

To appear in Proc.Confenis 2006

Maintaining Temporal Warehouse Models

Johann Eder¹, Christian Koncilia², and Karl Wiggisser³

1 University of Vienna, Dept. of Knowledge and Business Engineering
Rathausstrasse 19, 1010 Wien, Austria
johann.eder@univie.ac.at

2 Panoratio Database Images, Inc.
Theresienstrasse 4, 80333 Muenchen, Germany
christian.koncilia@panoratio.de

3 University of Klagenfurt, Dept of Informatics-Systems
Universitaetsstrasse 65-67, 9020 Klagenfurt, Austria
karl.wiggisser@isys.uni-klu.ac.at

Abstract. DWT is a tool for the maintenance of data warehouse structures based on the temporal data warehouse model COMET. Data warehouse systems do not provide support for maintaining changes in dimension data. DWT allows keeping track of modifications made in the dimension-structure of multidimensional cubes stored in an OLAP (On-Line Analytical Processing) system. We present the overall structure of the DWT system, which allows to upload and download warehouse models in different modeling notations in a time conscious manner, load edit scripts describing changes between versions of warehouse models and apply these edit scripts. We present the workflows for maintenance of warehouse models and discuss how maintenance can be supported with the various integrated tools of DWT .

1 Introduction

Data Warehouses are integrated materialized collections of data typically from different heterogeneous data sources. They provide sophisticated support for aggregating, analyzing and comparing data to support decision making. The most popular architecture for data warehouses is the multidimensional datamodel, where transaction data (also called cells or fact data) is described in terms of masterdata also called dimension members). Usually, members are hierarchically organized in dimensions.

Data warehouses are well prepared to deal with modifications in transaction data, e.g. the changing values of the fact *Turnover* over the time can be covered by introducing a dimension *Time*. Not surprisingly, most multidimensional models feature a time dimension. Surprisingly, however, data warehouses are not well prepared for changes of the structure of dimensions in spite of their requirement for serving as long term memory and the observation that such modifications happen frequently, too. It is, however, vital for the accuracy and correctness of results of OLAP queries that modifications in the structure of dimensions are correctly taken

into account, in particular, when comparing data over several periods, computing trends, or computing benchmark values from data of previous periods.

Maintenance of structural modifications in data warehouses is a crucial point for keeping track of structural modifications and considering these modifications in analytical queries. There are several approaches to cope with this problem, [1-5] are some of them. Some of them (e. g. [1, 2]) allow changes only on instance level, i. e. changing the members. Others (e. g. [5]) work only on the schema level, i. e. allow changing dimension and hierarchy definitions. In [6, 7] we presented the COMET metamodel for temporal data warehousing, which allows a versioning on both, schema and instance level.

Here, we present our temporal data warehouse maintenance system DWT built upon the COMET metamodel. We show, how the COMET model can be realized with a layered architecture where a temporal store is employed by non-temporal OLAP tools. We show the architecture of the system, its functionality, and the ways the system can be used. A discussion of design considerations and implementation issues of the prototype complement the paper.

2 System Overview and Functionality

Changes in the dimensional structures of OLAP cubes can cause serious problems, and today's data warehouse systems do not provide appropriate means for solving them. Based on the COMET temporal data warehouse metamodel, we present the DWT system, which is intended for dealing with such problems. The main functionalities of the DWT system are:

- *Import and Export of OLAP Cubes*: import cubes from and export cubes to virtually any OLAP system via generic interfaces.
- *Management of Structure Versions*: select a particular structure version from the DWT database, create new structure versions, and maintain relations and differences between two contiguous versions.
- *Detection of Differences*: tag differences between two structure versions, either by comparing them, importing a change list, or by manual input from the user.

A data warehouse administrator is able to adjust the OLAP cubes due to environmental changes: store the changes into the DWT database and create a set of different structure versions for a cube, each tagged with a timestamp, defining its valid time. Any structure version that was valid at an arbitrary point in time can be re-established.

