

Changes of Dimension Data in Temporal Data Warehouses

Johann Eder and Christian Koncilia

University of Klagenfurt
Dep. of Informatics-Systems
{eder,koncilia}@isys.uni-klu.ac.at

Abstract. Time is one of the dimensions we frequently find in data warehouses allowing comparisons of data in different periods. In current multi-dimensional data warehouse technology changes of dimension data cannot be represented adequately since all dimensions are (implicitly) considered as orthogonal. We propose an extension of the multi-dimensional data model employed in data warehouses allowing to cope correctly with changes in dimension data: a temporal multi-dimensional data model allows the registration of temporal versions of dimension data. Mappings are provided to transfer data between different temporal versions of the instances of dimensions and enable the system to correctly answer queries spanning multiple periods and thus different versions of dimension data.

1 Introduction and Motivation

Data warehouses or data marts are integrated materialized views over several often heterogeneous data sources. Their most important usage is On-Line Analytical Processing (OLAP) typically using a multi-dimensional view of the data. OLAP tools then allow to aggregate and compare data along dimensions relevant to the application domain. Typical examples of dimensions found frequently in business data warehouses include time, organizational structure (divisions, departments, etc.), space (cities, regions, countries) and product data.

This multi-dimensional view provides long term data that can be analyzed along the time axis, in contrast to snapshot-based OLTP systems. Available OLAP systems are therefore prepared to deal with changing values of fact data, e. g. , changing profit or turnover but surprisingly not for modifications in dimension data, e. g. , if a new branch or division is established, although time is usually explicitly represented as a dimension in data warehouses.

The reason for this disturbing property of current data warehouse technology is the implicitly underlying assumptions that the dimensions are orthogonal. Orthogonality with respect to the dimension time means the other dimensions ought to be time-invariant. This silent assumptions inhibits the proper treatment of changes in dimension data.

Naturally, it is vital for the correctness of results of OLAP queries that modifications of dimension data is correctly taken into account. E. g. , when the

economic figures of European countries over the last 20 years are compared on a country level, it is essential to be aware of the re-unification of Germany, the separation of Czechoslovakia, etc. Business structures and even structures in public administration are nowadays subject to highly dynamic changes. Comparisons of data over several periods, computation of trends, etc. have the necessity to correctly and adequately treat changes in dimension data. Otherwise we face meaningless figures and wrong conclusions triggering bad decisions. From our experience we could cite too much such cases.

The following extensions to a data warehouse are therefore necessary:

- **Temporal extension:** dimension data has to be time stamped in order to represent their valid time.
- **Structure versions:** by providing time stamps for dimension data the need arises that our system is able to cope with different versions of structures.
- **Transformation functions:** Our system has to support functions to transform data from one structure version into another.

In contrast to temporal databases, which have been well studied, e. g. , [3, 7, 8], few approaches are known in literature for temporal data warehouses, e. g. , [4, 13]. The same holds for schema evolution of databases, e. g. , [11, 5] vs. schema evolution of data warehouses, e. g. , [2].

[4] present necessary extensions to a data warehouse to cover temporal aspects, in particular to keep track of the history of hierarchical assignments. [2, 1] deal with schema evolution and schema versioning for data warehouse systems, transferring changes of the conceptual schema can be automatically into the logical and internal schema. However, these papers do not address the inevitable consequences of these scheme evolutions for analytical queries.

A formal definition of a temporal OLAP system and a temporal query language (TOLAP) is proposed in [12], however, without transformation of data between structural versions the system is not able to cope with changes in the time and fact dimensions.

In another proposal [13], the schema is extended with time stamps to enable the user to analyze data for different scenarios. However, this approach is limited to some basic operations on dimension data (e. g. , insert/delete a dimension member; change the “parent” of dimension member).

2 Temporal Multidimensional Systems

A multidimensional view on data consists of a set of dimensions. defining an n-dimensional data cube [14, 10]. Usually, a data cube is defined by a dimension *Time*, a dimension *Facts* and by several dimensions describing the managerial structures such as divisions, products, or branches.

