Data Warehouse Modeling and Quality Issues

Panos Vassiliadis

Knowledge and Data Base Systems Laboratory
Computer Science Division
Department of Electrical and Computer Engineering
National Technical University of Athens
Zographou 157 73, Athens, GREECE

Ph.D. Thesis

NATIONAL TECHNICAL UNIVERSITY OF ATHENS

ZOGRAPHOU 157 73, ATHENS, GREECE

JUNE 2000

ABSTRACT

Data warehouses can be defined as *'subject-oriented', integrated, time-varying, non-volatile collections of data that is used primarily in organizational decision making.* Nowadays, data warehousing became an important strategy to integrate heterogeneous information sources in organizations, and to enable On-Line Analytic Processing (OLAP). Unfortunately, neither the accumulation, nor the storage process, seem to be completely credible. For example, it has been suggested in the literature that more than $2 billion of U.S. federal loan money have been lost because of poor data quality at a single agency, that manufacturing companies spent over 25% of their sales on wasteful practices, a number which came up to 40% for service companies.

The question that arises, then, is how to organize the design, administration and evolution choices in such a way that all the different, and sometimes opposing, quality user requirements can be simultaneously satisfied.

To tackle this problem, this thesis contributes as follows:

The first major result that we present is a general framework for the treatment of data warehouse metadata in a metadata repository. The framework requires the classification of metadata in at least two instantiation layers and three perspectives. The metamodel layer constitutes the schema of the metadata repository and the metadata layer the actual meta-information for a particular data warehouse. The perspectives are the well known conceptual, logical and physical perspectives from the field of database and information systems.

We link this framework to a well-defined approach for the architecture of the data warehouse from the literature. Then, we present our proposal for a quality metamodel, which builds on the widely accepted Goal-Question-Metric approach for the quality management of information systems. Moreover, we enrich the generic metamodel layer with patterns concerning the linkage of (a) quality metrics to data warehouse objects and (b) of data warehouse stakeholders to template quality goals. Then, we go on to describe a metamodel for data warehouse operational processes. This metamodel enables data warehouse management, design and evolution based on a high level conceptual perspective, which can be linked to the actual structural and physical aspects of the data warehouse architecture. This metamodel is capable of modeling complex activities, their interrelationships, the relationship of activities with data sources and execution details.

The ex ante treatment of the metadata repository is enabled by a full set of steps, i.e., quality question, which constitute our methodology for data warehouse quality management and the quality-oriented evolution of a data warehouse based on the architecture, process and quality metamodels. 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.

Special attention is paid to a particular part of the architecture metamodel, the modeling of OLAP databases. To this end, we first provide a categorization of the work in the area of OLAP logical models by surveying some major efforts, including commercial tools, benchmarks and standards, and academic efforts. We also attempt a comparison of the various models along several dimensions, including representation and querying aspects. Our contribution lies in the introduction a logical model for cubes based on the key observation that a cube is not a self-existing entity, but rather a view over an underlying data set. The proposed model is powerful enough to capture all the commonly encountered OLAP operations such as selection, roll-up and drill-down, through a sound and complete algebra. We also show how this model can be used as the basis for processing cube operations and provide syntactic characterizations for the problems of cube usability.

Finally, this thesis gives an extended review of the existing literature on the field, as well as a list of related open research issues.

Table of Contents

Chapter 4: Data Warehouse Repository Exploitation

Chapter 5: Modeling Multidimensional Databases

Chapter 1
Introduction

1. DESCRIPTION

A *Data Warehouse* (DW) is a collection of technologies aimed at enabling the knowledge worker (executive, manager, analyst, etc) to make better and faster decisions. Many researchers and practitioners share the understanding that a data warehouse 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 layer 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* [BBC*99] and the *Metadata Interchange Specification* (MDIS) [Meta97] provide different models and application programming interfaces to control and manage metadata for OLAP databases. In Figure 1.1, a generic architecture for a data warehouse is depicted.

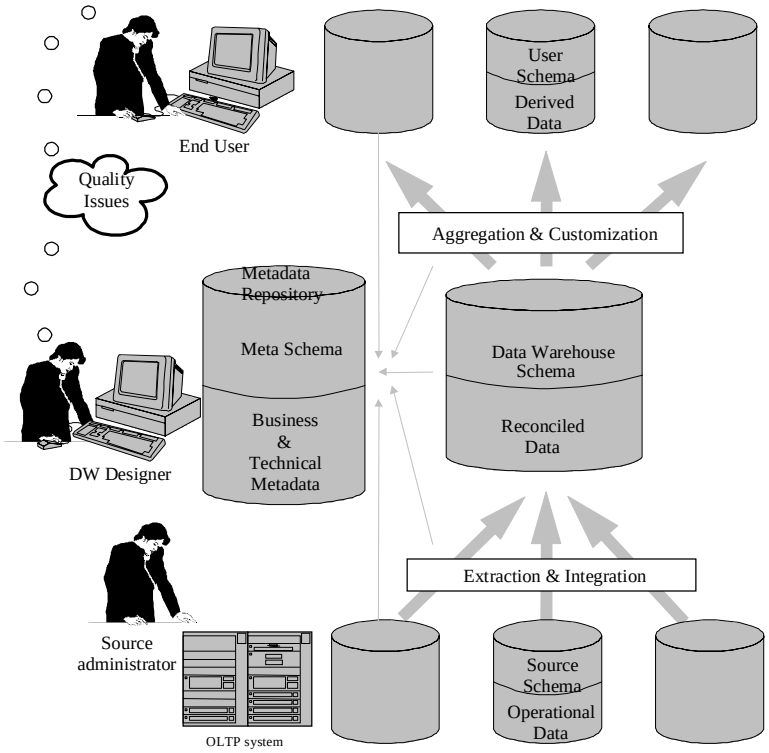


Figure 1.1. A generic architecture for a data warehouse

The emergence of data warehousing was initially a consequence of the observation by W. Inmon and E. F. Codd in the early 1990's that operational-level on-line transaction processing (OLTP) and decision support applications (OLAP) cannot coexist efficiently in the same database environment, mostly due to their very different transaction characteristics [Inmo96]. Inmon defined data warehouses as *'subject-oriented', integrated, time-varying, non-volatile collections of data that is used primarily in organizational decision making.* Since then, data warehousing became an important strategy to integrate heterogeneous information sources in organizations, and to enable On-Line Analytic Processing.
A report from the META group during the Data Warehousing Conference (Orlando, Fl) in February 1996, presented very strong figures for the area [JLVV99]:

- Data warehousing will be a $13,000 million industry within 2 years ($8,000m hardware, $5,000m on services and systems integration), while 1995 represents $2,000m levels of expenditure..
- The average expenditure for a data warehouse project is $3m. This is set to accelerate. 59% of the survey's respondents expect to support data warehouses in excess of 50GB by the middle of the year 1996 and 85% are claiming they will be supporting over 50 users in the same timeframe.

In 1998, the reality has exceeded these figures, reaching sales of $14,600m. The number and complexity of projects is indicative of the difficulty of designing good data warehouses. Still, it is an interesting question to discover why the big organizations and enterprises embark in such a costly, risky and painful task, as the construction of a data warehouse is. It is a fact, of course, that OLAP technology gives added value to the information system of an enterprise; still practice has proved that another issue is of equal or greater importance too.

It is well known that the ultimate goal of organizations, governments and companies, when they accumulate and store information is the ability to process it later and take advantage of it. Unfortunately, neither the accumulation, nor the storage process, seem to be completely credible. In [WaRK95] it is suggested that errors in databases have been reported to be up to ten percent range and even higher in a variety of applications. It is obvious that inaccurate, invalid, out-of-date or incomplete data may have a heavy financial, or even social, impact. Although the implementation of mechanisms for achieving data quality has financial risks and may prove not to be profitable for the

organization which decides to undertake the task of implementing it, it is at least accepted that there can be an equilibrium in the cost-quality tradeoff. In [WaSF95], it is reported that more than $2 billion of U.S. federal loan money had been lost because of poor data quality at a single agency. It also reported that manufacturing companies spent over 25% of their sales on wasteful practices. The number came up to 40% for service companies.

Data warehouses have proved their value to serve as repositories for integrated, homogenized and clean data. In other words, they do not serve only as information buffers for answering complex questions quickly but also as intermediate stages in the processing of information within the information system of an enterprise, where the information becomes more accurate and useful. Thus, at the end of the data processing chain, at the front end of an OLAP / DW application, is ultimately the overall quality of the information which is provided to the end user.

2. QUALITY AND DATA WAREHOUSES

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 [BBBB95]. We believe, though, that the best definition is the one found in [TaBa98, Orr98, WaSG94]: 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 [Orr98] mentions "the problem of data quality is fundamentally intertwined in how [...] users actually use the data in the system", since the users are actually 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.

As a decision support information system, a data warehouse must provide high level quality of data and quality of service. Coherency, freshness, accuracy, accessibility, availability and performance are among the quality features required by the end users of the data warehouse. Still, too many stakeholders are involved in the lifecycle of the data warehouse; all of them seem to have their quality requirements. As already mentioned, the *Decision Maker* usually employs an OLAP query tool to get answers interesting to him. A decision-maker is usually concerned with the *quality of the stored data*, their *timeliness* and the *ease of querying* them through the OLAP tools. The *Data Warehouse Administrator* needs facilities such as *error reporting*, *metadata accessibility* and knowledge of the *timeliness* of the data, in order to detect changes and reasons for them, or problems in the stored information. The *Data Warehouse Designer* needs to measure the *quality of the schemata* of the data warehouse environment (both existing and newly produced) and the *quality of the metadata* as well. Furthermore, he needs *software evaluation standards* to test the software packages he considers purchasing. The *Programmers of Data Warehouse Components* can make good use of *software implementation standards* in order to accomplish and evaluate their work. *Metadata reporting* can also facilitate their job, since they can avoid mistakes related to schema information. Based on this analysis, we can safely argue that different roles imply a different collection of quality aspects, which should be ideally treated in a consistent and meaningful way.

From the previous it follows that, on one hand, the quality of data is of highly *subjective* nature and should ideally be treated differently for each user. At the same time, the quality goals of the involved stakeholders are highly diverse in nature. They can be neither assessed nor achieved directly but require complex measurement, prediction, and design techniques, often in the form of an interactive process. On the other hand, the reasons for data deficiencies, non-availability or reachability problems are definitely *objective*, and depend mostly on the information system definition and implementation. Furthermore, the prediction of data quality for each user must be based on objective quality factors that are computed and compared to users' expectations. **The question that arises, then, is how to organize the design, administration and evolution of the data warehouse in such a way that all the different, and sometimes opposing, quality requirements of the users 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, another 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 user requests response time and the global data warehouse maintenance cost at the same time.

