Pergamon

Information Systems Vol. 25, No. 2, pp. 89–115, 2000

Printed in Great Britain
0306-4379/00 $20.00

PII: S0306-4379(00)00011-9

TOWARDS QUALITY-ORIENTED DATA WAREHOUSE USAGE AND EVOLUTION

PANOS VASSILIADIS[1], MOKRANE BOUZEGHOUB[2], and CHRISTOPH QUIX[3]

[1]Computer Science Division, National Technical University of Athens, Zographou 15773, Athens, Greece
[2]Laboratoire PriSM, Université de Versailles and INRIA Rocquencourt, France
[3]RWTH Aachen, Informatik V, Ahornstr. 55, 52074, Germany

(Received 15 December 1999; in final revised form 24 March 2000)

Abstract — As a decision support information system, a data warehouse must provide high level quality of data and services. In the DWQ project (Foundations of Data Warehouse Quality), we have proposed how semantically rich meta-information of a data warehouse can be stored in a metadata repository. This *static* representation of the various perspectives of data warehouse components and their linkage to quality factors is complemented by an *operational* methodology on how to use these quality factors and achieve the quality goals of the users. This approach is an extension of the Goal-Question-Metric (GQM) approach, based on the idea that a quality goal is operationally defined over a concrete set of questions, i.e., algorithmic steps. The proposed approach covers the full lifecycle of the data warehouse, allows capturing the interrelationships between different quality factors and helps the interested user to organize them in order to fulfill specific quality goals. Furthermore, we prove how the quality management of the data warehouse can guide the process of data warehouse evolution, by tracking the interrelationships between the components of the data warehouse. Finally, we present a case study, as a proof of concept for the proposed methodology. © 2000 Published by Elsevier Science Ltd. All rights reserved

Key words: Data Warehousing, Repositories, Evolution, Data Quality

1. INTRODUCTION

Many researchers and practitioners share the understanding that a Data Warehouse (DW) architecture can be formally understood as layers of materialized views on top of each other. A data warehouse architecture exhibits various layers of data in which data from one layer are derived from data of the lower layer. *Data sources*, also called *operational databases*, form the lowest layer. They may consist of structured data stored in open database systems and legacy systems, or unstructured or semi-structured data stored in files. The central layer of the architecture is the *global* (or *primary*) *data warehouse*. The global data warehouse keeps a historical record of data that result from the transformation, integration, and aggregation of detailed data found in the data sources. Usually, a data store of volatile, low granularity data is used for the integration of data from the various sources: it is called *Operational Data Store (ODS)*. The Operational Data Store, serves also as a buffer for data transformation and cleaning so that the data warehouse is populated with clean and homogeneous data. The next layers of views are the *local*, or *client* warehouses, which contain highly aggregated data, directly derived from the global warehouse. There are various kinds of local warehouses, such as the *data marts* or the *OLAP databases*, which may use relational database systems or specific multidimensional data structures.

All the data warehouse components, processes and data are – or at least should be – tracked and administered from a *metadata repository*. The metadata repository serves as an aid both to the administrator and the designer of a data warehouse. Indeed, the data warehouse is a very complex system, the volume of recorded data is vast and the processes employed for its extraction, transformation, cleansing, storage and aggregation are numerous, sensitive to changes and time-varying. The metadata repository serves as a roadmap that provides a trace of all design choices and a history of changes performed on its architecture and components. For example, the new version of the *Microsoft Repository* [1] and the *Metadata Interchange Specification* (MDIS) [15] provide different models and application programming interfaces to control and manage metadata for OLAP databases. In Figure 1, a generic architecture for a data warehouse is depicted.

As a decision support information system, a data warehouse must provide high level quality of data and service. Coherency, freshness, accuracy, accessibility, availability and performance are among the

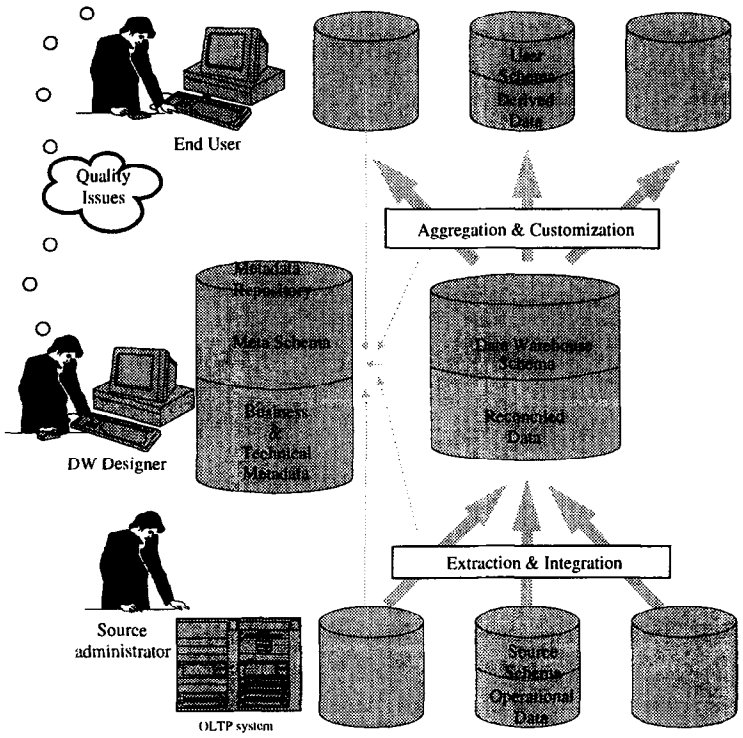

Fig. 1: A Generic Architecture for a Data Warehouse

quality features required by data warehouse users. The metadata repository plays a central role in this concern, as it provides the necessary knowledge to understand, evaluate and analyze the architecture of a data warehouse, in order to predict its behavior and the resulting quality of service and quality of data.

Data quality has been defined as the fraction of performance over expectancy, or as the loss imparted to society from the time a product is shipped [3]. We believe, though, that the best definition is the one found in [17, 20, 27, 29]: data quality is defined as "fitness for use". The nature of this definition directly implies that the concept of data quality is relative. For example, data semantics (the interpretation of information) is different for each distinct user. As [17] mentions "the problem of data quality is fundamentally intertwined in how [...] users actually use the data in the system", since the users are the ultimate judges of the quality of the data produced for them: if nobody actually uses the data, then nobody will ever take care to improve its quality.

From the previous it follows that, on the one hand, the quality of data is of highly *subjective* nature and should ideally be treated differently for each user. On the other hand, the reasons for data deficiencies, non-availability or reachability problems are definitely *objective*, and depend mostly on the definition and implementation of the information system. Furthermore, the evaluation of data quality for each user must be based on objective quality factors, which are computed and compared to the users' expectations. The question that arises, then, is how to tune the design choices in such a way that all the different, and sometimes opposing, user requirements can be simultaneously satisfied. As the number of users and the complexity of data warehouse systems do not permit to reach total quality for every user, a subsidiary question is how to prioritize these requirements in order to satisfy them with respect to their importance. This problem is typically illustrated by the physical design of the data warehouse where the problem is to find a set of materialized views that optimize the response time of user requests and the global data warehouse maintenance cost [14, 22, 23, 24].

In [13] a metadata modeling approach is presented that enables the capturing of all the crucial parts of the architecture of a data warehouse, along with information over different quality dimensions of these components. In this paper, we have refined the quality metamodel with a more detailed linkage between objective quality factors and user-dependent quality goals. Moreover, we have extended the Goal-Question-Metric (GQM) methodology [2] in order (a) to capture the interrelationships between different quality factors with respect to a specific quality goal, and (b) to define an appropriate lifecycle that deals with quality goal evaluation and improvement.

Our methodology comprises a set of steps aiming, on the one hand, to map a high-level subjective quality goal into the measurement of a set of interrelated quality factors, and, on the other hand, to propose improvement actions which may help in achieving the target quality goal. These steps involve the *design* of the quality goal, the *evaluation* of the current status, the *analysis* and *improvement* of this situation, and finally, the *re-evaluation* of the achieved plan. The metadata repository together with this quality goal definition methodology constitute a decision support system which helps data warehouse designers and administrators to take relevant decisions, in order to achieve reasonable quality level which fits the best user requirements. This work is integrated in a methodology for data warehouse quality design, which has been developed in the European DWQ project (Foundations of Data Warehouse Quality) [10].

We want to stress out that we do not follow the ISO 900x paradigm [8] in our approach; rather we try to present a computerized approach to the stakeholder, for both the storage and exploitation of information relevant to the quality of the data warehouse. The objective of this paper is to show how subjective quality goals can be evaluated using more objective quality factors, following an extended GQM approach.

The paper is organized as follows: Section 2 describes the DWQ quality metamodel and an example for its instantiation. In Section 3, we detail the DWQ methodology for quality management. Section 4 presents some hints on data warehouse evolution. A case study for the partial application of the methodology is presented in Section 5. Section 6 summarizes related work and finally, in Section 7 we discuss our results.

2. METADATA REPOSITORY AND QUALITY MODEL

This section summarizes the nature of metadata used in the DWQ framework and gives an overview of the DWQ quality model. The section particularly focuses on the quality dimensions and quality factors associated with the main data warehouse meta-objects. To further illustrate the relevance of data warehouse objects and quality factors, we use as examples, two of the most crucial processes in the data warehouse lifecycle, the design and the refreshment processes, along with their corresponding quality factors.

2.1. Architecture Components

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 [13], it is shown that the data warehouse metadata should track both architecture components and quality factors.

During the analysis of existing data warehouse frameworks we made the observation that these frameworks cover only logical schemata and their physical implementation; hence, interpretation of these representations is far from being natural for data warehouse users. Furthermore, since data warehouses may be built on a huge number of heterogeneous data sources, it is difficult to have an overall picture of what kind of data is available in each source and to keep track of the interdependencies between these data sources. Finally, any data warehouse design should satisfy some quality requirements without which the derived decision data become useless.

Therefore, we have extended the traditional data warehouse architecture in three ways:

(i) We have added a conceptual *perspective* which provides a clear understanding of the three data warehouse *levels*, namely the source level, the data warehouse level and the client level. We use the term *perspective* to denote the classical categorization of database models to *conceptual, logical* and *physical.*

(ii) We considered both source schemata and client schemata as views defined over a global enterprise model which describes the data warehouse level. The enterprise model is the central component in our framework; it gives a global understanding of the data warehouse subjects as well as the interrelationships between its components.

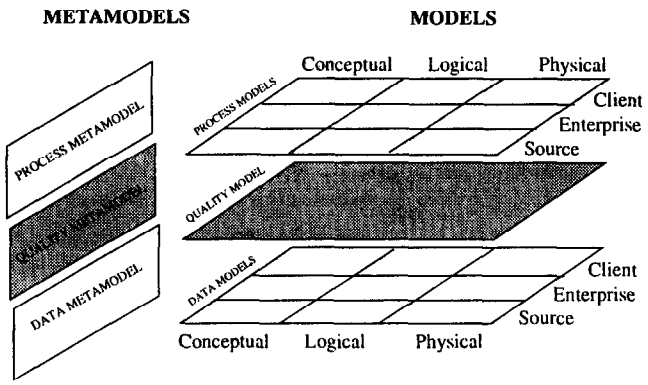

Fig. 2: The Data Warehouse Architecture Metamodel

(iii) We have associated each data warehouse component with a set of quality factors which characterize its usage and evolution. A quality model is then added as a corner stone to our design framework. The overall picture of the DWQ framework is depicted Figure 2.

