Body, M., M. Miquel, Y. Bédard & A. Tchounikine, 2003, Handling Evolutions in Multidimensional Structures, IEEE 19th Int. Conf. on Data Engineering (ICDE), March 5-8, Bangalore, India, Bangore, Inde

Handling Evolutions in Multidimensional Structures

Mathurin Body
CRG/LISI
Villeurbanne France

Maryvonne Miquel
LISI-INSA de Lyon
Villeurbanne, France
miquel@if.insa-lyon.fr

Yvan Bédard
CRG, Université Laval
Québec, Canada
yvan.bedard@scg.ulaval.ca

Anne Tchounikine
LISI-INSA de Lyon
Villeurbanne, France
atchouni@if.insa-lyon.fr

Abstract

Building multidimensional systems requires gathering data from heterogeneous sources throughout time. Then, data is integrated in multidimensional structures organized around several axes of analysis, or dimensions. But these analysis structures are likely to vary over time and the existing multidimensional models do not (or only partially) take these evolutions into account. Hence, a dilemma appears for the designer of data warehouses: either keeping trace of evolutions therefore limiting the capability of comparison for analysts, or mapping all data in a given version of the structure that entails alteration (or even loss) of data. We propose a novel approach that offers another alternative, allowing to track history but also to compare data, mapped into static structures. We define a conceptual model and provide possible logical adaptations to implement it on current commercial OLAP systems. At last, we present the global architecture that we used for our prototype.

1. Introduction

1.1 DW and OLAP technologies

Data Warehouse is a "subject-oriented, integrated, non-volatile and time-variant collection of data in support of management's decisions" [12]. A data warehouse is a single site repository of information collected from multiple sources. Information in the data warehouse is organized around major subjects and is modeled so as to allow precomputation and fast access to summarized data. "OLAP" as defined in [7,19] refers to analysis functionalities used to explore the data.

Data warehouse has become a leading topic in the commercial world as well as in the research community. Until now, data warehouse technology has been mainly used in business world, in retail or finance areas for examples. The leading motivation is to take benefits from the enormous amount of data that relies in operational databases.

According to Kimball [13], the data-modeling paradigm for a data warehouse must comply with requirements that are profoundly different from the data models in OLTP environments. The data model of the data warehouse must be easy for the end-user to understand and write queries, and must maximize the efficiency of queries. Data warehouse models are called multidimensional models or hypercubes and have been formalized by [5]. They are designed to represent measurable facts or indicators and the various dimensions that characterize the facts. As an example, in a retail area, typical facts are price and amount of a purchase, dimensions being product, location, time and customer. A dimension can be organized in hierarchy, for example the location dimension can be aggregated in city, state, country. The "star schema" models the data as a simple cube, in which the hierarchical relationship in a dimension is not explicit but is rather encapsulated in attributes. The "snowflake schema" normalizes dimension tables, and makes it possible to explicitly represent the hierarchies by separately identifying dimension in its various granularities. At last, when multiple fact tables are required, the "galaxy schema", or "fact constellation" model allows the design of collection of stars.

OLAP architectures adopt a multi-tier architecture (Figure 1). The first tier is a warehouse server, often implemented by a relational DBMS. Data of interest must be extracted from operational legacy databases, cleaned and transformed by ETL (Extraction, Transformation, Loading) tools before being loaded in the warehouse. This step aims to consolidate heterogeneous schema (structure heterogeneity, semantic heterogeneity) and to reduce data in order to make it conform to the data warehouse model (using aggregation, discretization functions…). Then the data warehouse contains high quality, historical, and homogeneous data.

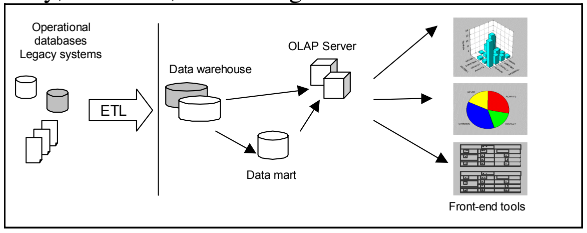

Figure 1: OLAP multi-tier architecture

The second tier is a data mart. A data mart handles data sourced from the data warehouse, reduced for a selected subject. The main advantage of data marts is to isolate data of interest for a smaller scope, thus permitting the focusing on optimization needs for this data and increase

security control. However this intermediate data mart is optional.
The OLAP server composes the 3rd level. It calculates and optimizes the hypercube, i.e. the set of fact values for all the tuples of instances of dimensions (also called members). In order to optimize accesses against the data, query results are pre-calculated in the form of aggregates. OLAP operators permit materializing various views of the hypercube, allowing interactive queries and analysis of the data. Common OLAP operators include roll-up, drill-down, slice and dice, rotate.
The fourth level is an OLAP client, and provides a user interface with reporting tools, analysis tools and/or data mining tools. Software solutions exist for a traditional use. If the studied data and analysis processes are complex, then a specific interface must be designed.

1.2 "Slowly Changing Dimensions"