Before answering these questions, though, it should be useful to **make a clear-cut definition of the major concepts** in these data warehouse quality management problems. To give an idea of the complexity of the situation let us present a verbal description of the situation. The *interpretability* of the data and the processes of the data warehouse is heavily dependent on the design process (the level of the description of the data and the processes of the warehouse) and the expressive power of the models and the languages which are used. Both the data and the systems architecture (i.e. where each piece of information resides and what the architecture of the system is) are part of the interpretability dimension. The integration process is related to the interpretability dimension, by trying to produce minimal schemata. Furthermore, processes like query optimization (possibly using semantic information about the kind of the queried data -e.g. temporal, aggregate, etc.), and multidimensional aggregation (e.g. containment of views, which can guide the choice of the appropriate relations to answer a query) are dependent on the interpretability of the data and the processes of the warehouse. The *accessibility* dimension of quality is dependent on the kind of data sources and the design of the data and the processes of the warehouse. The kind of views stored in the warehouse, the update policy and the querying processes are all influencing the accessibility of the information. Query optimization is related to the accessibility dimension, since the sooner the queries are answered, the higher the transaction availability is. The extraction of data from the sources is also influencing (actually determining) the *availability* of the data warehouse. Consequently, one of the primary goals of the update propagation policy should be to achieve high availability of the data warehouse (and the sources). The update policies, the evolution of the warehouse (amount of purged information) and the kind of data sources are all influencing the *timeliness* and consequently the *usefulness* of data. Furthermore, the timeliness dimension influences the data warehouse design and the querying of the information stored in the warehouse (e.g., the query optimization could possibly take advantage of possible temporal relationships in the data warehouse). The *believability* of the data in the warehouse is obviously influenced from the believability of the data in the sources. Furthermore, the level of the desired believability influences the design of the views and processes of the warehouse. Consequently, the source integration should take into account the believability of the data, whereas the data warehouse design process should also take into account the believability of the processes. The validation of all the processes of the data warehouse is another issue, related with every task in the data warehouse environment and especially with the design process. Redundant information in the warehouse can be used from the aggregation, customization and query optimization processes in order to obtain information faster. Also, replication issues are related to these tasks. Finally, quality aspects influence several factors of data warehouse design. For instance, the required storage space can be influenced by the amount and volume of the quality indicators needed (time, believability indicators etc.). Furthermore, problems like the improvement of query optimization through the use of quality indicators (e.g. ameliorate caching), the modeling of incomplete information of the data sources in the data warehouse, the reduction of negative effects schema evolution has on data quality and the extension of data warehouse models and languages, so as to make good use of quality information have to be dealt with.

3. FRAMEWORK AND MOTIVATION FOR RESEARCH

Our proposal is based on the assumption that the metadata repository of the data warehouse can serve as the cockpit for the management of quality. To make the repository functional, we must provide a coherent framework that captures the data warehouse from three different viewpoints, namely the *architecture*, *process* and *quality* viewpoints.

Why an architecture model?

Although many data warehouses have already been built, there is no common methodology, which supports database system administrators in designing and evolving a data warehouse. The problem with architecture models for data warehouses is that practice has preceded research in this area and continues to do so. Consequently, the task of providing an abstract model of the architecture becomes more difficult.

Formally, an architecture model corresponds to the schema structure of the meta-database that controls the usually distributed and heterogeneous set of data warehouse components and therefore is

the essential starting point for design and operational optimization. **The purpose of architecture models is to provide an expressive, semantically defined and computationally understood meta modeling language, based on observing existing approaches in practice and research**. Expressiveness and services of the metadata schema are crucial for data warehouse quality.

To facilitate complex analysis and visualization, information targeted for On-Line Analytical Processing is organized according to the *multidimensional data model*, which involves aggregation of data according to specific data dimensions. It has been argued that traditional data models (e.g., the relational one) are in principle not powerful enough for data warehouse applications, whereas multidimensional models provide the functionality needed for summarizing, viewing, and consolidating the information available in data warehouses [JLVV99]. Despite this consensus on the central role of multidimensional data cubes, and the variety of the proposals made by researchers, there **is little agreement on finding a common terminology and semantic foundations for a multidimensional data model**. The missing semantic enrichment will not only enhance data warehouse design but can also be used to optimize data warehouse operation, by providing reasoning facilities to exploit redundant pre-materialized (or cached) information which can be reused in more than one operations.

Why a process model?

The *static* description of the architecture parts of the data warehouse can be complemented, with a metamodel of the *dynamic* parts of the data warehouse, i.e. the data warehouse processes. **Providing a process model for data warehouses practically captures the behavior of the system and the interdependencies of the involved stakeholders.** Since data warehouse processes are data intensive, special care should be taken to capture the interrelationship of the process definition to the data involved to their operation. Moreover, due to the diversity and number of the stakeholders in the data warehouse environment, the tracking of data warehouse processes provides the ability to administer the operation and especially the evolution of the data warehouse, based on the knowledge of the possible effects of any actions, to the involved stakeholders or architecture objects.

Why a quality model? How to use it?

As already made explicit, apart from their role as passive buffers in the data propagation path between operational databases and end users, data warehouses can also serve as the bridge of the gap between subjective user requirements for information quality and objective detection and measurements of data deficiencies. The data warehouse, being a collective *off-line*, multi-layered system, can serve as a "data cleaner" for the information presented to the user. Recording the quality of the data warehouse data and processes in a metadata repository is providing, thus, the data warehouse stakeholders with sufficient power to evaluate, understand and possibly react against the structure and content of the warehouse. The quality model of the data warehouse, answers questions like:

- How can the quality of a data warehouse be evaluated and designed?
- How can the quality of a data warehouse be managed in the everyday usage of the data warehouse?

Still, although quality considerations have accompanied data warehouse research from the beginning, most of the research has focused on issues such as the trade-off between freshness of data warehouse data and disturbance of OLTP work during data extraction; the minimization of data transfer through incremental view maintenance; and a theory of computation with multi-dimensional data models [JLVV99].

From the viewpoint of information systems research, though, the problem of **how to design, construct and administer data warehouses that satisfy specific, well-defined quality criteria is still present**. Due to the versatile nature of quality requirements, it is desirable to **achieve breadth of coverage without giving up the detailed knowledge available for certain criteria.** The heavy use of highly qualified consultants in data warehouse applications indicates that we have not yet reached a high-level, systematic solution to the problem. The basic issues raised in this context are basically

- The introduction of formal models of data warehouse quality;

- Measurement techniques that populate the quality model, in order to make it useful;
- A methodology that ensure the quality of the data and processes during both the design and the everyday usage of the warehouse.

As a concluding general statement, the enrichment of the metadata of a data warehouse with the appropriate models will be beneficial for the design, administration and usage of the data warehouse.

4. ROADMAP FOR THE REST OF THE DOCUMENT

We answer all the aforementioned problems in the following chapters of this document. In Chapter 2, we adopt an architecture metamodel [JJQV99] as the basis for the metadata repository of the data warehouse. A framework for data warehouse metadata and a quality metamodel accompany the architecture metamodel. In Chapter 3, the metamodel for data warehouse processes is introduced. The three metamodels are accompanied by a methodology for quality-oriented data warehouse management in Chapter 4. In Chapter 5 we present a formal metamodel for multidimensional databases as well as query optimization techniques that exploit it. Chapter 6 presents information about a prototype OLAP tool, which implemented a previous version of the multidimensional metamodel. Finally, in Chapter 7, we conclude our results.

Specifically, the thesis is structured as follows:
The first major result that we present in Chapter 2, is a general framework for the treatment of data warehouse metadata in a metadata repository. The framework requires the classification of metadata in at least two instantiation layers and three perspectives. The metamodel layer constitutes the schema of the metadata repository and the metadata layer the actual meta-information for a particular data warehouse. The perspectives are the well known conceptual, logical and physical perspectives from the field of database and information systems.

We link this framework to a well-defined approach for the architecture of the data warehouse from the literature. Then, we present our proposal for a quality metamodel, which builds on the widely accepted Goal-Question-Metric approach for the quality management of information systems. Moreover, we enrich the generic metamodel layer with patterns concerning the linkage of (a) quality metrics to data warehouse objects and (b) of data warehouse stakeholders to template quality goals.

Then, in Chapter 3, we go on to describe a metamodel for data warehouse operational processes. This metamodel enables data warehouse management, design and evolution based on a high level conceptual perspective, which can be linked to the actual structural and physical aspects of the data warehouse architecture. This metamodel is capable of modeling complex activities, their interrelationships, the relationship of activities with data sources and execution details.

The ex ante treatment of the metadata repository is enabled by a full set of steps, i.e., quality question, which constitute our methodology for data warehouse quality management and the quality-oriented evolution of a data warehouse based on the architecture, process and quality metamodels. Our approach –presented in Chapter 4- 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.

Special attention is paid to a particular part of the architecture metamodel, the modeling of OLAP databases. To this end, in Chapter 5, we first provide a categorization of the work in the area of OLAP logical models by surveying some major efforts, including commercial tools, benchmarks and standards, and academic efforts. We also attempt a comparison of the various models along several dimensions, including representation and querying aspects. Our contribution lies in the introduction a logical model for cubes based on the key observation that a cube is not a self-existing entity, but rather a view over an underlying data set. The proposed model is powerful enough to capture all the commonly encountered OLAP operations such as selection, roll-up and drill-down, through a sound and complete algebra. We also show how this model can be used as the basis for processing cube operations and provide syntactic characterizations for the problems of cube usability. In Chapter 6, we

present an experimental prototype OLAP tool, implementing a previous version of the proposed model.

Finally, this thesis gives an extended review of the existing literature on the field, as well as a list of related open research issues.

Chapter 2
Architecture and Quality Metamodels

1. INTRODUCTION

We made an interesting point at the end of the previous chapter, by stating that the efficient management of the lifecycle of a data warehouse is enabled by the inspection of three *viewpoints* concerning the data warehouse: its *architecture*, *processes* and *quality*. The architecture constitutes of the static components comprising the data warehouse; the processes capture the dynamic part of the data warehouse environment. Finally, quality is a measure of the fulfillment of the expectations of the involved stakeholders in such an environment.
For each viewpoint, a respective metamodel can be derived. It is important, of course, that all three metamodels respect a coherent framework, and fit gracefully with each other. We will immediately proceed to present the architecture metamodel for a data warehouse, as well as the general framework for metadata management. In the next section we will make a proposal for a quality metamodel and in the fourth section we will present templates for quality management in a data warehouse environment. The data warehouse processes -which we believe to be the more interesting part of the whole environment- will be detailed in the next chapter.

1.1 State of the art in metadata management

As mentioned in [JLVV99], metadata is stored in a repository, where it can be accessed from every component of the data warehouse. The Metadata Coalition has proposed a *Metadata Interchange Specification* [Meta97]; additional emphasis has been placed on this area through the recent efforts of Microsoft to introduce a repository product in their Office tool suite, including some simple Information Models for data warehousing [BBC*99]. We refer the interested reader to [JLVV99] for a more detailed description of these approaches. The *Telos language* developed jointly between the University of Toronto and a number of European projects in the late 80's, is specifically dedicated to the goal of metadata management [MBJK90]. Telos, in the axiomatized form defined in [Jeus92], offers an unlimited classification hierarchy in combination with abstractions for complex objects and

for generalizations, both for objects and for links between them, i.e. both are first-class citizens of the language, offering maximum flexibility in modeling and re-modeling complex metadata. In particular, it becomes possible to define, not just syntactically but also semantically, meta models for new kinds of metadata introduced in the distributed system managed by the repository, and therefore in the repository itself. Such metamodels are often also called *Information Models* and exist typically for all kinds of objects and processes used in system analysis, design and evolution.
The ConceptBase system, developed since 1987 at RWTH Aachen, Germany [JaRo88, JGJ*95, NiJa98], as a knowledge-based repository, integrates a logical semantics with the basic abstractions for the purpose of analyzing the consistency of stored repository objects such as software specifications in different kinds of formalisms. Through the axiomatization of the Telos semantics, ConceptBase achieves a combination of structural object-orientation with the kinds of reasoning capabilities offered by deductive relational databases and relational query optimization. ConceptBase will be the metadata repository to which we will refer in the rest of this document.