The vertical plane on the left hand-side represents the metamodels which describe any data warehouse system. Each instantiation of these metamodels corresponds to a particular data warehouse system with its own data, process and quality models. These models are represented by the horizontal planes in Figure 2. The data warehouse data and process models are structured into the three levels and three perspectives. The different components of these models are interlinked with mappings between perspectives and levels. The intermediate layer describes the quality model for a given data warehouse system. It provides a bridge between data and process models by defining quality goals. Quality goals provide reasons why certain processes are executed and how they affect the data [25]. For all models, we have provided templates which guide the design and the maintenance of any data warehouse system [13]. These template models can be further refined and adopted to specific data warehouse applications. In Sections 2.3 and 2.4, we will present models for the data warehouse refreshment and design processes.

Different formalisms can be used to describe the various models and metamodels in the framework. The conceptual definition of the metamodels is expressed in an extended entity-relationship model which is viewed as a graphical interface to a description logic language [5]. Description logic allows a more complete and precise specification and a formal reasoning on data warehouse objects. The models at the logical perspective are assumed to be defined in the relational model. The description of the models at the physical perspective depends on each data source, the target data warehouse system and the client tools.

The metamodels are described in Telos, a conceptual modeling language for representing knowledge about information systems. Telos is implemented in the deductive object-oriented repository system ConceptBase [12] and supports the representation of models at any abstraction level. Thus we can represent the metamodels and their instantiations in a unified framework in ConceptBase. ConceptBase provides also a query language to analyze the contents of the repository. Furthermore, external applications like a reasoner for description logics can easily access the meta-information and check for the consistency of the conceptual models. Still, although empowered by the query and deductive facilities of ConceptBase, our approach can be applied to any metadata repository holding the respective information for the data warehouse.

2.2. *Quality Metamodel*

Each object in any level and perspective of the architectural framework can be subject to quality measurement. Since quality management plays an important role in data warehouses, we have incorporated it in 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.

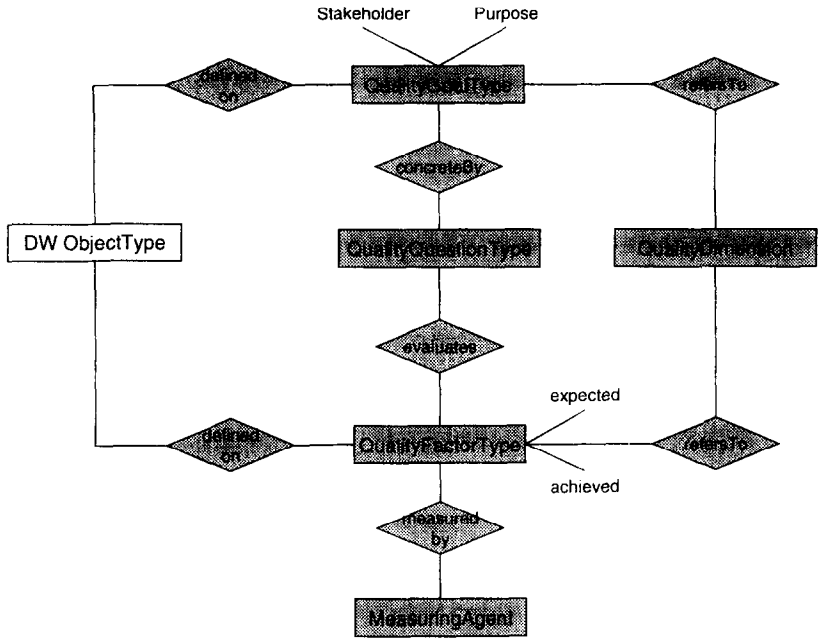

Fig. 3: The DWQ Quality Metamodel (Simplified)

The DWQ quality metamodel [11] is based on the Goal-Question-Metric approach (GQM) of [16] 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) we make a clear distinction between subjective quality goals requested by the stakeholders and objective quality factors attached to data warehouse objects, (ii) quality 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 3 shows a simplified conceptual view of the DWQ Quality Model. The class 'DW object type' refers to any meta-object of the DWQ framework depicted in Figure 2. A *quality goal* is an abstract requirement, defined on data warehouse object types, and documented by a purpose and the stakeholder interested in. This s expresses natural language requirements like 'improve the availability of source s1 until the end of the month in the viewpoint of the data warehouse 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. 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 the quality dimension of the quality factor. It also represents the expected range of the quality value, which may be any subset of a domain. Dependencies between quality factors are also stored in the repository. Finally, 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.

The overall setting is not separated from the automation of the solution of problems occurring in a data warehouse environment. Each problem can be expressed in terms of measurement, optimization or improvement of particular quality factors. Thus, we can exploit all the work produced by research and practice, in terms of automated techniques and algorithms, by linking it to a formal quality model. We discriminate between three categories of quality factors:

Primary quality factors, which are simple estimations of stakeholders or direct measurements and play the role of input of the employed algorithms.

- *Derived quality factors*, which are the outputs of the problem-solving techniques and can be assumed to be the result of a function application over the primary factors.
- *Design choices*, which are parameter values and control strategies and aim to regulate or tune the employed algorithm for the solution of a particular problem.

For example, as far as the problem of data accuracy is concerned, the accuracy of the data of the sources, their size and the desired/acceptable range for the accuracy of the data in the warehouse are primary quality factors, the employed data cleaning algorithm is a design choice and the produced accuracy after the cleaning has taken place, as well as the overall time that the cleaning algorithm took to successfully terminate are the derived quality factors of the problem. We will elaborate more on this discrimination in Section 2.3 and Section 2.4, where we examine the data warehouse refreshment and the design problems.

The quality repository is not instantiated simply with concrete quality factors and goals, but also 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.

In [13], based on our practical experience, we have identified the following *roles* (now: *viewpoints*) of users in a data warehouse environment. Note that these are roles and not different persons; i.e. a person can be involved in more than one role:

- *Decision maker;*
- *Data warehouse administrator;*
- *Data warehouse designer;*
- *Programmers of data warehouse components;*
- *Source data users, administrators, and designers.*

For each role we describe some template quality goals. We do not claim that we cover the subject exhaustively, but rather we provide an illustrative initial set of templates which can be refined during the use of the data warehouse. We have summarized the quality goals in Table 1. In the rest of this subsection we will detail these roles and template quality goals.

The *Decision Maker* is the final user of the data warehouse. He usually uses an OLAP query tool to get answers interesting to him. The main issues regarding the decision maker are the quality of the information (if it is relevant for his needs, fresh enough to reflect the real world, complete, consistent etc.) and the efficiency of the use of the system (how easy it is for him to work with the tools, how quickly are the responses given to him, etc.).

The *Data Warehouse Administrator* is concerned with keeping the data warehouse properly operating. Several aspects are involved in the everyday use of the data warehouse: the system must be available to the users for querying, the querying must be done as efficiently as possible, the refreshment must be performed successfully and the evolution should happen smoothly. Moreover, the quality of the stored data should be satisfying for the needs of the users. To accomplish the aforementioned tasks, the data warehouse administrator can be aided by facilities such as error reporting (e.g., for the detection of errors during the execution of any process) and metadata accessibility (for the high-level description of the system).

The *Data Warehouse Designer* is involved in a continuous design process, which is usually done incrementally. The data warehouse designer is concerned with two major issues, namely the construction of the data warehouse schema and the design of the data warehouse processes. For the former, the basic requirement is that the derived data warehouse schema is design efficiently and in accordance with the requirements of the application, the schemata of the existing sources and, most important, with the user needs. For the latter, it is important to take into account the derived schemata, so that the population processes are consistent with the underlying schema and the constraints posed by the user needs and the environment (freshness of data, granularity of the information, availability windows of the sources, etc.).

Stakeholder	Purpose	Issue	Object
Decision Maker	Evaluate, understand, improve	Overall data quality (freshness, accuracy, completeness, coherence, relevance)	Stored Data
		Query efficiency (easiness, responsiveness)	DW, OLAP tool
DW Admin.	Evaluate, understand, improve	Overall data quality (freshness, accuracy, completeness, coherence, relevance)	Stored data
		System availability	Data Warehouse
		Query efficiency	DW, OLAP tools
		Refreshment efficiency	DW, sources
		Evolution efficiency	Data Warehouse
		Error reporting	DW processes
		Metadata accessibility	Metadata
DW Designer	Evaluate, understand, improve	Schema quality (design efficiency, design consistence)	Source and DW schemata
		Software quality (implementation efficiency, consistency with schemata and any other constraints)	Software packages
		Metadata quality	Metadata
Programmers of DW Components	Evaluate, understand, improve	Overall software quality	DW components
		Metadata accessibility	Metadata
Legacy Systems Stakeholders	Evaluate, understand, improve	Reporting (feedback) on the data quality (freshness, accuracy, completeness, coherence, relevance)	Source data
		System availability	Source operational system

Table 1: Template Quality Goals for Different Stakeholders

Of course, the availability of all the meta-information is crucial for the success of the work undertaken by the data warehouse designer.

The *Programmers of Data Warehouse Components* are the people who develop the actual data warehouse applications. Obviously, their primary concern is the quality of all *the data warehouse software components*, produced or purchased. To this end, the data warehouse programmers need software implementation standards to test them (e.g., the respective ISO standard [8]). Especially for the production / customization of the software components, it is important that they have efficient access to the metadata of the warehouse, so that their implementation is successful.

The *Source Data Users/Administrators/Designers* are affected from the data warehouse in the sense that they could both benefit and be disturbed from its existence, at the same time. The benefit from the existence of the warehouse could possibly be the extra knowledge of the quality of the source data. On the other hand, this can also be the reason for the disturbance of the respective stakeholders. Also, the source data administrators are obviously affected from the existence of the data warehouse, due to the extra workload both of the system and their own.

Based on the metamodel of data warehouse architecture, we have also developed a set of quality factor templates which can be used as an initial set for data warehouse quality management [13]. The following sections give an intuition of the quality factors, associated to the data warehouse refreshment process and to the data warehouse optimal design.

2.3. Example 1: Data Warehouse Refreshment

The refreshment process aims to propagate changes raised in the data sources to the data warehouse stores. This propagation is done through a set of independent activities (extraction, cleaning, integration, etc.) that can be organized in different ways. The ordering of these activities and the context in which they are executed define the semantics of the refreshment process and influence its quality [4]. As shown in [13], it is not sufficient to describe a data warehouse as layers of materialized views on top of each other. For example, a view definition is not sufficient to capture the semantics of the refreshment process. Indeed, a view definition does not include information such as whether this view operates on a history or not, how this history is sampled, which transformations are necessary during the refreshment, whether the changes of a given source should be integrated each hour or each week, and which data timestamp should be taken when integrating changes of different sources. Consequently, based on the same view definitions, a refreshment process may produce different results depending on all these extra parameters which have to be fixed independently, outside the queries which define the views.

