Repository Support for Data Warehouse Evolution

Christoph Quix
Informatik V, RWTH Aachen, Germany
quix@informatik.rwth-aachen.de

Abstract

Data warehouses are complex systems consisting of many components which store highly-aggregated data for decision support. Due to the role of the data warehouses in the daily business work of an enterprise, the requirements for the design and the implementation are dynamic and subjective. Therefore, data warehouse design is a continuous process which has to reflect the changing environment of a data warehouse, i.e. the data warehouse must evolve in reaction to the enterprise's evolution. Based on existing meta models for the architecture and quality of a data warehouse, we propose in this paper a data warehouse process model to capture the dynamics of a data warehouse. The evolution of a data warehouse is represented as a special process and the evolution operators are linked to the corresponding architecture components and quality factors they affect. We show the application of our model on schema evolution in data warehouses and its consequences on data warehouse views. The models have been implemented in the metadata repository ConceptBase which can be used to analyze the result of evolution operations and to monitor the quality of a data warehouse.

1 Introduction

Data warehouses are complex systems consisting of many components which store highly-aggregated data for decision support. Most requirements stem from managers and analysts using the data warehouse system to support them in decision making in the daily business work of an enterprise. The nature of the work of managers and analysts implies that their requirements are often changing and do not reach a final state, i.e. their requirements are dynamic and subjective. They do not only demand faster response time to their queries (which may be achieved by ordering new and faster hardware), they also want more information, e.g. access to data which are currently not present in data warehouse, or a higher quality of their data, e.g. query results with less incorrect values.

Therefore, a data warehouse can not be designed in one step, usually it evolves over many years. A common methodology to construct data warehouses is to start with some local data marts (e.g., one data mart for each department). The knowledge acquired during this phase can be used to construct in parallel a global enterprise schema for the data warehouse. Data marts are usually easier to implement than an enterprise-wide data warehouse, and after a relatively short time analysts can work with the system. The requirements of the analysts will grow in time, and after some time they want to make queries across several data marts of the departments. At this point, the enterprise-wide data warehouse comes into play: it can either be a virtual/distributed data warehouse, i.e. there is common interface to data warehouse but the queries are delegated to the data marts, or a materialized data warehouse, which has loaded the data from the data marts and other information sources.

In data warehouses, changes may happen or be required in many different situations. The data warehouse is usually separated from the OLTP systems and the OLTP systems are important for the daily business of the enterprise. Therefore, the data warehouse must be adapted to any changes which occur in the underlying data sources, e.g. changes of the schemata, changes of the physical location of a data source, or a change of the time window for the extraction of source data. Beside these changes on the source level, the client level (analysts) often change their requirements as already mentioned above. Furthermore, new versions of software components may also require a change in the data warehouse.

Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'99)
Heidelberg, Germany, 14. - 15.6. 1999
(S. Gatziu, M. Jeusfeld, M. Staudt, Y. Vassiliou, eds.)
http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-19/

Data quality is also important in traditional online transaction processing systems (OLTP). In the research on these systems, techniques were developed to ensure a certain data quality level. For example, most relational databases today support referential integrity constraints and the SQL standard, so that data can be easily queried from the database, and the semantics of the result is well understood. However, online analytical processing (OLAP) has refocused the attention on data quality, because of several reasons. First, in data warehouses the data are loaded from many different sources and often problems with the format, encoding or interpretation of data are encountered. Furthermore, data quality is always relative since the quality of data depends on how the data are suited for a particular use. In OLTP systems, the intended use is known before the system is designed and implemented and usually does not change over time. In contrast, the use of OLAP systems is not as static as in OLTP systems and may even be not known at design time of the warehouse. The information demand of managers and analysts changes very often, and if new information is required, it must be delivered in a short time to be useful [TB98].

In the European *DWQ project (Foundations of Data Warehouse Quality)* [JV97], we have developed an architecture and quality model for data warehouses [JJQV99]. This model allows the representation of the data warehouses in three different perspectives:

- the conceptual perspective, which represents an overall business perspective on the information resources and analysis tasks of an enterprise,
- the logical perspective, which describes the schemata used in the sources, the data warehouse, and the data marts, and
- the physical perspective, which shows where the data is physically stored (host, disk, etc.).

Each perspective has three different levels: the source, enterprise and client level. A central role in this model plays the enterprise model which should be a conceptual representation of the data which is available in the enterprise. In [JJQV99], we also presented a preliminary approach of linking quality information to the architecture model. This approach was extended and more formally presented in [JQJ98] and is shortly summarized in section 2.

