published in: Proceedings of the 3rd International Workshop on Knowledge Management in Electronic Government (KMGov 2002), Trauner Druck, Schriftenreihe Informatik, Vol. 7, ISBN 3-85487-409-X, pp 186-198

Representing Knowledge about Changes in Data Warehouse Structures

Johann Eder and Christian Koncilia

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

Abstract. *Already Ovid claimed that everything is in flux and we see continuing metamorphoses. Knowledge about changes is essential for Knowledge management in particular for the correct interpretation of data stemming from different periods. Data Warehouses are increasingly deployed in public administrations to provide analytical data for decision making, for monitoring or for revisions.*

Changes in transaction data are recorded in data warehouses and sophisticated tools allow to analyze these data along time and other dimensions. But changes in master data and in structures cannot be represented in current data warehouse systems impeding their use in dynamic areas and leading to erroneous query results. For an example: if the definition of "unemployment rate" changes, then the figures cannot be compared to those of previous years. Trend calculations on basis of the available data is irrelevant or severely misleading.

We propose a temporal data warehouse architecture named COMET to represent structural changes and permit correct analysis of data over periods with changing structures.

1. Introduction

Data Warehouses are structured collections of data supporting controlling, decision making and revision. Data Warehouses build the basis for analyzing data by means of OLAP (online analytical processing) tools which provide sophisticated features for aggregating, analyzing, and comparing data and for discovering irregularities [14],[10].

Data Warehouses are designed and tuned for answering complex queries rather than for high throughput of a mix of updating transactions, and they typically have a longer memory, i.e. they do not only contain the actual values (snapshot data) but also historical data needed for the purposes outlined above.

The most popular architecture for data warehouses are multidimensional data cubes, where transaction data (called cells, fact data or measures) are described in terms of master data hierarchically organized in dimensions.

Surprisingly, data warehouses are not well prepared for changes in spite of their requirement for serving as long term memory. Of course they are very well designed for dealing with changes in transaction data, which are represented in the cells of multidimensional data cubes. However, they cannot adequately represent changes in master data which span the dimension structures of such cubes, with changes in units, changes in formulas computing derived transaction data, or schema changes.

Consider the following example: Diagnoses for patients were represented in a data warehouse using the “International Statistical Classification of Diseases and Related Health Problems” (ICD) code. However, codes for diagnoses changed from ICD Version 9 to ICD Version 10. Even worse, the same code described different diagnoses in ICD-9 and ICD-10. Other ICD-10 codes are a subset of ICD-9 codes, i.e. the granularity of codes changed. Let us say that in ICD-9 we a code X for the diagnose D and in ICD-10 we have two codes X1 for diagnose D1 and X2 for diagnose D2 where D1 and D2 are subsets of D. The question is now: how can we get correct results for queries like "did liver cancer increase over the last 5 years". Without *knowing* the above changes we will end up with incorrect results.

Data Warehouse and OLAP tools are increasingly employed to provide access to large data collections and to automate and distribute reports derived from these data collections. This proliferation of reports and OLAP statistics unfortunately does not go hand in hand with the proliferation of knowledge about the data in these statistical data nor with knowledge about their sources. Frequently, we find that legends and other informative texts accompanying usually statistical tables are missing from quickly available OLAP reports, and of course are typically missing in ad-hoc query results. This lack of knowledge causes wrong information of even well-minded receivers of these data.

The problems associated with this lack of information is very severe in governmental information systems. On one hand government agencies are collecting enormous amounts of data and provide these data to the public, on the other hand these collected data are under closer observation and frequently are used by parties, media and other participants in public debates.

The goals of our research described in this paper are to make data warehouses more useful, and to improve the correctness of OLAP-results. For this purpose we need provisions to correctly answer queries for data in a particular point in time (snapshot data), and for retrieving data stemming from different periods in a way that they can be compared or commonly processed

further on. As a direct consequence, it is not sufficient to just maintain schema and structure of a data warehouse to represent the "new" view, since this would result in the loss of the ability to answer queries regarding previous periods correctly, restricting the value of data warehouses for revision purposes. On the other hand, just keeping the correct master data and units for the transaction data is not sufficient either, since aggregation over data from different periods would give wrong results.

We propose an architecture for representing the changes of a data warehouse schema and of the dimension data in a way that correct analysis of data is made possible. Since this means recognizing that the shape and content of a *Star*-Schema may change over time we call this a COMET schema. This COMET schema not only contains the information about the structure of a data warehouse, but also about all the changes and serves as knowledge base for the interpretation of query results.