Quality Dimension	DW objects	Primary Quality Factors	Derived Quality Factors	Design Choices
Coherence	- Sources - ODS - Views	- Availability window of each source - Expected response time for a given query	- Extraction frequency of each source - Estimated response time of extraction for each source	- Granularity of data - Extraction and cleaning policy - Integration policy
Completeness	- Sources - ODS	- Availability window of each source - History duration for each DW store - Percentage of present vs. estimated full data items	- Extraction frequency of each source	- Extraction policy - Integration policy
Freshness	- Sources - ODS - Views	- Availability window of each source - Expected freshness for a given query - Estimated response time of extraction for each source, of integration and of propagation - Volume of data extracted and integrated	- Extraction frequency of each source - Actual freshness for a given query - Actual response time for a given query	- Extraction policy - Integration policy - Update policy

Table 2: Different Levels of Abstraction for the Management of Quality for the Refreshment of the Data Warehouse

As mentioned before, the refreshment process is one of the main data warehouse processes for which the quality is an important issue. The associated quality template includes quality dimensions such as coherence, completeness and freshness.

- *Data coherence*: the respect of (explicit or implicit) integrity constraints from the data. For example, the conversion of values to the same measurement unit allows also doing coherent computations.
- *Data completeness*: the percentage of data found in a data store, with respect to the necessary amount of data that should rely there.
- *Data freshness*: the age of data (with respect to the real world values, or the date when the data entry was performed).

Given a quality dimension, several low level quality factors of this dimension may be defined in a data warehouse. For example, one can define quality factors like the *availability window* or the *extraction frequency* of a source, the *estimated values for the response time* of an algorithm or the *volume of the data extracted each time*, etc. In Table 2, which was derived both from practical experience and the study of research results in the field of data warehousing, we mention several quality factors which are relevant to the refreshment process and link them to the corresponding data warehouse objects and quality dimensions. One can also notice that some quality factors may belong to more than one dimension. Some of them are primary quality factors, arbitrarily assigned by the data warehouse administrator whereas others are derived. The deriving procedures can be mathematical functions, logical inferences or any *ad hoc* algorithms. The values of derived quality factors depend on design choices which can evolve with the semantics of the refreshment process. Underlying the design choices are design techniques, that comprise all the rules, events, optimizations and algorithms which implement the strategies on which refreshment activities are based.

We believe that quality dimensions, quality factors and design choices are tightly related. For example, in the case of the refreshment process, the design choice 'extraction policy' is related to the derived quality factor 'extraction frequency' of each source, which is computed from the corresponding primary quality factor 'availability window' of each source. As another example, the completeness of a source content may be defined with respect to the real world this source is supposed to represent. Hence, the completeness of this source is a primary quality factor, since it is a subjective value directly assigned by the data warehouse administrator or the source administrator. On the other hand, the completeness of the operational data store content is a derived quality factor, since it can be defined as a formula over the completeness of the sources. The extraction, integration and update policies are algorithms and parameters tuned to complete the task of data warehouse refreshment. The designer can pick from a variety of alternatives for each of these policies. The particular design choice he makes ultimately affects the quality of the involved dimensions and factors of the problem.

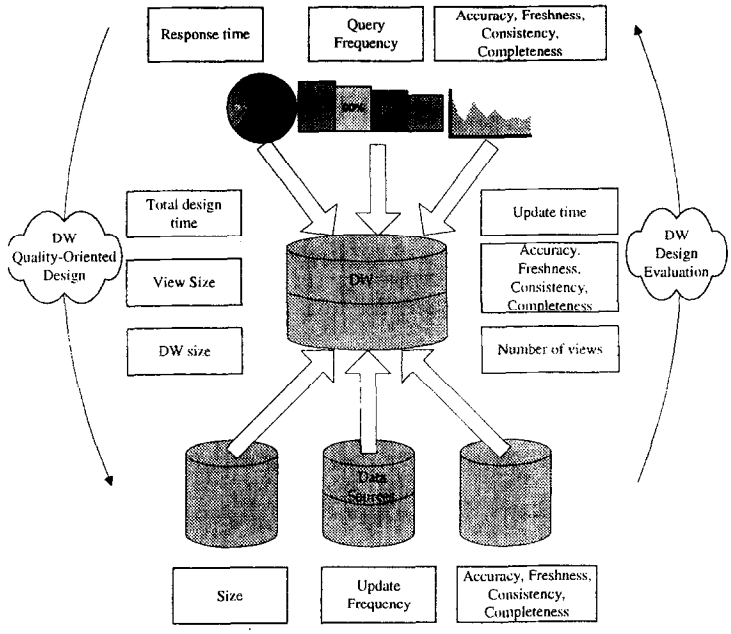

Fig. 4: The Quality Scenario for the Data Warehouse Optimal Design Problem

2.4. Example 2: Data Warehouse Design Optimization

In [24], a data warehouse is seen as a set of materialized views defined over distributed heterogeneous databases. All the queries posed to the data warehouse are evaluated locally using exclusively the data that are stored in the views. The materialized views have also to be maintained when changes occur to the data of the sources. Note that the term "queries" includes both the possible ad-hoc user queries (seeking for detailed information) and the batch, regularly posed queries used for the updating of the high level, aggregated views of the data warehouse and its data marts.

For a given set of different source databases and a given set of queries over the data warehouse, there is a number of alternative sets of materialized views that the administrator can choose to maintain. Each of these sets has different refreshment and query answering cost while some of them may require more disk space than the space which is available for the data warehouse. Also, each of these views can be differently evaluated with respect to the quality of its data (in terms of accuracy, freshness, consistency etc.).

The data warehouse design problem is the selection of the set of materialized views with the maximum overall "benefit" (minimum overall "cost") that fulfill all the predefined requirements, set by the involved stakeholder. In the sequel, we will use an exemplary constraint, demanding that the solution fits into the available space. These views have to answer all the queries posed to the data warehouse without accessing the heterogeneous sources. Obviously, the benefit (or cost) can be quantified with respect to the aforementioned parameters as well as with respect to the total execution time of the design algorithm.

The overall problem of data warehouse design optimization, composed of its objects and quality factors is graphically depicted in Figure 4. We present the basic object types and for each of them we give their relevant quality factors.

We can determine three basic object types for the problem of optimal data warehouse design: the *sources*, which are considered to be relational (or at least provide a relational interface through a wrapper), the *queries* posed from the user over these sources and the *intermediate materialized views* that the algorithms produce, which serve as buffers to speed up the answering of the queries. The design choices of the problem, which are present in the problem definition (although implicit), are the *employed policies for the propagation* and the *cleaning of the information*, as well as the *employed algorithm* itself.

We believe that the basic quality dimensions affecting the problem are related to time and space:

1. *Query Timeliness*: the answering of the queries must be done as quickly as possible.

Quality Dimension	DW Object	Primary Quality Factors	Derived Quality Factors	Design choices
Query timeliness	- DW views - User queries	- Query cost over the sources	- Query cost over the views - Total query cost for the DW	- View definition
System availability	- DW views - Source relations	- View definition - Refreshment window	- Update cost for view - Total update cost	- Refreshment policy
Data quality	- DW views - Source relations	- Source accuracy - Source completeness - Source consistency - Source freshness	- DW view accuracy - DW view completeness - DW view consistency - DW view freshness	- Propagation and cleaning policies
Design consistency	- DW views - Source relations	- Disk space - Space occupied by each view - Size of propagated data from the sources	- DW space	- Model for size prediction
Design efficiency	- DW views - Queries	- Query frequency - Update costs - Cost model	- Design time - Quality of the solution	- Employed algorithm

Table 3: Quality Factors for the Data Warehouse Design Optimization Problem

2. *System Availability*: the downtime of the system due to refreshment reasons must be the smallest possible (the maintenance of the materialized views must be done as quickly as possible or at least within a specified time interval).
3. *Data Quality*: the data delivered to the users must be accurate, complete, up to date and abiding by the internal rules (constraints) of the data warehouse.
4. *Design Consistency*: the constraints over the result of the design process must be respected (e.g. the volume of the produced materialized views must be such, that fits in the disk space provided for the data warehouse).
5. *Design Efficiency*: the choice of the materialized views must be done as quickly as possible, without a severe impact in the quality of the solution.

For each of the related objects of the problem, one can define relevant quality factors that affect the result of the data warehouse design. As far as the *sources* are concerned, one can identify the *size of source relations*, which will be used for estimations in the cost model and the *original cost of the queries over the sources*. The data warehouse views are affected by several quality factors such as the *final number of views in the data warehouse*, the *available disk space for the data warehouse*, the *space occupied by each view* (as well as the *total space occupied by al the views*), the *update cost for each view* (i.e. the time to perform the update of an data warehouse view) as well as the *total update time for the data warehouse*. The queries are characterized, in general, from their *query cost* (i.e., the time needed to perform the query over the data warehouse). The sources, the views and the queries are also characterized by their data quality. The quality factors related to data quality are the *accuracy* (i.e., the validity of the data, with respect to real world values), the *completeness* (i.e., the percentage of available information with respect to expected volume of information), the *consistency* (i.e., the percentage of information obeying the database rules), and *the freshness* (i.e., the age of the data with respect to their transaction time). Finally, the algorithm itself is characterized by the *design time* (i.e., the total time needed for the algorithm to terminate) and the *quality of its solution* (i.e., the fraction of the cost of the obtained solution over the best possible solution which can be obtained). All the aforementioned quality factors are listed in Table 3.

3. EXPLOITATION OF THE METADATA REPOSITORY AND THE QUALITY METAMODEL

In the GQM approach, each goal is defined from a set of questions, in order to help the transition from a very general, high level, user request to a set of specific measurements. Yet, the selection of the right set of questions for a specific goal, or better, for a specific type of goals, remains an open issue. Basili gives some hints [2, 16]: there are questions informative on the current status of an object (or process),

questions objectively quantifying this situation through specific measures and finally questions subjectively judging the current status from the viewpoint of the user.

Naturally, these guidelines are too general, since they are supposed to open a path for the development of specific algorithms/methodologies for the different fields of applications. As a result the suggested guidelines do not really provide a concrete set of steps for the operational usage of the metadata repository. So, in our approach we attack this problem from a methodological point of view: we try to come up with a set of steps in order to be able to exploit the information residing inside the data warehouse metadata repository. To perform this task, we customize the GQM process to fit with the DWQ approach as far as the problems of data warehousing and the given solutions are concerned. More specifically, *we base our approach on the idea that a goal is operationally defined over a set of questions.* Thus, we provide specific "questions" for the full lifecycle of a goal, not only for the identification of a situation, but also for the interrelationships between its crucial components and quality factors. Moreover, we do not restrict our approach to the detection of the anomalies in the quality of a data warehouse: we extend GQM towards the re-action to encountered problems by providing guidelines for the improvement of an undesired situation as well as for the re-evaluation of the usage of a goal in the presence of a continuously evolving environment as a data warehouse. Underlying our methodology, we exploit:

- A metadata repository, which provides all the necessary knowledge to understand quality goals, quality factors and their related data warehouse objects. This repository allows to trace design decisions, and to report on the history of quality goals with their successive evaluations and improvements.