Our models represent only a "snapshot" of a data warehouse system without taking into account anything of the *dynamics* in a data warehouse environment. In this paper, we want to describe how our repository approach developed in [JJQV99] and [JQJ98] can be extended to deal also with the dynamics of a data warehouse. On the one hand, this includes a *process model* which represents the usual data warehouse processes like data loading or update propagation. On the other hand, the process model is specialized to deal also with *evolution processes*, which are processes which evolve the data warehouse like the materialization of a new view or the addition of a new source.

The advantage of our proposed approach is that all relevant metadata of a data warehouse (architecture, quality, process and evolution information) are stored in a central repository. The different types of information are interrelated, and therefore provide a semantically rich representation of the data warehouse. The query facilities of our metadata repository ConceptBase [JGJ+95] enable data warehouse users to analyze the data warehouse and to find deficiencies in architecture, quality, processes or the evolution of the data warehouse.

This paper is structured as follows. In section 2, we first recall the principles of the architecture and quality models shown in [JJQV99] and [JQJ98] before we present the data warehouse process model. Section 3 specializes the process model to the case of data warehouse evolution. In section 4, we present related work which addresses evolution in data warehouses, in particular schema evolution. Finally, we provide a summary and conclusions and give an outlook to future work.

2 A meta model for Data Warehouse Architecture, Quality and Processes

This section summarizes the nature of metadata used in the DWQ framework and gives an overview of the DWQ quality model. In section 2.3, the framework is extended by a process model for data warehouses.

2.1 Data Warehouse Architecture

In the DWQ project we have advocated the need for enriched metadata facilities for the exploitation of the knowledge collected in a data warehouse. In [JJQV99], it is shown that the data warehouse metadata should track both architecture components and quality factors.

The proposed categorization of the DW metadata is based on a 3x3 framework, depicted in figure 1: we identified three *perspectives* (conceptual, logical and physical) and three *levels* (source, data warehouse, client). We made the observation, that the conceptual perspective, which represents the real world of an enterprise, is missing in most data warehousing projects, with the risk of incorrectly representing or interpreting the information found in the data warehouse.

The proposed metamodel (i.e. the topmost layer in figure 1) provides a *notation* for data warehouse generic entities, such as schema or agent, including the business perspective. Each box shown in figure 1 is decomposed into more detailed data warehouse objects in the metamodel of [JJQV99]. This metamodel is instantiated with the metadata of the data warehouse (i.e. the second layer in figure 1), e.g. relational schema definitions or the description of

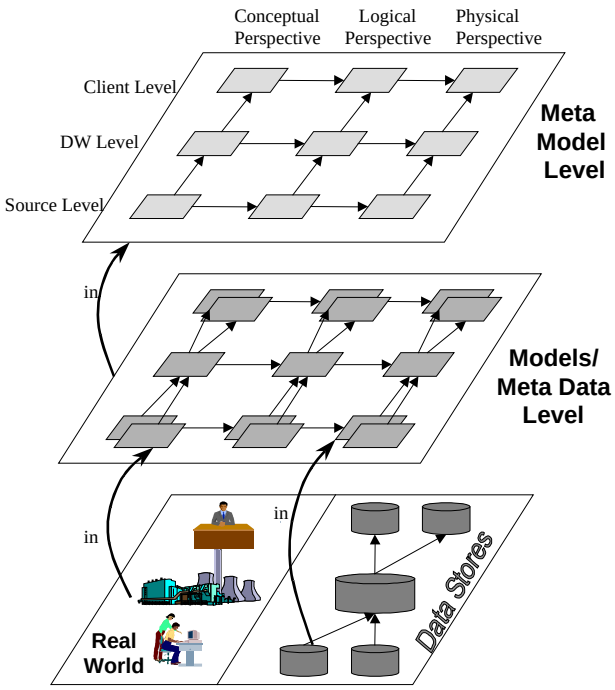

Figure 1: The Data Warehouse Architecture Meta Model

the conceptual data warehouse model. The lowest layer in figure 1 represents the real world where the actual data reside: in this level the metadata are instantiated with data instances, e.g. the tuples of a relation or the objects of the real world which are represented by the entities of the conceptual model.

2.2 Quality Meta Model

Each object in the three levels and perspectives of the architectural framework can be subject to quality measurement. Since quality management plays an important role in data warehouses, we have incorporated it into our metamodeling approach. Thus, the quality model is part of the metadata repository, and quality information is explicitly linked with architectural objects. This way, stakeholders can represent their quality goals explicitly in the metadata repository, while, at the same time, the relationship between the measurable architecture objects and the quality values is retained.

The *DWQ quality metamodel* [JQJ98] is based on the *Goal-Question-Metric* approach (GQM) of [OB92] originally developed for software quality management. In GQM, the high-level user requirements are modeled as *goals*. *Quality metrics* are values which express some measured property of the object. The relationship between goals and metrics is established through *quality questions*.