In this paper we discuss the use of our temporal data warehouse metamodel in public administrations by means of an example. We present a series of changes representing the evolution of structures. We show how the knowledge about changes can be recorded in the COMET data warehouse model and how this knowledge is used for reducing incorrect OLAP results.

2. Temporal Data Warehouse Architecture

Changes in dimension data cannot be represented adequately in current data warehouse technology because of 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. The consequences of this deficiency are the restricted applicability of data warehouse technology in dynamic domains with frequent changes of structural data. The storing of data over several periods, foundational for data warehouses is of rather limited use, if these data cannot be compared and aggregated over these periods to allow for trend computations and multi-period comparisons. On the other hand we often found incorrect data as results of OLAP analysis, sometimes the users were aware of the errors stemming from structural changes, more often not.

In particular, we propose a temporal data warehouse architecture which extends multidimensional data warehouses with knowledge about changing structures, master data and relationships between master data to achieve the following features:

- representation of changes in master data, units and schema
- identification of structure versions as changeless periods.

- provision of mappings of transaction data between structure versions.
- Support of queries touching data from different structural versions.
- analysis of data according to new and old versions of the structure
- detect whether the result of a query might be wrong because of changes in structure or schema

Our concept for a temporal data warehouse model called COMET proposed in [4] and [5] builds on the techniques developed for temporal databases [1],[8],[9], in particular time stamping information, and temporal selection and projection, schema evolution and schema versioning [12],[3],[2],[7],[13] and extends and adapts them for the particularities of data warehouses.

All dimension members and all hierarchical links between these dimension members have to be time stamped with a time interval $[T_S, T_E]$ representing the valid time where T_S is the beginning of the valid time, T_E is the end of the valid time and $T_E \geq T_S$. According to [11] a time interval "is the time between two instants" and it "may be represented by a set of contiguous granules". Furthermore, we timestamp all schema definitions, i.e. dimensions, categories and their hierarchical relations, in order to keep track of all modifications of the data warehouse schema [5].

If we represent all time stamps of all modifications within our data warehouse on a linear time axis the interval between two succeeding time stamps on this axis represents a structure version. This means that a structure version is a view on a temporal data warehouse valid for a given time period $[T_S, T_E]$. Within a structure version, the structure of dimension data is stable on both schema and instance level. Information about structure versions can be gained from our temporal data warehouse using temporal projection and temporal selection [11].

The data returned by a query may originate in several (different) structure versions. Hence, it is necessary to check whether the data needed for answering the query (the relevant sub-cube) was affected by structural changes. This is important since not all structural changes affect all data. If the data was affected by structural changes, it is necessary to provide transformation functions mapping data from one structure version to a different structure version.

Transformation functions enable us to assure that a successful analysis can be made when dimension data and dimension structure changed. The combination of structure versions and transformation functions enables the user to analyze data with dimension data and structures "backward" or "forward" in the time axis.

3. COMET @ work

In this section, we will show by means of an example how the COMET metamodel can be used to deal with modifications in the structure of data warehouses, i.e., how it can be used to deal with changing knowledge.

3.1. Running Example

Throughout the rest of this paper, we will use the following running example. Consider a data warehouse that stores information about the gross domestic product (*GDP*) in the local currency and the number of inhabitants (*#Inhabitants*) of European countries. In the COMET metamodel, facts are modelled as a unique dimension. The users stated, that they would like to analyze these facts along two dimensions: *Geography* and *Time*.

The dimension *Geography* has the following structure: *Country* → *Province*, where "A → B" means that B rolls-up to A. In our COMET approach both, *Country* and *Province*[1] are called **Categories**. Furthermore, users want to store the size of each country in square kilometres (*Size*). However, they do not want to analyze this data, nor do they want to analyze the *GDP* or the *#Inhabitants* along a unique dimension Size. Therefore, we can treat this information as an **user defined attribute** (UDA) that is applicable to all dimension members of the Category *Country*.

The data stored in this data warehouse stems from several information systems. These systems are able to deliver data once per quarter. Therefore, dimension *Time* has two Categories, namely *Year* and *Quarter*.

Figure 1 shows the resulting cube called "Europe Info" and the user interface of the COMET Administration Tool. The right part of the toolbar consists of a selection box that allows to select a structure version. As until now we did not change any master data within the cube, only one structure version exists, namely the structure version valid from 01/01/1980 until NOW. The right part of the user interface shows a graphical representation of the schema (dimensions and categories) of the cube valid in the selected structure version. The left part shows all dimension members valid in the selected structure version for the selected dimension. In Fig. 1, it shows all members assigned to the dimension *Geography*.