Multidimensional models [5,13,14] usually consider facts as the dynamic part of data warehouses, and dimensions as static entities. Time is recorded in a Time dimension. Other dimensions are supposed to be time-invariant. However, in many real-life cases, changes occur on the analysis structures. To take such evolutions into account, most of the current OLAP systems map data in the most recent analysis structure. In this case, research focuses on updating models [1,2,10,11]. These models provide a pragmatic way of handling evolutions since they allow the temporal comparison of data and then, drawing conclusions that would not be achievable in a changing analysis structure. But in this case, some data are corrupted, or even lost (e.g. deletion of members that do not exist anymore). Moreover, working only with the latest version hides the existence of evolution and information that may be critical for data analysis. False conclusions may even occur from analysis that does not take into account the explicit evolution of information over time.
As early as 1996, Kimball gave good bases on this issue, by introducing three types of "Slowly Changing Dimensions" [13] (now abbreviated by SCDs) that are in fact three possible ways of handling changes in multidimensional structures. The first one is to update the data structure (as is done by updating models). But as Kimball underlines it, this only "avoids the real goal", which is the tracking of history. As an answer, the Type Two SCD proposes to keep all 'versions' of members. However, in such a representation, comparisons across the transitions cannot be made, since links between them are not kept, even if evolutions are. That is why Kimball proposes a Type Three SCD, in which all evolutions are kept 'inside' members. Nevertheless, limitations exist also for this solution, since overlapping between versions may occur and cannot be handled. It is also "equipped to handle only changes" on members attributes. Although partial, this first study takes into account most users needs and points out the necessity of both keeping track of history and links between transitions.
The remainder of the paper is organized as follows. Section 2 provides the bases of our work, which are motivating examples and a study of related work in this research area. Section 3 formally proposes a conceptual model. In section 4 we present a way of adapting our model to a logical level, which takes existing tools limitations into account. In section 5, we focus on physical level and the implementation of metadata. Finally, we conclude in section 6.

2. Overview of our proposal

In this section, we present examples that motivated our work and an overview of possible changes on multidimensional structures, which both provide guidelines to our work. Then, we consider related work and identify the contribution of our work with respect to the existing approaches.

2.1 Case study

Let us assume that we are studying the restructuring of an institution. We work with a multidimensional schema, having a fact table containing a measure Amount, a Time dimension with hierarchy {year}, and a dimension Organization with hierarchy {division>departement}. For readability, we identify a member of the Department table with the name of its leader and identify a member of the Division table with its name. Let us suppose the data of table 1 for the Organization dimension, Year 2001 and suppose Smith's department is reorganized and moved in the R&D division in Year 2002 (see Table 2).

Division	*Department*
Sales	Dpt.Jones
Sales	Dpt.Smith
R&D	Dpt.Brian

Table 1. The *organization* dimension in 2001

Division	*Department*
Sales	Dpt.Jones
R&D	Dpt.Smith
R&D	Dpt.Brian

Table 2. The *organization* dimension in 2002

Suppose the following snapshot of data:

Year	*Division*	*Department*	*Amount*
2001	Sales	Dpt.Jones	100
2001	Sales	Dpt.Smith	50
2001	R&D	Dpt.Brian	100
2002	Sales	Dpt.Jones	100
2002	R&D	Dpt.Smith	100
2002	R&D	Dpt.Brian	50
2003	Sales	Dpt.Bill	150
2003	Sales	Dpt.Paul	50

2003	R&D	Dpt.Smith	110
2003	R&D	Dpt.Brian	40

Table 3. Snapshot of data for year 2001, 2002, 2003

A query Q1 asking for total amount by year and division could have three different interpretations:
The first one computes amounts in 'Consistent Time'. It returns the following table:

Year	*Division*	*Amount*
2001	Sales	150
	R&D	100
2002	Sales	100
	R&D	150

Table 4. Result of Q1 in consistent time

The second one maps the results with the organization of 2001:

Year	*Division*	*Amount*
2001	Sales	150
	R&D	100
2002	Sales	200
	R&D	50

Table 5. Result of Q1 mapped on 2001 organization

The last one maps the organization at year 2002:

Year	*Division*	*Amount*
2001	Sales	100
	R&D	150
2002	Sales	100
	R&D	150

Table 6. Result of Q1 mapped on 2002 organization

This first query shows that, depending on the interpretation given to a simple request, results and then conclusions may greatly vary and even be contradictory. As an example, Amounts in the Sales Division seem to decrease, increase or be the same depending on the different interpretations. It is therefore primordial to let the end-user choose, but also to guide him, throughout all these different interpretations.

As a second example, let us suppose now that Jones 's department is splitted into Paul's and Bill's departments in 2003:

Division	*Department*
Sales	Dpt.Bill
Sales	Dpt.Paul
R&D	Dpt.Smith
R&D	Dpt.Brian

Table 7. The *organization* dimension in 2003

With the data shown in table 3 the query Q2 asking for the total amounts per department for the years 2002, 2003 has the three following possible results:
In consistent time we obtain:

Year	*Department*	*Amount*
2002	Dpt.Jones	100
	Dpt.Smith	100
	Dpt.Brian	50
2003	Dpt.Bill	150
	Dpt.Paul	50
	Dpt.Smith	110
	Dpt.Brian	40

Table 8. Result of Q2 in consistent time

Mapping the data on 2002 organization we obtain:

Year	*Department*	*Amount*
2002	Dpt.Jones	100
	Dpt.Smith	100
	Dpt.Brian	50
2003	Dpt.Jones	200
	Dpt.Smith	110
	Dpt.Brian	40

Table 9. Result of Q2 on 2002 organization

For the mapping of data on 2003 current organization, we have knowledge on the way Jones's department has been divided. Let us assume that we know that the repartition of the turnover in 2002 is estimated as 40% in Bill's department and 60% in Paul's. The result is the following table:

Year	*Department*	*Amount*
2002	Dpt.Bill	40
	Dpt.Paul	60
	Dpt.Smith	100
	Dpt.Brian	50
2003	Dpt.Bill	150
	Dpt.Paul	50
	Dpt.Smith	110
	Dpt.Brian	40

Table 10. Result of Q2 on 2003 organization

This second example points out that in most cases, if the 'Consistent Time' view gives 'true' data, it does not allow the tracing of data evolution. That is why we need to map data in a given structure version. In this case, note that the "older" version is less detailed but more truthful than the "current" one, in which approximations are made. These different points of view on data are not only different: they also are complementary. Finally, this example also shows that it is essential to be able to distinguish source from mapped data, and to eventually report on data reliability.