- A computational engine composed of all the deriving procedures of quality factors. The techniques underlying this engine can be simple functions and procedures or more sophisticated reasoning mechanisms. For example, in the case of performance evaluation of a given query, a mathematical function is generally sufficient while in the case of coherence validation of a conceptual schema we need a more sophisticated inference mechanism.

Based on this, the DWQ methodology for quality management is composed of four main *phases*: (i) the *design phase* which elaborates a quality goal by defining its "ingredients" and their interrelationships at the type level; (ii) the *evaluation phase* which deals with the computation of quality factors; (iii) the *analysis and improvement phase* which gives an interpretation to the quality goal evaluation and suggests a set of improving actions; (iv) the *re-evaluation and evolution* phase, which deals with the problem of continuous change both of the data warehouse and the status of the quality goals of the users.

In Figure 5, we graphically present our methodological approach for quality management. This methodology is influenced by the TQM paradigm, which has also been adopted by other approaches such as TDQM [27]. In the sequel we provide a detailed presentation for the different steps / questions of each phase. Before proceeding, we would like to mention that the proposed methodology does not consist of a strict algorithm: one may choose to ignore several steps, according to the specific situation he is tackling.

3.1. The Design Phase

Naturally, when dealing with a quality goal, we assume that there is always a first time when an involved *stakeholder* defines the goal. The *design process* of the goal is the first phase of the interaction between the stakeholder and the repository and should result in the selection of the involved object types and their quality factors.

There are two steps that can take place at the same time: *the identification of the object types which are related to the goal* and the respective low level *quality factors.* The identification of the object types tries to reuse the experience stored in the metadata repository. The repository is powerful enough to model the relationships not only at the instance but at the type level as well.

Take for example, the refreshment process, described in Section 2.3. Several object types of the data warehouse are involved, e.g., "source data stores", "ODS", "materialized views" (Table 2). Each of these template object types can be linked to template quality factors (e.g., "availability window of each data source"). Actually, there are two kinds of template object types that can be reused this way. First, at the metamodel level (Figure 2), we can find relationships between object and quality factor types applicable to any data warehouse. In the DWQ project, we have provided a "list" of such template interrelationships for all the crucial phases of the data warehouse lifecycle [13]. Second, these interrelationships, found at

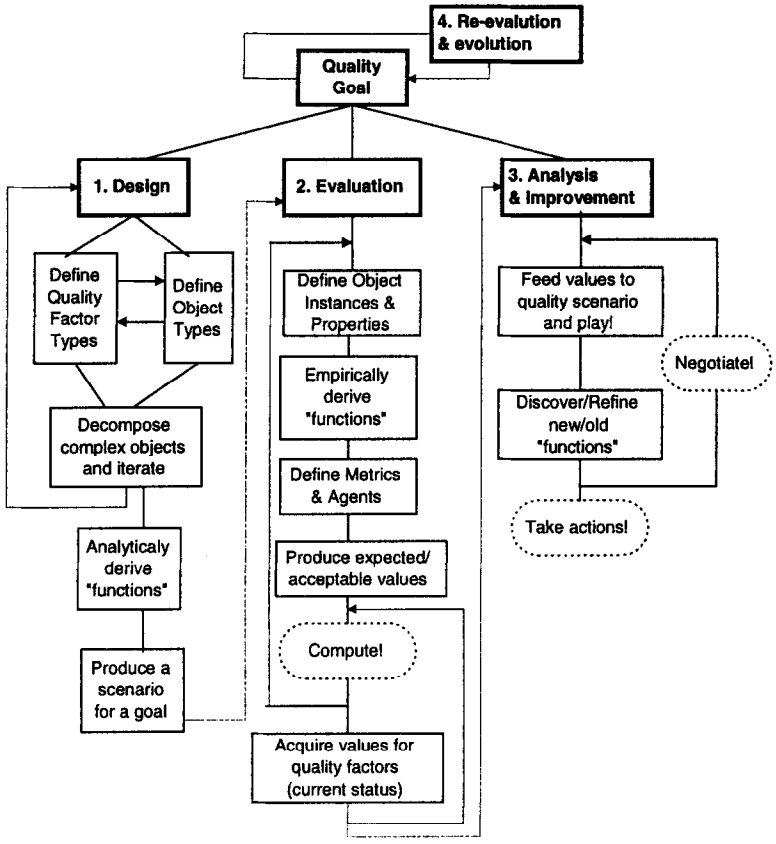

Fig. 5: The Proposed Methodology for Data Warehouse Quality Management

the metamodel level, can be enriched with template patterns at the metadata level (i.e., concerning the architecture of the particular data warehouse that the involved stakeholder considers). This can be the case, when a general pattern is followed for a certain kind of processes, throughout all the data warehouse. We will exemplify this situation in Section 5, where we present a real-world case study.

The identification of the involved object and quality factor types is accompanied by a complementary, but necessary step. Since the identified object types are most probably composite (e.g. a schema is composed from several relations) one has to *decompose them at a satisfactory level of detail.* For example, if the examined type is the refreshment process, one can try to decompose it into more refined objects such as data extraction, data cleaning and transformation, data integration and high level aggregation.

The next step deals with the *identification of the interrelationships between objects and quality factors.* Each object can be viewed as a node in a graph. Every node in the graph has input and output arcs, determining the interdependencies of the data warehouse components with respect to their quality factors. Several design choices are by default encapsulated in the graph (e.g. the simple fact that the data of a materialized view stem from source data). The graph is enriched by the tracking of high-level quality dimensions, expressed by the user. The final output of the application of the methodology will be a set of specific quality factors, measuring these quality dimensions.

The goal of this process is, not only to set up a list of the "ingredients" of the problem, but also, to come up with a list of "functions", determining the outcome of the quality of an object, in terms both of its own characteristics and of the quality of other objects affecting it. We call the outcome of the process, the *scenario* of the quality goal.

More specifically, to produce the list of functions, the involved stakeholder has to try to define the interrelationships between the determined object types, by inspecting the peculiarities of the problem. Take for example the problem of determining the timeliness of a materialized view. The stakeholder should use a standard statistical methodology, like a Pareto diagram [3], or a specific algorithm (acting like a function) to take into account the availability of the sources, the frequency of updates and queries and the capacity of the propagators.

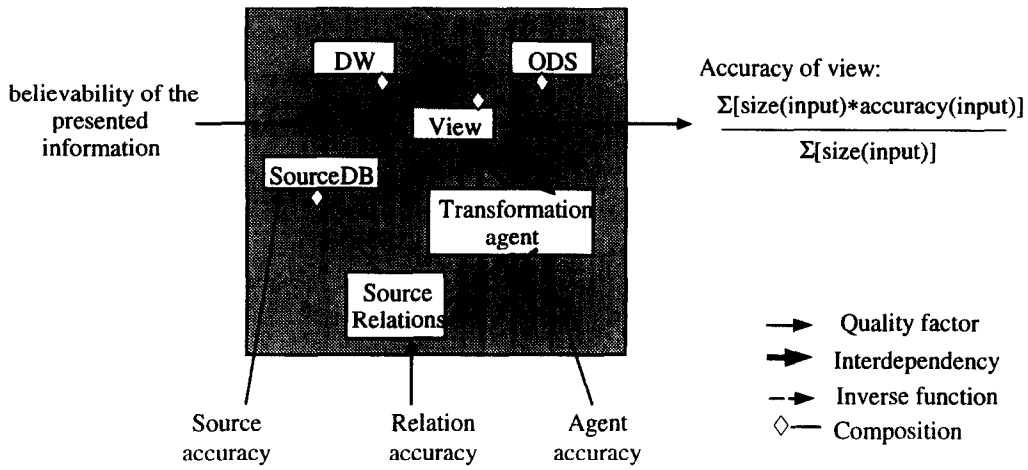

Fig. 6: The Scenario of a Quality Goal

We do not advocate that these functions can always be derived or discovered in an analytic form. Before proceeding, we feel that it is important to stress that the presence of an analytical function, or a concrete algorithm, can be the case in several occasions. We will demonstrate this with an elementary example in the sequel.

Still, even if this is not the case, we can complement the lack of an analytical function to describe the relationship of two quality factors, in various ways. First, it is quite common -as we have observed in our practical experience- that the involved stakeholders have a detailed empirical knowledge of the domain in which they are involved. This kind of knowledge can be captured both in the design and the evaluation stage (as we shall also see in the sequel). Moreover, it is important to note that even the existence of an interdependency link can be used as a boolean function to denote dependency, between the involved objects. We will demonstrate how this kind of interrelationship works in Section 5, where we present a real-world case study.

Example 1 In the example of Figure 6, we try to quantify a quality dimension: the believability of the information delivered to the final user. To achieve this goal, we decide that we have to measure a specific quality factor: the accuracy of the data in the views used by the final users. The scenario is composed from all the components participating in the refreshment of a view: the source database (which in terms is decomposed to a set of source relations), the transformation agents converting the data to the desired format and the data warehouse / ODS views, each one possibly defined on top of another view. We also provide an analytical function for the accuracy of a view, calculating it from the size and the accuracy of the input data. □

A fascinating feature of a scenario is the tracking of the *inverse relationships* between the quality factors. In other words, by describing the interdependencies of the quality factors, not only do we get a clear view of the way the overall quality of our final "product" is influenced, but also we get a first insight of how to remedy an undesired situation. For example, in Figure 6, we can improve the believability of our information by increasing its accuracy, something which, in terms, can be achieved through the improvement of the accuracy of the transformation agents and/or the source relations. In the case of redundant information, one can also increase the volume of the utilized data from a source with higher accuracy. In any case, to generalize this observation, the inverse functions can be either analytical relationships or the inverse interdependency path on the scenario.

3.2. The Evaluation Phase

After the design process, the following step is the *evaluation of the current status*. The purpose of the evaluation phase is to construct a detailed *map* based on the constructed scenario, which describes accurately all the interplaying components and factors at the instance level. This can also be the first step, when a goal has already been defined in the past and a scenario has been developed and can be currently reused.

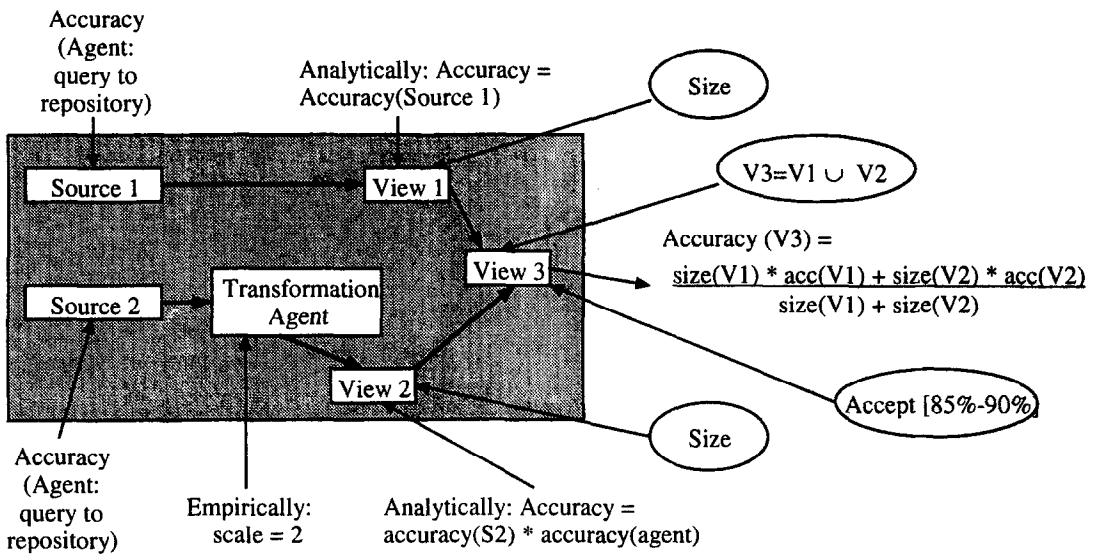

Fig. 7: The Map of a Quality Goal

First, we must *determine the specific object instances* of the specific evaluation through a query to the metadata repository. In the example of Figure 6, one can identify two source relations (S_1, S_2), pumping data to two views in the ODS (V_1, V_2), through a respective transformation agent and a final view (V_3), of which we have to quantify the accuracy (Figure 7).