---

[1] For sake of readability we modelled only four different European countries (and some provinces): Germany, Austria, Czechoslovakia, Yugoslavia and their different "versions"

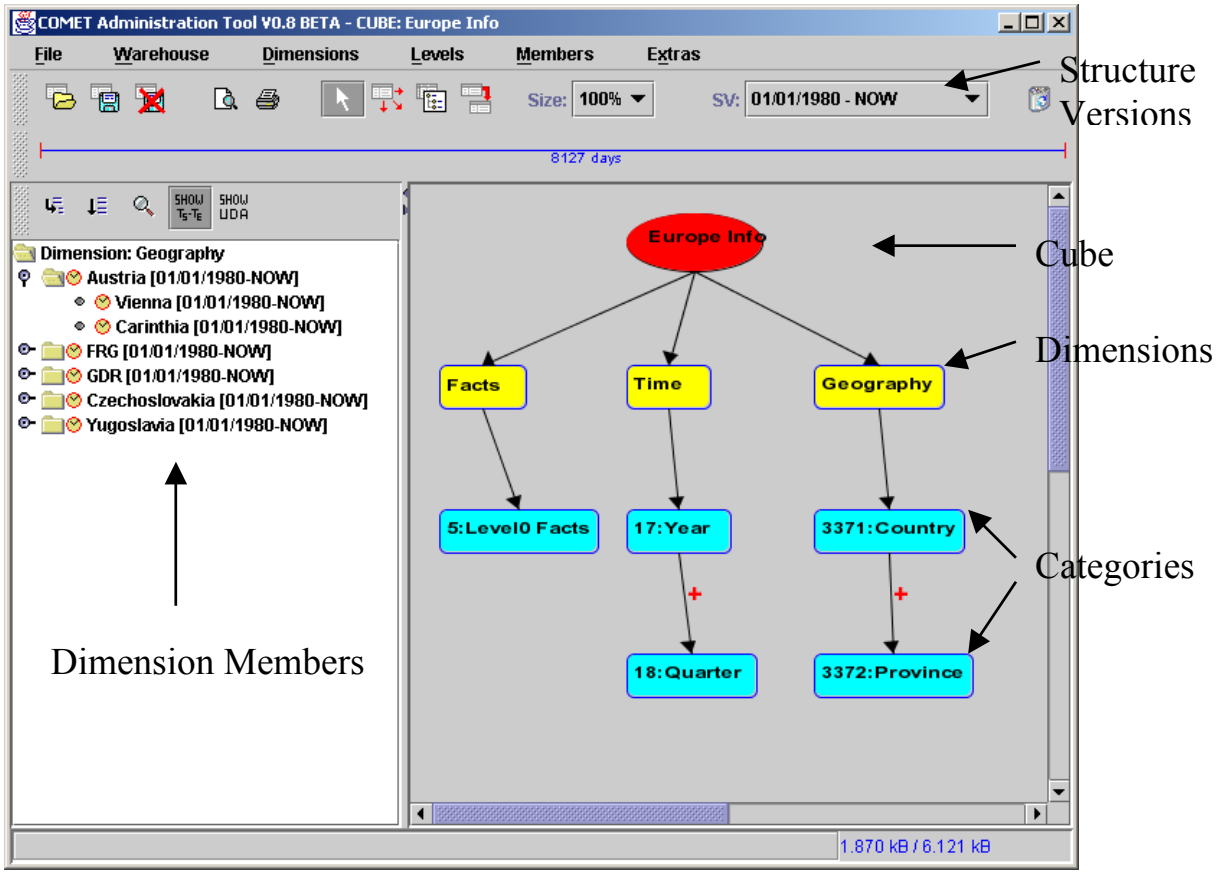

Figure 1: Cube created with the COMET Administration Tool

3.2. Changes on the Instance Level

We will now discuss how to model changes on the instance level, i.e., changes of dimension members, in the COMET Administration Tool. We will show some typical types of complex modifications by means of an example.

3.2.1. MERGE

A merge operation is an operation where several dimension members merge into one dimension member. In our running example, the re-unification of FRG and GDR into Germany in 1990 is a merge operation.

Representing this re-unification in the COMET data warehouse, leads to a cube with two structure versions. The first one is valid from the start of the data warehouse until 1989, the second is valid from 1990 until now.