The main difference in our approach resides in the following points: (i) a clear distinction between *subjective* quality goals requested by stakeholder and *objective* quality factors attached to data warehouse objects, (ii) quality

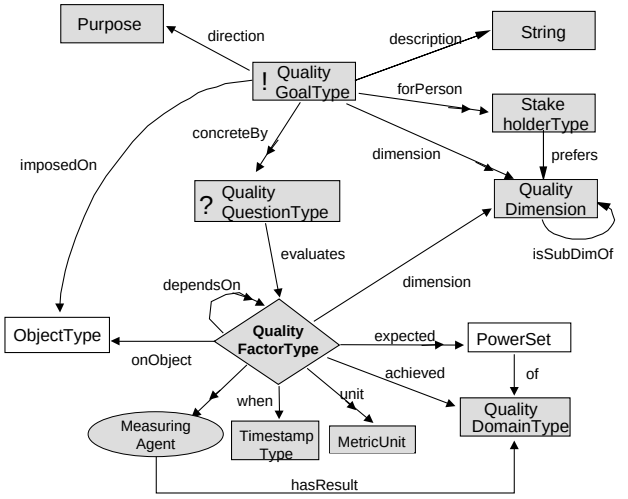

Figure 2: DWQ Quality Meta Model [JQJ98]

goal resolution is based on the evaluation of the composing quality factors, each corresponding to a given quality question, (iii) quality questions are implemented and executed as *quality queries* on the semantically rich metadata repository.

Figure 2 shows the DWQ Quality Model[1]. The class *"ObjectType"* refers to any meta-object of the DWQ framework depicted in the first layer of figure 1. A *quality goal* is an abstract requirement, defined on an object types, and documented by a purpose and the stakeholder interested in. A quality goal roughly expresses natural language requirements like "improve the availability of source s1 until the end of the month in the viewpoint of the DW administrator". *Quality dimensions* (e.g. "availability") are used to classify quality goals and factors into different categories. Furthermore, quality dimensions are used as a vocabulary to define quality factors and goals; yet each stakeholder might have a different vocabulary and different preferences in the quality dimensions. Moreover, a quality goal is operationally defined by a set of questions to which quality factor values are provided as possible answers. As a result of the goal evaluation process, a set of improvements (e.g. design decisions) can be proposed, in order to achieve the expected quality [VBQ99]. A *quality factor* represents an actual measurement of a quality value, i.e. it relates quality values to measurable objects. A quality factor is a special property or characteristic of the related object with respect to a quality dimension. It also represents the expected range of the quality value, which may be any subset of a *quality domain*. Dependencies between quality factors are also stored in the repository. Finally, the method of measurement is attached to the quality factor through a *measuring agent*.

The quality meta-model is not instantiated directly with

[1] The different colors in this and the following figures refer to the abstraction level of the object: meta-meta classes are white, meta-classes are light-gray, simple classes are dark-gray, and data objects are black.

concrete quality factors and goals, it is instantiated with patterns for quality factors and goals. The use of this intermediate instantiation level enables data warehouse stakeholders to define *templates* of quality goals and factors. For example, suppose that the analysis phase of a data warehouse project has detected that the availability of the source database is critical to ensure that the daily online transaction processing is not affected by the loading process of the data warehouse. A source administrator might later instantiate this template of a quality goal with the expected availability of his specific source database. Thus, the programmers of the data warehouse loading programs know the time window of the update process.

Based on the meta-model for data warehouse architectures, we have developed a set of quality factor templates which can be used as a initial set for data warehouse quality management. The exhaustive list of these templates can be found in [QJJ+98]. In [VBQ99], we have shown a methodology for the application of the architecture and quality model. The methodology is an adaptation of the *Total Quality Management* approach [BBBB95] and consists of the following steps:

- design of object types, quality factors and goals,
- evaluation of the quality factors,
- analysis of the quality goals and factors and their possible improvements, and
- re-evaluation of a quality goal due to the evolution of data warehouse.

The basic idea of [VBQ99] is to add (analytical) functions to the quality model which formalize the dependencies between the quality factors. Their inverse functions are use to find possibilities for the improvement of data warehouse quality.

2.3 A Quality-Oriented Data Warehouse Process Model

As described in the previous section it is important that all relevant aspects of a data warehouse are represented in the repository. Yet the described architecture and quality model does not represent the workflow which is necessary to build and run a data warehouse, e.g. to integrate data source or to refresh the data warehouse incrementally. Therefore, we have added a *data warehouse process model* to our meta modeling framework. Our goal is to have a simple process model which captures the most important issues of data warehouses rather than building a huge construction which is difficult to understand and not very useful due to its complexity.