2.2 Related work

Recently, literature has brought forward the problem of evolutions in the multidimensional structures and new models have been proposed to handle some of these issues. Some of them, the updating models [1,2,10,11], focus on mapping data into the most recent version of the structure, whereas tracking history models [3,6,9,15,18] keep trace of evolutions of the system. Among these tracking history approaches, some [3,6] choose to represent data in the temporally consistent mode of presentation. They therefore present the same limits as the Type Two SCD of Kimball, i.e. to not be able to draw comparisons across time. Only the three most recent models of Mendelzon and Vaisman [15], Pedersen et al. [18] and Eder and Koncilia [9], aware of the users needs of both accurately tracking history and comparing data, provide a way of mapping data in an "unchanged structure", chosen by the user.

The model of Eder and Koncilia [9] proposes mapping functions that allow conversions between structure versions. These mapping functions are used to store the links across transitions (e.g. links between an 'old' member and its new version). They are based on knowledge around evolution operations. Yet, it provides a partial solution, which neither takes schema evolution and time consistent presentation into account, nor considers complex dimension structures.

Pedersen et al. [18] propose a conceptual model focusing on imprecision and complex dimension structures. Handling evolutions is considered as one aspect. Their approach is however closed to the one previously proposed by Mendelson and Vaisman.

The approach of Mendelzon and Vaisman [15] gives a good overview of the end-user needs. The authors define a temporal multidimensional model that reuses contributions of the temporal databases research area: mainly, the timestamps on the elements of their multidimensional database. Using these valid times, they build the TOLAP Query Language that lets the user choose in his request the way he wants data to be aggregated. He can therefore choose between a temporally consistent representation and the last version. Moreover, they also extend their model to track the links between transitions, and then be able to handle merging and splitting evolutions. This can be seen as the equivalent of the mapping functions of Eder and Koncilia. Nonetheless, their model does not provide the means of reporting data in any other version than the latest one and to only partially take merges and splits evolutions into account, in comparison with the approach of Eder and Koncilia.

2.3 Our approach

In light of the above, the Mendelzon and Vaisman, and Eder and Koncilia approaches seem complementary. The first one lays down bases for handling evolutions in multidimensional structures, whereas the second one gives solutions to exploit knowledge on evolutions in order to map data in a given representation. Using these notions, we introduce a conceptual temporal multidimensional model and a way of translating it into a pragmatic logical model.

In order to take into account all changes in multidimensional structure, we first draw up a taximomy and we divide possible evolutions into schema evolution and evolution on instances of dimensions.

The following changes may occur on a multidimensional schema:

- Creation and deletion of a dimension.
- Creation and deletion of a hierarchy.
- Creation and deletion of a level.
- Move of a level in the hierarchical schema structure.
- Creation and deletion of a measure.

Some may also add creation and deletion of a member attribute, depending on their point of view and model.

The evolutions on instances of dimensions are more difficult to list exhaustively. Hence, we choose to introduce six simple operations that could be combined to build complex operations. The simple operations are:

- Creation of a dimension member.
- Deletion of a dimension member.
- Transformation of a member (change of an attribute, its name or meaning…).
- Merging of n members into one member.
- Splitting of one member into n members.
- Reclassification of a member in the dimension structure.

Some examples of complex operations can be:

- Decreasing: splitting followed by a deletion.
- Increasing: creation followed by a merging.
- Partial annexation: splitting followed by a merging.

We think that as long as a model will not clearly take all these changes into account the problematic of evolution in multidimensional structures will remain unsolved.

In our model, hierarchical levels are deduced from the dimensions instances. It is therefore no longer necessary to previously define the schema structure. Two main reasons lead us to this choice. First, Pedersen et al. [18] point out the fact that the current multidimensional models do not support complex hierarchical structures. By not imposing any explicit schema structure, our model greatly gains in genericity. Indeed, we can easily handle non-onto, non-covering and multiple hierarchies [18].

Second, every evolution on schema has necessarily an impact on its instance. With such an approach, we take into account both evolutions on schema and instances, at the same time: changes on schemas will be treated by the changes they imply on their instances.
Furthermore, the biggest advantage of having explicit hierarchies in the dimension lies in the better guidance for the user navigating in the cube that it gives. But it turns out that our approach does not in any way restrict these navigation possibilities since, as we will show, schemas structure can be deduced from dimension instances.

3. Conceptual-Level Approach

In this section, our temporal multidimensional model is developed in detail. We then introduce the evolution operators that allow one to take the evolutions of multidimensional structures into account.

3.1 Temporal multidimensional model

As most of the proposed multidimensional models, our approach is based on a fact table and dimensions. We redefine these elements with valid times and add the notions of MultiVersion Fact Table and mapping relationships.

Definition 1 (Member Version). A Member is an object or an abstract entity of special interest for the end-user. Because changes may occur on this member, a **Member Version** can therefore be seen as a state of a member, unchanged and coherent over a given time slice. It is represented by a tuple *<MVid, Name, [A], [Level],* t_i, t_f>, where *MVid* is a unique identifier for each Member Version, *Name* is the name of the associated member. *A* may be a set of user-defined attributes and values for this Member Version. *Level* is optional and could be used to explicitly define the schema structure. Finally, $[t_i, t_f]$ defines the valid time of this Member Version.
Notice that a Member may have several valid Member Versions for a given time (when valid times overlap). Therefore, there is no need of accurate history partitions (as was needed in Type Two Slowly Changing Dimensions of Kimball).

Example 1. In the motivating example we presented earlier, we have a department *Dpt.Jones* that is split into *Dpt.Paul* and *Dpt.Bill*, on and after 2003. In our approach, we have three member versions:
<Dpt.Jones _id, 'Dpt.Jones', Department, 01/2001, 12/2002>, *<Dpt.Paul_id, 'Dpt.Paul', Department, 01/2003, Now>* and *<Dpt.Bill_id, 'Dpt.Bill', Department, 01/2003, Now>*.

