' := R-UP ∪ { $r-up_{l1}^{l2}$ }, ∀ m ∈ dom(l_1): $r-up_{l1}^{l2}$ (m):= k, k∈ dom(l_2).

Additionally, $r-up_{l1}^{l2}$ (dom(l_1))⊆ dom(l_2), i.e. $r-up_{l1}^{l2}$ is well-defined ∀ m ∈ dom(l_1).

10. **delete classification relationship**: removes a classification relationship without deleting the corresponding dimension levels. Parameter (l_1,l_2)∈ *class*
$\mathcal{M}'$=<F, L, A, *gran*, *class'*, *attr*>
class' = *class* – {(l_1,l_2)}
Instances: $\mathfrak{I}'_{\mathcal{M}'}$ = <R-UP', C, AV>, C not changed, AV not changed.
R-UP' := R-UP - { $r-up_{l1}^{l2}$ }

11. **insert fact:** this operation extends the MD model by a new fact. The operation extends the set of facts without attaching dimension levels to this fact. Dimensions for this fact have to be defined separately. Parameter: new fact $f_{new} \notin F$
$\mathcal{M}'$=<F' := F ∪{f_{new}}, L, A, *gran'*, *class*, *attr'*>

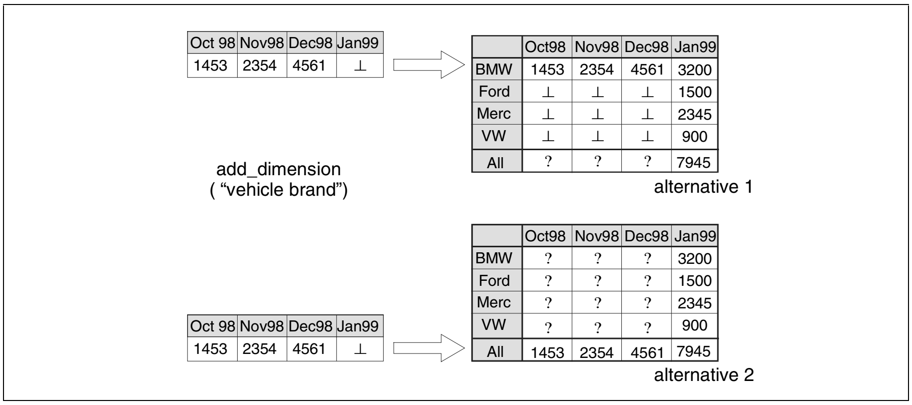
$$gran'(f) : F' \rightarrow 2^{L'} \qquad gran'(f) := \begin{cases} \emptyset & \text{if } f = f_{new} \\ gran(f) & \text{else} \end{cases}$$

attr': A → F' ∪ L ∪ {⊥}; *attr*'(a) := *attr*(a)
Instances: $\mathfrak{I}'_{\mathcal{M}'}$ = <R-UP, C', AV>, R-UP not changed, AV not changed.
C': C ∪ { $c_{f_{new}}$ }, $c_{f_{new}}$: dom (f_{new}) → codom(f_{new}), define c(x):= ⊥ ∀ x∈ dom(f_{new})

12. **delete fact:** removes a fact f_{del} from an MD model. The fact must not be connected to a dimension (gran(f_{del})=∅) and must also not contain any attributes (*attr*(a) ≠ f_{del} ∀ a∈ A). Parameter: name of fact to be deleted f_{del} ∈ F
$\mathcal{M}'$=<F' := F -{f_{del}}, L, A, $gran|_{F'}$, *class*, *attr'*>
attr': A → F' ∪ L ∪ {⊥} *attr*'(a) = *attr*(a)
Instances: $\mathfrak{I}'_{\mathcal{M}'}$ = <R-UP, C', AV>, R-UP not changed, AV not changed.
C': C - { $c_{f_{del}}$ }

13. **insert dimension into fact**: inserts a dimension at a given dimension level into an existing fact, thus increasing the number of dimensions by one. Parameters: level name $l \in L$ and fact name $f_{ins} \in F$. Additionally, a function *nv* is provided defining how to compute the new values for the fact based upon the now extended set of dimensions and the old value of the fact. Each cell of the old cube now becomes a set of cells, exactly reflecting the new dimension. This means that each old value of the fact is now related to all elements of the new dimension. For instance, assume we have daily repair cases of cars stored without the brand (i.e. we have no distinction between the brand of cars). Now we want to include the brand meaning that we insert a dimension at the level brand (cf. figure 1).