Figure 3 shows the meta model for data warehouse processes. A data warehouse *process* is composed of several

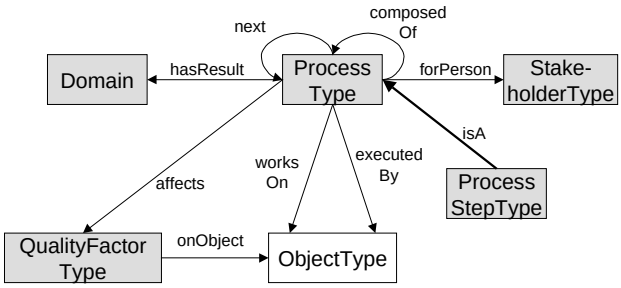

Figure 3: A process model for data warehouses

processes or *process steps* which may be further decomposed. Process steps and the processes itself are executed in a specific order which is described by the *"next"* relation between processes. A process *works on* an object type, e.g. data loading works on a source data store and a data warehouse data store. The process itself must be *executed by* some object type, usually an agent which is represented in the physical perspective of the architecture model. The *result* of a process is some value of a domain, the execution of further processes may depend on this value. For example, the data loading process returns as a result a boolean value representing the completion value of the process, i.e. if it was successful or not. Further process steps like data cleaning are only executed if the previous loading process was successful. The process is linked to a stakeholder which controls or has initiated the process. Moreover, the result of a process is the data which is produced as an outcome of the process, e.g. the tuples of a relation.

Processes *affect* a quality factor of an object type, e.g. the availability of data source or the accuracy of a data store. It might be useful to store also the expected effect on the quality factor, i.e. if the process improves or decreases the quality factor. However, the achieved effect on the quality factor can only be determined by a new measurement of this factor. A query on the metadata repository can then search for the processes which have improved the quality of a certain object.

The processes can be subject to quality measurement, too. Yet, the quality of a process is usually determined by the quality of its output. Therefore, we do not go into detail with process quality but quality factors can be attached to processes, too.

As an example for a data warehouse process we have partially modeled the data warehouse loading process in figure 4. The loading process is composed of several steps, of which one in our example is data cleaning. The data cleaning process step works on a data store, where the data which have to be cleaned reside. It is executed by some data cleaning agent. It affects among others the quality factors accuracy and availability, in the sense that accuracy is hopefully improved and availability is decreased because of locks due to read-write operations on the data store. The data cleaning process may also store some results of its ex-

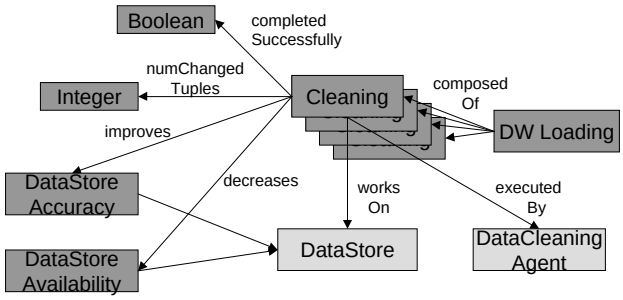

Figure 4: An example for a data warehouse process pattern

ecution in the metadata repository, for example, a boolean value to represent the successful completion of the process and the number of changed tuples in the data store. As already mentioned in section 2.2, the first instantiation level provides only a pattern for data warehouse processes, and not the "real" processes. The data of a "real" process is stored as an instance of this pattern (see below).

The information stored in the repository may be used to find deficiencies in data warehouse. To show the usefulness of this information we use the following query. It returns all data cleaning processes which have decreased the availability of a data store according to the stored measurements. The significance of the query is that it can show that the implementation of data cleaning process has become inefficient.

```
GenericQueryClass DecreasedAvailability
  isA DWCleaningProcess with
parameter
    ds : DataStore
constraint c :
 $ exists qf1,qf2/DataStoreAvailability
   t1,t2,t3/TransactionTime v1,v2/Integer
 (qf1 onObject ds) and (qf2  onObject ds) and
 (this worksOn ds) and (this executedOn t3) and
 (qf1 when t1) and (qf2 when t2) and (t1<t2) and
 (t1<t3) and (t3<t2) and (qf1 achieved v1) and
 (qf2 achieved v2) and (v1 > v2) $
end
```

The query has a data store as parameter, i.e. the query will return only cleaning processes which are related to the specified data store. The query returns the processes which have worked on the specified data store and which were executed between the measurements of quality factors `qf1` and `qf2`, and the measured value of the newer quality factor is lower than the value of the older quality factor. The query can be formulated in a more generic way to deal with all types of data warehouse processes but for reasons of simplicity and understandability, we have shown this more special variant.

Finally, figure 5 shows the trace of a process at the instance level. The process pattern for DW Loading has been instantiated with a real process, which has been executed on the specified date "April 15, 1999". An instantiation of the links to the quality factors is not necessary, because the information that "data cleaning" affects the accuracy and the availability of a data store is already recorded in the process pattern shown in figure 4.