A dimension is a set of dimension members and their hierarchical structure. For example “*VCR X-25*”, “*VCRs*” and “*All Products*” are dimension members of the dimension “Products” and are in the hierarchical relation *All Products* \rightarrow *VCRs* \rightarrow *VCR X-25*. The hierarchical structure of all dimensions defines all

possible consolidation paths, i. e. , it defines all possible aggregation and disaggregation paths.

We will now extend this description of a multi-dimensional system to define a temporal data warehouse supporting valid time relations:

- **Chronons:** A chronon Q is defined as “a non-decomposable time interval of some fixed, minimal duration” [9]. This means a chronon is the finest dimension member in the dimension time and the time axis defined through the dimension time is a series of chronons.
- **Time Intervals:** All dimension members and all hierarchical links between these dimension members are associated with a time interval $[T_s, T_e]$ representing the valid time beginning at T_s and ending at T_e (with $T_e \geq T_s$).

More formally, a temporal multidimensional system consists of:

- i.) A number of dimensions $N + 1$.
- ii.) A set of dimensions $\mathcal{D} = \{D_1, \dots, D_N, F\}$ where F is the dimension describing the required facts and D_i are all other dimensions including a time dimension if required.
- iii.) A number of dimension members M .
- iv.) A set of dimension members $\mathcal{DM} = \mathcal{DM}_{D_1} \cup \dots \cup \mathcal{DM}_{D_N} \cup \mathcal{DM}_F = \{DM_1, \dots, DM_M\}$ where \mathcal{DM}_F is the set of all facts, \mathcal{DM}_{D_i} is the set of all dimension members which belong to dimension D_i . A dimension member DM_i is defined as $DM_i = \langle DM_{id}, Key, D_i, UDA, [T_s, T_e] \rangle$. DM_{id} is a unique identifier for each dimension member that cannot be changed (similar to $O_{id's}$ in object-oriented database systems). $[T_s, T_e]$ represents the valid time of the dimension member. D_i is the dimension identifier to which the dimension member belongs. Key is a user defined key (e. g. , the number of a product) which is unique within D_i for each timepoint $T_s \leq T \leq T_e$. UDA is a set of user defined attributes (e. g. , the name and/or color of a product).
- v.) A set of hierarchical assignments $\mathcal{H} = \{H_1, \dots, H_O\}$ where $H_i = \langle DM_{id}^C, DM_{id}^P, Level, [T_s, T_e] \rangle$. DM_{id}^C is the identifier of a dimension member, DM_{id}^P is the dimension member identifier of the parent of DM_{id}^C or \emptyset if the dimension member is a top-level dimension member. $Level$ is a value $0 \dots L$ where L is the number of layers and $Level$ is the level of DM_{id}^C . All “leaves” (dimension members without successors) are at level 0. $[T_s, T_e]$ is the time stamp representing the valid time for the relation between DM_{id}^C and DM_{id}^P . No dimension member may be its own parent/child and cycles within \mathcal{H} are not admissible.
- vi.) A function $cval : (DM_{D_1}, \dots, DM_{D_N}, DM_F) \rightarrow value$ which uniquely assigns a value to each vector $(DM_{D_1}, \dots, DM_{D_N}, DM_F)$ where $(DM_{D_1}, \dots, DM_{D_N}, DM_F) \in \mathcal{DM}_{D_1} \times \dots \times \mathcal{DM}_{D_N} \times \mathcal{DM}_F$. Therefore, a cube (or n-cube) C is defined by this function $cval$. The domain of this cube $dom(C)$ is the set of all cell references. The range of this cube $ran(C)$ are all cell values.

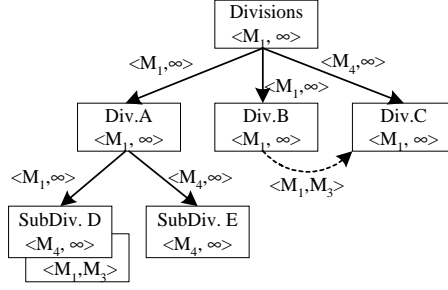


Fig. 1. A Dimension *Divisions* with time stamps

3 Structure Versions

Temporal projection and selection [9] allows us to define a *Structure Version* (SV) of a temporal data warehouse. Intuitively, a structure version is a view on a multidimensional structure that is valid for a given time interval $[T_s, T_e]$. All dimension members and all hierarchical relations are also valid for the given time interval. In other words: within one structure version no dimension member is changed nor a hierarchical relation. Vice versa each modification of a dimension member or a hierarchical relation leads to a new structure version.