Definition 2 (Temporal Relationship). A **Temporal Relationship** establishes an explicit hierarchical link between two Member Versions and represents a rollup function. It consists of a tuple *<Id_from, Id_to, ti, tf>* where *Id_from* is the identifier of the Member Version child in the relationship and *Id_to* is the one of the Member Version, parent in it. $[t_i, t_f]$ gives the valid time of this relationship. Note that this valid time has to be included in the intersection of the valid times of both Member Versions.

Example 2. Keeping our example, let us now suppose that our three departements belong to the Division *Sales* defined by the Member Version *<Sales_id, 'Sales', Division, 01/2001, Now>*. We then have the following temporal relationships: *< Dpt.Jones_id, Sales_id, 01/2001, 12/2002>*, *< Dpt.Paul_id, Sales_id, 01/2003, Now>* and *< Dpt.Bill_id, Sales_id, 01/2003, Now>*.

Definition 3 (Temporal Dimension). A **Temporal Dimension** $\mathcal{D}$ is a tuple $<\mathcal{D}id, \mathcal{D}name, D, G>$ where $\mathcal{D}id$ is a unique identifier for the dimension and $\mathcal{D}name$ its name. *D* is a set of Member Versions and *G*, a set of Temporal Relationships, defining the hierarchical structure of this dimension. Then, a Temporal Dimension can be seen as a directed graph, where nodes are Member Versions of *D* and arcs are relationships of *G*.
For any instant *t*, let $\mathcal{D}(t)$ be $<\mathcal{D}id, \mathcal{D}name, D(t), G(t)>$ where $D(t) = \{d \in D \mid t \in [t_i^d ; t_f^d]\}$ and $G(t) = \{g \in D \mid t \in [t_i^g ; t_f^g]\}$. More explicitly, *D(t)* (resp. *G(t)*) is the restriction of *D* (resp. *G*) to its elements valid at time *t*. Then, $\mathcal{D}(t)$ must be a Directed Acyclic Graph (DAG) that represents the dimension structure at this instant *t*.
In the rest of the paper we will call Leaf Member Versions, all Member Versions that have no children at, at least, one instant.

Example 3. In our example, we can define a *Org* dimension as *< Org_id, Org, D, G>* where *D* only contains the three department member versions and the division *Sales*, and *G* only has the three temporal relationships previously defined.

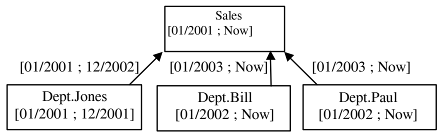

Figure 2. The *Org* Dimension.

Definition 4 (Levels in a dimension). Given a Temporal Dimension $\mathcal{D}$, a **level** in this dimension is a set of Member

Versions that may be defined in two ways. First, if the optional *Level* field is given for all Member Versions of $\mathcal{D}$, then the levels are defined by the sets of Member Versions having the same value for this field. Levels are actually equivalence classes of D, for the relation "has same *level* field as". Otherwise, if this option is not defined for all Member Versions, levels are the set of Member Versions of same depth in the DAG of $\mathcal{D}(t)$. In both cases, note that the notion of level emerged from the instances of $\mathcal{D}$ and that it evolves then over time.

Example 4. In our trivial case, the *Org* dimension schema contains two levels: *Department*, with the three departments Dpt.Jones, Dpt.Paul, and Dpt.Bill and *Division* with the member version Sales.

Definition 5 (Temporally Consistent Fact Table). Given n temporal dimensions $\mathcal{D}_1, \ldots, \mathcal{D}_n$, a Time Dimension T and a set of m measures $M=\{m_1, \ldots, m_m\}$, a **Temporally Consistent Fact Table** f can be seen as a function :

$$f : D_1 \times \ldots \times D_n \times T \to dom(m_1), \ldots, dom(m_m)$$
$$d_1, \ldots, d_n, t \mapsto v_1, \ldots, v_m$$

where D_i is the set of Member Versions of each dimension $\mathcal{D}_i$ and $dom(m_k)$ is the range for the measure m_k. This function associates a set of Leaf Member Versions d_i, valid at time t, with the values v_k obtained for the measure m_k.

Furthermore, in order to keep the links across transitions, i.e. between Members Versions, we have to introduce Mapping Relationships and Confidence Factor. This will be used to build a Multiversion Fact Table.

Definition 6 (Confidence Factor). A **Confidence Factor** is a value that describes the reliability of data. It allows to distinguish source from mapped data.
Moreover, an aggregate function $\otimes_{cf}$ has to be defined by the designer to obtain this notion of confidence factor for aggregated data in the cube. This aggregate function can either be defined by a function, in case of quantitative Confidence Factors, or by a truth table, if Confidence Factors are given in a qualitative way.

Example 5. In our example, we define the following range of Confidence Factors: CF={*sd*, *em*, *am*, *uk*} where *sd* stands for source data, or temporally consistent data. *em* and *am* mean respectively exact and approximated mapped data, in order to translate that, and how, data is mapped. Finally, *uk* indicates that the mapping relationship is unknown. We define the following truth table as the aggregate function:

$\otimes_{cf}$	*sd*	*em*	*am*	*uk*
sd	*sd*	*em*	*am*	*uk*
em	*em*	*em*	*am*	*uk*
am	*am*	*am*	*am*	*uk*
uk	*uk*	*uk*	*uk*	*uk*

Definition 7. (Mapping Relationship). A Mapping Relationship is defined as a tuple <*Id_from, Id_to, F,* F^{-1}>, where *Id_from* is the Leaf Member Version identifier of the Member before changing, and *Id_to*, the identifier of the one after changing. F is a set of m pairs <fm_k, cf_k> where fm_k is a mapping function from $dom(m_k)$ into itself that specifies how the measure m_k must be mapped ; and cf_k is a confidence factor for this mapping function. Finally, F^{-1} is also a set of m pairs <fm'_k, cf'_k>, describing the reverse mapping from *Id_to* to *Id_from*. This lets us deal with the problem exposed by Kimball in his Type Three Slowly Changing Dimensions: keeping links between transitions.
Notice that these mapping relationships are only relevant for Member Versions of the Temporally Consistent Fact Table, i.e. Leaf Member Version. For all non-Leaf Member Versions, mappings will be calculated from the aggregation of their children values (eventually mapped).
Note also that Confidence Factors are linked here to mapping functions. They will be associated to data later on, once data have been mapped using the mapping functions.