Next, one must take into account several *design choices*, i.e., the properties of the interplaying objects which influence the quality of the outcome. In our example, one can take into account the size of the propagated data, the time windows of the sources, the regularity of the refreshment, etc. For reasons of simplicity of the presentation and since we deal only with the accuracy factor, we retain only the size of the propagated data and the view definitions.

Apart from the component refinement, we can also refine the interrelationships between the quality factors. The refinement can be performed either through the use of analytical formulae or direct instantiations in the scenario, based on the empirical knowledge of a specific situation. Empirical knowledge can be obtained from simple observation, user expertise, or through the use of well-tested techniques such as statistical process control (SPC), concurrent engineering, etc. [3]. In our example, simple sampling could show that in the past, the transformation agent increased the accuracy of the data by a scale of 2.

Then, for each quality factor one should also determine the *metrics* and *measuring agents*. If no measuring agent(s) has ever been defined, one must determine the computation procedure for the actual values of the quality factors. Also, the parameters of the measuring procedures should be set accordingly.

The final step is the addition of *acceptable/expected values for each quality factor*, wherever necessary. This is a crucial step for the evaluation of the current status later on. The accepted range of values will be the basic criterion for the objective judgment of a subjective quality goal. The outcome of this step should provide the stakeholder with a well-defined *map* of the problem (see also Figure 7).

With respect to the scenario of Figure 6, the map is enriched with (a) agents for the computation of primary quality factors (e.g. the queries at the metadata repository), (b) formulae for the computation of the derived quality factors, (c) properties of the components such as the view definition, or the size of the propagated data and (d) acceptable ranges of values (e.g. accuracy of view *V3*).

After that, the only thing left is the acquisition/calculation of the specific values of the selected quality factors, through the necessary computation. In Figure 8, a certain *instance of the quality map* is depicted.

The acquisition of these values is performed through the use of the already defined measuring agents. In fact, we anticipate that if the values are regularly (i.e. not on-demand) computed and stored in the metadata repository, then their acquisition can be done through a simple query to the metadata repository.

Here we must clarify again, that several steps can be omitted. In fact, if we consider that the metadata repository is regularly refreshed through an external agent, then some of the intermediate steps of this process can be avoided.

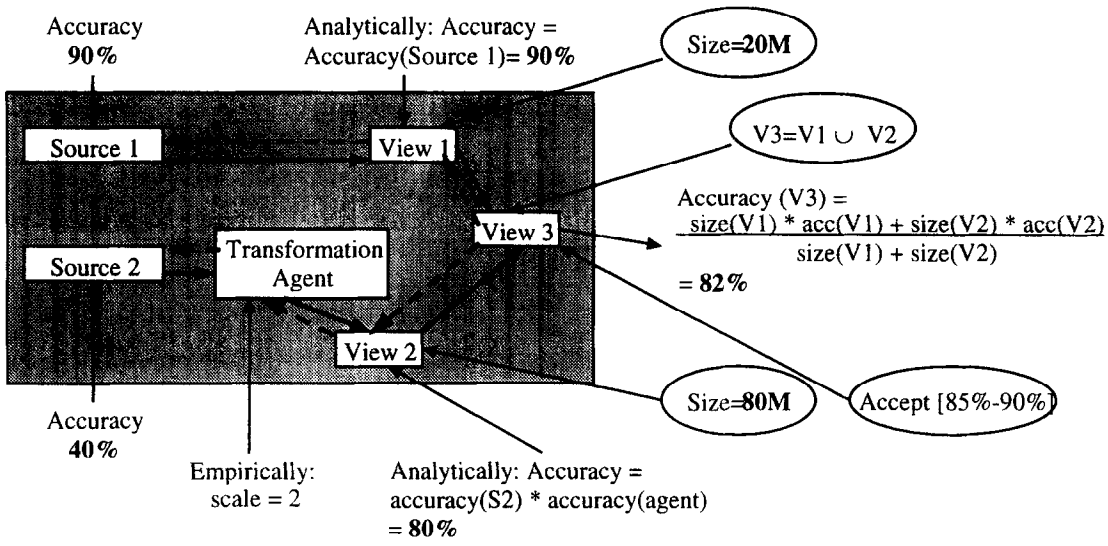

Fig. 8: Instance of a Quality Map

3.3. The Analysis and Improvement Phase

At the end of the second phase, the map of the problem is fully instantiated with actual values (e.g., like in Figure 8). Yet, if the situation is not satisfactory, the stakeholder may choose to react against it. Although this is a process with different characteristics each time, we can still draw some basic guidelines for the steps that can be taken. Consider for example the case in Figure 8, where the computed accuracy for view *V3* is not within the accepted range. Obviously there must be some reaction against this undesired situation.

One of the main advantages of our approach is that if we have an understanding of the mechanism that produces the problem, we can attack the problem directly through the use of the inverse quality functions, which have been derived during the design phase or detected during the evaluation phase. Again, by 'inverse functions' we mean both the possible analytical functions and the inverse interrelationships in the map of the problem.

The inverse functions in our example suggest that an increase of 10% for the accuracy of view *V3* calls for one of the following actions:

a) Use the analytical formulae directly: increase of 10% to the accuracy of views 1 and 2 (directly through the formula), which in terms implies:
- increase of the accuracy of source 1 by 10%;
- increase of the accuracy of source 2 by 5% or the accuracy of the agent by 10% or a combination of the two.

b) Customize the reaction to the specific characteristics of the situation: Through the use of the specific measurements one could also try to derive a plan taking into account the sizes of the input views. For example, elementary calculations prove that it suffices to increase the accuracy of source 2 to 45%, for the quality of the view 3 to be in the accepted range.

We call the final result of the negotiation process, the *final instance of the quality map*. In Figure 9, the final map instance of the motivating example is depicted, according to the second proposed solution (b).

Nevertheless, there is always the possibility that this kind of approach is not directly feasible. If our understanding of the problem is not full, then steps must be taken so that we deepen our knowledge. Moreover, it is possible that the derived solution is not feasible -or is too costly to achieve.

In the first case, we must go all the way back to the design process and try to *refine the steps of the function discovery*. In the second case, we must try to use the inverse functions in order to *determine which are the feasible limits of values* that we can negotiate. The negotiation process is a painful task, since one has to deal with contradictory goals and priorities. Yet, several specific techniques exist which

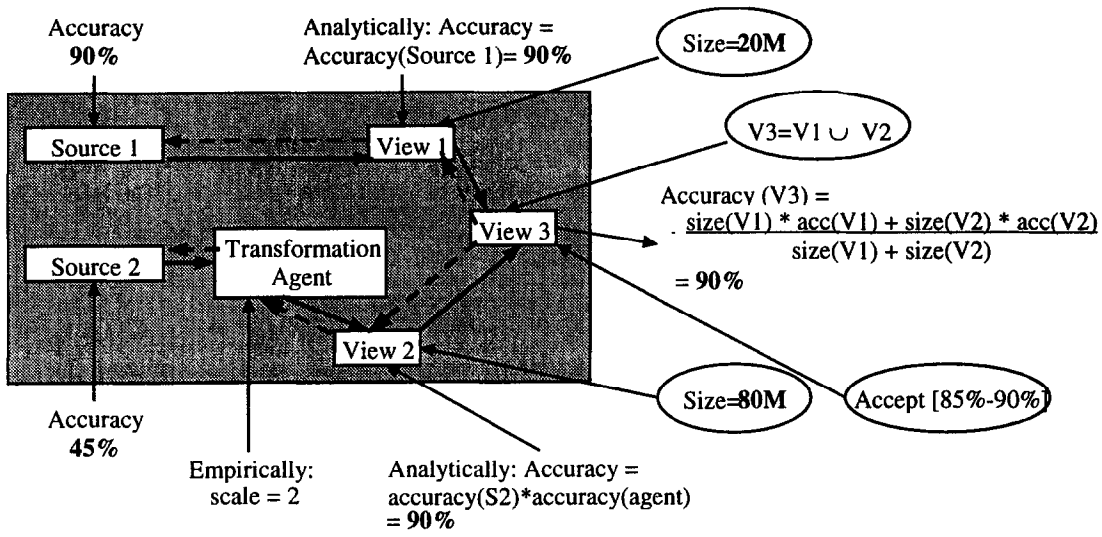

Fig. 9: Final Instance of a Quality Map

can be applied to the negotiation problem. In Section 6, we present as example the QFD and the Statistical Process Control methodologies. Other examples are the experimental design, the Taguchi quality engineering etc. [3].

4. DATA WAREHOUSE EVOLUTION

Complementing the three previous steps of the DWQ methodology is the last step: *Re-evaluation and evolution*. A data warehouse is a very complex system whose components evolve frequently independently of each other. New materialized views can be created and old ones can be updated. Some sources may stop being used, while others are added. The enterprise model can evolve with new enterprise objectives and strategies. The technical environment constantly changes due to product evolution and updates. Design choices at the implementation level can also evolve in order to achieve user requirements and administration requirements.

As a result of evolution and errors, our 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 that is not appropriate), outdated information of the repository with respect to the data warehouse, etc.

In this section, we do not deal with the problem of schema evolution or the redefinition of data warehouse views, but rather we focus on *how the evolution of quality factors and quality goals fits into our methodology*. In our view, *the quality of the data warehouse is a view of the metadata and data of the warehouse*. For example, the quality of the data warehouse depends on the quality of the sources, the quality of the extraction process and the quality of the data warehouse components itself. One can think of the quality factors as materialized views over the metadata and data of the warehouse; thus the evolution of the data warehouse can be seen as a view maintenance problem on the aggregated quality views.

The consequence of this observation is that, exactly as in the case of view maintenance, the relevance of data warehouse evolution and quality factors is two-fold. On the one hand, changes in the architecture of a data warehouse result in the evolution of its quality factors. On the other hand, a change in the user goals can impose a change in the architecture of the data warehouse, in any perspective: conceptual, logical or physical. In the former case, which we will describe in Subsection 4.1, we are dealing with a situation similar to the view refreshment problem, whereas in the latter case, which we present in Section 4.2, we have a situation similar to the view evolution problem. Both these cases are efficiently supported by the results of the application of our methodology. Still, in Section 4.3 we present how the use of the data warehouse metadata repository can provide further support to both cases, in a uniform fashion.