With respect to multi-period comparisons like “Show GDP of ‘Germany’ for the last 20 years”, we could introduce a set of transformation functions to map data from FRG and GDR into Germany (forward in time, i.e. from a structure version into a subsequent structure version) and vice versa (backwards in time, i.e. into a preceding structure version). This could be done by

introducing a transformation function for all facts (*GDP* and *#Inhabitants*), or by introducing separate transformation functions for a subset of facts.

In our example, we would like to transform all values for all facts with the same transformation function. Hence, we will use the same weighting factor to map data. The weighting factors are stated by the administrator and may be a result of analysing the data stored in the date warehouse. In our example, the factors to transform cell values from "Germany" into "FRG" and "GDR" are the proportional number of inhabitants[2] of these countries.

This allows us to introduce the following transformation functions:

$$\text{Germany} = \text{FRG} * 1 + \text{GDR} * 1$$

$$\text{GDR} = \text{Germany} * 0{,}21$$

$$\text{FRG} = \text{Germany} * 0{,}79$$

Figure 2 shows how to introduce transformation functions within the COMET Administration Tool. As can be seen, a transformation function always maps data from one structure version SV_N, into the preceding (SV_{N-1}) or succeeding (SV_{N+1}) structure version. In [4] we showed, that these transformation functions can be represented at matrices. By simply multiplying these matrices, we are able to automatically compute transformation functions to map data from one structure version SV_N into a non-contiguous structure version, e.g. SV_{N+2} or SV_{N-3}.

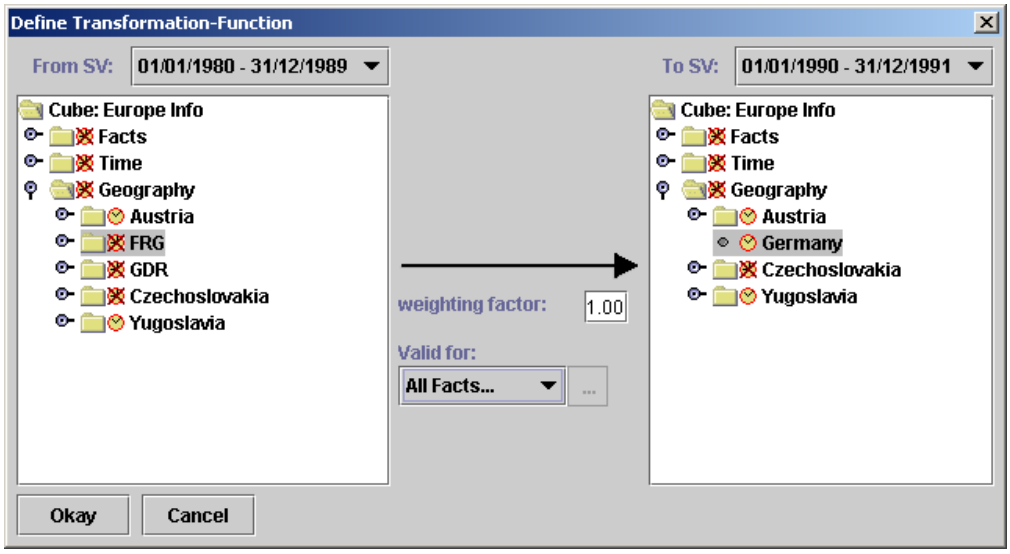

Figure 2: Defining Transformation Functions

[2] To be more precise, in 1989 the GDR had about 16.4 million inhabitants and the FRG had about 62.6 million inhabitants.

3.2.2. SPLIT

A split operation is an operation where one dimension member splits up into several other dimension members. Hence, a split operation is the opposite of a merge operation. In other words, the re-unification of Germany (a merge operation) is also a split operation in the reverse direction of time, e.g., from the "future" into the "past". Therefore, a split operation is also a merge operation in the reverse direction of time.

In our running example, Czechoslovakia split up into the Czech Republic and the Slovak Republic in 1992.

After the re-unification of Germany in 1990, and the separation of the Czech and Slovak Republic in 1992, the given cube comprises of three different structure versions. The first one is valid from the start of the data warehouse until 1989, the second is valid from 1990 until 1991, and the third is valid from 1992 until now.

Again we are able to introduce a set of transformation functions to transform cell values from Czechoslovakia into the Czech Republic and the Slovak Republic and vice versa. And once again, we use the proportional number of inhabitants[3] of these countries to compute the weighting factors that have to be stated for the transformation functions.