1.2 The proposal for a data warehouse architecture model

In [JJQV99] the need for a data warehouse metadata structure that offers three *perspectives* is argued:
- a *conceptual* business perspective developed around the *enterprise* model of the organization;
- a *logical* perspective covering all the schemata of the data warehouse environment, and
- a *physical* perspective representing the storage and execution properties of the data warehouse components.

Each of these perspectives, and their interrelationships, are orthogonally linked to the three traditional layers of data warehousing, namely sources, data warehouse, and clients. The metamodel is reproduced in Figure 2.1. In this section, we will quickly present the extended metamodel resulting from this approach, and the reasons for which we adopt it. This description is fully elaborated in [JJQV99]. Note that the architecture model was basically developed in RWTH, the Technical University of Aachen, by C. Quix, M. Jeusfeld, M. Jarke.

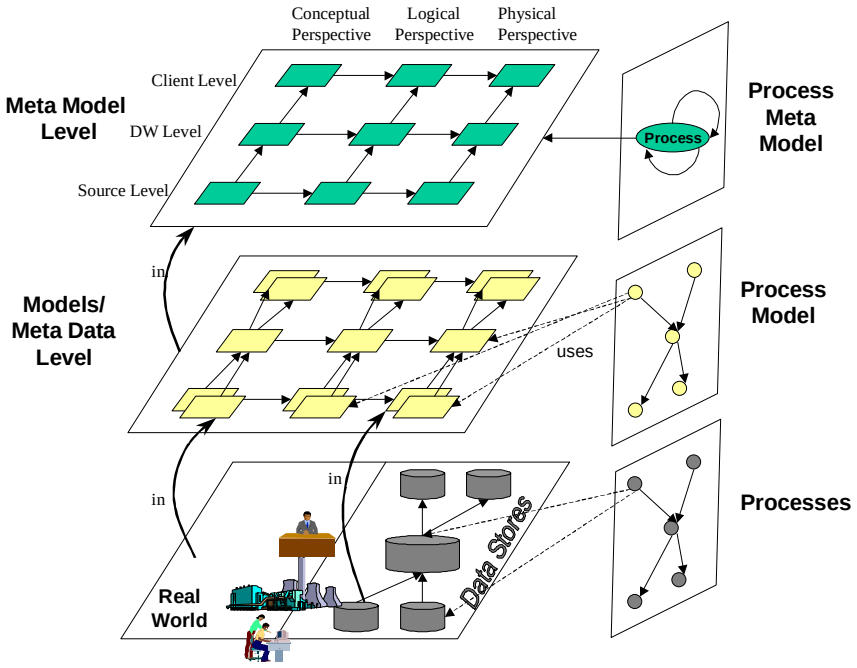


Figure 2.1. The [JJQV99] Data Warehouse MetaData Framework

At the beginning of this description we would like to remind the reader that (a) data warehouse systems are unique in the sense that they rely on a run-time *meta-database* (or *metadata repository*)

that stores information about the data and processes in the system and (b) our approach is based on the Telos language [MBJK90], and the ConceptBase metadata repository [JGJ*95].
A condensed ConceptBase model of the architecture notation is given in Figure 2.2, using the graph syntax of Telos. Bold arrows denote specialization/instantiation links (depending on the respective tag each time). The top-level object is *MeasurableObject*. It classifies objects at any perspective (conceptual, logical, or physical) and at any level (source, data warehouse, or client). Within each perspective, we distinguish between the modules it offers (e.g. client model) and the kinds of information found within these modules (e.g. concepts and their subsumption relationships). The horizontal levels of the information structures are coded by the three subclasses attached to *Model*, *Schema*, and *DataStore*, each for a different perspective. The horizontal links *isViewOn* and *hasStructure* establish the way how the horizontal links in Figure 2.2 are interpreted: the types of a schema (i.e., relational or multidimensional structures) are defined as logical views on the concepts in the conceptual perspectives. On the other hand, the components of the physical perspective get a schema from the logical perspective as their schema.
Each object can have an associated set of materialized views called *QualityMeasurements*. These materialized views (which can also be specialized to the different perspectives – not in the figure) constitute the bridge to the quality model discussed in the Section 3.

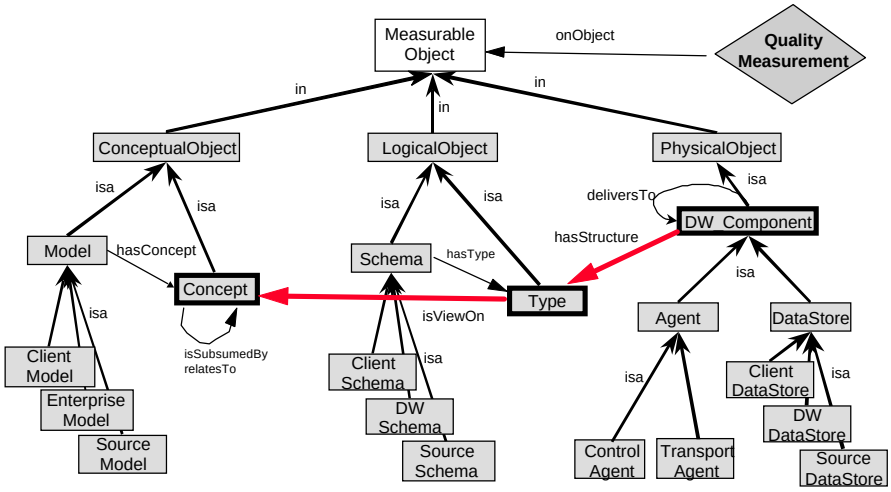


Figure 2.2. Structure of the Repository Meta Model

Note that all the objects in Figure 2.2 are meta-classes: actual conceptual models, logical schemas, and data warehouse components are represented as instances of them in the meta-database. Actually, as depicted in Figure 2.1, we can discriminate between three levels of instantiation. The *metamodel* (i.e. the topmost layer in Figure 2.1) provides a notation for data warehouse generic entities, such as schema, or agent, including the business perspective. Each box shown in Figure 2.1 is decomposed into more detailed data warehouse objects in the metamodel of [JJQV99]. This metamodel is instantiated with the *metadata* of the data warehouse (i.e. the second layer in Figure 2.1), e.g. relational schema definitions or the description of the conceptual data warehouse model. The lowest layer in Figure 2.1 represents the real world where the actual processes and data reside: in this level the metadata are instantiated with data instances, e.g. the tuples of a relation or the objects of the real world which are represented by the entities of the conceptual model.
We adopt the approach of [JJQV99] for specific reasons. During the analysis of previous data warehouse frameworks (see [JLVV99] for a broader discussion), one could observe that these frameworks cover only logical schemata and their physical implementation. However, data warehouse users may interpret the data in different ways because they think of different concepts. This situation leads often to misunderstandings of the data. Furthermore, enterprises have a huge number of heterogeneous data sources and it is difficult to have on overall picture of what kind of data is available in the source and to keep track of the interdependencies between the data sources. Thus, it is important to note that the proposal of [JJQV99] extended the traditional data warehouse architecture in two ways: (i) a conceptual perspective is clearly added and (ii) a clean separation of locality (here: *level*) between the *source*, *data warehouse* and *client* level is made. The central object in the proposed

metamodel depicted in Figure 2.2 is the conceptual *enterprise model*, which gives a conceptual description of the data in the enterprise. All the information that is available in the sources and requested by the clients of the data warehouse is expressed in terms of this enterprise model.

To summarize, we stress that the **metadata framework** that we adopt is based on the following assumptions / requirements:

1. A clear distinction exists between different *layers of instantiation*. Thus, for all the metamodels that we will present, we will require that the generic *metamodel layer* deals abstractly with entities applicable to any data warehouse; the *metadata layer* deals will the schemata of a specific data warehouse under examination; and finally the *instance layer* intantiates the previous ones at the real-world domain. Out of the three layers of instantiation, we require that the two first should be treated clearly by a metadata repository. The instance layer, though, is optional as far as the metadata repository is concerned, since it deals with the actual information (and not its "meta" description).
2. A clear distinction exists between *perspectives*. Following a classical database / information systems categorization, we require that a *conceptual perspective* captures the world in a vocabulary really close to the one of the final user; the *physical perspective* covers the data warehouse environment in terms of real world, computer-oriented components or events; and finally, the *logical perspective* acts as an intermediate between these two extremes in order to provide a concrete vocabulary of the data warehouse environment, being independent, though, from strict implementation details.

As we saw, the architecture model of [JJQV99], not only covers these requirements, but provides an extra useful categorization, namely the locality attribute of any data warehouse component. In Section 3, we will introduce a quality metamodel that covers the second viewpoint for data warehouses. Still, before that, we feel that it is necessary to present the state of the art in the field of quality definition and management, in information systems.

2. STATE OF THE ART ON DATA WAREHOUSE QUALITY

Models and tools for the management of data warehouse quality can build on substantial previous work in the fields of data quality.

2.1 Quality Definition

In [BBBB95], a definition and quantification of *quality* is given, as the fraction of *Performance* over *Expectance*. Taguchi defined quality as the loss imparted to society from the time a product is shipped [BBBB95]. The total loss of society can be viewed as the sum of the producer's loss and the customer's loss. It is well known that there is a tradeoff between the quality of a product or service and a production cost and that an organization must find an equilibrium between these two parameters. If the equilibrium is lost, then the organization loses anyway (either by paying too much money to achieve a certain standard of quality, called "target", or by shipping low quality products of services, which result in bad reputation and loss of market share).

2.2 Data Quality Research

Quite a lot of research has been done in the field of data quality. Both researchers and practitioners have faced the problem of enhancing the quality of decision support systems, mainly by ameliorating the quality of their data. In this section we will present the related work on this field, which more or less influenced our approach for data warehouse quality. A detailed presentation can be found in [VaSL97] and [WaSF95].

Wang et al. [WaSF95] presents a framework of data analysis, based on the ISO 9000 standard. The framework consists of seven elements adapted from the ISO 9000 standard: management responsibilities, operation and assurance cost, research and development, production, distribution, personnel management and legal function. This framework reviews a significant part of the literature

on data quality, yet only the research and development aspects of data quality seem to be relevant to the cause of data warehouse quality design. The three main issues involved in this field are: analysis and design of the data quality aspects of data products, design of *data manufacturing systems* (DMS's) that incorporate data quality aspects and definition of data quality dimensions. We should note, however, that data are treated as products within the proposed framework. The general terminology of the framework regarding quality is as follows: *Data quality policy* is the overall intention and direction of an organization with respect to issues concerning the quality of data products. *Data quality management* is the management function that determines and implements the data quality policy. A *data quality system* encompasses the organizational structure, responsibilities, procedures, processes and resources for implementing data quality management. *Data quality control* is a set of operational techniques and activities that are used to attain the quality required for a data product. *Data quality assurance* includes all the planed and systematic actions necessary to provide adequate confidence that a data product will satisfy a given set of quality requirements.