Formally, each structure version is a 4-tuple $\langle SV_{id}, T, \{\mathcal{DM}_{D_1, SV_{id}}, \dots, \mathcal{DM}_{D_N, SV_{id}}, \mathcal{DM}_{F, SV_{id}}\}, \mathcal{H}_{SV_{id}} \rangle$ where SV_{id} is a unique identifier and T represent the valid time of that structure version as a time interval $[T_s, T_e]$. $\mathcal{DM}_{D_i, SV_{id}}$ ($\mathcal{DM}_{D_i, SV_{id}} \subseteq \mathcal{DM}_{D_i}$) is a set of all dimension members which belong to dimension D_i and which are valid at each timepoint P with $T_s \leq P \leq T_e$. $\mathcal{DM}_{F, SV_{id}}$ ($\mathcal{DM}_{F, SV_{id}} \subseteq \mathcal{DM}_F$) is the set of all facts which are valid at each timepoint P with $T_s \leq P \leq T_e$. $\mathcal{H}_{SV_{id}}$ ($\mathcal{H}_{SV_{id}} \subseteq \mathcal{H}$) is a set of hierarchical assignments valid at each timepoint P with $T_s \leq P \leq T_e$.

Conceptually each structure version SV has a corresponding cube with the same valid time interval. Fig. 1 shows an example for the consolidation tree of the dimension “Divisions” including time intervals. Each node and each edge in this figure has a time interval $[T_s, T_e]$. An attribute of “*SubDiv.D*” was modified at M_4 , a new subdivision “*SubDiv.E*” was introduced at M_4 and *Div.C* was a subdivision of *Div.B* from M_1 until M_3 (dotted line). Two structure versions can be identified in this example:

- i.) $\langle SV_1, [M_1, M_3], \{\{Divisions, Div.A, Div.B, Div.C, SubDiv.D\}, \{Sales\}\}, \{Div.A \rightarrow Divisions, SubDiv.D \rightarrow Div.A, \dots\} \rangle$
- ii.) $\langle SV_2, [M_4, \infty], \{\{Divisions, Div.A, Div.B, Div.C, SubDiv.D, SubDiv.E\}, \{Sales\}\}, \{Div.A \rightarrow Divisions, SubDiv.D \rightarrow Div.A, \dots\} \rangle$.

In this example we have two different structure versions SV_1 and SV_2 . SV_1 and all given dimension members (*Divisions*, *Div.A*, *Div.B*, ...) and hierarchical assignments (*Div.A* \rightarrow *Divisions*, ...) are valid from M_1 to M_3 . SV_2 is valid from M_4 to ∞ , i. e., until now.

4 Structural Changes

We define the structure of a data warehouse (DWH) as a non-empty, finite set of structure versions $DWH = \{SV_1, \dots, SV_n\}$, where each structure version SV_i is a 4-tuple $\langle SV_{id}, T, \mathcal{DM}_{SV_{id}}, \mathcal{H}_{SV_{id}} \rangle$ (see above) forming a dense sequence of tuples $\langle SV_{id}, T_i, \mathcal{DM}_{SV_{id}}, \mathcal{H}_{SV_{id}} \rangle$ with respect to chronon Q , i. e., $T_i = [T_{i,s}, T_{i,e}]$ such that $T_{i,s} = T_{(i-1),e} + Q$.

We provide three basic operations INSERT, UPDATE and DELETE to modify the structure of a temporal data warehouse, i. e., the dimension data within the granularity defined through the chronon Q . *Key*, D_i , *UDA* and DM_{id}^P are defined as described in Sect. 2

INSERT(DM, T_s): inserts the new dimension member DM as $\langle Key, D_i, UDA, DM_{id}^P \rangle$. T_s defines that DM is valid for the time interval $[T_s, \infty]$. A unique DM_{id} is assigned to the new element.