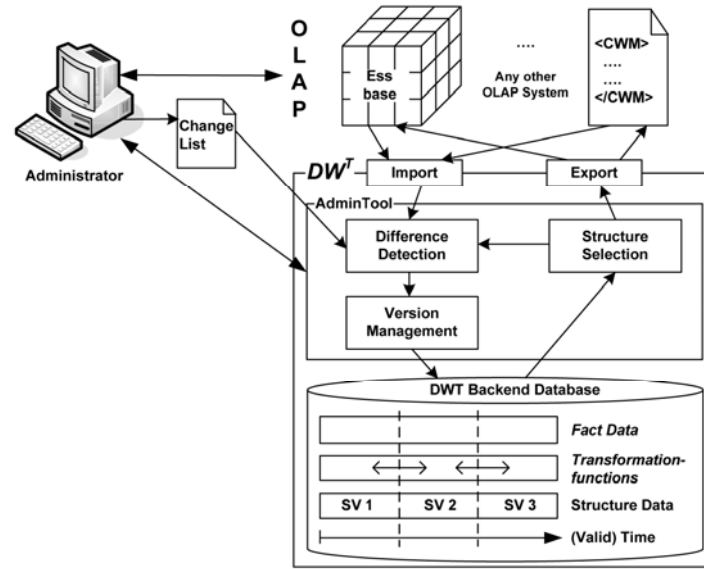


Fig. 1. Overall architecture of the system

2.1 General System Architecture

Figure 1 shows the overall architecture of the DWT system. The system's basis is a relational database that holds all versioning information. It consists of:

- The *Structure Data*, which holds the structure versions of the cubes and the changes between them (e. g. changed members, inserted dimensions)
- The *Transformation Functions*, which describe the relations between members of different structure versions.
- The *Fact Data*, which holds the cell data to be transformed with the help of
- the transformation functions.

The database is queried and filled by the DWT administration tool. This is the central part for managing the versioning process. The main components of the administration tool are the Structure Selection, the Version Management, and the Difference Detection component. The structure selection component is responsible for extracting one particular structure version from the database. The version management component is responsible for creating new structure versions from existing OLAP cubes combined with results from the difference detection component. The difference detection describes differences between imported cubes and stored structure versions (details in 2.5).

2.2 Interfaces

To interact with OLAP systems, the DWT application has two generic interfaces: to *import* cubes from and to *export* cubes to an OLAP system. For each OLAP system to be supported, one has to implement these two interfaces.

The import interface reads the data from an OLAP system. Within the DWT tool, the data is temporalized, i. e. every element is timestamped, defining its valid time and augmented with versioning information, i. e. relations to elements already stored in the database. The export interface works vice versa. It gets a single version of a cube with all its temporal and versioning information from the structure selection component. As the external OLAP system does not support such temporal information, it is removed during the export and the pure OLAP data is written to the external OLAP system.

The interfaces are shown in the top of Fig. 1. At the moment we have implementations for Hyperion Essbase [8] and a subset of the Common Warehouse Metamodel (CWM) [9], as shown in the architecture. The administrator interacts with both, the selected OLAP system (e. g. Hyperion Essbase), and the DWT administration tool. With the OLAP tool, he can do all update operations on cubes, as usual. With the DWT tool he is able to incorporate these changes into the database.

2.3 Conceptual Database Model and Temporalization

A sketch of our conceptual model for the backend database is given in [7]. Here we can only give a brief summary of the main design ideas.

As the database has to store structure data and fact data, the model includes tables for all integral parts of an OLAP cube, i. e. the cube itself, dimensions, members, hierarchies, cell data, and all necessary relations between them. All these elements, except the cell data, are subjects to versioning and are, therefore, temporalized with respect to the schema given in Fig. 2. Two of the main elements in an OLAP cube are a hierarchy and members belonging to that hierarchy. Figure 2a shows the nontemporal model of these two elements and the relation between them. Figure 2b shows the same elements and relation in a temporalized environment. The Hierarchy and the Member classes have both been split into two classes. The Member class is still the class representing the concept, and therefore holding all associations to other classes. But as all attributes may change over the time, we introduced a new class named MemberVersion which holds the values for the attributes at the given valid time. The ValidTime attribute of the Member is the sum of all valid times of the different versions. For each point in time, a Member is valid, there must exist *exactly one* valid MemberVersion, and for each point in time, a MemberVersion is valid, the corresponding Member has to be valid too. The same principles also apply for hierarchies, dimensions, and cubes. The association between Hierarchy and Member gets timestamped too. As such an association may be valid in more than one structure version, the ValidTime is a multiple attribute here. The constraint for such an association defines that there must not be any point in time,

where the association is valid, but one of the associated classes is not. Of course, this schema does also apply to all other relations.