In [WaSF95] the research regarding the design of data manufacturing systems that incorporate data quality aspects is classified into two major approaches: the *development of analytical models* and the *design of system technologies to ensure the quality of data*.

In the papers concerning the development of analytical models, one can find efforts such as the ones presented in [BaPa85], [BaPa87] and [BWPT93]. In [BaPa85] and [BaPa87] a model that produces expressions for the magnitude of errors for selected terminal outputs, is presented. The model can be deployed to investigate the quality of data in existing systems. Furthermore, an operations research model is presented for the analysis of the effect of the use of quality control blocks, in the management of production systems. In [BWPT93] a data-manufacturing model to determine data product quality is proposed. The model is used for the assessment of the impact quality has on data delivered to 'data customers'.

In the field of the design of system technologies to ensure the quality of data, there exist several interesting efforts. In [HKRW90], [PaRe90], [Redm92] a data tracking technique is proposed. This technique uses a combination of statistical control and manual identification of errors and their sources. The basis of the proposed methodology is the assumption that processes that create data are often highly redundant. The aim of the methodology is to identify pairs of steps in the overall process that produce inconsistent data. In [WaKM93], [WaRG93], [WaRK95], [WaMa90] an attribute-based model that can be used to incorporate quality aspects of data products has been developed. The basis of this approach is the assumption that the quality design of an information system can be incorporated in the overall design of the system. The model proposes the extension of the relational model as well as the annotation of the results of a query with the appropriate quality indicators. In [WaKM93], in particular, an extension of the ER model is proposed, in order to incorporate data quality specification. The data quality design is handled as an internal part of the whole database design. The methodology deals with data quality requirements collection and definition. Quality parameters are initially treated as subjective (user requirements) and then mapped to objective technical solutions. [Svan84] reports on a case study, regarding the testing of an actual development of an integrity analysis system. A methodology -consisting of seven steps- is suggested: i) defining database constraints, ii) selecting statistical techniques for sampling the database, iii) selecting the integrity analysis to be performed, iv) defining suitable quality measures, v) specifying outputs to be produced from a defect file, vi) developing and testing program code, and finally vii) executing the integrity analysis system.

Finally, there is a great deal of work related to the definition of data quality dimensions. In [BaPa82], [BaPa85], [BaPa87], [BaTa89], [BWPT93], the following quality dimensions are defined: *accuracy* (conformity of the stored with the actual value), *timeliness* (recorded value is not out of date), *completeness* (no information is missing), *consistency* (the representation of data is uniform). In [StLW94] data quality is modeled through the definition of intrinsic, contextual, representation, and accessibility aspects of data. In [Jans88], [LiUp90], [WaRG93] *validation*, *availability*, *traceability* and *credibility* are introduced. In [More82] a method that estimates the 'true' stored error rate is proposed. In the field of information systems, many efforts are focused on the evaluation of the success of a system from the user's point of view. In [Hall*78] various factors are proposed, such as *usability*, *reliability*, *independence*, etc. [Krie79] proposes attributes such as *accuracy*, *timeliness*, *precision*, *reliability*, *completeness* and *relevancy*. In [AgAh87] the *reliability* of an information system is measured. The following measures are proposed: *internal reliability* ("commonly accepted" characteristics of data items), *relative reliability* (data compliance to user requirements) and *absolute reliability* (the level of data resemblance to reality).

In [WaSF95] it is suggested that the establishment of data quality dimensions can be achieved through two possible ways. The first is the usage of a scientifically grounded approach in order to achieve a rigorous definition. In the context of this approach, an ontological-based [Wand89], [WaWa96], [WaWe90] approach has been proposed in order to identify the possible deficiencies that exist when mapping the real world into an information system. In [DeMc92] information theory is used as the basis for the foundation of data quality dimensions, whereas in [WaSG94] marketing research is used as the basis for the same cause. The second way to establish data quality dimensions is the use of pragmatic approaches. E.g. data quality dimensions can be thought as user defined. Another proposal [WaSF95] is the formation of a data quality standard technical committee for this task. A previous example of similar practice is the IEEE standard for software quality dimensions [Schn90].

2.3 Software Quality Research

Apart from the research in the quality of data, quite a lot of research has been done in the field of software quality. The research in this field is actually centered towards the design, implementation, testing and evaluation of software.

Several models for software quality have been proposed. The GE Model of software quality [McRW78], suggests 11 criteria of software quality. The model developed by B. Boehm [Boeh89] suggests 19 quality factors for software. ISO 9126 [ISO97] suggests six basic quality factors that are further analyzed to an overall of 21 quality factors. In [HyRo96] a comparative presentation of these three models is done. Furthermore, the SATC software quality model is presented, along with the metrics for the software quality dimensions.

The GE Model suggests that the dimensions software should have are: *correctness*, *reliability*, *integrity*, *usability*, *efficiency*, *maintainability*, *testability*, *interoperability*, *flexibility*, *reusability* and *portability*.

Boehm's model suggests *correctness*, *reliability*, *integrity*, *usability*, *efficiency*, *maintainability*, *flexibility*, *reusability*, *portability*, *clarity*, *modifiability*, *documentation*, *resilience*, *understandability*, *validity*, *generality*, *economy*.

ISO 9126 model suggests *functionality* (suitability, accuracy, interoperability, compliance, security), *reliability* (maturity, fault tolerance, recoverability), *usability* (understandability, learnability, operability), *efficiency* (time behavior, resource behavior), *maintainability* (analyzability, changeability, stability, testability), *portability* (adaptability, installability, conformance, replacability).

Finally, SATC suggests *requirements quality* (ambiguity, completeness, understandability, volatility, traceability), *product (code) quality* (structure/architecture, maintainability, reusability, internal documentation, external documentation), *implementation effectivity* (resource usage, completion rates), *testing effectivity* (correctness).

The interested reader can find quite a lot of standards developed from ISO and IEEE in the field of software quality. A list of them is found in Appendix A.

2.4 Quality engineering and management

2.4.1 Total Quality Management

In our days, one of the most popular trends in the field of quality engineering is *Total Quality Management (TQM)*. Total Quality Management is a philosophy for the improvement of an organization in order to achieve excellence. It involves the whole structure and operation of an organization. In [BBBB95], Total Quality Management is considered to be heavily dependent upon 6 crucial concepts:

1. A management committed to TQM ideas, in order to provide long term organizational support at all levels.
2. An unwavering focus both to the internal and the external customers.
3. The effective involvement of all the work force of the organization.
4. The continuous improvement of all the processes of the organization.
5. The treatment of suppliers as partners.
6. The establishment of performance measures for all the processes.

As already mentioned, TQM considers quality as the fraction of performance over expectance. The greater quality is, the better a customer feels about a product or service. In [Garv88] the dimensions of quality (regarding TQM) are presented:

- performance (primary product characteristics);
- features (secondary product characteristics);
- conformance (to specification standards);
- reliability (consistency of performance over time);
- durability (useful life of a product);
- service (resolution of problems, ease of repair);
- response (human-to-human interface);
- aesthetics (sensory characteristics);
- reputation (past performance).

For reasons of completeness, we will proceed in a very short presentation of the basic principles of TQM. TQM involves a strategic planning for the achievement of quality. In [BBBB95] it is suggested that several steps have to take place in order to complete the strategic planning successfully:

1. Discovery of the customers needs.
2. Positioning of the organization towards its customer base.
3. Future prediction.
4. Analysis of the gaps between current and future states of the organization.
5. Closing of these gaps through the construction of an action plan.
6. Alignment of this plan to the mission, vision and core values of the organization.
7. Implementation of the plan.

Nevertheless, the original design is never sufficient enough. Continuous control and improvement of processes must take place. In [Jura88], a specific approach is presented for the achievement of this goal. This approach is called the *Juran Trilogy* and consists of several steps:

- planning
  - ⇒ determination of customers;
  - ⇒ discovery of their needs;
  - ⇒ development of product/service features responding to the customers' needs;
  - ⇒ development of processes able to produce these necessary features;
- control
  - ⇒ evaluation of the actual operating performance;
  - ⇒ comparison of this performance to the goals of the organization;
  - ⇒ action on the difference;
- improvement

To be useful, quality has to be measured. As reported in [Boed87], performance measures are used to establish baseline measures, determine processes which need to be improved, indicate process gains and losses, compare goals with actual performance, provide information for team and personal evaluation and finally, manage by fact rather than by gut feeling. Several criteria are defined for the evaluation of the measures employed:

- simplicity;
- number: measures should be few in number so that all users (i.e., persons that perform the measurement) can concentrate on them;
- development by the user;
- relevance to the customer;
- focus on the improvement of processes, prevention of errors and long-term planning;
- cost;
- knowledge of the measures from the side of the users.

Several characteristics of a process or function can be used to measure the performance of a process, as reported in [Boed87]:

- quantity of products or business processes produced;

- cost (the amount of resources required to produce a given output);
- time (timeliness of the output of a process);
- accuracy (number of non-conformances in the output of a process);
- function (extent of performance of the specified function);
- service;
- aesthetics (quantification come through the percentage of people who like the product's exterior design).

2.4.2 Statistical Process Control (SPC)

Statistical Process Control (SPC) is one of the best tools for monitoring and improving product and service quality. In [Best94], [BBBB95], seven basic techniques are proposed for this purpose:

- Pareto diagrams.
- Process flow diagrams.
- Cause and effect (Ishikawa) diagrams.
- Check sheets.
- Histograms.
- Control charts.

Pareto diagrams are used to identify the most important factors of a process, based on the assumption that most of the errors result from very few causes. A Pareto diagram is a graph that ranks data classifications (e.g. system failures) in a descending order. The process flow diagrams show the flow of product (services) through the various subsystems of a bigger system. Possible errors can be detected through the analysis of such a diagram. The cause and effect diagrams, introduced by Dr. Ishikawa in 1943, are composed of lines and symbols representing meaningful relationships between an effect and its causes, in a recursive fashion. Check sheets and histograms are very known techniques for representing graphically cumulative information about failure phenomena. Finally, control charts are used in order to relate an average (possibly expected or predefined) value of quality to statistical data representing the behavior of the actual products (services).

2.4.3 ISO 9000

The International Organization for Standardization (ISO) was founded in Geneva, Switzerland, in 1946. Its goal is to promote the development of international standards to facilitate the exchange of goods and services worldwide.
The original ISO 9000 [ISO92], [ISO97] standards were a series of international standards (ISO 9000, ISO 9001, ISO 9002, ISO 9003, ISO 9004), developed by ISO Technical Committee 176 (TC176) to provide guidance on selection of an appropriate quality management program (system) for a supplier's operations. The series of standards serves the purpose of common terminology definition and demonstration of a supplier's capability of controlling its processes. The titles and content of the 1994 edition of the ISO 9000 series are described in the following paragraphs:

ISO 9000 -1, Quality Management and Quality Assurance Standards - Guidelines for Selection and Use. This standard explains fundamental quality concepts, defines key terms and provides guidance on selecting, using and tailoring series. Furthermore, it helps in the selection and use of the standards in the ISO 9000 family.
ISO 9001-1, Quality Systems - Model for Quality Assurance in Design/Development, Production, Installation and Servicing. This is the most comprehensive standard. It addresses all elements in the including design. The 1994 edition of the standard improved the consistency of the terminology and clarified or expanded the meaning of some of the standards clauses. Several new requirements, such as that for quality planning, were added. The standard contains 20 elements describing the quality parameters, from the receipt of a contract through the design/delivery stage, until the service required after delivery.
ISO 9002, Quality Systems - Model for Quality Assurance in Production and Installation and Servicing. This standard is now identical to ISO 9001 except for the design requirements. Consequently, it addresses organization not involved in the design process.