We have to provide a function that computes the new fact (repair cases by brand) based on the old dimensions (without brand) and the (old) number of repair cases. The old number of repair cases could be repair cases for a specific brand (alternative 1 in figure 1), a summarization over all brands (alternative 2), or other. The idea how the new values can be computed is stored in nv. For example, if we only had BMW cars before, then we would use the old fact value for BMW and $\perp$ for all other cars (because the values cannot be computed, alternative 1). If the old value was a sum over all brands, we could only take this value as a sum, whereas values for the single brands are unknown (corresponding to "?" in figure 1).



Oct 98	Nov98	Dec98	Jan99
1453	2354	4561	⊥

	Oct98	Nov98	Dec98	Jan99
BMW	1453	2354	4561	3200
Ford	⊥	⊥	⊥	1500
Merc	⊥	⊥	⊥	2345
VW	⊥	⊥	⊥	900
All	?	?	?	7945

Oct 98	Nov98	Dec98	Jan99
1453	2354	4561	⊥

	Oct98	Nov98	Dec98	Jan99
BMW	?	?	?	3200
Ford	?	?	?	1500
Merc	?	?	?	2345
VW	?	?	?	900
All	1453	2354	4561	7945

figure 1: different alternatives for the instance adaptation

More formally, the situation before is that $c(x_1,...,x_n)=z$. Now, after insertion of the new dimension we have $c(x_1,...,x_n,x_{n+1}) = y \ \forall \ x_{n+1} \in \ dom(l)$ and thus

$$nv : codom(f)\ x\ dom(l) \rightarrow codom(f), \quad nv(z, x_{n+1}) = y \quad \forall x_{n+1} \in dom(l)$$

Schema: $\mathfrak{M}'$=< F, L, A, *gran'*, *class*, *attr*>

$$gran'(f) : F \rightarrow 2^L \qquad gran'(f) := \begin{cases} gran(f) & for \quad f \neq f_{ins} \\ gran(f) \cup \{l\} & \qquad f = f_{ins} \end{cases}$$

Instances: $\mathfrak{I}'_{\mathfrak{M}'}$ = <R-UP, C', AV>, R-UP not changed, AV not changed.

c_{fins}*' : dom* (f_{ins}) $\rightarrow$ *codom* (f_{ins}),

$$c'_{f_{ins}}(x_1,\dots,x_n,x_{n+1}) = nv(c_{f_{ins}}(x_1,\dots,x_n),x_{n+1}) \quad with\ x_{n+1} \in dom(l)$$

14. delete dimension: deletes a dimension (connected to the fact at level l) from a fact. Parameters: level name $l \in L$ and fact name $f_{del} \in F$. Additionally, an aggregation function agg_z has to be provided which defines how the fact values have to be aggregated over the deleted dimension (e.g. summation).

$\mathcal{M}'$=< F, L, A, *gran'*, *class*, *attr*>

$$gran'(f) : F \rightarrow 2^L \qquad gran'(f) := \begin{cases} gran(f) & for\quad f \neq f_{del} \\ gran(f) - \{l\} & \quad f = f_{del} \end{cases}$$

Instances: $\mathfrak{I}'_{\mathcal{M}'}$ = <R-UP, C', AV>, R-UP not changed, AV not changed.

c_{fdel}*' : dom* (f_{del}) $\rightarrow$ *codom* (f_{del}),

$$c'_{f\,del}(x_1,\dots,x_{n-1}) = agg_{x_n}(c_{f\,del}(x_1,\dots,x_n)) \quad with\ x_n \in dom(l)$$

6 Tool Support for the Evolution Process

When a considerable amount of requirements change for an MDIS that is operational, a new iteration of the development process is initiated (see section 1). First, the new and changed requirements are compiled and documented. The schema evolution process begins when these changed requirements are to be incorporated into the conceptual data model. Our vision is that the warehouse modeller works with the conceptual schema in a design tool based on a graphical representation (e.g. using the ME/R notation [18]). The tool enables the warehouse modeller to successively apply evolution operations $\varphi_1,\dots\ \varphi_n$ to the schema. When the designer is satisfied with the results, he commits his changes. At this time, the system checks the integrity of the resulting model and propagates the changes of the schema and instances to the implementation level, e.g. by transforming the evolution operations to a sequence of SQL commands. Thus, the warehouse modeller does not need to have knowledge about the specific implementation because the tool-supported environment allows him to work purely on the conceptual level.