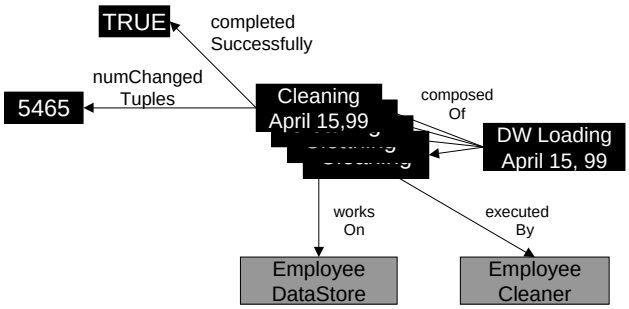

Figure 5: Trace of a data warehouse process

3 Data Warehouse Evolution

This section presents a framework for data warehouse evolution. It is based on the process model for data warehouses presented in the previous section. We will first discuss what types of evolution may occur in data warehouse. Finally, we will present the application of our framework to the evolution of data warehouse views.

3.1 Evolution in a Data Warehouse Environment

A data warehouse is a very complex system whose components evolve frequently independently of each other. Users can create new views or update old ones. Some sources may disappear while others are added. The enterprise model can evolve with the enterprise objectives and strategies. The technical environment changes with evolution of products and updates. Design choices at the implementation level can also evolve to achieve users requirements and administration requirements.

The data stores can produce changes due to reasons of schema evolution in the logical and conceptual perspective, changes to the physical properties of the source (e.g. location, performance etc.), insertions or deletions of data stores, and other reasons particular to their nature (e.g. in the sources, the time window for extraction or the data entry process can change). The software components can be upgraded, completed, debugged, etc. The propagation agents of all types (loaders/refreshers/wrappers/mediators/source integrators) can obtain new schedules, new algorithms, rules, physical properties, etc. Needless to say that the user requirements continuously change, too. New requirements arise, while old ones may become obsolete, new users can be added, priorities and expected/acceptable values change through the time, etc. Moreover, the business rules of an organization are never the same, due to changes in the real world.

As a result of evolution and errors, our goals, components, and quality factors are never to be fully trusted. Each time we reuse previous results we must always consider cases like: lack of measurement of several objects, errors in the measurement procedure (e.g. through an agent which is not appropriate), outdated information of the repository with respect to the data warehouse, etc.

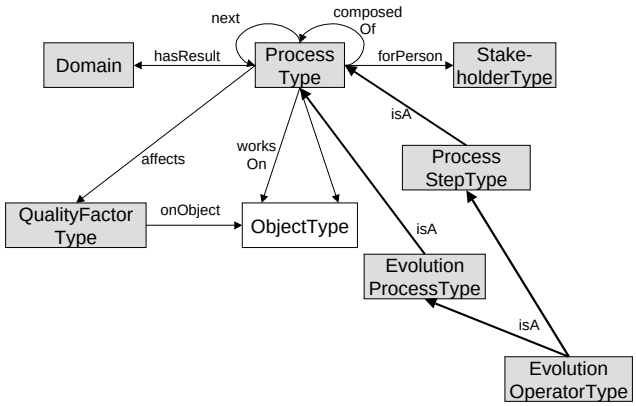

Figure 6: DW process model specialized for DW Evolution

3.2 A Meta Model for Data Warehouse Evolution

A way to control the evolution in data warehouses is to provide complementary meta-data which tracks the history of changes and provides a set of consistency rules to enforce when a quality factor has to be re-evaluated. To do so, it is necessary to link quality factors to evolution operators which affect them. The idea behind this is to enrich the meta-data repository in order to ease the impact analysis of each evolution operator and its consequences on the quality factor measures.

Our meta model for data warehouse evolution is a specialization of the data warehouse process model (see figure 6). An *evolution process* is composed of *evolution operators*, but also of "normal" data warehouse processes. For example, the materialization of new data warehouse is an evolution process of the data warehouse (cf. figure 7). This process includes the schema evolution operations such as "Add a new relation to the data warehouse schema" as well as the loading, extraction and writing process to evaluate the view and store its extent.

The example shown in figure 7 is also a pattern for an evolution process like the example in figure 4 is a pattern for a data warehouse process. Therefore, the pattern has to be further instantiated with an evolution process which has been executed on the data warehouse system. The information stored in the metadata repository can then be used to analyze the impact of certain evolution operations on the data warehouse.