ISO 9003, Quality Systems - Model for Quality Assurance in Final Inspection and Test. This is the least comprehensive standard. It addresses the detection and control of problems during final inspection and testing. Thus, it is not a quality control system. The 1994 edition added additional requirements including: contract review, control of customer supplied product, corrective actions, and internal quality audits.
ISO 9004 -1, Quality Management and Quality System Elements - Guidelines. This standard provides guidance in developing and implementing an internal quality system and in determining the extent to which each quality system element is applicable. The guidance in ISO 9004-1 exceeds the requirements contained in ISO 9001, ISO 9002 and ISO 9003. ISO 9004-1 is intended to assist a supplier in improving internal quality management procedures and practices. Yet, it is not intended for use in contractual, regulatory or registration applications.

The full list of the developed standards is found in Appendix A. Out of them, the *«ISO/DIS 9000-3 Quality management and quality assurance standards -- Part 3: Guidelines for the application of ISO 9001:1994 to the development, supply, installation and maintenance of computer software (Revision of ISO 9000-3:1991)»* standard, is specifically intended for use in the computer software industry. Furthermore, there are several standards developed from ISO, concerned with the achievement of quality in the development and evaluation of software. Yet, these standards are not directly concerned with ISO 9000, so they are covered in the section of software quality research.

A framework for quality analysis is developed in ISO 9001-1 and consists of the following 20 elements [BBBB95]:

- management responsibility (quality policy, responsibility - authority, management review);
- the quality system (establishment and maintenance of a documented quality system);
- contract review (clear definition of contracts, existence of unusual quality requirements, feasibility of the implementation);
- design control (establishment and maintenance of procedures to control and verify that product design meets specified requirements);
- document control (establishment and maintenance of procedures to control all document and data that affect the quality of a product or service);
- purchasing (conformance of purchased materials or products, to specified requirements);
- customer supplied support (identification and isolation of customer supplied materials from any similar organization-owned items);
- product identification and traceability (establishment of procedures for the identification of the products during all stages of production, delivery and installation);
- process control (monitoring and documentation to ensure that the design is successfully implemented);
- inspection and testing (receiving, in-process and final inspection);
- inspection, measuring and test equipment (ownership status, control, calibration and maintenance of all equipment used);
- inspection and test status (identification of the condition of product);
- control of nonconforming product (definition and documentation of the responsibility for product review and the authority for the disposition of nonconforming products);
- corrective action (detection of any suspected nonconformance and responsibility for the undertaking of the correction of the deficiency and the prevention of its recurrence);
- handling, storage, packaging and delivery (protection of the products from damage and deterioration);
- quality records (records used to demonstrate the achievement of the required quality and verify the effective and economical operation of the quality system);
- internal quality audits (to ensure that the quality system is working according to plan and to provide opportunities for improvement);
- training (very general requirements, practically leaving the decision of appropriate training to the organization involved);
- servicing (establishment of procedures for service after delivery, and verification that the service meets the contract's specified requirements);
- statistical techniques (for the improvement and/or control of the quality of the processes or the product).

2.4.4 Quality Function Deployment (QFD)

As reported in [Dean97], 'the concept of QFD was introduced in Japan by Yoji Akao in 1966. By 1972 the power of the approach had been demonstrated and in 1978 the first book on the subject was published in Japanese. Unfortunately, for those of us who do not read Japanese, it was not translated into English until 1994 [MiAk94]'.
QFD [Dean97], [HaCl88], [Akao90], [Day93], [MiAk94], [BBBB95] is a team-based management tool, 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'. An example of a House of Quality is shown in Figure 2.3.

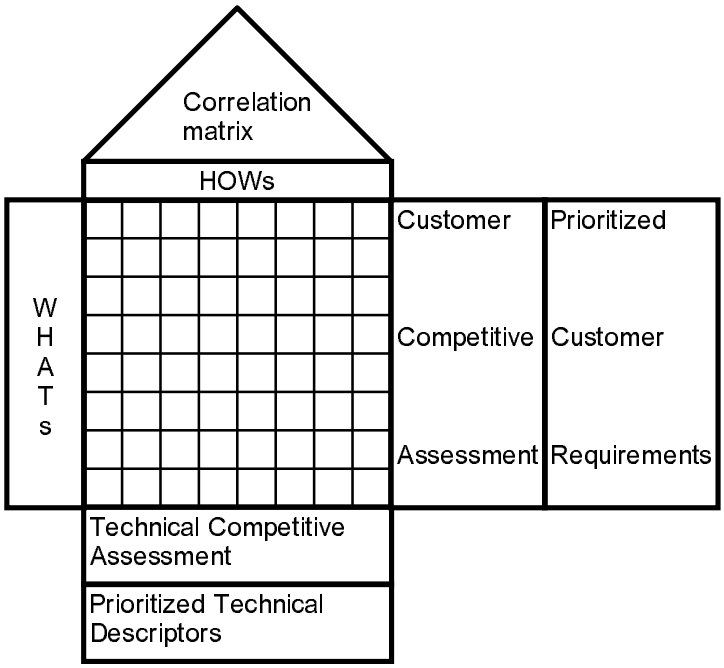


Figure 2.3. QFD, the House of Quality.

The basic methodology for building a house of quality is composed of several steps. The first step involves the modeling of the needs or expectations of the customers. This step produces a list of goals-objectives, often referred as the 'WHATs' [BBBB95]. It is very possible that a customer requirement is expressed rather generally and vaguely; so the initial list is refined and a second, more detailed, list of customer requirements is produced. If it is necessary, this procedure is repeated once more.
The second step involves the suggestion of technical solutions (the 'HOWs') which can deal with the problem that was specified at the previous step. This process can also be iterated, as it is rather hard to express detailed technical solutions at once.
The third step involves the combination of the results of the two previous steps. The basic aim of this process is to answer the question 'how are customer requirements and possible technical solutions interrelated?'. To achieve that, the interior of the house of quality, called the relationship matrix, is filled in. Symbols are usually used, determining how strong a relationship is. It is also important to note that both positive and negative relationships exist.
The fourth step is also very crucial. It involves the identification of any interrelationships between the technical factors. The roof of the house, also known as the 'correlation matrix' is also filled in. All the conflicting points represent tradeoffs in the overall technical solution.
Next, competitive assessments must be made. The competitive assessments are a pair of weighted tables which depict analytically, how competitive products compare with the organization products. The competitive assessment is separated in two categories: customer assessment and technical assessment.
The following step is the definition of the prioritized customer requirements. The prioritized customer requirements are a block of columns corresponding to each customer requirement and contain

columns for importance rating, target value, scale-up factor, sales point and absolute weight for each customer requirement.
Finally, prioritized technical descriptors are also defined. Each of the proposed technical solutions is annotated with degree of technical difficulty, target value, absolute and relative weights.
Previous research has exploited the usefulness of the QFD approach. Formal reasoning in such a structure has been investigated in works about the handling of non-functional requirements in software engineering, e.g. [MyCN92]. Visual tools have shown a potential for negotiation support under multiple quality criteria [GeJJ97].

2.4.5 Taguchi's Quality Engineering

Taguchi defined quality as the loss imparted to society from the time a product is shipped [BBBB95]. Dr. Taguchi's loss function aims in the minimization of the total loss by combining cost, target and variation into a single metric. Furthermore, signal-to-noise ratio measurements can be used in order to be used as a proactive equivalent to the reactive loss function. The analyst tests the response of the system by determining signal and noise factors and measuring the response of the system.
Orthogonal arrays is a method of conducting experiments. The goal of using an orthogonal array is the detection of degrees of freedom between factors influencing a process. The array is composed of columns, representing the possible factors and rows, representing the treatment conditions and the analyst is experiment by changing the number of rows and columns in order to find out the correct number of the degrees of freedom of a process. The decision upon this matter is obviously related to the quality of a process, as it can determine the key factors from which the process is dependent upon.
As reported in [BBBB95], [Kack86], [ASII91], there are three stages in the development of a product: *product design*, *process design* and *production*. The results of each stage are influenced by several noise and variation factors (e.g. product deterioration, environmental variables, etc). Taguchi's techniques introduced a new way of facing the design of a product: the overall design can be viewed as the sequence of system design, parameter design and tolerance design. The system design is the traditional R & D development of a prototype. The parameter design is the selection of the optimal conditions (parameters) for the reduction of the dependency of the product from noise factors. It is suggested to start the parameter design with low-cost and inferior grade materials and gradually ameliorate the quality of the product. Finally, tolerance design is the process where the statistical tolerance around the quality target must be defined. The key notion here is that, even if parameter design used low-cost tolerance, tolerance design should tighten the tolerances and achieve a good equilibrium of cost and quality, while eliminating excessive variation of values.

2.4.6 Miscellaneous techniques

Quality systems often make use of several other techniques. *Benchmarking* 'is a systematic method, by which organizations can measure themselves against the best industry practices' [BBBB95]. Benchmarking is used to discover what are the weaknesses of a product or service and to borrow ideas, which led other organizations to excellence.
Concurrent engineering is another technique used to enhance the quality of a product or service. Concurrent engineering is a way of conducting business and structuring the processes of an organization. It involves the formulation of several teams, populated by people from several departments of an organization. The teams work simultaneously in the conceptual modeling, design and production planning of a product or service. The teams have to cooperate strongly in order to produce a quality system. In fact, the basic philosophy of concurrent engineering is the broadening of communication paths between people working on different departments of an organization.
Finally, *experimental design* is also a powerful technique for quality design and implementation. The experimental design approach requires the introduction of intentional changes in the system in order to observe their effects. The difference between experimental design and SPC is that SPC is a passive statistical method, assuming that the right variable is controlled and the right target has been set, with a right tolerance; whereas experimental design is an active statistical method, where everything is extracted from the performed tests.

2.5 Commercial tools concerning data warehouse quality

As the data warehouse market is rapidly evolving in the last few years, all the major database companies (IBM, Oracle, Informix, Sybase, Red Brick Systems, Software AG, Microsoft, Tandem) have already created tools and products in order to support data warehouse solutions. A large number of smaller companies have managed to develop and market specialized tools for data warehouses. Most (if not all) of those tools affect in some way the quality of the resulting data warehouse, and several of the tools deal with quality explicitly. An up-to-date list of products can be found in [Gree97].
Obviously quality of the data stored in the data warehouse depends on the quality of data used to load/update the data warehouse. Incorrect information stored at the data sources may be propagated in the data warehouse. Still, the data is inserted in the data warehouse through a load/update process which may (or may not) affect the quality of the inserted data. The process must correctly integrate the data sources and filter out all data that violate the constraints defined in the data warehouse. The process may also be used to further check the correctness of source data and improve their quality. In general, the most common examples of dirty data [Hurw97, JLVV99] are:
- Format differences.
- Information hidden in free-form text.
- Violation of integrity rules (e.g., in the case of redundancy).
- Missing values.
- Schema differences.