7 Conclusions and Future Work

We suggested a framework for multidimensional schema evolution that enables the design of a tool supported schema evolution environment on a conceptual level. In this paper, we presented the core of this framework: a schema evolution algebra based on a formal description of multidimensional schema and instances. This algebra offers 14 atomic evolution operations that can be used to build more complex evolution operations. Furthermore, we described how this formalism can be embedded into the iterative tool-based design and maintenance process of multidimensional information systems. The future research work of our group will investigate the automatic propagation of effects to different implementations (e.g. multidimensional, relational) taking into account effects on predefined aggregates, indexing schemes and predefined reports.

References

[1] J. Banerjee, W. Kim, H.-J. Kim, H.F. Korth: *Semantics and Implementation of Schema Evolution in Object-Oriented Databases*, Proc. SIGMOD Conference, 1987.

[2] M. Blaschka: *FIESTA : A Framework for Schema Evolution in Multidimensional Information Systems*, Proc. of 6th CAiSE Doctoral Consortium, 1999, Heidelberg, Germany

[3] E. Baralis, S. Paraboschi, E. Teniente: *Materialized view selection in a multidimensional database*, Proceedings Conference on Very Large Databases, Athens, Greece, 1997.

[4] L. Cabibbo, R. Torlone: *A Logical Approach to Multidimensional Databases.* Proc. EDBT 1998.

[5] A. Datta ,H. Thomas: *A Conceptual Model and an algebra for On-Line Analytical Processing in Data Warehouses*, Proc. WITS 1997

[6] A. Gupta, I. S. Mumick: *Maintenance of Materialized Views: Problems, Techniques, and Applications*, Data Engineering Bulletin, June 1995.

[7] M. Golfarelli, D. Maio, S. Rizzi, *Conceptual design of data warehouses from E/R schemes*, Proc. 31st Hawaii Intl. Conf. on System Sciences, 1998.

[8] M. Golfarelli, S. Rizzi, *A Methodological Framework for Data Warehouse Design*, ACM DOLAP Workshop, Washington 1998

[9] C.A. Hurtado, A.O. Mendelzon, A.A. Vaisman: *Maintaining Data Cubes under Dimension Update*s, Proceedings of the ICDE' 99, Sydney Australia

[10] G. Höfling, *Schema Evolution in Object-oriented Databases*,. PhD Thesis, Technische Universität München, 1996 (In German)

[11] R. Kimball: *Slowly Changing Dimensions,* Data Warehouse Architect, DBMS Magazine, April 1996, URL: http://www.dbmsmag.com

[12] W. Lehner, *Multidimensional Normal Forms,* Proceedings of SSDBM'98, Capri, Italy

[13] W. Lehner: *Modeling Large Scale OLAP Scenarios*. Proc. of the EDBT 98, Valencia Spain http://www6.informatik.uni-erlangen.de/dept/staff/lehner-publications.html

[14] M. Mohania: *Avoiding Re-computation: View Adaptation in Data Warehouses.* Proc. 8th International Database Workshop, Hong-Kong, Springer LNCS, 1997

[15] C. Quix: *Repository Support for Data Warehouse Evolution*. Proc. CAiSE99 Workshop on Design and Management of Data Warehouses (DMDW99), 1999.

[16] E.A. Rundensteiner, A.J. Lee, A. Nica: *On Preserving Views in Evolving Environments*. Proc. of the 4th KRDB Workshop, Athens, Greece, August 1997

[17] C. Sapia: *On Modelling and Predicting Query Behavior in OLAP Systems*. Proc. CAiSE99 Workshop on Design and Management of Data Warehouses (DMDW99), 1999.

[18] C. Sapia, M. Blaschka, G. Höfling, B. Dinter, *Extending the E/R Model for the Multidimensional Paradigm*, Proc. DWDM Workshop (ER98 Conference), Singapore.

[19] C. Sapia, M. Blaschka, G. Höfling, *An Overview of Multidimensional Data Models*, FORWISS Technical Report FR- 1999-001

[20] M. Tresch, *Evolution in Object Databases*, Teubner, Stuttgart, 1995 (In German)

[21] Panos Vassiliadis: *Modeling Multidimensional Databases, Cubes and Cube Operations*. Conference on Statistical and Scientific Databases (SSDBM) 1998