3.3 Evolution of Data Warehouse Views

To be useful, the described framework for data warehouse evolution must be filled with patterns of evolution processes. As an example, we will discuss the evolution of views in data warehouses. The evolution of data warehouse views has been studied recently in the research fields of schema evolution [RLN97, Bell98, Blas99] and maintenance of data warehouse views under view redefinition [GMR95]. In this section, we do not provide a new technique for schema evolution or view maintenance of data

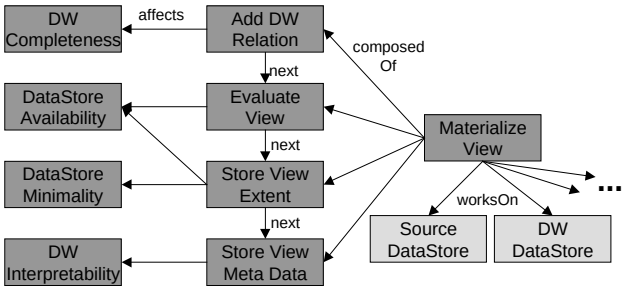

Figure 7: DW Evolution: Materialization of a view

warehouse views. Our goal is to provide a framework for the quality-oriented evolution of a data warehouse and the existing techniques are integrated into our framework to make use of the semantically rich meta database.

One application of our framework is the monitoring of data warehouse quality under the evolving environment of a data warehouse. As described in section 2.2 quality measurements should be repeated periodically to monitor how the quality of the data warehouse evolves. In addition to the architecture and quality model, the meta model for data warehouse evolution keeps track of the (evolution) processes which have changed the configuration of the data warehouse or have changed the data of the warehouse. With this information, it is possible to trace the evolution of the data warehouse. If a quality problems occurs, the meta-data repository can be used to find the (evolution) process which has caused the quality problem. In the rest of this section, we will use the example for the evolution of data warehouse views to show the usefulness of our approach.

In [CNR99] a taxonomy for schema evolution operators in object-oriented databases is given. We have adapted this taxonomy to relational databases, which are often used in data warehouses. Table 1 summarizes the evolution operators for base relations and views, and relates them to the quality factors which are affected by this evolution operator.

The evolution operators for base relations and views in data warehouse mainly work on the representation of the relation in the logical perspective of the architecture model, i.e. the relation itself and the logical schema it belongs to. Moreover, they affect the physical objects where the data of the relation is stored or where the view is materialized, i.e. the data stores. In addition, if there exists another view which is based on the evolved relation or view, then the view definition, the materialization of the view, and the maintenance procedure must be updated, too.

The completeness, correctness and consistency of the logical schema with respect to conceptual model are the most important quality factors affected by these evolution operators. Furthermore, the deletion of a base relation or an attribute might have a positive impact on the minimality or the redundancy of the logical schema. The renaming of attributes and relations to more meaningful names improves

Table 1: Evolution Operators for base relations and views in DWs and their effect on DW quality

Evolution Operator	**Affects Quality Factor**	**Works On**
Add base relation/view	- Completeness, correctness and consistency of the logical schema wrt. the conceptual model - Usefulness of schema - Availability of the data store	- Relation - Logical Schema - Data Store
Delete base relation/view	- Minimality of logical schema - Completeness, correctness and consistency of the logical schema wrt. the conceptual model - Availability of data store	- Relation, Log. Schema - Data Store - View - View Maintenance Agent
Add attribute to base relation/view	- Completeness, correctness and consistency of the logical schema wrt. the conceptual model - Interpretability of the relation - Redundancy of the attributes	- Relation - Data Store - View - View Maintenance Agent
Delete attribute from base relation/view	- Completeness, correctness and consistency of the logical schema wrt. the conceptual model - Interpretability of the relation - Redundancy of the attributes	- Relation - Data Store - View - View Maintenance Agent
Rename Relation, View, or Attribute	- Interpretability and understandability of the relation and their attributes	- Relation, View - Data Store, VM Agent
Change of attribute domain	- Interpretability of data	- Relation, View - Data Store, VM Agent
Add Integrity Constraint	- Credibility and Consistency of data in data store	- Logical Schema - Data Store
Delete Integrity Constraint	- Consistency of data wrt. integrity constraints	- Logical Schema - Data Store
Change to view definition	- Completeness, correctness and consistency of the logical schema wrt. the conceptual model - Usefulness of schema	- View - Data Store - View Maintenance Agent

the interpretability and the understandability of the logical schema. The change of the domain of an attribute to a more applicable domain, e.g. changing the domain from string to date, improves the interpretability of data. New integrity constraints in the logical schema may improve the credibility and the consistency of the data. Finally, if the view definition is changed without an impact on the structure of the view (e.g. the WHERE clause in a SQL statement is changed) the view may become useful for more client applications.

As an example to show the usefulness of the data warehouse evolution model, we suppose that an analyst has detected that the views he is using are changed often, and that he wants to get notified about future changes. We can establish a view on the metadata repository for the analyst which monitors the changes to the view he is interested in.

```
View EvolutionOperationsOnView
  isA RelationalEvolutionProcess with
parameter
    v : DWView
constraint
    c: $ (this worksOn v) $
end
```

This view returns all evolution operations which are made to the given data warehouse view assuming that all evolution processes concerning relational schema evolution are instances of the process type RelationalEvolutionProcess. A similar view might be useful for data warehouse administrators which notifies them if base relations have changed. Our repository system ConceptBase is able to maintain views on the metadata and supports the notification of external client applications if a view has changed they are interested in [SQJ98].