UPDATE(Key, D_i, DM', T_s): changes an existing dimension member identified by Key and D_i to a new dimension member DM' as $\langle Key, UDA, DM_{id}^P \rangle$. An UPDATE operation consists of two actions: set the ending time of an existing dimension member to $T_s - Q$, and insert a new dimension member, with the valid time interval $[T_s, \infty]$.

DELETE(DM, T_e): changes the end time of the dimension member DM to T_e .

Using these basic operations we can modify the structural dimensions of the multidimensional cube. We distinguish among the following modifications:

- i.) SPLIT: One dimension member splits into n dimension members.
- ii.) MERGE: n dimension members are merged into one dimension member.
- iii.) CHANGE: An attribute, e.g. the product number, of a dimension member changes.
- iv.) MOVE: Modify the hierarchical position of a dimension member.
- v.) NEW-MEMBER: Insert a new dimension member.
- vi.) DELETE-MEMBER: Delete a dimension member.

5 Mappings between Structure Versions

We will now extend the temporal model of a data warehouse presented in chapter 2 with the definition of mapping functions between structure versions.

A structure version is a view on a temporal data warehouse valid for a given time period $[T_s, T_e]$. We distinguish between the structure of a structure version (the set of all valid dimension members of the structure version together with their hierarchies) and the data of a structure version (the cube defined by mapping the structure to the value domain).

For answering queries on the data warehouse the user always has to define which structure version should be used. The data returned by the query can, however, originate in several (different) temporal versions of the cube. Therefore, it is necessary to provide transformation functions mapping data from one structure version to a different structure version.

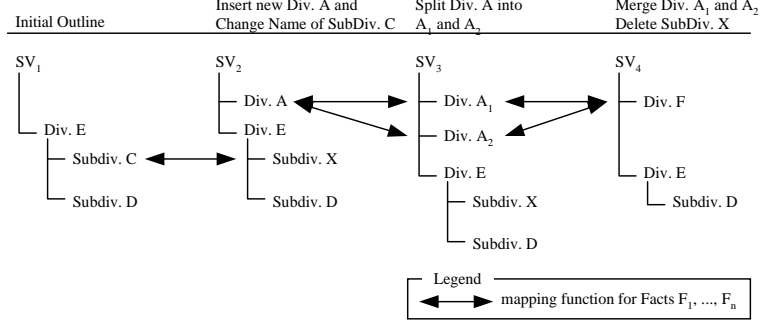


Fig. 2. An example for structural changes and mapping functions

In the rest of the paper we make the following assumptions: Relations between different structure versions depend on the contemplated fact. For sake of simplicity and understandability we only consider the cube for a single fact. Furthermore, the cell values of upper-level dimension members are always computed from their subordinate lower level dimension members. Therefore, without loss of generality, we do not consider the upper levels here and assume that the dimensions are flat. Or, in other terms: before we transform the data we select the cube of the dimension members at level-0 and transform only this subset of cell values and compute the upper-levels of the resulting cube bottom-up as usual.

5.1 Definition of Inter-Structure Relationships

Mapping functions are employed to map data (cell values) for numeric facts and a particular dimension member from one structure version into another using a weighting factor. $MapF$ is defined as $MapF(SV_j, SV_k, DM_{id}, DM'_{id}, \{M_{id}^1, \dots, M_{id}^n\}, w)$ where SV_j and SV_k are different structure versions. DM_{id} and DM'_{id} are unique IDs for dimension members for which $DM_{id} \in SV_j$ and $DM'_{id} \in SV_k$ is true. DM_{id} and DM'_{id} must be dimension members of the same dimension. $\{M_{id}^1, \dots, M_{id}^n\}$ is a non-empty, finite set of fact IDs and $\exists f : f \in F \wedge f_{id} = M_{id}^i$. w is the weighting factor to map data from one structure version into another.

We implicitly introduce a mapping function for each dimension member which does not change from one structure version into another with $w = 1$.