Example 6. Until now, we didn't have any relationship between our different departments. We were then unable to map measures from one to another. To solve this problem, let us introduce the following mapping relationships: <*Dpt.Jones_id, Dpt.Bill_id,* {($x \mapsto 0.4x$, *am*)}, {($x \mapsto x$, *em*)}> and <*Dpt.Jones_id, Dpt.Paul_id,* {($x \mapsto 0.6x$, *am*)}, {($x \mapsto x$, *em*)} >, which mean that all values obtained for Dpt.Bill or Dpt.Paul will be mapped (just as they are and with the em confidence factor) to Dpt.Jones, whereas data associated to Dpt.Jones will approximately be reported to Dpt.Bill and Dpt.Paul, by a given proportion.

Definition 8 (Temporal Multidimensional Schema). A **Temporal Multidimensional Schema**, denoted $\mathcal{TMD}$, is a tuple <{$\mathcal{D}_1, \ldots, \mathcal{D}_n, T$}, $\mathcal{MR}$, f>, where $\mathcal{D}_i$ are temporal dimensions, T is the Time Dimension, $\mathcal{MR}$ is a set of Mapping Relationships and f is a Temporally Consistent Fact Table.

Definition 9 (Structure Version) . A **Structure Version** $\mathcal{V}$ of a Temporal Multidimensional Schema $\mathcal{TMD}$ is a tuple <VS_{id}, {$\mathcal{D}_{1,VSid}, \ldots, \mathcal{D}_{n,VSid}$}, t_i, t_f>, where VS_{id} is a unique identifier and $[t_i, t_f]$ defines the valid time of this Structure Version. Each $\mathcal{D}_{i,Vsid}$ is the restriction of $\mathcal{D}_i$ to its elements (Member Versions and Temporal Relationships) valid for all t in $[t_i, t_f]$. We can see a Structure Version as a valid and unchanged structure over its given valid time. Note that the Structure Versions of a given Temporal Multidimensional Schema $\mathcal{TMD}$ partition history and that they can be inferred from $\mathcal{TMD}$ Schema, as the intersections of the valid time intervals of all Member Versions and Temporal Relationships.

Example 7. To illustrate this definition, let us pursue our case study. With the evolution of the department Dpt.Jones, note that we obtain two Structures Versions: the first one laying from 01/2001 to 12/2002, in which only the department Dpt.Jones exists; the second one laying from 01/2002 to Now, and where Dpt.Paul and Dpt.Bill are valid.

As reported in our analysis of end-user needs, multidimensional requests may have several ways of presentating the information: either in the temporally consistent mode or in one of the available Structure Versions. That is why we introduce here the concept of Temporal Mode of Presentation.

Definition 10 (Temporal Mode of Presentation). Given N structure versions $\mathcal{V}_1, \ldots, \mathcal{V}_N$ (deduced from a Temporal Multidimensional Schema $\mathcal{TMD}$), we define $TMP = \{tcm, \mathcal{VM}_1, .., \mathcal{VM}_N\}$ as the set of **Temporal Modes of Presentation** for the data. *tcm* stands for the temporally consistent mode of presentation, and $\mathcal{VM}_i$ denotes the temporal mode where data is mapped into the structure version $\mathcal{V}_i$.

Example 8. In our example, we have three temporal modes of presentation: the two structure versions we earlier detected, plus the temporally consistent mode, *tcm*.

Definition 11 (MultiVersion Fact Table). Given a Temporal Multidimensional Schema and its associated set *TMP* of Temporal Modes of Presentation, the MultiVersion Fact Table is a function f' such as:

$$\begin{aligned} f' : D_1 \times \ldots \times D_n \times T \times TMP &\to dom(m_1) \times \ldots \times dom(m_m) \times CF^m \\ d_1, \ldots, d_n, t, tmp &\mapsto v_1, \ldots, v_m, cf_1, \ldots, cf_m \end{aligned}$$

where *CF* is the range of the Confidence Factors. This function associates a set of values v_k and their respective confidence cf_k, to a set of Leaf Member Versions d_i, valid for the given mode *tmp* (but not necessarily for the given time t, if *tmp* is different than the temporally consistent mode), a time t and a given Temporal Mode of Presentation *tmp*.

Note that the Temporally Consistent Fact Table is included into this MultiVersion Fact Table. We have indeed:

$$f'\big|_{D_1 \times \ldots \times D_n \times T \times \{tcm\}} = f \times \{sd\}^m$$

where *tcm* is the temporally consistent mode of presentation and *sd* is the confidence for source data.

This MultiVersion Fact Table can be inferred from the Temporal Multidimensional Schema. It can then be automatically calculated from the temporal dimensions, Mapping Relationships and the Temporally Consistent Fact Table.