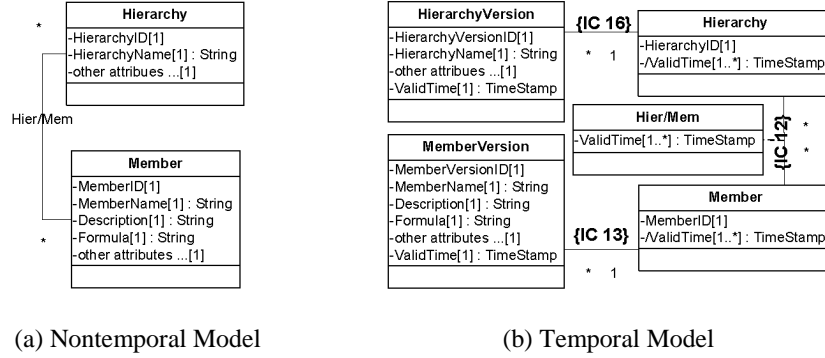


Fig. 2. Concept of Temporalization

2.4 Functionality and Workflow

We have to consider, where to really alter the OLAP cubes and where to do the integrity checking, with respect to DWT. We decided to use the *external/external* approach for our system for the following reasons:

With *external/external* all updates and checks are not done in the DWT tool, but in the external OLAP application. DWT then provides the means for importing cubes, detecting and tagging changes, and incorporate them into the database. The advantages of this approach are the low implementation costs, easy extensibility, and that users can work with the OLAP tool they are familiar with. A disadvantage is that there is no direct control of the data- and controlflow outside the DWT system.

With *internal/external* all updates are done within the DWT tool, but the integrity checking is done by the external OLAP tool. The advantages of this approach are the possibility of logging changes and the partial control of the data- and control flow. Disadvantages are the high implementation costs, because each supported OLAP system needs its own implementation of the maintenance component. Furthermore the users have to get familiar with a new tool for altering OLAP cubes.

With *internal/internal* all changes and checks are done within the DWT tool. The advantages of this approach are the complete control of the data by the DWT tool, the logging of update operations, and that there is no need for an online connection to the OLAP system. The disadvantages are again the high implementation costs and the high export for an extension to additional OLAP systems.

Figure 3 shows a set of statecharts which describe the workflow in the different components. Figure 3a shows the flow for the complete application.

The *Export* (see Fig. 3b) is quite simple: The user selects one particular structure version, and a cube representing this version is created in the OLAP system. Referring back to the main functionalities, this is composed of *Structure Selection* and *Cube Export*.