The tools that may be used to extract/transform/clean the source data or to measure/control the quality of the inserted data can be grouped in the following categories [Orli97]:

- Data Extraction.
- Data Transformation.
- Data Migration.
- Data Cleaning and Scrubbing.
- Data Quality Analysis.

SQL*Loader module of Oracle [OrCo96a] which can extract, transform and filter data from various data sources (including flat files and relational DBMS's). A powerful feature of SQL*Loader is its reporting capabilities with respect to the inserted, ignored and problematic data.
Carleton's Pure Integrate [PuIn98] (formerly known as Enterprise/Integrator), ETI Data Cleanse [ETI98], EDD Data Cleanser tool [EDD98] and Integrity [InVa98] from Vality can be used for the application of rules that govern data cleaning, typical integration tasks (esp. postal address integration), etc.

3. THE QUALITY METAMODEL

After presenting the metadata framework and the architecture metamodel, as well as the related work in the field of quality in information systems, we proceed to discuss the quality metamodel that we propose. As we saw in the presentation of quality management techniques, there are several methods for the engineering of quality in an information system. However, although these techniques certainly have a useful role in rough quality planning and cross-criteria decision making, using any of them alone could possibly 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. 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 [LSTV99,ThLS99,ThSe97,ThSe99] and (b) the tuning of the refreshment process based on specific quality factors (see also Chapter 4 and [ThBo99]).
Thus, we had to look for a new approach to the problem of quality management of the data warehouse. The formal requirements for the quality metamodel can be summarized as follows:
- The categorizations of the metadata framework that we have adopted should be respected. Thus, all three perspectives and layers of instantiation should be clearly present in the quality metamodel. This would increase both the usability of the approach and the re-usability of any achieved solutions.

- Moreover, the exploitation of quality should be done through the use of a repository, enabling thus the centralized collection, storage and querying of all relevant meta-information in a consistent fashion.
- Finally, the metadata repository must allow the exploitation of the involved quality factors, through the use of well-established automated techniques and algorithms.

3.1 The Goal-Question-Metric (GQM) approach

A solution to this problem builds on the widely used Goal-Question-Metric (GQM) approach to software quality management [OiBa92, BaCR94]. The idea of GQM is that quality goals can usually not be assessed directly, but their meaning is circumscribed by questions that need to be answered when evaluating the quality. Such questions again can usually not be answered directly but rely on metrics applied to either the product or process in question; specific techniques and algorithms are then applied to derive the answer of a question from the measurements. In this subsection we provide a quick review of the GQM model, based primarily on [BaCR94].

A GQM Model has 3 levels:

- *Conceptual* (Goal);
- *Operational* (Question);
- *Quantitative* (Metric).

A **goal** is defined for an *object*, for a variety of *reasons*, with respect to various *models of quality*, from *various points of view*, relative to a *particular environment*. Objects of measurement are:

- *Products* (e.g., specifications, designs, programs, test suites).
- *Processes* (e.g., specifying, designing, testing, interviewing).
- *Resources* (e.g., personnel, hardware, software, office space).

In [BaCR94] it is also mentioned that a goal can be refined into several other subgoals.

A set of **questions** is used to characterize the way the assessment/achievement of a specific goal is going to be performed based on some characterizing model. Questions try to *characterize the object of measurement* (product, process, resource) with respect to a *selected quality issue* and to determine its quality from the selected viewpoint.

Metrics is a set of data is associated with every question in order to answer it in a quantitative way. The data can be:

- *Objective* (E.g., number of versions of a document, staff hours spent on a task, size of a program).
- *Subjective* (e.g., readability of a text, level of user satisfaction).

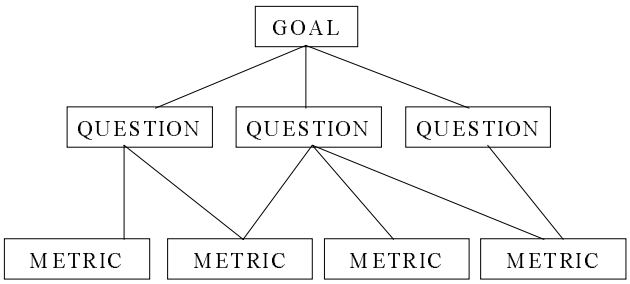


Figure 2.4. Goals, Questions and Metrics

The GQM process consists of the following steps:

- *Identify a set of quality and/or productivity goals*, at corporate, division or project level; e.g., customer satisfaction, on-time delivery, improved performance.
- From those goals and based upon models of the object of measurement, we *derive questions that define those goals* as completely as possible.
- *Specify the measures* that need to be collected in order to answer those questions.
- *Develop data collection mechanisms*, including validation and analysis mechanisms.

A goal has three coordinates:

- *Issue* (e.g. Timeliness);
- *Object* (e.g. Change request processing);
- *Viewpoint* (e.g. project manager);

and a purpose:

- *Purpose* (e.g. improve).

The *Issue* and the *purpose* of the Goal are obtained from the *policy and the strategy of the organization* (e.g. by analyzing corporate policy statements, strategic plans and, more important, interviewing relevant subjects in the organization). The *object* coordinate of the Goal is obtained from a description of the process and products of the organization, *by specifying process and product models*, at the best possible level of formality. The *viewpoint* coordinate of the Goal is obtained from the *model of the organization*.

There are 3 types of questions:
Group 1. How can we characterize the object (product, process, or resource) with respect to the overall goal of the specific GQM model? E.g.,

- *What is the current change request processing speed?*
- *Is the change request process actually performed?*

Group 2. How can we characterize the attributes of the object that are relevant with respect to the issue of the specific GQM model? E.g.,

- *What is the deviation of the actual change request processing time from the estimated one?*
- *Is the performance of the process improving?*

Group 3. How do we evaluate the characteristics of the object that are relevant with respect to the issue of the specific GQM model? E.g.,

- *Is the current performance satisfactory from the viewpoint of the project manager?*
- *Is the performance visibly improving?*

The development of metrics is a customized process! Suggestions for this development include:

- try to *maximize the use of existing data sources* if they are available and reliable;
- apply *objective* measures to more *mature* measurement objects, and more *subjective* evaluations to *informal or unstable* objects.
- *the measures we define must help us in evaluating not only the object of measurement but also the reliability of the model used to evaluate it.*

In [OiBa92] Oivo and Basili mention the fact that GQM models are constructed through 2 parallel (and iterative) processes: (a) a generic process, which involves the creation, tailoring and reuse of GQM template objects to build a GQM model base and (b) a specialized process, which involves the rule-based construction and instantiation of the GQM model base into a collection of operational GQMs. In the same paper, the authors mention that the process of construction is somehow automated: a given goal implies a set of questions and a given question implies a set of metrics based on the templates. The implication is produced through the "firing" of several rules.

Goal	Purpose	Improve
	Issue	the timeliness of
	Object (process)	change request processing
	Viewpoint	from the project manager's viewpoint
Question	Q1	What is the current change request processing speed?
Metrics	M1	Average cycle time
Metrics	M2	Standard deviation
Metrics	M3	% cases outside of the upper limit
Question	Q2	Is the (documented) change request process actually performed?
Metrics	M4	Subjective rating by the project manager
Metrics	M5	% of exceptions identified during reviews
Question	Q3	What is the deviation of the actual change request processing time from the estimated one?
Metrics	M6	$\frac{\text{Current average cycle time} - \text{Estimated average cycle time}}{\text{Current average cycle time}}$
Metrics	M7	Subjective evaluation by the project manager
Question	Q4	Is the performance of the process improving?
Metrics	M8	$\frac{\text{Current average cycle time}}{\text{Baseline average cycle time}}$
Question	Q5	Is the current performance satisfactory from the viewpoint of the project manager?
Metrics	M4	Subjective evaluation by the project manager
Question	Q6	Is the performance visibly improving?
Metrics	M8	$\frac{\text{Current average cycle time}}{\text{Baseline average cycle time}}$

Figure 2.5. An Example of the GQM process [BaCR94].

In Figure 2.5, we present a brief example of how the GQM methodology can be applied for the fulfillment of a certain quality goal [BaCR94]. In the first column, the type (Goal, Question or Metric) of the respective step is depicted; the second column holds an ID for its identification and the third column holds the description of each step.

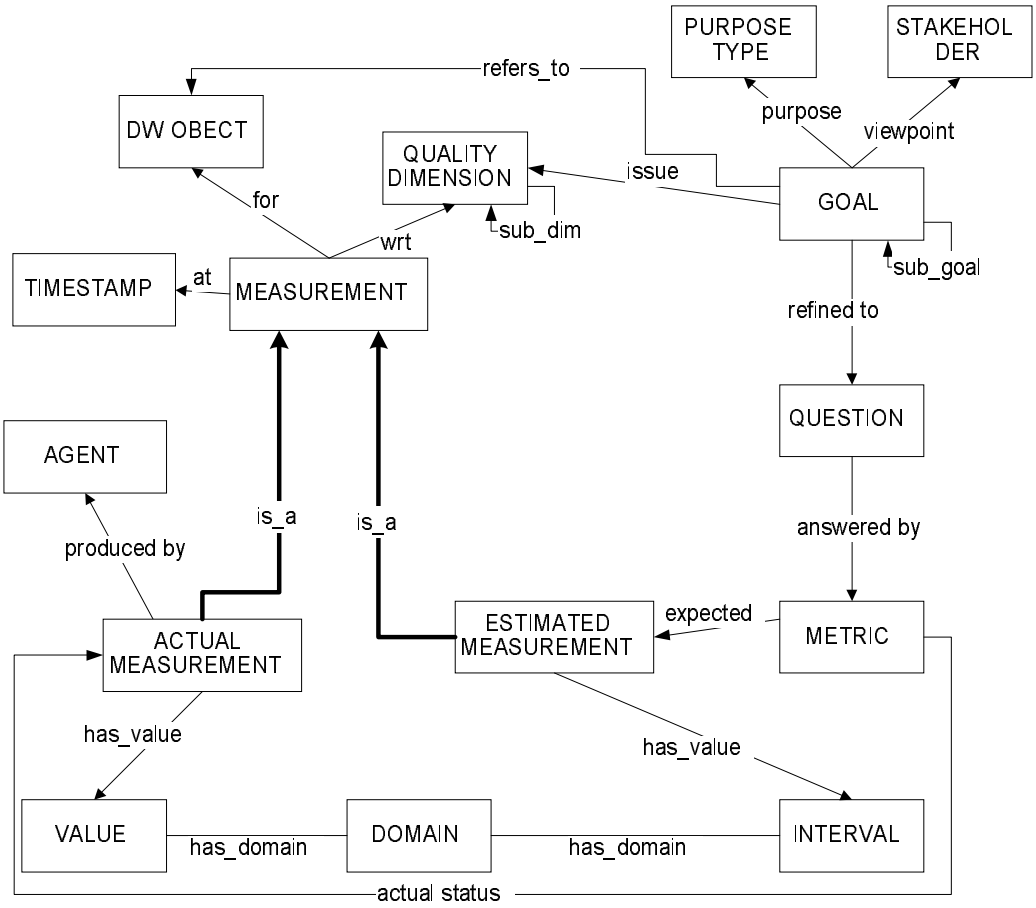


Figure 2.6. The proposed quality metamodel

3.2 The proposed quality metamodel

Our quality metamodel coherently supplements the architecture metamodel. We adopt again three layers of instantiation. At the metamodel layer, a generic framework that follows GQM is given, extended with associations applicable to any data warehouse environment. At the metadata layer, specific quality goals concerning each particular data warehouse are given. Finally, concrete values are the traces of measurement in the real world. Needless to say that the quality metamodel is part of the metadata repository of the data warehouse, in order to be consistent with the overall approach of data warehouse management. The definition of the metamodel has been influenced by -or better: developed in parallel with- the proposals of [JeQJ98] and [LeBo98].