4.1. Evolution of Quality Factors

Quality factors can evolve because of changes in the architecture of the data warehouses. The data stores can produce changes due to reasons of schema evolution in logical and conceptual perspective, changes to the physical properties of the source (e.g., location, performance etc.), insertions or deletions of data stores, or specific 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, or debugged. The propagation agents of all types (e.g., loaders, refreshers, wrappers, mediators and source integrators) can obtain new schedules, new algorithms, rules, or physical properties. Moreover, the business rules of an organization are never the same, due to real world changes.

In all these cases, the evolution of the quality factors can take many forms: new factors can be needed for the precise tracking of the new situation, while existing ones maybe useless. The measurement techniques may need to change too and the values of the quality factors have to be recomputed. For example, if a source is changed – either its data or its properties that are captured by the system metadata – the quality of the source must be recomputed. All objects and their quality factors which depend on this source must be adapted to the new situation. As another example, in the case a new source is integrated, we just have to compute the quality of this source and recompute the data warehouse and data mart quality using the information of the process quality, which describes how the data is transformed and what improvement or debasement to the data quality has been made.

Our methodology is powerful enough to support this kind of evolution efficiently, both at the metamodel and the metadata levels. The metamodel level captures interdependencies of generic types (e.g., a view depends on its sources, or the data freshness inside the data warehouse depends on the extraction frequency of the sources). The quality scenarios trap this meta-information explicitly, through the respective dependency functions. Note that, due to their nature, it is rather straightforward to hardcode any generic technical results from database research and practice into the quality scenarios. For example, the operational cost for loading and maintaining the data warehouse, as a result of the design process [22], depends on the query and update frequencies: this meta-information can be incorporated into the respective quality scenarios of the repository. Thus, any change in the instances of any of the involved types in a quality scenario signifies the need to redesign (or simply re-evaluate) the respective instances of the quality factor types appearing in this scenario.

At the same time, at the metadata level, the peculiarities of the interdependencies in the data warehouse that the interested stakeholder examines, are captured in the quality maps of the metadata repository. Thus, any changes in the particular object instances, appearing in the quality maps calls for the re-evaluation – or even redesign – of the affected quality factors.

4.2. Evolution of Quality Goals

Similarly to the view evolution problem, where a change in the view definition signifies a new way to materialize it, the user requirements continuously change, possibly resulting in a new data warehouse architecture. New requirements arise, while old ones may become obsolete, new users can be added, priorities and expected/acceptable values change through the time, etc. In the evolving context of a data warehouse, the re-evaluation of a goal and of the strategy to achieve it is a strict contingency. There are 3 main reasons for this:

(a) *evolution reasons*: there are natural changes happening in such a complex environment;
(b) *failure in the achievement of the desired quality*, and
(c) *meta-quality:* we can never be sure for the quality of our measuring processes.

All these changes, or observations, may lead to the evolution of the data warehouse architecture, so that the new quality goals of the users are met. Consequently, in addition to the maintenance process of the quality, the inverse of the computation functions for the quality factors can be used, to find the data warehouse object that has to be improved to reach a certain quality goal. This process can be compared with the view update process in databases systems, where updates to views (here: derived quality factors, expressed as "views") are translated to updates in base data (here: primary quality factors). As an example, we can mention the very common case where the users demand more up-to-date data: this evolving user quality goal affects directly the data warehouse architecture, at least at its physical level, since new definitions of the data warehouse materialized views (in the logical perspective) are employed,

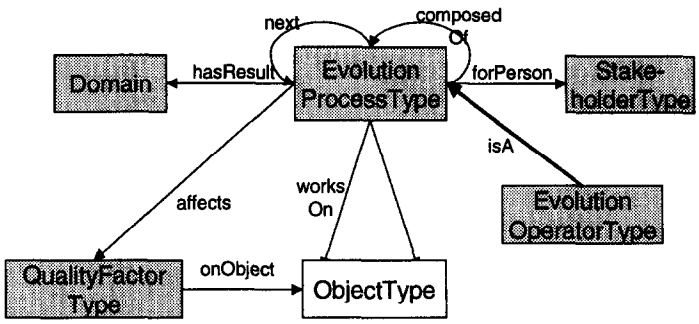

Fig. 10: A Meta Model for Data Warehouse Evolution

along with the appropriate physical changes in the clustering and indexing of the data warehouse tablespaces. Alternatively, or when these techniques are proved to be inadequate, new refreshment techniques and tools have to be applied, in order to achieve the requested timeliness.

Our methodology can support the evolution of the user quality goals through the use of its intermediate results. Whilst the direct dependency functions support the architecture evolution, the inverse dependency functions can enable the evolution of quality goals. Take for instance the working example of Section 3: it was the existence of the inverse functions that led to the solution of the problem. In a similar manner, the inverse functions indicate which quality factors are affected or should interplay in an evolved user requirement. Again, as in the case of architecture evolution, this can happen both at the metamodel (here: scenario) and the metadata (here: map) level.

4.3. Repository Support for Data Warehouse Evolution

Apart from the facilities provided by the methodologically derived scenarios and maps, we can extend the support provided by the repository for the task of data warehouse evolution. The repository, thus, gains added value since, *ex ante* the data warehouse stakeholders can use it for design purposes (e.g., to perform what if analysis through the application of the methodology) and *ex post*, people can relate the data warehouse objects to decisions, tools and the facts which have happened in the real world [9].

A way to control data warehouse evolution is to provide complementary metadata which track 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 that affect them. The idea behind this is to enrich the metadata repository in order to ease the impact analysis of each evolution operator and its consequences on the quality factor measures. In [18], we have proposed a metamodel to capture the semantics of evolution operations and to their effect on data warehouse quality (see Figure 10).

A data warehouse *evolution process* is *composed of* several sub-processes, which may be further decomposed. These sub-processes are executed in a specific order, which is described by the *next* relationship between evolution processes. An evolution process *works on* an object type and its *result* is some value of a *domain*. The process is linked to a *stakeholder* that controls or has initiated the process. 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.

While this description provides the general framework under which the evolution operators function, we also provide interrelationships between specific data warehouse objects and the impact of their evolution on quality. In [6] a taxonomy for schema evolution operators in object-oriented databases is given. We have adapted this taxonomy to relational databases, which constitute the most popular platform used in data warehouses. Table 4 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 (a) 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, and (b) on the physical objects where the data of the relation is stored or

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 (VM) 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 - VM 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 - VM 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 - VM Agent

Table 4: Evolution Operators for Base Relations and Views in Data Warehouses and Their Effect on Data Warehouse Quality

where the view is materialized, i.e. the data stores. In addition, if there exists a view, which is based on the evolved relation or view, 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 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 meta model, we suppose that an analyst has detected that the views he is using are often changed, and that he wants to get notified about future changes. We can establish a view on the metadata repository for the analyst that 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 that 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 that notifies data warehouse administrators whether base relations have changed might be also useful for them. Our repository system ConceptBase is able to maintain views on the metadata and supports the notification of external client applications if a view, which they are interested in, has changed [19].

5. CASE STUDY

To demonstrate the feasibility of our approach, we will present its partial application to a specific case study. The case study involves an organization of the Greek public sector, which is not revealed for reasons of confidentiality. In the rest of this section we will briefly present the architecture of the data warehouse which was built, the problems that occurred during its testing and the way we applied our methodology to resolve the respective situations.

5.1. The System Architecture

The role of this organization is to support the general policy of Greek State towards issues of health. In the past, various data about the yearly activities of all the Greek hospitals were collected from all the hospitals once a year and an annual report was produced from a legacy system. The data warehouse that we built aimed to replace and extend the old system. In the sequel, we will present a small subset of the data warehouse, concerning a specific problem, which we resolved with our methodology.

The system relies on operational data coming from COBOL files (see Figure 11). We focus on two COBOL files, the first presenting the annual information by department and the second by class of beds for each hospital. Each COBOL file yields a specific attribute for each type of department (or class of beds respectively), along with various other information. Each year, the COBOL files are transferred from the operational system to the data warehouse and stored in "buffer" tables of the data warehouse, acting as mirrors of the files, inside the DBMS. Then, the tuples of the buffer tables are used by computation procedures to further populate normalized tables in the data warehouse. Several materialized views are then populated with aggregate information and used by client tools for querying. In this study, we will present a case where the materialized view could be populated from any of two different data flows. The wrong choice of data flow led to incorrect data.

5.2. Problems and Quality Goals of the Project

Among the requirements that were originally set for the system, we distinguish two that will concern us in this paper:

- *Data Completeness.* All the information for all the hospitals should be in place after the annual refreshment process.
- *Data Consistency.* Any information presented to the users should be identical to the one that would be produced by the legacy system (for the parts that provided the same functionality).

The testing of the deployed application showed that the first goal was achieved. Nevertheless, the application seemed to fail the second goal. In particular, we observed that there were hospitals where the total number of beds did not match the number calculated for the legacy system. Thus, the following quality goal was set: *"Firm up the consistency for the data stored inside the warehouse"*.

Purpose: firm up	**Issue:** consistency
Object: data stored inside the warehouse	**Viewpoint:** data warehouse designer

In the sequel, we will show the results and the actions undertook in the context of each step of the methodology.

5.3. The Design Phase: Scenario of the Quality Goal

To find out what had gone wrong we first designed the scenario of the loading process (we do not present the querying process here, for reasons of simplicity).

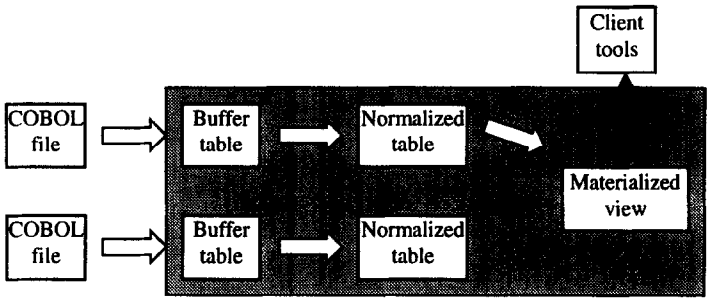

Fig. 11: The Data Flow in the Architecture of the Case Study

Step 1. *Decomposition of complex objects and identification of object types.* The only object participating in the definition of the quality goal is the data residing in the warehouse. A general statement like this had to be decomposed to more detailed object types. The metamodel layer can be exploited in this stage, since it provides all the information about the object types and processes that constitute a composite object. In our case study, the data warehouse, which we previously described, primarily consisted of data stores and loading processes. The interplay of these two high-level object types abided by a general pattern that was followed throughout the whole data warehouse. This way, the scenario that we built could be reused later, for testing other parts of the data warehouse architecture, too.

We identified the following important object types for the map of the quality goal: the *source COBOL file*, the *"buffer" mirror table*, the *normalized table* and the *materialized view* in the data warehouse. Apart from the static objects, the *loading*, *cleaning* and *computing applications* of the warehouse affected the overall quality of the stored data.