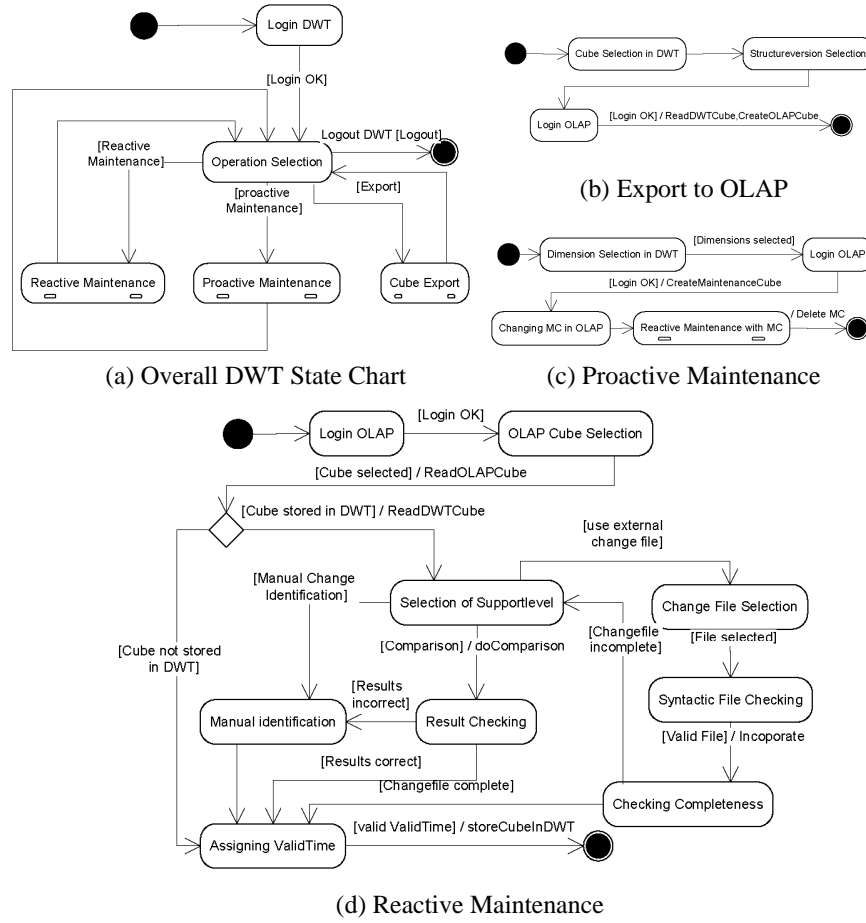


Fig. 3. State Charts describing the Behavior of the DWT Application

The *Reactive Maintenance* (see Fig. 3d) is a bit more complicated: After selecting and reading the cube from the OLAP system, there is either a corresponding version for this cube stored in the DWT database to be updated, or there is not. In the first case, the first thing the user has to choose, is the level of support for the change detection: No Support, Structure Comparison and Change List (details in 2.5). After all changes are correctly tagged the user specifies the valid time for the new version and the system stores it into the database. In the latter case, i. e. no prior structure version for the imported cube exists, the user just gives the valid time for the cube. Then the system stores the cube structure into the database as initial structure version. Referring back to the main functionalities, the reactive maintenance uses all main functionalities except for the cube export, i. e. *Cube Import*, *Cube Selection*, *Difference Detection*, *Version Creation*, and *Relation Management* between versions.

For the *Proactive Maintenance* (see Fig. 3c) the user has to select the dimensions he wants to change. The DWT tool exports these dimensions into a *temporary*

maintenance cube in the OLAP system, where the user does all desired changes. After all changes are done in the OLAP system, the user triggers a reactive maintenance with the maintenance cube, so all differences are detected and stored into the database. Referring back to the main functionalities, the proactive maintenance uses all of them, i. e. *Cube Selection*, *Cube Export*, *Cube Import*, *Difference Detection*, *Version Creation* and *Relation Management* between versions.

```
MatchingLine = Delete|Insert|Change;
Delete = Identifier Separator;
Insert = Separator Identifier;
Change = Identifier Separator Identifier;
Identifier = Path|Name;
Name = ValidCharacter {ValidCharacter};
Path = PathDelimiter Name {PathDelimiter Name};
Separator = ";";
PathDelimiter = "\";
ValidCharacter = any character valid in a member's name
```

Fig. 4. EBNF Syntax of a Change File

2.5 Identification of Changes

Identification of changes between structure versions and establishing relations between two versions of a changed element is a crucial part during the versioning process. As a changelog may not be available, we have to define other means of change detection.

The naive approach is not providing *Any Support* at all. Thus, the user is responsible for tagging all differences between the structure versions. This method is the last fallback solution, as it is time consuming and error prone.

The second possibility is *Structure Comparison*. The system applies a feasible comparison algorithm to the cube structures and detects a list of differences. Such a comparison can for instance be graph based, as described in [10]. Due to the heuristic and inductive nature of comparison algorithms, the results may contain errors. Therefore, the administrator must have the possibility to review the results and manually correct them if necessary.

If, on the other hand, the OLAP system provides the functionality to create a log of the changes applied to a cube, or there is any other possibility to obtain a list of changes outside the DWT tool, it is not necessary to identify them again during the import, but the user may import a change file consisting of a number of *MatchingLines* with the syntax describe in Fig. 4. The characters for the *Separator* or *PathDelimiter* are implementation dependent and may vary for different OLAP systems, as they may occur in a member's name. Members are identified by a path from the root to this member, or, if member names are unique, just by this name.