Definition 12 (Data Aggregation). Suppose given an aggregate function $\oplus_{m_k}$ for each measure m_k, $\otimes_{cf}$ the aggregate function of the Confidence Factors, *tmp* an element of *TMP* and t an instant of the time axis. **Data Aggregation** in the cube can easily be processed, using the MultiVersion Fact Table and Temporal Relationships between Members Versions. As an example, let us assume that we want to aggregate data for a non-Leaf Member Version d from the dimension $\mathcal{D}_1$. From G_1, set of temporal relationships of $\mathcal{D}_1$, we can find $d^1, \ldots, d^J$ the children of d. Let us also suppose that we have :

$\forall j \in [1, J], \quad f'(d^j, d_2 \ldots, d_n, t, tmp) = v_1^j, \ldots, v_m^j, cf_1^j, \ldots, cf_m^j$

We can then obtain the aggregated values for d as:

$$f'(d, d_2, \ldots, d_n, t, tmp) = \bigoplus_{m_1}{}_{j=1}^{J} v_1^j, \ldots, \bigoplus_{m_m}{}_{j=1}^{J} v_m^j, \bigotimes_{cf}{}_{j=1}^{J} cf_1^j, \ldots, \bigotimes_{cf}{}_{j=1}^{J} cf_m^j$$

Such operations will have to be proceeded in order to build an OLAP cube.

3.2 Structural Evolution Operators

To modify the structure of a Temporal Multidimensional Schema, we provide four basic operators: *Insert*, *Exclude*, *Associate* and *Reclassify*. The administrator integrates changes in the structure by means of these operators.

Insert($\mathcal{D}$id, mvID, mName, [A], [level], t_i, [t_f], P, C) inserts a new Member Version *mvID* with the name *mName* in dimension $\mathcal{D}$*id* as <*mvID, mName, [A], [level],* t_i*, [*t_f*]*>. Its position into the hierarchical structure of the dimension is specified by *P* and *C*, respectively set of its parents and children. Temporal Relationships are created from t_i [to t_f] to integrate this change. Note that t_f is optional and, if omitted, is replaced by *Now*.

Exclude($\mathcal{D}$id, mvID, t_f) excludes the Member Version *mvID* from dimension $\mathcal{D}$*id* on and after t_f. It means that the end time of *mvID* and all Temporal Relationships involving this Member Version is set to t_f*-1*.

Associate($\mathcal{R}$map). $\mathcal{R}$*map* is a Mapping Relationship that defines a new transition link between two versions of a member. It will simply be checked for consistency and added to $\mathcal{MR}$, the set of Mapping Relationships.

Reclassify($\mathcal{D}$id, mvID, t_i, [t_f], OldParents, NewParents) changes the position of member version *mvID* in the hierarchical structure of dimension $\mathcal{D}$*id,* by redefining the set of its parents. *NewParents* (resp. *OldParents*) is a set of the member versions that are (resp. are no more) parents of *mvID* on and after t_i. This operator updates the end time of all temporal relationships concerned (i.e. involving member versions from *OldParents* and *mvID*) and/or inserts new ones.

Note that *OldParents* or *NewParents* may be empty.

Using these four basic operators, we cover most of the evolution operations introduced in 2.2. By combining them we are indeed able to translate simple and complex operations into a sequence of basic operators. Examples of such translations are given in Table 11. We use the confidence factor range introduced in example 5.

Simple Operations
Creation of *V* at time *T* in the dimension *Org* as a Child of *P1*:
- Insert(Org, idV, V, T, {idP1},∅)
Change from *V* to *V'* at time *T* and in the dimension *Org (equivalence Relationship)*
- Exclude(Org, idV, T)
- Insert(Org, idV', V', T, {idP1},∅)
- Associate(idV, idV',{(x→x,em)}, {(x→x,em)})
Merge of *V1* and *V2* into *V12* at time *T*. Half of the values obtained for *V12* will be mapped for *V1*, with approximation and mapping is unknown from *V12* to *V2*:
- Exclude(Org, idV1, T)
- Exclude(Org, idV2, T)
- Insert(Org, idV12, V12, T, {idP1},∅)
*-Associate(idV1,idV12,{(x→x,em)}, {(x→0.5*x,am)})*
- Associate(idV2, idV12,{(x→x,em)}, {(-,uk)})
Complex Operations
Increase *V* in V^+ at time *T*. Values increase with a factor 2, with *approximation*
- Exclude(Org, idV, T)
- Insert(Org, idV^+ V^+*, T, {idP1},∅)*
- Associate(idV,idV^+*, {(x→2.x,am)}, {(x→0.5*x,am)})*
Partial Annexation of a portion of *V1* to *V2* at time *T*. (10% of the measure of V1 will go for V2, what is an increasing of 20% for V2, with *approximation*)
- Exclude(Org, idV1, T)
- Exclude(Org, idV2, T)
- Insert(Org, $idV1^-$*,* $V1^-$*, T, {idP1},∅)*
- Insert(Org, $idV2^+$*,* $V2^+$*, T, {idP1},∅)*
- Associate(idV1, $idV1^-$*, {(x→0.9*x,am)}, {(x→x,em)})*
- Associate(idV2, $idV2^+$*,{(x→x,em)}, {(x→0.8*x,am)})*
-Associate(idV1,$idV2^+$*,{(x→0.1*x,am)},{(x→0.2*x,am)})*

Table 11. Examples of simple and complex operations

Therefore, we handle all possible changes on dimension instances. Moreover, as underlined in section 2.4, we choose a 'bottom-up' approach for hierarchical schemas that allows us to handle evolutions on schema and on instances in the same time. There is consequently no more need to introduce new operators dealing with schema evolutions. For example, introducing (resp. deleting) a level is equivalent to creating (resp. excluding) the members of this level.

4. Logical-Level Approach

In this section we focus on how to adapt our conceptual approach to a logical-level, by taking into account current commercial tools constraints. This allows us later on, to implement our model on existing commercial systems.

4.1 Logical temporal multidimensional model

Current multidimensional systems are only made of dimensions and fact table(s). Therefore, integrating our notions of temporal mode presentation and the confidence factor in commercial tools seems difficult. In definition 11, the temporal mode of representation is similar to a dimension and the confidence factors appear as a measure. In the logical level, we represent the set *TMP* as a 'flat' dimension, i.e. without hierarchical structure. This choice offers all the flexibility provided by a usual dimension, during cubes exploration (comparing different structure versions, switching between temporal modes, rotating…). Each confidence factor, which is characterizing a value, may be seen as a measure in the fact table, associated to the same members in the multidimensional structure. Moreover, the aggregate function becomes a 'usual' aggregate function, as it must be defined for all measures in a multidimensional structure.