Mapping functions may be applied to map data between contiguous or non contiguous structure versions. Two structure versions SV_i and SV_k are contiguous if $T_{s,i} = T_{e,k} + Q$ or if $T_{s,k} = T_{e,i} + Q$.

For a split or a merge operation we need several mapping functions, e. g. , if department A splits up into A_1 , A_2 and A_3 we would need three functions to map data from A to A_n and three functions to map data from A_n to A . We do not restrict the user regarding the weighting factor w . This means that the sum of all weighting factors for all functions $A \rightarrow A_n$ (split) does not have to be 1, i. e. , 100%. Vice versa not all weighting factors of the functions $A_1 \rightarrow A, \dots, A_n \rightarrow A$ (merge) need to be 1.

The example given in Fig. 2 shows several structural changes in a dimension “Divisions”, e. g., “Div.A” splits up into “Div.A₁” and “Div.A₂” from SV_2 to SV_3 and that for the fact “Turnover” “Div.A₁” in SV_3 corresponds to 30% of the “Div.A” in SV_2 (see function 1). Or vice versa that the “Div.A” in SV_2 is equal to the sum of A_1 and A_2 in SV_3 (see functions 3 and 4). This example would result in the following mapping functions for the fact “Turnover”:

- 1.) $MapF(SV_2, SV_3, Div.A, Div.A_1, \{\text{Turnover}\}, 0.3)$
- 2.) $MapF(SV_2, SV_3, Div.A, Div.A_2, \{\text{Turnover}\}, 0.7)$
- 3.) $MapF(SV_3, SV_2, Div.A_1, Div.A, \{\text{Turnover}\}, 1)$
- 4.) $MapF(SV_3, SV_2, Div.A_2, Div.A, \{\text{Turnover}\}, 1)$, and so on...

5.2 Transformation Matrices

On a conceptual level we can represent each multidimensional cube and the relationships between dimension members of different structure versions as matrices.

Let SV_i be a structure version with N dimensions. Each dimension D_N consists of a set $\mathcal{DM}_N^{L_0}$ which represents all Level-0 dimension members of that dimension. We can represent this structure version as a $\mathcal{DM}_1^{L_0} \times \mathcal{DM}_2^{L_0} \times \dots \times \mathcal{DM}_N^{L_0}$ matrix.

Let SV_1 and SV_2 be two structure versions. We define a transformation matrix $T_{SV_1, SV_2, D_i, F}$ for each dimension D_i and each fact F . Where $T(d_i, d_j)$ is a number representing the weighting factor for mapping a fact F of dimension member d_i of structure version SV_1 to a fact of dimension member d_j of structure version SV_2 .

These transformation matrices are merely another way of representing the information contained in the $MapF$ relation described above. We want to emphasize that the construction of these matrices is a conceptual view on the transformation. Any meaningful implementation will take into account that these matrices are usually sparse and will not implement the matrices in a naive way.

Example: Consider a cube C representing the structure defined through a structure version SV_1 with the dimensions $A = \{a_1, a_2, a_3\}$ and $B = \{b_1, b_2, b_3\}$ (a_i and b_j are dimension members). We represent the cell values for a specific fact in this cube as a matrix. Therefore, a value in this matrix represents a cell value in the given 2-dimensional cube.

$$C = \begin{matrix} & a_1 & a_2 & a_3 \\ b_1 & \left(\begin{matrix} 3 & 7 & 5 \end{matrix} \right) \\ b_2 & \left(\begin{matrix} 10 & 8 & 6 \end{matrix} \right) \\ b_3 & \left(\begin{matrix} 20 & 13 & 5 \end{matrix} \right) \end{matrix}$$

As mentioned above we need one transformation matrix for each dimension D_i to map data from structure version SV_1 into structure version SV_2 . In the following example we split the dimension member a_1 into a_{11} and a_{12} and we merge b_1 and b_2 into b_{12} . The functions between SV_1 and SV_2 for a fact “Fact” are defined by the following operations:

- $MapF(SV_1, SV_2, a_1, a_{11}, Fact, 0.3)$
- $MapF(SV_1, SV_2, a_1, a_{12}, Fact, 0.7)$
- $MapF(SV_1, SV_2, b_1, b_{12}, Fact, 1)$
- $MapF(SV_1, SV_2, b_2, b_{12}, Fact, 1)$

To represent these functions we define two transformation matrices. T_A for dimension A , and T_B for dimension B :

$$T_A = \begin{matrix} & a_{11} & a_{12} & a_2 & a_3 \\ \begin{matrix} a_1 \\ a_2 \\ a_3 \end{matrix} & \begin{pmatrix} 0.3 & 0.7 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{matrix} \quad T_B = \begin{matrix} & b_1 & b_2 & b_3 \\ \begin{matrix} b_{12} \\ b_3 \end{matrix} & \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{matrix}$$

5.3 Transformation of Warehouse Data

The goal of the transformation is to map the warehouse data (cube) of a certain structure version SV_1 to the structure of a different structure version SV_2 . We first define a function to transform the cube in one dimension:

$f_{SV_1, SV_2, D, F}$ transforms the values of fact F of structure version SV_1 to the structure version SV_2 in the dimension D as follows:

$$f_{SV_1, SV_2, D, F}(C_{D=j}) = \sum_{j \in DM_{D, SV_1}} T_{SV_1, SV_2, D, F}(i, j) * C_{D=j} \text{ for all } i \in DM_{D, SV_2}$$

where C is a cube with the dimension members of SV_1 in dimension D and C' is the transformed cube where all values in the cube have been transformed to the members of the dimension D in the structure version SV_2 according to the transformation matrix T . $C_{D=j}$ is the (n-1) dimensional sub-cube of an n-dimensional cube associated with the member j in dimension D .

It is easy to see, that transforming a cube in dimension D_x first, and then in dimension D_y yields the same result as the transformation in the reverse sequence. The transformation of a fact F in a cube C from structure version SV_1 to structure version SV_2 is now defined as a sequence of functions successively transforming the cube in all dimensions D_i :

$$f_{SV_1, SV_2, F} = f_{SV_1, SV_2, D_1, F}(f_{SV_1, SV_2, D_2, F}(\dots f_{SV_1, SV_2, D_n, F}(C_{SV_1}) \dots))$$

As seen from the observation above the result does not depend on the sequence of transformation used. Again, we emphasize that this is the specification of a transformation function, and the actual implementation will efficiently make use of the sparseness of the involved matrices, etc.

Example: By using the defined transformation functions we are now able to transform data from SV_1 into SV_2 . The cube C and the transformation matrices T_A and T_B are given in the example in Sect. 5.2.

$$\begin{aligned} C' &= f_{SV_1, SV_2, D_A, F}(f_{SV_1, SV_2, D_B, F}(C)) \\ &= \begin{matrix} & a_{11} & a_{12} & a_2 & a_3 \\ \begin{matrix} b_{12} \\ b_3 \end{matrix} & \begin{pmatrix} 3.9 & 9.1 & 15 & 11 \\ 6 & 14 & 13 & 5 \end{pmatrix} \end{matrix} \end{aligned}$$

The matrix C' represents the cube with the structure defined through structure version SV_2 and the values of structure version SV_1 .

6 Queries

When a user issues a query within such a system, he/she has to define a timepoint T_q . This timepoint specifies a certain base structure version where $T_s \leq T_q \leq T_e$ and $[T_s, T_e]$ defines the valid time interval of the base structure version.

This base structure version determines which structure has to be used for the analysis. In most cases this will be the current structure version. However, in some cases it will be of interest to use an “older” structure version. Suppose the structure versions given in Fig. 2 are valid for the following time periods and the chronon is a month:

Table 1. Valid time periods

Version	T_s	T_e
SV_1	Jan. 1998	Mar. 1998
SV_2	Apr. 1998	Jan. 1999
SV_3	Feb. 1999	Dec. 1999
SV_4	Jan. 2000	∞

We might assume the user chooses SV_4 as base structure version and requests data for March 2000 and March 1999 for the analysis. In this case the system needs functions to map data which is valid for the structure version SV_3 into the structure version SV_4 . The same analysis however could also be made with SV_3 as base structure version. For this query the system needs functions to map data from SV_4 to SV_3 .