Hence, we introduce the following transformation functions:

Czechoslovakia = Slovak Republic * 1 + Czech Republic * 1

Slovak Republic = Czechoslovakia * 0,35

Czech Republic = Czechoslovakia * 0,65

Figure 3 shows the structure version that is valid from 1992 until now of the resulting cube after the re-unification of Germany in 1990 and the separation of the Czech and Slovak Republic in 1992.

The second toolbar visualizes all structure versions that are stored in the data warehouse, i.e., it visualizes the linear time axis and the timepoints of all modifications within the cube. The currently selected structure version is depicted in a different colour. Furthermore, each part of this time axis is labelled with the number of time units (in our example, the number of days) between to ticks.

---

[3] The Czech Republic had about 10.2 million inhabitants and the Slovak Republic had about 5.4 million inhabitants.

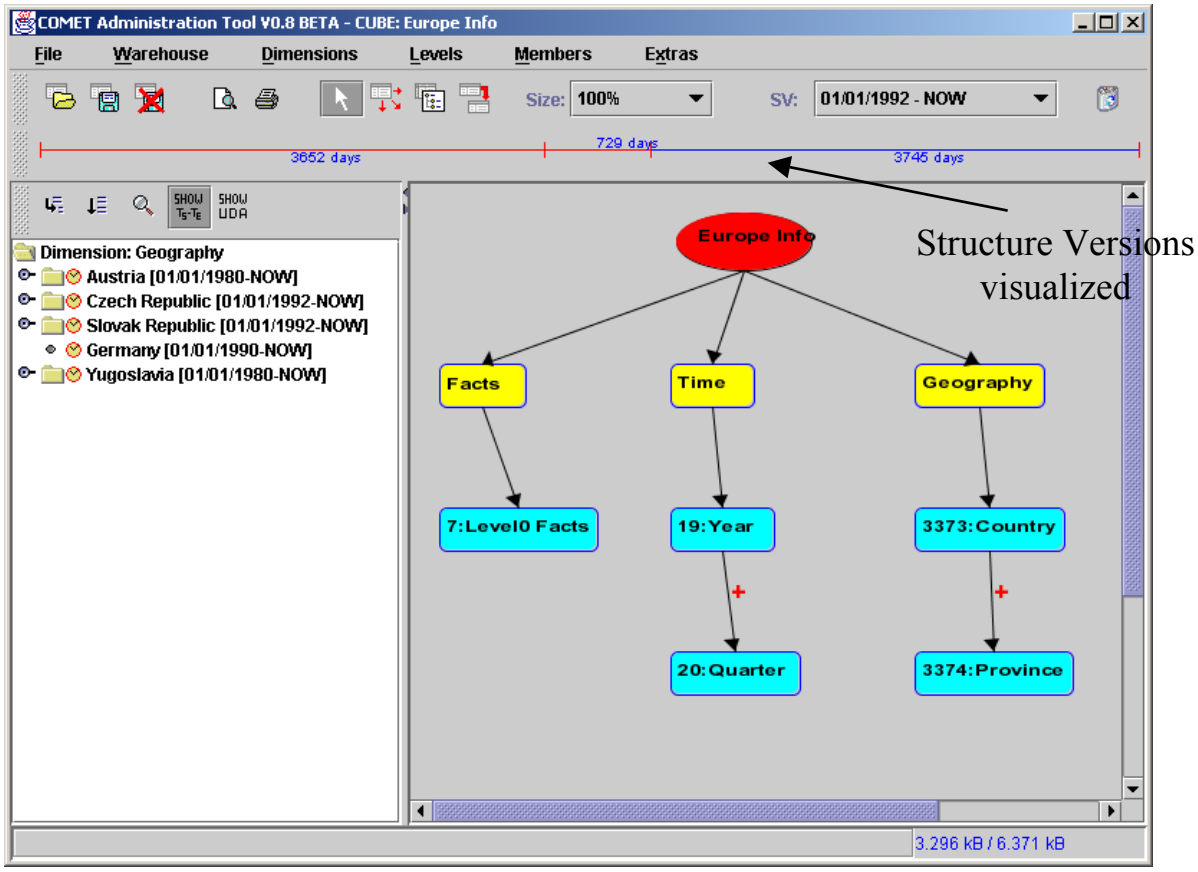

Figure 3: The cube after the re-unification of Germany, and the separation of the Czech and Slovak Republic

3.2.3. CHANGE

A change operation is an operation where the name of a dimension member, or one of its user defined attributes changes. In our running example, there are for instance two different changes: In 2002 the name of "Yugoslavia" changed to "Serbia and Montenegro". Furthermore, in 2002 the local currency changed in twelve countries within the European Community to EURO.