3.4 Case Study

In [Lehm97], a commercial case study is described in which an early version of the approach described above has been used to link changes in the definition of materialized views of the data warehouse to changes in the view maintenance strategy.

We have developed a tool for data warehouse design at the relational level, aiming at several data warehouse quality goals like reusability of solutions, sufficient and flexible freshness of data, ability for evolution of source or data warehouse schemas, and clear process definitions for data integration and refreshment. Prior to the development of this tool, especially the goal of flexibility was hampered by the need to re-program scripts whenever schema or policy changes happened.

Due to the constraints in the project, we decided to decompose the data warehouse views into several self-

maintainable views [HZ96]. The design tool records the schema definitions of the source systems and the view definitions of the warehouse. It then decomposes the views and creates automatically the SQL statements to initialize and incrementally maintain the views.

If the schema of the sources or of the data warehouse has changed, only little effort is necessary to update the maintenance processes, i.e. only the SQL code has to be re-generated. However, we did not support the adaptation of the tuples in the relations to the new schema.

The tool has been integrated into a commercial product suite for sales force automation and has significantly reduced the effort of data warehouse maintenance [JQB+99].

4 Related Work

An approach for the management of views in a federated database system is proposed in [KGF98]. The approach is based on a knowledge base which stores what information is available in the federated database, how it has been combined previously, and how the information is related semantically. A workbench of tools assist users to create and evolve the knowledge base and their views on the federated database system.

Research in data warehouses addresses the evolution problem from two different perspectives. The first aspect is the schema evolution of base relations and views, which has been studied in [Bell98], [RLN97] and [Blas99]. [Bell98] provides a set of algorithms to maintain the definitions of views if the schema of the base relations is changing. Furthermore, different versions of a view are constructed and maintained if the view definition has changed. The versioning of views is necessary because not every client application of the data warehouse can be adapted to the new version of the view.

[RLN97] provides a taxonomy of view adaptations problems. The taxonomy is based upon the types of changes to the view, the desired level of view adaptability in the context of changes, and the changes related to the base information system, e.g. data updates, capability changes or metadata changes. They present an environment for the view synchronization problem, i.e. the view definition adaptation is triggered by capability changes of information systems. Other work in the context of schema evolution has been devoted to the evolution of schemas in object-oriented databases like [CNR99].

[Blas99] presents a framework for the evolution of conceptual multidimensional schemata. In this approach, the data warehouse is designed and maintained at a conceptual level. Each evolution operation at the conceptual level has well-defined semantics and is mapped to a physical implementation level. The framework supports among other features the automatic adaptation of instances, change notification for applications, and forward compatibility of schemata.

The second perspective which addresses the problem of evolution of data warehouse views, is maintenance of the extent of a view. In [GMR95], the problem of incremental view maintenance under view redefinition is studied. An overview and a taxonomy of view maintenance problems is given in [GM95]. [HMV99] studies the problem of maintaining multi-dimensional data cubes under dimension updates. They define a basic set of operators why modify the dimensions of a data cube. Moreover, they provide an algorithm for maintaining the data cube under these update operations.

5 Conclusions

We have extended our meta modeling framework for data warehouse architecture and quality by a model for data warehouse processes and have specialized this model to the case of data warehouse evolution. In detail, we have addressed the problem of evolution of data warehouse views. The management of the metadata in our repository system ConceptBase allows us to query and analyze the stored metadata for errors and deficiencies. In addition, features like client notification and active rules of ConceptBase support the maintenance of the data warehouse components and keep data warehouse users up-to-date on the status of the data warehouse.

In the DWQ project, we are currently studying some data warehouses processes like update propagation, querying, and conceptual design. Furthermore, the different types of data warehouse evolution mentioned in section 3.1 have to be studied in more detail. In this context, the proposed models will be refined and extended to cover new aspects of data warehouse processes. A validation of the data warehouse process model with one of our industrial cooperation partners - a small data warehouse application vendor - is also planned for the future.

Acknowledgments

This research is sponsored by the European Esprit Project “DWQ: Foundations of Data Warehouse Quality”, No. 22469. We would like to thank all our DWQ partners who contributed to the progress of this work, and especially Matthias Jarke, Manfred A. Jeusfeld, Mokrane Bouzeghoub, and Panos Vassiliadis.

References

[BBBB95] D.H. Besterfield, C. Besterfield-Michna, G. Besterfield and M. Besterfield-Sacre, *Total Quality Management*, Prentice Hall, 1995.