For each query, the systems checks which structure versions are necessary to answer the query. E. g., for SV_4 as base structure version and the valid time intervals according to Tab. 1, the structure versions SV_4 , SV_2 and SV_1 are necessary to answer the query “return costs for all divisions for January 1999 and January 1998”. For each fact the system checks for a mapping function from SV_1 to SV_4 and from SV_2 to SV_4 .

7 Conclusion

We presented a novel approach for representing changes in dimension data of multi-dimensional data warehouses, by introducing temporal extension, structure versioning and transformation functions. This representation can then be used to pose queries (analysis) against the structure valid at a given point in time and correctly admit data from other periods into the computation of the result.

This effort is necessary as changes in these data have the combined characteristics of temporal databases and schema evolution, as these dimension data

serve in multi-dimensional systems as data as well as schema elements. Our approach thus overcomes the implicit orthogonality assumption underlying multi-dimensional data warehouses.

The transformation function we propose here can only be seen as a first step and will be elaborated in the future. The simple transformation matrices however proved themselves surprisingly powerful. We were able to represent several cases of structural changes with these data (at least approximatively). Changes which can be covered by our model comprise:

- Changes in the organizational structure of enterprises, of the regional structure of distribution systems, of product portfolios, etc.
- Changes of Units, like actually the changes from ATS to EURO.
- Changes in the way economic figures like unemployment rate, consumer price index, etc. are computed.

We also expect that our approach improves the correctness of interpretation of answers to OLAP queries and relieves the user from the need to have detailed knowledge about the change history of dimension data. In particular, our approach provides for multi-period comparisons of facts which currently requires stability in dimension data.

References

- [1] M. Blaschka. FIESTA: A Framework for Schema Evolution in Multidimensional Information Systems. In *Proc. of 6th Doctoral Consortium*, Germany, 1999.
- [2] M. Blaschka, C. Sapia, and G. Höfling. On Schema Evolution in Multidimensional Databases. In *Proc. of the DaWak99 Conference*, Florence, Italy, 1999.
- [3] M. Böhlen. Temporal Database System Implementations. *SIGMOD*, 24(4), 1995.
- [4] P. Chameni and S. Stock. Temporal Structures in Data Warehousing. In *Data Warehousing and Knowledge Discovery (DaWaK) 1999*, p. 353–358, Italy, 1999.
- [5] S. M. Clamen. Schema Evolution and Integration. In *Distributed and Parallel Databases: An International Journal*, p. 2(1):101–126.
- [6] O. Etzion, S. Jajodia, and S. Sripada, editors. *Temporal Databases: Research and Practise*. LNCS 1399. Springer-Verlag, 1998.
- [7] Goralwalla, Tansel, and Zsu. Experimenting with Temporal Relational Databases. *ACM*, CIKM95, 1995.
- [8] Gregersen and Jensen. Temporal Entity-Relationship Models - a Survey. *Time-Center*, 1997.
- [9] C. S. Jensen and C. E. Dyreson, editors. *A consensus Glossary of Temporal Database Concepts - Feb. 1998 Version*. Springer-Verlag, 1998. In [6].
- [10] C. Li and X. Wang. A Data Model for Supporting On-Line Analytical Processing. *ACM*, CIKM 96, 1996.
- [11] C. Liu, S. Chang, and P. Chrysanthis. Database Schema Evolution using EVER Diagrams. In *Proc. of the Workshop on Advanced Visual Interfaces*, 1994.
- [12] A. Mendelzon and A. Vaisman. Temporal Queries in OLAP. In *Proc. of the 26th VLDB Conference, Egypt*, 2000.
- [13] SAP America, Inc. and SAP AG. Data Modelling with BW - ASAP for BW Accelerator. 1998. White Paper: URL: <http://www.sap.com>.
- [14] P. Vassiliadis and T. Sellis. A Survey of Logical Models for OLAP Databases. In *SIGMOD Record* 28, 1999.