After we incorporated both modifications (the changing name of Yugoslavia and the changing currencies in several countries) within our cube, this leads to a new structure version that is valid from 2002 until now.

We are able to allow multi-period queries that include the dimension member "Yugoslavia" by simply introducing a transformation function between the dimension member "Yugoslavia" in structure version [1992, 2001] (valid from 1992 until 2001) and the dimension member "Serbia and Montenegro" in structure version [2002, NOW] (valid from 2002 until now) with a weighting factor equal to 1.

Transforming the *GDP* in local currencies into EURO and backwards from EURO into the local currencies can be done by defining transformation functions with the corresponding weighting factors. For instance, as 1 Euro = 1.95583 Deutschmarks we can introduce a transformation function with a weighting factor 0.5113 to map Deutschmarks into EURO. On the other hand, we may introduce a transformation function with a weighting factor 1.95583 to map EURO into Deutschmarks.

However, to map the *#Inhabitants* of Germany in structure version [1992, 2001] to Germany in structure version [2002, NOW] we have to introduce another, different transformation function with a weighting factor equal to 1.

3.2.4. MOVE

A move operation changes the parent member of a dimension member. If, in our running example, we would classify all European countries into members and non-members of the European Community, then Austria, Sweden and Finland would have moved from the group "*Non-EC-Member*" to the group "*EC-Member*" in 1995.

As in our COMET approach, we do not only timestamp all nodes (dimensions, categories and dimension members) but also all edges defining the hierarchical relation between nodes, this could be done by setting the end time of the valid time for the hierarchical relation between *Austria* and *Non-EC-Member* to 1994, and creating a new hierarchical relation between *Austria* and *EC-Member* with a timestamp [1995, NOW].

As we implicitly introduce a transformation function for each dimension member which does not change from one structure version into another with a weighting factor equal to 1, the administrator of the cube would not have to introduce a transformation function to map data from *Austria* as a *Non-EC-Member* into *Austria* as a *EC-Member* and backwards.

3.3. Changes on the Schema Level

The COMET metamodel and its implementation, the COMET Administration tool does not only allow changes on the instance level, e.g. modifications of dimensions members and their hierarchical relation. It does also support changes on the schema level, for instance deleting a whole dimension or inserting a new category.