4.2 Adaptation of Evolution operators

In our conceptual model we clearly distinguished in a dimension the member (or member version) from its relationships. It is indeed of prime necessity to do so, since changes may occur on a structure independently of members (reclassification operation as described in section 2.2). But for implementation, none of current commercial tools, as far as we know, supports such an approach. Hierarchical links are always stored as foreign keys in children attributes. We can nonetheless circumvent this problem by interpreting a change in the hierarchical structures as a change in this hierarchical-link attribute (i.e. as a transformation operation as defined in section 2.2).

For example, if department D1 is reclassified from Division Di1 to Di2, we create two versions of D1, the first one having Di1 in his hierarchical-link attribute, and the second one, having Di2. These two versions are linked by an equivalence mapping relationship, translating that data are mapped as source data.

Nonetheless, such an adaptation is not without consequences. For example, if a change occurs on a non-leaf member version, then all its descendants have to be changed into new versions, to reflect the evolution of its parent(s) and therefore the evolution of their hierarchical-link attribute. We underline that this solution is not satisfying but could be avoided by designing an appropriate physical model for our multidimensional structure.

Adopting this solution, the *Reclassify* operator becomes useless and has to be redefined. Then, *Reclassify*($\mathcal{D}$*id, mvID,* t_i*,* [t_f]*, OldParents, NewParents)* is treated as:

- *Insert(*$\mathcal{D}$*id, mvID', mvName, [A], [level],* t_i*,* [t_f]*, P', E)* where, if P is the set of the parents of *mvID*, *P'=(P-OldParents)*$\cup$*NewParents* and *E* is the set of the children of *mvID*. If *E* is not empty then, as we highlight it earlier, each element of *E* has to be reclassified recursively to the new version *mvID'*.
- *Exclude(*$\mathcal{D}$*id, mvID,* t_i*)*
- *Associate*$(mvID, mvID', \bigcup_{k=1}^{m}\{(x \rightarrow x, sd)\}, \bigcup_{k=1}^{m}\{(x \rightarrow x, sd)\})$ where $(x \rightarrow x)$ means that the mapping function is the identity function and *sd* is the confidence for source data.

These adaptations provide a way of implementing our conceptual model on current commercial tools. It can therefore be implemented either on ROLAP, MOLAP or HOLAP servers, just as all usual models. The pros and cons of these architectures is then out of our concern.

5. Physical level approach

5.1.Architecture

To implement our model on existing tools (SQL Server 2000 Analysis Services, OLE DB for OLAP [16] and ProClarity 4.0 Components) we choose an architecture divided into three parts:

- A Temporal Data Warehouse, that contains the Temporal Multidimensional Schema (temporally consistent data), and metadata related to it, including Mapping Relationships.
- A MultiVersion Data Warehouse in which the 'temporal mode of presentation' dimension has been proceeded and the MultiVersion fact table has been inferred from the Temporally Consistent fact table and Mapping Relationships.
- An OLAP cube built from the MultiVersion Data Warehouse, using aggregations, and that allows requests to integrate the temporal modes of presentation concept.

Up to now, to make our system run on current OLAP tools we have to duplicate the values in all versions. This obviously implies a high level of useless redundancies inside the MultiVersion Data Warehouse and the OLAP Cube, since we could only store differences between versions instead of replicating all values.

The most recent commercial tools now handle several ways of structuring dimensions into data warehouses. Most of them propose either a "normalized" representation where levels are stored in distinct relational tables, corresponding to the snowflake schema, or a "denormalized" representation, corresponding to the star schema, and where the whole dimension is stored in a single table. Some also propose a third type of data structure that is of first interest in our case. It is called Parent-Child Dimension in Microsoft SQL Server 2000 [17] and denotes a data structure where dimension do not have explicit hierarchy and where hierarchy is only deduced from links between members. Such a data structure presents the advantage of being close to our conceptual model and allows us to deal with most of the evolutions over dimensions schemas. Nevertheless some limitations exist with such a physical representation: multi-hierarchies are not supported. Designer wishing to implement our conceptual model on existing commercial tools will therefore have to choose between handling multi-hierarchy (and therefore using normalized or denormalized data structure) or evolutions on schema (by means of the Parent-children data structure).

5.2 Implementing the meta-data

As seen before, the metadata is introduced in the temporal data warehouse and used at various stages of the design of the multiversion structure. We distinguish two main categories of metadata:

- metadata related to the versions of members
- metadata related to the evolution of the members.

This choice simplifies the building of the multiversion facts table.

The first category of metadata is stored in the relational dimensions tables and is then integrated in the hypercube. Thus, the user can have information on the temporal interval of validity of a member version, the name of its associated member, its position in the hierarchy of dimension, for each member version.

The second type of metadata relates to information on the evolution of the members' versions, in particular the mapping relations and their associated functions. In the prototype, the mapping functions are introduced as linear functions, i.e. $f(x) = k.x$ *(k∈ ℜ).*

The *k* factor represents a percentage or a weighting of a given measure. As an example, suppose that Jones's department is splitted into Paul's and Bill's departments. For the *Turnover* measure (noted *m1*), one estimate that 60 % of the values obtained by Jones's department is affected to Paul's and 40 % to Bill's. Let us suppose a *Profit* measure (noted *m2*), whose k factor is estimated at 0.8 for Paul's department and 0.2 for Bill's department.

In the prototype, we do not affect a confidence factor for each mapping function but only for each mapping relation (and its symmetrical). The confidence factors are qualitative and are coded as follows:

- 3 for the source data
- 2 for the exact mapped data
- 1 for the approximated mapped data
- 4 for the data with unknown mapping function.