[Bell98] Z. Bellahsène. Structural View Maintenance in Data Warehousing Systems. *Journées Bases de Données Avancées (BDA ’98)*, Tunis, October 1998.

[Blas99] M. Blaschka. FIESTA: A Framework for Schema Evolution in Multidimensional Information Systems. In *Proc. 6th CAiSE Doctoral Consortium*, Heidelberg, Germany, June 1999.

[CNR99] K.T. Claypool, C. Natarajan, E.A. Rundensteiner. Optimizing the Performance of Schema Evolution Sequences. *Technical Report WPI-CS-TR-99-06*, Worcester Polytechnic Institute, Dept. of Computer Science, March 1999.

[GM95] A. Gupta, I.S. Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Warehousing*, **18**(2), 1995.

[GMR95] A. Gupta, I.S. Mumick, K.A. Ross. Adapting Materialized Views after Redefinitions. In *Proc. ACM SIGMOD International Conference on Management of Data*, pp. 211–222, 1995.

[HMV99] C.A. Hurtado, A.O. Mendelzon, A.A. Vaisman. Maintaining Data Cubes under Dimension Updates. In *Proc. 15th Intl. Conference on Data Engineering (ICDE '99)*, Sidney, Australia, 1999.

[HZ96] R. Hull, G. Zhou. A framework for supporting data integration using the materialize and virtual approaches. *Proc. ACM SIGMOD Intl. Conf. Management of Data*, Montreal, Canada, 1996.

[JGJ+95] M. Jarke, R. Gallersdörfer, M.A. Jeusfeld, M. Staudt and S. Eherer. ConceptBase - a deductive object base for meta data management, *Journal of Intelligent Information Systems*, **4**(2), 1995, pp. 167–192.

[JJQV99] M. Jarke, M.A. Jeusfeld, C. Quix, P. Vassiliadis. Architecture and Quality in Data Warehouses: An Extended Repository Approach. *Information Systems*, **24**(3), pp. 229–253, 1999 (a previous version appeared in *Proc. of the 10th Conference on Advanced Information Systems Engineering (CAiSE '98)*, pp. 93–113, Pisa, Italy, 1998).

[JQB+99] M. Jarke, C. Quix, G. Blees, D. Lehmann, G. Michalk, S. Stierl. Improving OLTP Data Quality Using Data Warehouse Mechanisms. In *Proc. ACM SIGMOD Intl. Conf. Management of Data*, Philadelphia, PA, 1999.

[JQJ98] M.A. Jeusfeld, C. Quix, M. Jarke. Design and Analysis of Quality Information for Data Warehouses. In *Proc. of the 17th International Conference on the Entity Relationship Approach (ER'98)*, Singapore, 1998.

[JV97] M. Jarke, M. Vassiliou. Foundations of data warehouse quality: an overview of the DWQ project. In *Proc. of the 2nd International Conference on Information Quality*, Cambridge, Mass, 1997.

[KGF98] D.D. Karunaratna, W.A. Gray, N.J. Fiddian. Organising Knowledge of a Federated Database System to Support Multiple View Generation. In *Proc. 5th KRDB Workshop (Knowledge Representation meets Data Bases)*, Seattle, May 1998.

[Lehm97] D. Lehmann. View Maintenance in a Data Warehouse Environment. *Diploma Thesis*, RWTH Aachen, November 1997 (in german).

[OB92] M. Oivo, V. Basili. Representing software engineering models: the TAME goal-oriented approach. *IEEE Transactions on Software Engineering*, **18**(10), 1992.

[QJJ+98] C. Quix, M. Jarke, M.A. Jeusfeld, M. Bouzeghoub, D. Calvanese, E. Franconi, M. Lenzerini, U. Sattler, P. Vassiliadis. Quality Oriented Evolution of DW Designs. *Technical Report, DWQ 9.1*, DWQ Consortium, 1998.

[RLN97] E.A. Rundensteiner, A.J. Lee, A. Nica. On Preserving Views in Evolving Environments. In *Proc. 4th KRDB Workshop (Knowledge Representation meets Data Bases)*, Athens, 1997.

[SQJ98] M. Staudt, C. Quix, M.A. Jeusfeld. View Maintenance and Change Notification for Application Program Views. *ACM Symposium on Applied Computing*, Atlanta, Georgia, 1998.

[TB98] G.K. Tayi, D.P. Ballou. Examining Data Quality. *Communications of the ACM*, **41**(2), pp. 54–57, Feb. 1998.

[VBQ99] P. Vassiliadis, M. Bouzeghoub, C. Quix. Towards Quality-oriented Data Warehouse Usage and Evolution. In *Proc. of the 11th Conference on Advanced Information Systems Engineering (CAiSE '99)*, Heidelberg, Germany, 1999.