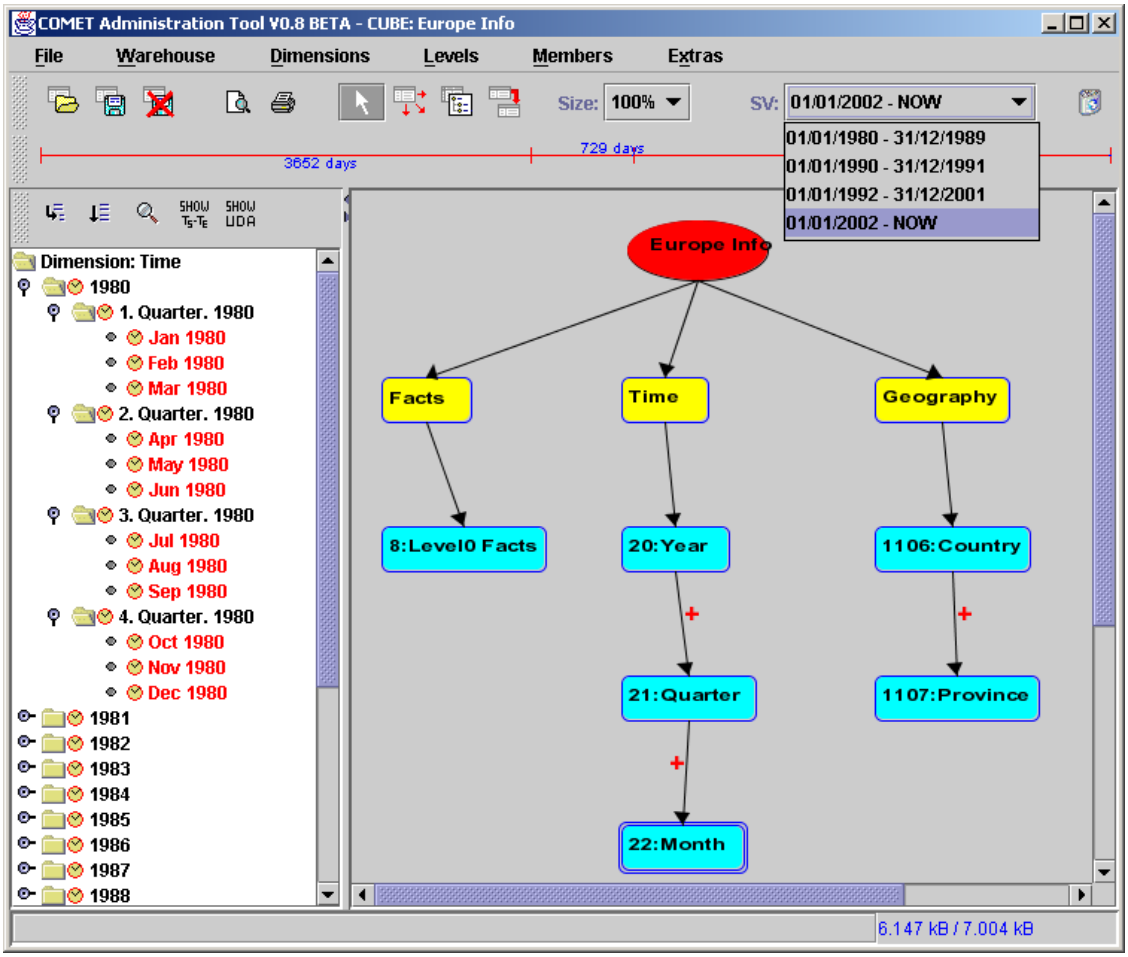

Figure 4: Resulting cube after inserting a new category *Month*

Imagine for our running example, that the information systems that are the sources for our data warehouse are changing. From 2002 on, they do not only support data on a quarterly basis, but could also provide data on a monthly basis. Therefore, we would like to adopt our cube, i.e., to insert a new category *Month* beneath the category *Quarter* in the dimension *Time*.

Hence, we introduce a new category *Month* with the valid time [2002, NOW] and a new hierarchical relation between the two categories *Month* and *Quarter* with the same valid time. Figure 4 shows the resulting cube after inserting a new category *Month*.

To allow the user to state queries like “Show GDP of Germany in January for the last 20 years”, we could introduce a set of transformation functions to map data from the old structure into the new structure version. This is similar to the split operation mentioned: we split each quarter into three months.

Therefore, we could introduce the following transformation functions:

1.Quarter = January * 1 + February * 1 + March * 1

2.Quarter = April * 1 + May * 1 + June * 1

…

January = 1.Quarter * 0,333

February = 1.Quarter * 0,333

March = 1.Quarter * 0,333

April = 2.Quarter * 0,333

…

Although the same could be done when inserting new dimensions, this would require a vast number of transformation functions. However, inserting new dimensions should not occur frequently after careful requirements analysis. Furthermore, we are working for providing generic transformation functions.

3.4. Analyzing the Cube

In order to enable the user to use multidimensional querying techniques like Drill-Down, Roll-Up, Slice and Dice we implemented a tool called *COMET Transformer*. The Transformer generates a data mart for each structure version needed by the user. In its current version, the Transformer tool generates *Hyperion Essbase* data marts, a popular and widely-used multi-dimensional database management system. Figure 5 shows a data mart generated by the Transformer. This data mart corresponds to the structure valid in the structure version [2002, NOW] and can now be analysed using a standard OLAP tool like Microsoft Excel or Wired For OLAP.

In most cases, this data mart will have the structure defined by the actual structure version. Each data mart consists of all data that are valid for the same time interval as the corresponding structure version plus it consists of all data that could be transformed by the defined transformation functions from all other structure versions.