3.2.1 The Metamodel layer: the framework for quality management

A *quality goal* is a project where a stakeholder has to manage (e.g., evaluate or improve) the quality of the data warehouse, or a part of it. This roughly expresses natural language requirements like 'improve the availability of source s1 until the end of the month in the viewpoint of the DW administrator'. *Quality dimensions* (e.g. 'availability') are used as the vocabulary to define abstractly different aspects of quality, as the stakeholder perceives it. Of course, each stakeholder might have a different vocabulary and different preferences in the quality dimensions. Moreover, a quality goal is *operationally* defined by a set of *quality questions.* Each question is dispatched to concrete *quality metrics* (or *quality factors*) which are groupings for concrete measurements of quality. A metric, thus, is defined over a specific data warehouse object and incorporates expected / acceptable values, actually measured values (*measurements*), methodology employed, timestamps etc.

Formally, the definitions of the involved entities are as follows:

GOAL: is a project where a stakeholder has to manage (e.g., evaluate or improve) the quality of the data warehouse or a part of it. A quality goal can be decomposed into subgoals, which are recursively described as quality goals. A GOAL is defined by the following assertions:
1. it refers to an DW_OBJECT,
2. it has a purpose taken from the PURPOSE TYPE possible instances,
3. it has an issue taken among the intances of QUALITY DIMENSION entity,
4. it is defined with respect to a specific viewpoint of a given STAKEHOLDER.

A GOAL is refined to several QUALITY QUESTIONS.

PURPOSE-TYPE: is any action to take in order to reach a certain quality goal (improve, optimize, enforce, etc.).

STAKEHOLDER: a person who is in some way related to the data warehouse project (administrator, analyst, designer, etc.). A listing of data warehouse stakeholders and of the templates for their quality goals can be found in Section 4.1 and in [JJQV99, JLVV99].

QUALITY DIMENSION: a subjective, high-level, user-oriented characterization of a given object. Actually, the dimensions serve as the stakeholder's vocabulary for different aspects of quality. A quality dimension could be decomposed into sub-dimensions, which are in turn also considered as dimensions, too.

DW-OBJECT: is any object of the data warehouse framework, of any abstraction perspective (conceptual, logical, physical) and any level (client, enterprise, source). We use DW-OBJECT to represent processes, too (although they will be presented extensively in the next chapter). In the sequel we will also show how this fits naturally with the different instantiation layers of the metadata framework.

QUALITY QUESTION: is an effort to characterize the way the assessment/achievement of a specific goal is going to be performed based on some characterizing model (GQM definition). A question is concretely answered (i.e., evaluated and assessed) by a set of QUALITY METRICS.

QUALITY METRIC (or QUALITY FACTOR): a characterization of the action performed, made in order to achieve a set of data (objective or subjective) which answer a question in a quantitative way. (rephrasing GQM). For the moment we consider only calculated and expected values; yet other data can also be of interest. A METRIC instant is related to a set of values (MEASUREMENTs). Also, a METRIC is characterized by an extra attribute, namely DESCRIPTION (not shown in the Figure 2.6 for reasons of efficiency) characterizing the default algorithm for its MEASUREMENT's of the METRIC (any deviation from this default algorithm are covered by the description of the executing agent).

MEASUREMENT: is a datum used to evaluate the quality goal. A MEASUREMENT is done for a specific DW-OBJECT, with respect to a specific QUALITY METRIC, at a specific point in time (TIMESTAMP) (since we need present and previous values).

ACTUAL MEASUREMENT: IS_A MEASUREMENT representing the fact that some quantification of the quality of a DW-OBJECT has been performed. This is done using a certain AGENT (i.e., software program in the architecture model) for the computation and producing a specific VALUE (which is the final quantification of the question made to a answer a GOAL).

EXPECTED-MEASUREMENT: IS_A MEASUREMENT defining the INTERVAL of allowed values of the ACTUAL MEASUREMENTs. The INTERVAL must have the same DOMAIN with the produced VALUES.

VALUE: the value of a specific measurement.

INTERVAL: a range of values for a metric (actually, here, the acceptable range).

DOMAIN: the domain of expected and achieved values.

AGENT: a software program of the architecture model. Each agent is characterized by a DESCRIPTION for its functionality.

3.2.2 Specialisation at the metamodel layer: the patterns for quality management

The abstract framework for quality management, presented in the previous subsection is enriched with "patterns", or "templates" for specific cases of quality management in the data warehouse environment. Take for example, the case where we want to improve the schema minimality for *any* data warehouse schema. This also means that we are not interested in any particular stakeholder. A set of template questions is also provided, which are mapped to generic quality factors (such as 'Number of Redundant Views'). A query at the repository can produce the involved objects.

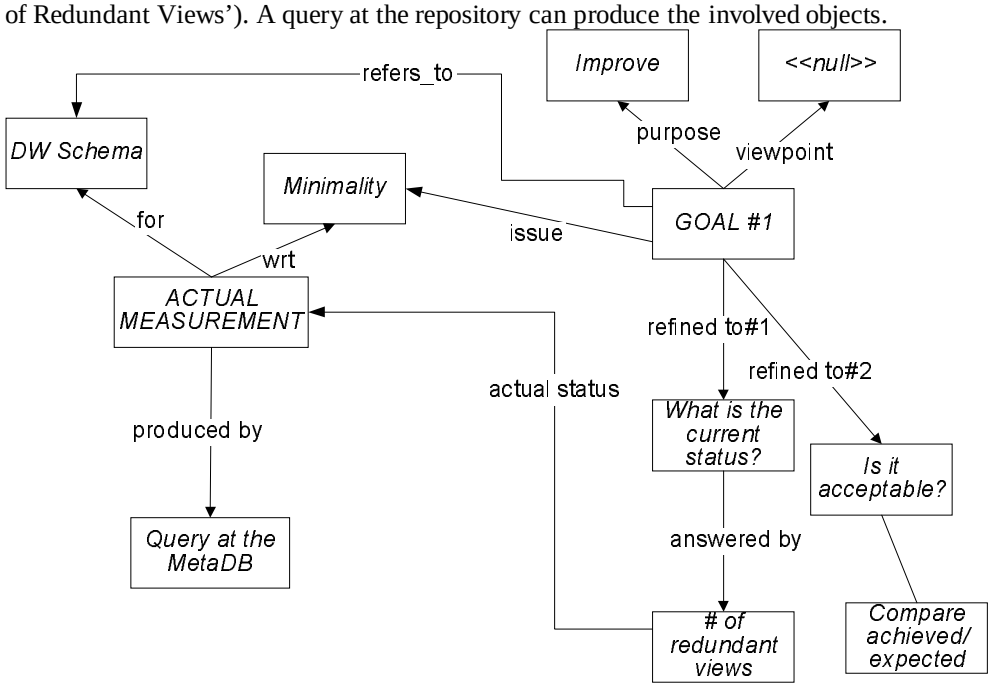


Figure 2.7 Template Quality Goals and Factors at the Metamodel layer.

3.2.3 Instantiation of the quality framework at the metadata layer.

The generic templates for quality management, are of course, instantiated to specific cases for each particular data warehouse. Suppose the instantiation of Figure 2.8 for our motivating example. One can observe that

(a) The interplaying objects are refined from generic metamodel entities (e.g., DW-SCHEMA) to specific warehouse dependent objects (e.g., K.'s DW-SCHEMA).
(b) The goals, questions, factors and measurements are refined too, accordingly.
(c) The dimension and purpose of the quality goal remain the same, since they provide rather documentation than reasoning information to the repository.
(d) The stakeholder is instantiated to a specific instance (Mr. K.) for the particular quality goal under consideration.

3.2.4 Adherence to the metadata framework

It must be evident, by now, that the proposed quality metamodel adheres by the generic metadata framework that was introduced in Section 1. As far as the issue of *perspective dicrimination* is concerned, we believe that is clear that all the perspectives of the metadata framework play a very important role in the quality metamodel.

- *Conceptual quality goals*. The quality goals are expressed directly by the stakeholders, in their own high-level, subjective vocabulary, involving abstract requirements.
- *Physical quality measurements.* The measurements are performed by specific algorithms / tools, over specific data warehouse objects, at specific points of time.
- *Logical quality questions*. The questions are, by definition, the intermediate stage between the high-level, abstract user requirements and the concrete, physical measurements and actions.

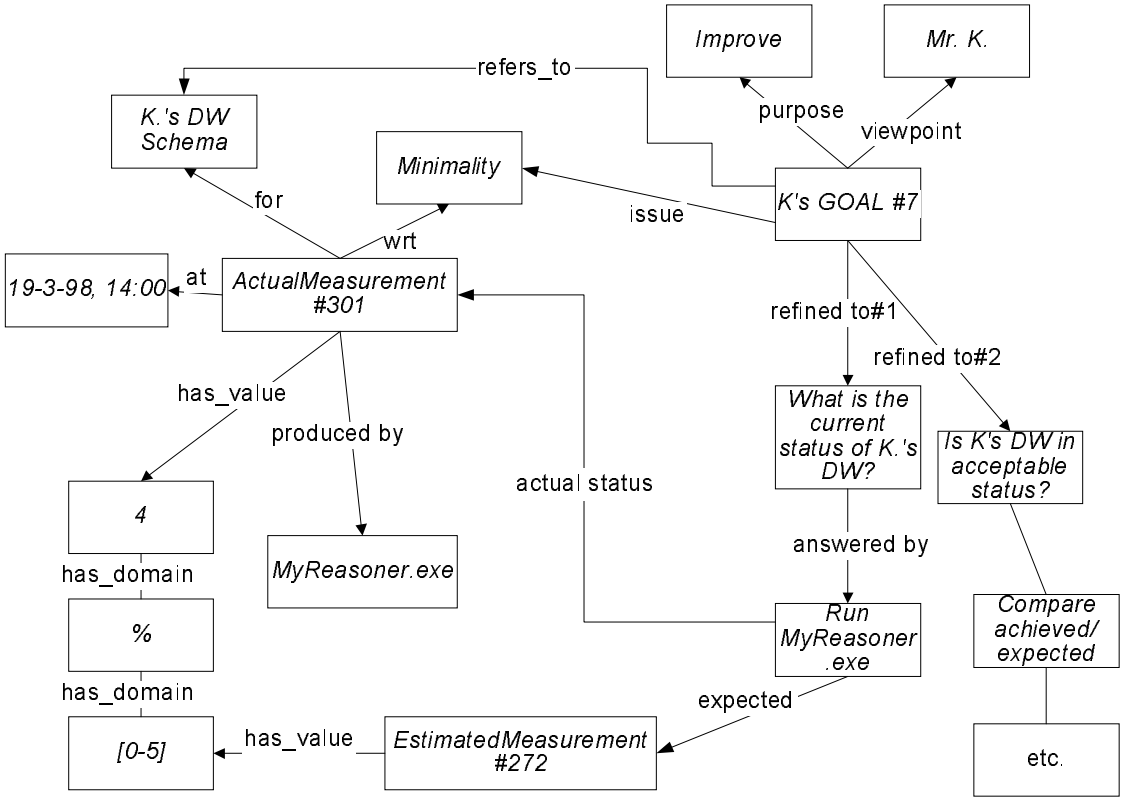


Figure 2.8 Quality Goals and Factors at the Metadata layer.