Step 2. *Identification of quality factor types.* In general, we can rely on two beacons in order to define the relevant quality factor types for a quality scenario: the *quality dimensions* participating in the definition of the quality goal and the *quality factors related to the interplaying objects* of the scenario. Obviously, quality factor and object types, as well as quality dimensions are interrelated: thus, the definition of quality factor and object types for a particular scenario is an iterative process.

In our case, we identified the relevant quality factors by inspecting (a) the quality dimension of the quality goal (namely the *consistency* of the stored data) and (b) the quality factors of the interplaying *data stores* and *applications*, which are listed in the previous paragraph. The outcome of this process comprises quality factors such as the *correctness* of the developed software applications and the *completeness* and *consistency* of the involved data stores.

Step 3. *Analytical derivation of "functions".* The goal of this step is to analytically define the computational procedure for the derived quality factors and to identify any kind of interrelationships over the quality scenario. This means that the involved stakeholder must (a) detect any dependency between the object and quality factor types and (b) employ known algorithms, or self-defined heuristics for the definition of the computation procedure of the derived quality factors.

To identify the interrelationship of the quality factors of the problem we were based on the following observations:

- Given a data flow abiding by the general pattern of the particular warehouse of the case study, the consistency of the data in a certain stage depends on the consistency of the data in a previous stage.
- The consistency of a data store also depends on its completeness, since missing information affects the result of any process operating over this data store (e.g., aggregation or cleaning).
- The correctness of the employed processes also affects the consistency and the completeness of a data store, since erroneously rejected or propagated data affect the amount of information residing in the data store.

These observations led to a general picture as suggested by Table 5.

Quality Dimension	DW objects	Primary Quality Factors	Derived Quality Factors	Design Choices
Consistency	- COBOL Source file - Buffer table - Normalized table - Materialized View - Loading process - Cleaning process - Computation process	- Consistency of a data store - Completeness of a data store - Correctness of an application	- Consistency of a data store - Completeness of a data store	- Data flow - Chosen source files

Table 5: Quality Factors for the Data Warehouse Design Optimization Problem

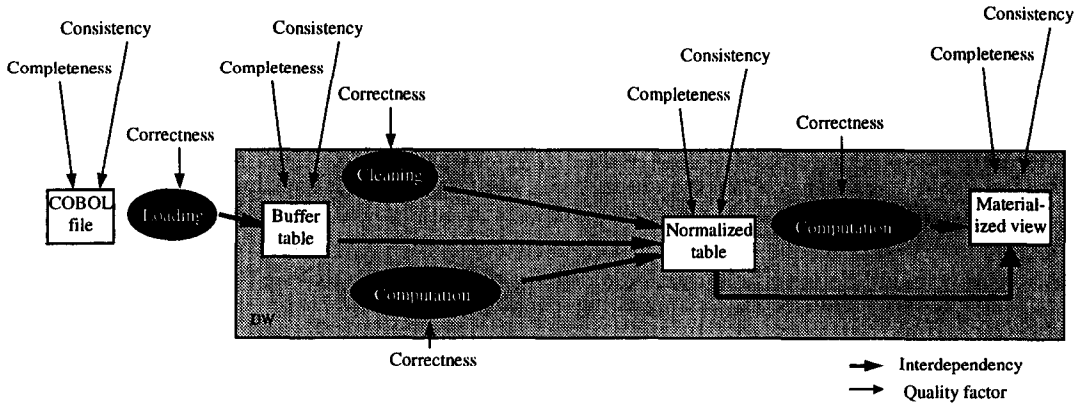

Fig. 12: The Scenario of the Quality Goal of the Case Study

In our case study, we could not possibly exploit the results of previous research or industrial efforts, since we were not aware of any related work, performed in the context of data warehouse quality management, by that time. Thus, in order to compute the consistency and completeness of the data stores of the quality scenario, we had to resort to some rather naïve analytical functions which seemed to be reasonable. In our scenario, the completeness of a data store equals the completeness of its previous data store in the data flow by the correctness of the intermediate programs. The same holds for consistency, too. For example,

*Norm_Table.completeness=Computation.correctness*Cleaning.correctness*Buffer_Table.completeness (1)*
*Norm_Table.consistency = Computation.correctness * Cleaning.correctness * Buffer_Table.consistency (2)*

The resulting scenario after the three aforementioned steps of the design phase is depicted in Figure 12. We should note that in our case study we did not have the possibility of using a metadata repository. Thus, we did not have the possibility of storing this template and reusing it electronically; still the meaning of the approach was kept, even on hard-copy. The same applies for the rest of the steps that will be presented in the sequel.

5.4. The Evaluation Phase: Map of the Quality Goal

The construction of the map of the quality goal was rather easy, due to the simplicity of the processes involved.

Step 1. *Identification of object instances.* To perform this particular step, one would normally pose a simple query to the metadata repository, in order to instantiate the template object and quality factor types. In our case, the absence of a repository was compensated by the simplicity of the problem that we considered. The particular instantiation of the quality scenario is depicted in Figure 13. To give an example, we instantiated the "template" types, such as *Buffer_table*, *Materialized View*, *Computation Process*, to specific architecture objects such as tables *Buff_class*, *Hospital_info* and processes *Computation_c1* and *Computation_c2*.

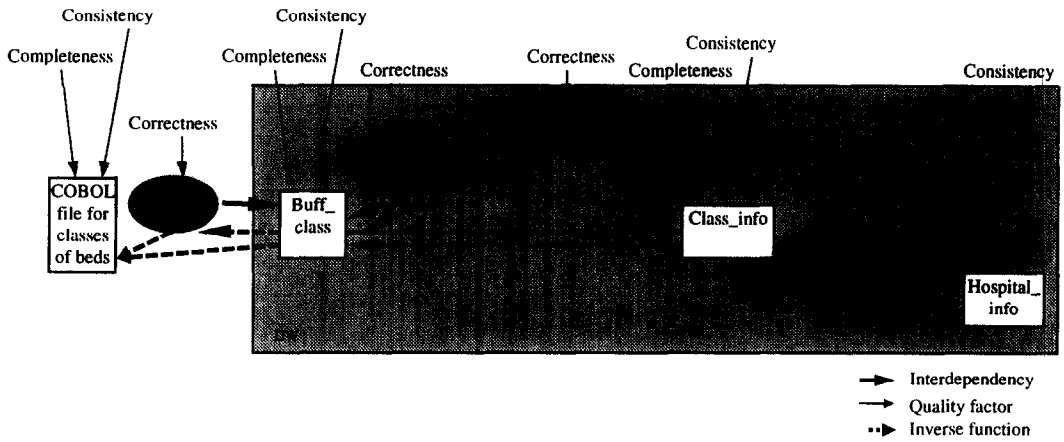

Fig. 13: The Map of the Case Study

Step 2. *Definition of metrics and agents.* Since we had no personal knowledge of the particularities of the data and the warehouse was under construction (i.e., no empirical knowledge had been acquired from its run-time usage), we were not able to enrich the derived scenario with empirical functions. Thus, we proceeded to define metrics and agents to test the assigned quality factors. The task was dealt with standard techniques. The correctness of the deployed applications would be examined by white box testing. The completeness and consistency of the data stores would be tested using SQL queries with respect to the previous data store in the data flow.

Step 3. *Definition of accepted ranges for the quality factors.* In our case study, the definition of the quality goal and the simplicity of the data flow posed clear restrictions for the quality of the stored data: only 100% completeness and consistency would be allowed. The analytical formulae, which were produced during the design phase, imposed a similar constraint for 100% correctness of the deployed applications.

Step 4. *Value acquisition for the quality factors.* This is the final step in the construction of the quality map. Normally, the acquisition of the values of the quality factors is performed either through a query to the metadata repository (in the case where the repository is periodically kept up-to-date with respect to the real world values), or the execution of the defined agents and metrics. Obviously, we followed the latter approach in our case. Initially, we performed the computation of the hospital beds from the tables involving the "class of beds". We followed the path of the "inverse functions" to evaluate the quality factors of the map of the problem: we started the tests from the application performing the load / update of the materialized view, then we tested the consistency of the "normalized" tables, and so on. We were surprised to see that the consistency of the data, with respect to the previous data stores in the data flow, was 100%, all the way back to the buffer table.

5.5. The Analysis and Improvement Phase: Final Map of the Problem

There was not too much of a negotiation during the analysis and improvement phase. Clearly, the path of inverse functions pointed towards the source COBOL file as the reason for data deficiencies. The mystery was solved when the source administrators verified our suspicion that the COBOL file for the "class of beds" was inconsistent. Moreover, they empirically knew that the COBOL file involving departments was the right one to choose. Of course, this file was also the one used in the production of the reports of the legacy system. Our testing (following again the constructed map) verified their – rather late to arrive – story. Our scripts were changed accordingly, so that the correct information was delivered to the users.

The final map of the problem is depicted in Figure 14, where one can notice that the computation of the materialized view *Hospital_Info* is performed by the new computation process *Computation_d2*.

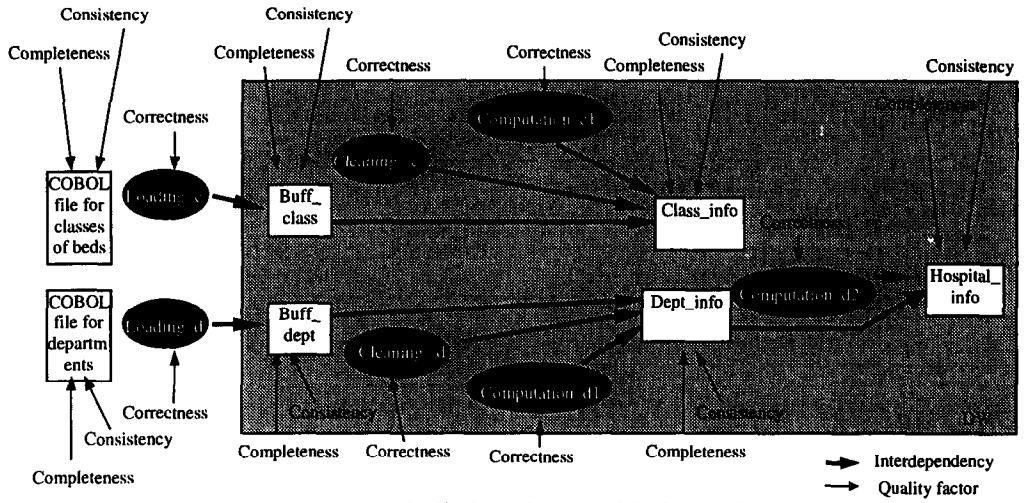

Fig. 14: The Final Map Instance of the Case Study

6. RELATED WORK

There has been much research on the definition and measurement of data quality dimensions [20, 26, 29, 30]. A very good review is found in [28]. The *GQM* methodology is best presented in [16, 2].

The *TDQM* methodology [27] follows the Total Quality Management approach, adapted for the evaluation of data quality in an information system (by assuming that each piece of produced information can be considered a product). The TDQM methodology also follows the TQM cycle: *Definition, Measurement, Analysis* and *Improvement.* The Definition part identifies the important quality dimensions and the corresponding requirements. The Measurements step produces the quality metrics. The Analysis step identifies the roots of any quality problems and their interplay, while the Improvement step provides techniques for improving the quality of information.