A *MatchingLine* may either represent the deletion, the insertion, or the change of a member, which may be any combination of update, rename, and move. Generally, it has the form OLDID;NEWID with the following semantics: The member identified

by OLDID in the old structure version represents the same element as the member identified by NEWID in the new structure version. If no OLDID is given, the member identified by NEWID was inserted into the structure. If no NEWID is given, the member identified by OLDID was deleted from the structure. If both of them are present, this indicates a change. In this case, the correct operations can easily be detected by searching the members in both structures and comparing their properties and position. A change file may not be complete, i. e. not describe all changes between the structure versions. In this case, the user has to select additional means for identifying the remaining differences until all changes are tagged. After all changes are identified and tagged, the results are passed to the version management component. The administrator assigns a valid time and the new structure version is stored.

3 Implementation

The implementation is in Java 1.4, the backend database is Oracle 9i. The communication between database and the DWT tool uses JDBC. The interface implementation to Hyperion Essbase is done via the native Hyperion Essbase Java API. The relational schema for the DWT database was highly optimized for achieving good performance.

As the main target OLAP system on this stage is Hyperion Essbase, data warehouse outlines are represented by trees and the tree comparison algorithm defined in [10] is used to compare the two structure versions.

Figure 5 shows the screen after the matching between the two trees. The left tree denotes the structure version stored in the DWT database, the right tree represents the imported cube. Members in the DWT tree that are marked with a cross (e. g. Phantom V, Silver Spirit) could not be matched to any member in the imported tree. Members in the imported tree that are marked with a triangle (e. g. BMW 1, Silver Spirit II) could not be matched to any member in the DWT tree.

After the matching is completed, all yet unmatched nodes could have either been deleted/inserted or renamed. As the fully automated determination of renamings is not possible, we proposed a heuristic approach which calculates the most likely renamings [10]. The user has to check them and do corrections if necessary. Each accepted renaming results in an additional node matching. As the matching and renaming detection of graph nodes heavily relies on heuristics, the algorithm may return wrong results. Thus, the user must have the possibility to correct the node matchings before the change detection is executed. The user may break up a detected matching and/or define new matchings between unmatched nodes.

Figure 6 shows the screen after the comparison algorithm has finished. Members marked with a triangle (e. g. BMW 1, a new car in the product portfolio) have been inserted, members marked with a **C** have been changed (the engine power for Rolls-Royce cars is no longer given in kW but in HP), the **R** indicates a renaming and the **M** denotes a move (e. g. BMW is now a part of the united BMW&Rolls-Royce). During the calculation of the differences, the algorithm is transforming the old version of the tree, thus after the algorithm has finished, both trees have to be

identical, therefore deleted nodes (e. g. Phantom V, which was taken out of the product portfolio) cannot be seen any longer. The dialog box in Fig. 6 shows how to assign the valid time to the new structure version after having clicked the save button.