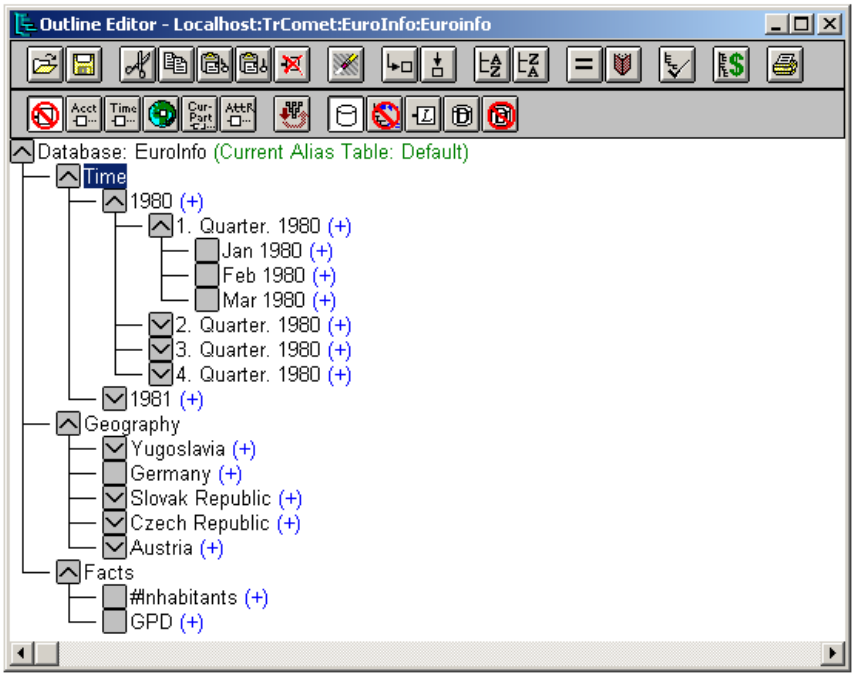

Figure 5: A data mart in Hyperion Essbase generated by the *Transformer* tool

Therefore, the user defines his/her base structure version by selecting a specific data mart. 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, e.g. for auditing purposes, it will be of interest to use an "older" structure version.

4. Conclusions

Unfortunately, many information systems are ill prepared to represent changing knowledge. Surprisingly, multidimensional data warehouse systems are among those. For the correctness of results of OLAP queries, it is vital that modifications of structures are 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. This knowledge can be used in different ways. First it is necessary for interpreting the results of OLAP queries. However, this is usually not sufficient. The Data Warehouse should proactively use knowledge about changes to provide subtexts (legends) for results, reject queries because their results would be meaningless, or perform transformations to achieve correct results.

We showed an easy-to-use and powerful approach that enables temporal data warehousing, namely the COMET metamodel. We gave a description of

some of the problems that can be solved with the implementation of this metamodel by means of an example. Furthermore, we showed how this implementation – the *COMET Administration Tool* and the *COMET Transformer* – works. It is our ambition to contribute to a more knowledgeable use of information collected in data warehouses and to hopefully better decisions.

5. References

[1] M. Böhlen. Temporal Database System Implementations. *SIGMOD*, 24(4), 1995.

[2] C. D. Castro, F. Grandi, and M. R. Scalas. Schema Versioning for Multitemporal Relational Databases. In *Information Systems*, volume 22(5):249–290, 1997.

[3] S. M. Clamen. Schema Evolution and Integration. In *Distributed and Parallel Databases: An International Journal*, pages 2(1):101–126, 1994.

[4] J. Eder and C. Koncilia. Changes of Dimension Data in Temporal Data Warehouses. In *Proc. of the DaWak 2001 Conference*, Munich, Germany, 2001.

[5] J. Eder, C. Koncilia, and T. Morzy. The COMET Metamodel for Temporal Data Warehouses. In *Proc. of the 14th Int. Conference on Advanced Information Systems Engineering (CAISE'02)*, Toronto, Canada, 2002.

[6] O. Etzion, S. Jajodia, and S. Sripada, editors. *Temporal Databases: Research and Practise*. Number LNCS 1399. Springer-Verlag, 1998.

[7] E. Franconi, F. Grandi, and F. Mandreoli. Schema Evolution and Versioning: a Logical and Computational Characterisation. In *Workshop on Foundations of Models and Languages for Data and Objects*, 2000.

[8] I. Goralwalla, A. Tansel, and M. Özsu. Experimenting with Temporal Relational Databases. *ACM*, CIKM95, 1995.

[9] H. Gregersen and C. Jensen. Temporal Entity-Relationship Models - a Survey. *TimeCenter Publication*, 1997.URL: http://www.cs.auc.dk/TimeCenter/.

[10] B. Hüsemann, J. Lechtenbörger, and G. Vossen. Conceptual Data Warehouse Design. In *Proc. of the International Workshop on Design and Management of Data Warehouses (DMDW 2000)*, Stockholm, 2000.

[11] C. S. Jensen and C. E. Dyreson, editors. *A consensus Glossary of Temporal Database Concepts - Feb. 1998 Version*, pages 367–405.Springer-Verlag, 1998. in [6].

[12] C. Liu, S. Chang, and P. Chrysanthis. Database Schema Evolution using EVER Diagrams. In *Proceedings of the Workshop on Advanced Visual Interfaces*, pages 123–132, 1994.

[13] J. Roddick. A Survey of Schema Versioning Issues for Database Systems. In *Information and Software Technology*, volume 37(7):383–393, 1996.

[14] M. Wu and A. Buchmann. Research Issues in Data Warehousing. *BTW'97*, 1997.