Negotiation techniques enable the negotiation over the desired quality of a system. *Statistical Process Control* (SPC) is one of the best tools for monitoring and improving product and service quality [3]. SPC comprises of several techniques, such as Pareto diagrams (used to identify the most important factors of a process), process flow diagrams, cause and effect (or Ishikawa) diagrams, check sheets, histograms and control charts.

Quality Function Deployment, (QFD) [7, 3] is a team-based management technique, used to map customer requirements to specific technical solutions. This philosophy is based on the idea that the customer expectations should drive the development process of a product. The basic tool used in QFD is the so-called "House of Quality", mapping user expectations to technical solutions, taking into account priorities and conflicts.

However, while the two aforementioned techniques certainly have a useful role in rough quality planning and cross-criteria decision making, using any of them alone would throw away the richness of work created by research in measuring, predicting, or optimizing individual data warehouse quality factors. In other words, these techniques are mostly based on human expert participation and statistical models for ad-hoc problem resolving. Our proposal, on the other hand, suggests a treatment of the quality problems at two levels, namely the type and instance level, increasing thus the reusability of the solutions. Moreover, the exploitation of quality is done through the use of a repository, enabling in this way the potential measurement of the involved quality factors through the use of well-established automated techniques and algorithms. We mention two prominent examples to support this claim: (a) the solution to the data warehouse design problem can be based on the use of concrete quality factors like the query and update frequency or the overall data warehouse operational cost [14, 22, 23, 24] and (b) the tuning of the refreshment process based on the quality factors of Section 2.3 [3, 21].

DWQ methodology	Nature	Orientation	GQM
Goal and dimensions	subjective	User	Goal for Issue
Quality Scenario & Map	hybrid	Process	Question
Quality factors	objective	System components	Metric

Table 6: Different Levels of Abstraction for the Management of Quality in a Data Warehouse

7. CONCLUSIONS

In this paper, we deal with the problem of quality-oriented design, usage and evolution of data warehouses. Following the approach of previous papers [11, 13], we store semantically rich meta-information of a data warehouse in a metadata repository concerning the conceptual, logical and physical perspective of the data warehouse. In addition, the information on the quality of the stored objects is recorded in this repository, too.

Our approach extends GQM, based on the idea that a goal is operationally defined over a set of questions. Thus, we provide specific "questions" for the full lifecycle of a goal: this way the data warehouse metadata repository is not simply defined statically, but it can be actually exploited in a systematic manner. These questions are expressed as a set of steps aiming, on the one hand, to map a high-level subjective quality goal into the measurement of a set of interrelated quality factors, and, on the other hand, to propose improvement actions which may help in achieving the target quality goal. These steps involve the *design* of the quality goal, the *evaluation* of the current status, the *analysis* and *improvement* of this situation, and finally, the *re-evaluation* of the achieved plan. Specific products stem out of each case: a quality *scenario* is the outcome of the design phase, capturing the problem at the type level. This reusable component is instantiated in the second step resulting in the specific *map* of the problem. The third step modifies this map, so that the user receives an acceptable value for his quality goal.

The benefit from the use of the methodology is not only the obtained solution to a specific problem. Maybe of greater importance is the fact that the involved stakeholder gets a more clear view of the data warehouse interdependencies. This is achieved through the systematic application of the methodological steps, which convert a subjective problem, expressed in a high-level vocabulary, to specific measurable factors that affect the solution to the problem. In Table 6 we can clearly depict this fact.

The *subjective, user-oriented* GQM *Goal*, shown in the first row of Table 6, is captured by the proposed *Quality Goal* of our methodology. The *objective* solution to the problem, obtained with respect to data warehouse architecture *components* is achieved through the application of specific *metrics* (in GQM vocabulary), expressed as *quality factors* in our approach, as shown in the last row of Table 6. The mediator between the problem and the solution is the proposed methodology, expressed as a *process*, which produces specific *quality scenarios and maps* (instead of more abstract GQM *questions*).

The application of our GQM-like methodology also helps us to design and maintain the knowledge about the data warehouse evolution efficiently. We make extensive use of our metadata repository, so that the information is obtained in a controlled, efficient fashion. We have elaborated on our quality metamodel, in order to track the basic primitives of the interrelationships between data warehouse components and quality factors. Our GQM extension gives us the advantage of exploiting the interrelationships of components and tracks the full lifecycle of a stakeholder's requirement.

We have verified our methodology in a set of case studies. One of these cases has also been presented in this paper as an example of the partial application of the methodology. We believe that the full application of the methodology in a wider extent in the future will provide the academic community with the insight for further tuning. In any case, we sincerely hope that people in the data warehouse research and industry will find our results fruitful, useful and helpful.

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 Jeusfeld and Maurizio Lenzerini. Many thanks are also due to the anonymous reviewers of a previous version of this paper for useful comments.

REFERENCES

[1] P.A. Bernstein, T. Bergstraesser, J. Carlson, S. Pal, P. Sanders, and D. Shutt. Microsoft repository version 2 and the open information model. *Information Systems*, **24**(2):78-93 (1999).

[2] V.R. Basili, G.Caldiera, and H.D. Rombach. The goal question metric approach. *Encyclopedia of Software Engineering - 2 Volume Set*, pp 528-532, John Wiley & Sons, Inc., also available at http://www.cs.umd.edu/users/basili/papers.html (1994).

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

[4] M. Bouzeghoub, F. Fabret, and M. Matulovic-Broqué. Modeling the data warehouse refreshment process as a workflow application. In *Proc. of the International Workshop on Design and Management of Data Warehouses (DMDW'99)*, Heidelberg, Germany, also available at http://sunsite.informatik.rwth-aachen.de/DMDW99/ (1999).

[5] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Source integration in data warehousing. In *Proc. of the 9th Int. Workshop on Database and Expert Systems Applications (DEXA'98)*, Vienna, Austria, pp. 192-197, IEEE Computer Society Press (1998).

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

[7] E.B. Dean. *Quality Functional Deployment from the Perspective of Competitive Advantage*. Available at http://mijuno.larc.nasa.gov/dfc/qfd.html (1997).

[8] ISO. ISO 9000 International Standards for Quality Management, Geneva: International Organization for Standardization (1992).

[9] M. Jarke, M.A. Jeusfeld, and T. Rose. A software process data model for knowledge engineering in information systems. *Information Systems*, **15**(1):85-116 (1990).

[10] M. Jarke and Y. Vassiliou. Foundations of data warehouse quality – a review of the DWQ project. In *Proc. 2nd Intl. Conference Information Quality (IQ-97)*, Cambridge, Mass., Available at http://www.dbnet.ece.ntua.gr/~dwq/ (1997).

[11] M.A. Jeusfeld, C. Quix, and 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)*, pp. 349-362, Singapore, Springer LNCS (1998).

[12] M. Jarke, R. Gallersdörfer, M.A. Jeusfeld, M. Staudt, and S. Eherer. ConceptBase - a deductive objectbase for meta data management. In *Journal of Intelligent Information Systems, Special Issue on Advances in Deductive Object-Oriented Databases*, **4**(2):167-192 (1995).

[13] M. Jarke, M.A. Jeusfeld, C. Quix, and P. Vassiliadis. Architecture and quality in data warehouses: an extended repository approach. *Information Systems*, **24**(3):229-253 (1999). A shorter version appeared also in *Proc. 10th Conference on Advanced Information Systems Engineering (CAiSE '98)*, Pisa, Italy, pp. 93-113, Springer-Verlag (1998).

[14] S. Ligoudistianos, T. Sellis, D. Theodoratos, and Y. Vassiliou. Heuristic algorithms for designing the global data warehouse with SPJ views. In *Proc. Internaitonal Conference on Data Warehousing and Knowledge Discovery, (DaWaK'99)*, Florence, Italy, pp. 66-105, Springer LNCS 1676, available at http://www.dbnet.ece.ntua.gr/~dwq/ (1999).

[15] Metadata Coalition. *Meta Data Interchange Specification, (MDIS Version 1.1)*. Available at http://www.he.net/~metadata/standards/ (1997).

[16] M. Oivo and V. Basili. Representing software engineering models: the TAME goal-oriented approach. *IEEE Transactions on Software Engineering*, **18**(10):886-898 (1992).

[17] K. Orr. Data quality and systems theory. *Communications of the ACM*, **41**(2):66-71 (1998).

[18] C. Quix. Repository support for data warehouse evolution. In *Proc. of the International Workshop on Design and Management of Data Warehouses (DMDW'99)*, Heidelberg, Germany, available at http://sunsite.informatik.rwth-aachen.de/DMDW99/ (1999).

[19] M. Staudt, C. Quix, and M.A. Jeusfeld. View maintenance and change notification for application program views. In *Proc. ACM Symposium on Applied Computing*, San Antonio, Texas, pp. 197-203, ACM Press (1998).

[20] G.K. Tayi, and D.P. Ballou. Examining data quality. In *Communications of the ACM*, **41**(2):54-57 (1998).

[21] D. Theodoratos and M. Bouzeghoub. Data currency quality factors in data warehouse design. In *Proc. of the International Workshop on Design and Management of Data Warehouses (DMDW'99)*, Heidelberg, Germany, available at http://www.dbnet.ece.ntua.gr/~dwq/ (1999).

[22] D. Theodoratos, S. Ligoudistianos, and T. Sellis. Designing the global data warehouse with SPJ queries. In *Proc. of the 11th International Conference on Advanced Information Systems Engineering (CAiSE'99)*, Heidelberg, Germany, pp. 180-194, Springer LNCS 1626, available at http://www.dbnet.ece.ntua.gr/~dwq/ (1999).

[23] D. Theodoratos, and T. Sellis. Data warehouse configuration. In *Proc. 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece,* pp. 126-135, Morgan Kaufmann (1997).

[24] D. Theodoratos, and T. Sellis. Designing data warehouses. *Data and Knowledge Engineering (DKE),* **31**(3):279-301, available at http://www.dbnet.ece.ntua.gr/~dwq/ (1999).

[25] P. Vassiliadis, C. Quix, Y. Vassiliou, and M. Jarke. A model for data warehouse operational processes. To appear in *Proc. 12th Conference on Advanced Information Systems Engineering (CAiSE '00),* Stockholm, Sweden, Springer LNCS (2000).

[26] R.Y. Wang, H.B. Kon, and S.E. Madnick. Data quality requirements analysis and modeling. In *Proc. of 9th International Conference on Data Engineering,* pp. 670-677, IEEE Computer Society, Vienna, Austria (1993).

[27] R.Y. Wang. A product perspective on total data quality management. *Communications of the ACM,* **41**(2):58-65 (1998).

[28] R.Y. Wang, V.C. Storey, and C.P. Firth. A framework for analysis of data quality research. *IEEE Transactions on Knowledge and Data Engineering,* **7**(4):623-640 (1995).

[29] R.Y. Wang, D. Strong, and L.M. Guarascio. *Beyond Accuracy: What Data Quality Means to Data Consumers.* Technical Report TDQM-94-10, Total Data Quality Management Research Program, MIT Sloan School of Management, Cambridge, Mass. (1994).

[30] Y. Wand and R.Y. Wang. Anchoring data quality dimensions in ontological foundations. *Communications of the ACM,* **39**(11):86-95(1996).