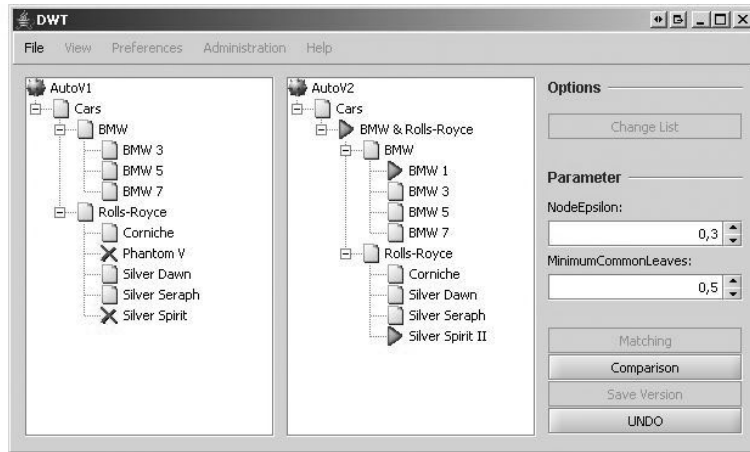


Fig. 5. Screen after Node Matching

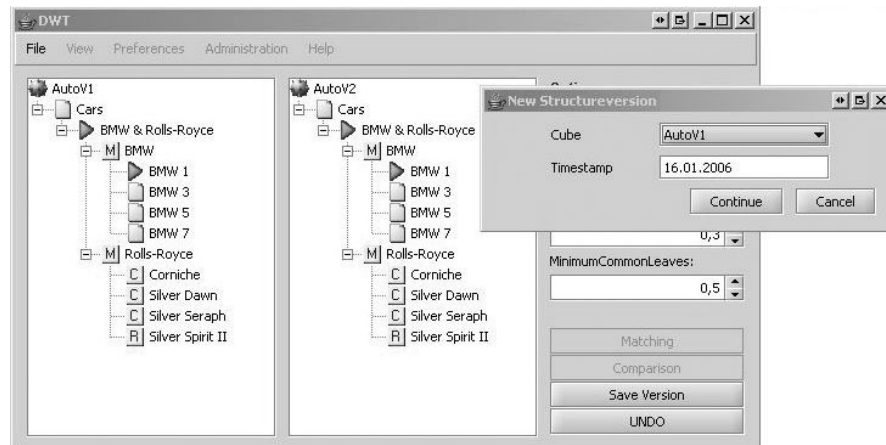


Fig. 6. Final Result of the Structure Comparison

Additional functionalities - e. g. administrative tasks or user management - which do not contain much scientific challenges are not described here.

4 Conclusion

Due to changes of the represented real world, OLAP structures have to change as well. As current implementations of DWH systems, surprisingly, do not support such changes, we defined the COMET Metamodel for temporal data warehouses. In this paper we present the DWT tool for maintaining temporal warehouse models. We describe the principal use cases and their representation in the main workflows within the tool. The administrator is enabled to store versions of a warehouse structure into a database, identify changes between different structure versions, and to re-establish any previously stored structure version. We present the general architecture and the conceptual database model, comprising temporal and versioning information. DWT offers three ways for identifying differences between two subsequent versions: a semiautomatic structure comparison if only snapshots are available, changelog application, and manual change identification.

The major advantage of such a backend tool is that the users and administrators can use their favorite OLAP frontend for doing the analysis of data and for registration of changes. The backend then makes the whole architecture temporal, i.e. provides temporal versions of the dimension structure of a data warehouse and allows that any structure version can be selected by valid time and be uploaded into the OLAP frontend. Using standard interfaces, this allows also for a mapping of dimension structures between different OLAP tools.

5 References

1. P. Chamoni and S. Stock. Temporal Structures in Data Warehousing. In *Proc. 1st DaWaK*, 1999.
2. R. Kimball. Slowly Changing Dimensions, Data Warehouse Architect. *DBMS Magazine*, 9(4), April 1996. <http://www.dbmsmag.com/>.
3. A. Vaisman. *Updates, View Maintenance and Time Management in Multidimensional Databases*. Ph.D. thesis, Universidad de Buenos Aires, 2001.
4. J. Yang. *Temporal Data Warehousing*. Ph.D. thesis, Stanford University, 2001.
5. M. Blaschka. *FIESTA: A Framework for Schema Evolution in Multidimensional Information Systems*. Ph.D. thesis, Technische Universität München, 2000.
6. J. Eder, C. Koncilia, and T. Morzy. A Model for a Temporal Data Warehouse. In *Proceedings of OES-SEO 2001 Workshop*, pages 48--54. 2001.
7. J. Eder, C. Koncilia, and T. Morzy. The Comet Metamodel for Temporal Data Warehouses. In *Proc. of the 14th CAISE*. 2002.
8. Hyperion Solutions Corporation. *Hyperion Essbase*. <http://www.hyperion.com/>.
9. Object Management Group. *The Common Warehouse Metamodel V1.0*, 2001. <http://www.omg.org/cwm>.
10. J. Eder, C. Koncilia, and K. Wiggisser. A Tree Comparison Approach to Detect Changes in Data Warehouse Structures. In *Proc. of the 7th DaWaK*. 2005.