Then, it is not necessary to repeat the same factor for each function. Table 12 is a snapshot of the table where each tuple represents a mapping relation.

From	To	*k* for m1	*k* for m2	k^{-1} for m1	k^{-1} for m2	Confi-dence	Confi-dence^{-1}
Dpt.Jones	Dpt.Paul	0.6	0.8	1	1	1	2
Dpt.Jones	Dpt.Bill	0.4	0.2	1	1	1	2

Table 12 : Table of mapping relations between version members (extract)

The table of mapping relations is used to build the multiversions facts table. It is also used to find the metadata related to calculations of a mapped measure. The user has a direct access to very precise information on the way the data were calculated and on the factors applied in conversions.

Information related to the evolution of the members of a dimension is stored in a similar way and the user can obtain a short textual description of the transformations that have affected a member (Leaf Member Versions or not)

The metadata is also used to help navigation. The background color of the cell showing a measure can be used to represent the associated confidence level (ex. white for source data, green for exact mapping, yellow for approximated mapping and red for impossible cross-point in the grid). The quality of a version is also computed. Once a request is built, we calculate for each temporal mode, a global quality factor as follows:

$$Q = \left(\sum_{i}^{Ni} \sum_{j}^{Nj} pds(fb(i,j)) \right) / (Ni * Nj * 10)$$

where pds() is a function pondered by the user who assigns to each confidence a weight ranging between 0 (weakest) and 10 (best). Ni and Nj are respectively the number of lines and columns in the table, result of request R. The user can choose his best version among all temporal modes of presentation, according to its own criteria of quality.

6. Conclusion

We have presented a novel temporal multidimensional model for supporting evolutions on multidimensional structures. On the one hand, our attention is on the designer needs, which are mainly: handling most of the possible evolution operations and allowing to have non trivial hierarchical structures in dimensions, to match with real-life cases and as was pointed out by Pedersen et al [18]. On the other hand, we consider query semantics and how to meet user needs, facing this problem of changes in their analysis framework. In this way we have introduced the notion of temporal mode of presentation to let the user choose which interpretation he wants to give to his request. We have also provided confidence factors that allow the user to detect at a glance, values that are mapped, and then altered somehow, from those that come from source data. Moreover, we extended this notion to allow it to take into account approximated, exact and unknown mappings. Meat data is used at various stages of the design. With respect to users needs, we extract information to guide him and help him to deal with evolutions in analysis structures.

Our model still suffers from the fact that a structure version is composed of the set of the temporal dimensions validated for that version. An improvement would allow the building of a structure version by selecting the temporal dimensions in different versions.

7. References

[1] M. Blaschka, C. Sapia and G. Höfling, "On Schema Evolution in Multidimensional Databases", *Proceedings of DaWak'99 Conference*, Florence, Italy, 1999.

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

[3] R. Bliujute, S. Saltenis, G. Slivinskas, C.S. Jensen, "Systematic Change Managment in Dimensional Data Warehousing", *Proceedings of the 3rd International Baltic Workshop on DB and IS*, 1998.

[4] M. Body, M. Miquel, Y. Bedard, A. Tchounikine, A Multidimensional and Multiversion Structure for OLAP Applications *ACM Fifth International Workshop on Data Warehousing And Olap (DOLAP 2002)* in McLean, VA, USA, on November 8, 2002

[5] L. Cabibbo and R. Torlone, "A Logical Approach to Multidimensional Databases", *Proceedings of the 6th International Conference on Extending Database Technology (EDBT'98)*, Valencia, Spain, 1998.

[6] P. Chamoni and S. Stock, "Temporal Structures in Data warehousing", *In Proceedings of the DaWaK'99 Conference*, Florence, Italy, 1999.

[7] S. Chaudhuri, U. Dayal "An Overview of Data Warehousing and Olap Technolog", *SIGMOD Record 26(1)*, 1997

[8] E.F. Codd, S.B. Codd and C.T. Salley, "Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate", *Technical Report*, E.F. Codd and Associates, 1993.

[9] J. Eder and C. Koncilia, "Evolution of Dimension Data in Temporal Data Warehouses", *Proceedings of the DaWaK'01 Conference*, Munich, Germany, 2001.

[10] C. Hurtado, A.O. Mendelzon and A. Vaisman, "Maintaining Data Cubes Under Dimension Updates", *In Proceedings of the IEEE/ICDE'99 Conference*, 1999.

[11] C. Hurtado, A.O. Mendelzon and A. Vaisman, "Updating OLAP Dimensions" *Proceedings of the ACM Second Int. Workshop on Data Warehousing and OLAP*, USA, 1999.

[12] W.H. Inmon *Building the Data Warehouse* 3rd Edition, Eds.Wiley and Sons, 1996

[13] R. Kimball, *The Data Warehouse Toolkit*, J.Wiley and Sons, Inc, 1996.

[14] W. Lehner, "Modeling large OLAP scenarios" *Proceedings of the International Conference on Extending Database Technology*, Valencia, Spain, 1998.

[15] A.O. Mendelzon and A. Vaisman, "Temporal Queries in OLAP" *Proceedings of the 26th VLDB'00 Conference*, Cairo, Egypt, 2000.

[16] Microsoft Corporation, "OLE DB for OLAP Specification Version 2.0", *Microsoft Technical Document*, 2001.

[17] J. Mundy, "Microsoft SQL Server 2000 as a "Dimensionally Friendly System", Microsoft Corporation, 2001.

[18] T.B. Pedersen, C.S. Jensen and C.E. Dyreson, "A foundation for capturing and querying complex multidimensional data", *Information Systems Special Issue: Data Warehousing*, Vol 26, No 5, 2001.

[19] P. Vassiliadis P., T. Sellis, A Survey of Logical Models for OLAP Databases, *SIGMOD Record* Volume 28, Number 1, March, 1999