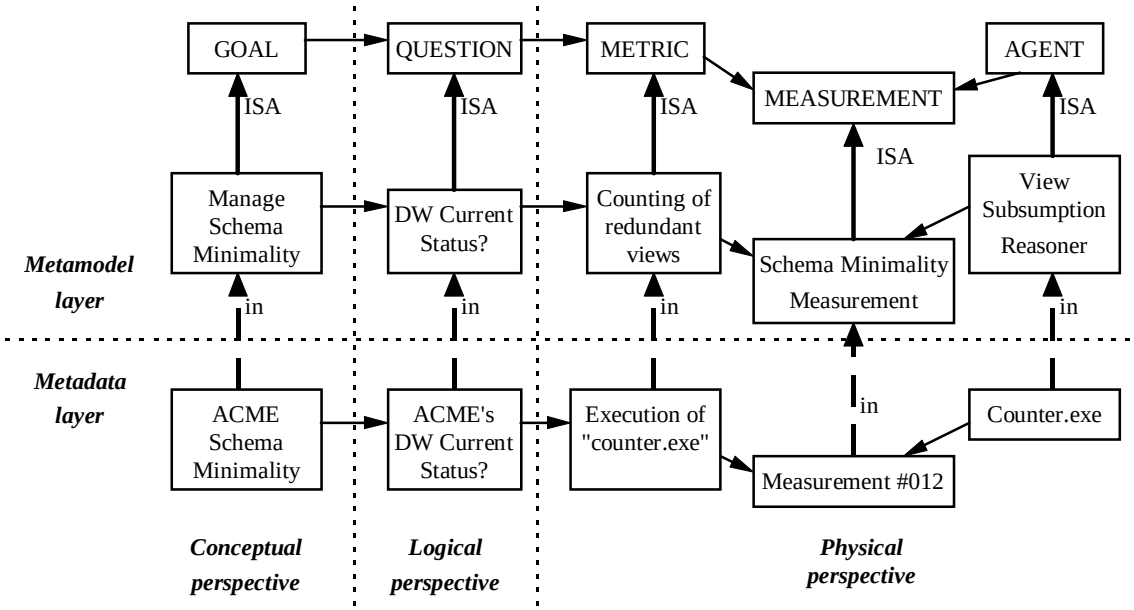


Figure 2.9. Perspectives and layers of instantiation for the quality metamodel.

At the same time, as far as the *different layers of instantiation* are concerned, the quality metamodel is not instantiated directly with concrete quality factors and goals, but also with patterns for quality factors and goals. 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 and can tune accordingly the loading processes they construct.

Based on the metamodel of data warehouse architecture, we have developed a set of quality factor templates, which can be used as a initial set for data warehouse quality management [QJJ*98]. The following section gives an intuition of some of them as far as (a) quality goals and (b) quality metrics and their related processes / objects are concerned.

4. PATTERNS FOR THE DATA WAREHOUSE QUALITY MANAGEMENT

In this section, we will detail the extension of the quality metamodel with specific templates for quality goals and metrics. In subsection 4.1 we will provide the linkage of quality goals to data warehouse stakeholder and, most importantly, in subsection 4.2 we will proceed to show how data warehouse objects and processes interrelate with quality metrics.

4.1 Templates for Quality Goals

In [JJQV99] 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, designers;*
- *Executive managers;*

The *Decision Maker* is the final user of the data warehouse. He usually uses reporting and/or OLAP tools to get answers to his questions.

Stakeholder: *Decision Maker* **Purpose**: *evaluate, understand, improve*	
Issue	**Object**
overall quality	*stored data*, in the degree that they represent real world facts.
timeliness	*stored data* are useless when out-of-date
ease of querying	*data warehouse, OLAP tool*

The *Data Warehouse Administrator* is concerned with keeping the data warehouse properly operating.

Stakeholder: *Data Warehouse Administrator* **Purpose**: *evaluate, understand, improve*	
Issue	**Object**
error reporting	*all the data warehouse processes*
metadata accessibility	*metadata*
timeliness	*stored data*

The *Data Warehouse Designer* is involved in a continuous process, the data warehouse design, which is usually done incrementally.

Stakeholder: *Data Warehouse Designer* **Purpose**: *evaluate, understand, improve*	
Issue	**Object**
schema quality	*existing and derived schemata*, making most use of the information directory.
metadata quality	*metadata*
software quality	*software packages* he considers purchasing - needs *software evaluation standards* to test them.

The *Programmers of Data Warehouse Components* are the people who develop the actual data warehouse applications.

Stakeholder: *Programmers of Data Warehouse Components* **Purpose**: evaluate, understand, improve	
Issue	**Object**
implementation quality	*the data warehouse components* they produce - they need *software implementation standards* to test them
overall SW quality	all *the data warehouse components*, produced or purchased - they need *software evaluation standards* to test them.
metadata accessibility	*Metadata*

The *Source Data Users/Administrators/Designers* are affected from the data warehouse in the sense that they could both benefit (by learning more for the quality of their data) and be disturbed from its existence (due to loss of political power and extra work and technical load), at the same time.

Stakeholder: *Legacy System Stakeholders* **Purpose**: *evaluate, understand, improve*	
Issue	**Object**
reporting (feedback) on the quality	*source data*
system availability	*source operational system*

Finally, the *Executive Manager* of an organization using a data warehouse is mostly concerned with the financial information regarding the data warehouse.

Stakeholder: *Executive Manager* **Purpose**: *evaluate, understand, improve*	
Issue	**Object**
value	*the data warehouse (cost benefit, return of investment* etc.*)*

Furthermore, there are also people who are in direct contact with a data warehouse environment or they can be affected by it:

- *People affected by the data warehouse related or supported decisions*
- *Vendors of data warehouse components*

People can be easily affected from the existence of an information system and consequently of a data warehouse in both good and bad ways, e.g. they can get information more easily, more quickly and more credibly; on the other hand they can possibly loose their job because of these aspects of an

information system. Vendors of data warehouse components can also be affected by the way their parts are constructed, used and functioning, since these factors can affect sales. Although we do not model they way these people are affected from the warehouse, or quality aspects interesting to them, we would like to mention them for reasons of scientific completeness and social sensitivity.

4.2 Templates for quality metrics

In this paragraph we present the various data warehouse quality metrics and their relevance to the methods for quality measurements as well as specific mathematical formulae. We categorize the quality metrics according to the *processes* that take place during the data warehouse operation. Still, their relevance to data warehouse objects is also discussed. We consider four basic processes during the operation of a data warehouse. Furthermore, we incorporate the quality of the stored data in the quality model. Consequently we have:

- *design and administration*
- *software implementation and/or evaluation*
- *data loading*
- *data usage*
- *data quality*

In the rest of this subsection we will detail the quality factors related to the aforementioned processes of a data warehouse. For each one of them we will present the way they are refined and link them to the appropriate data warehouse architecture components.

4.2.1 Design and Administration Quality

The *schema quality* refers to the ability of a schema to represent adequately and efficiently the information. The *correctness* factor is concerned with the proper comprehension of the entities of the real-world entities, the schemata of the sources as well as with the user needs. The *completeness* factor is concerned with the preservation of all the crucial knowledge for the data warehouse schema. The *minimality* factor describes the degree up to which undesired redundancy is avoided during the source integration process. The *traceability* factor is concerned with the fact that all kinds of requirements and decisions of users, designers, administrators and managers should be traceable in the data warehouse schema. The *interpretability* factor ensures that all components of the data warehouse are well described, in order to be administered easily. The *metadata_evolution* is concerned with the way the schema evolves during the data warehouse operation.

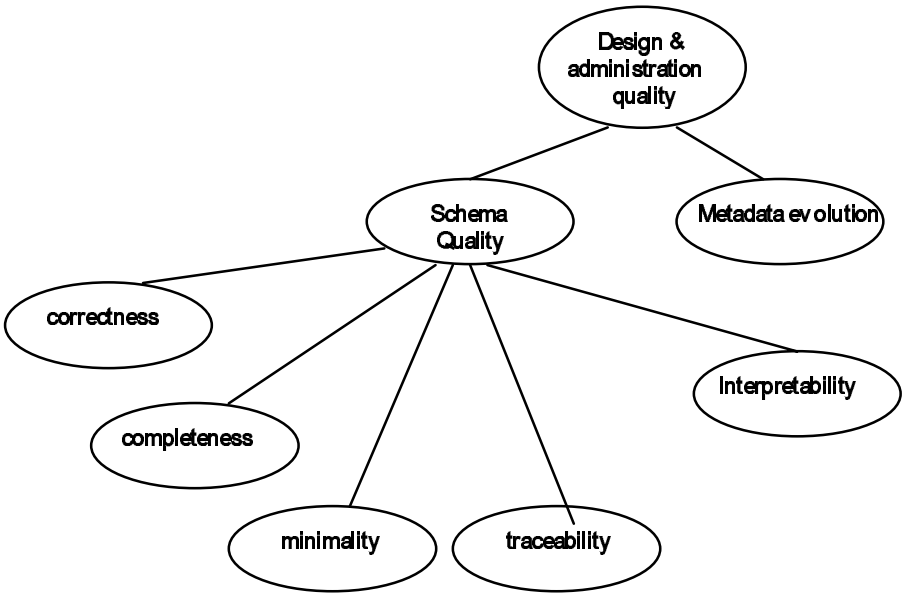


Figure 2.10 Design and Administration Quality Factors

As it is obvious from the above, the design and administration quality factors are related to all kinds of schemata in the architecture model. Note that the quality of models and schemata does not apply

only for the data warehouse, the sources and the clients but for the information directory itself. Factors like interpretability and quality of metadata evolution deal mostly with the quality of the information directory and less with the data warehouse itself. In Figure 2.10 the overall hierarchy of design and administration quality is depicted.
The design and administration major quality factors are schema quality and metadata evolution quality, which are in terms analyzed to other factors. The correctness, completeness, minimality and traceability factors should be measured after the source integration process in order to guarantee the absence of errors. The interpretability factor should be fairly documented in order to help the administrator know the system and data architecture, the relationship models, schemata and physical storage of the data and the processes of the data warehouse. The metadata evolution factor is used to track down the way the schema and models of the data warehouse evolve.

Factor	**Methods of measurement**	**Formulae**
Schema quality		
Correctness	final inspection of data warehouse schema for each entity and its corresponding ones in the sources	number of errors in the mapping of the entities
Completeness	final inspection of data warehouse schema for useful entities in the sources, not represented in the data warehouse schema	number of useful entities, not present in the data warehouse
Minimality	final inspection of data warehouse schema for undesired redundant information	number of undesired entities in the data warehouse
Traceability	final inspection of data warehouse schema for inability to cover user requirements	number of requirements not covered
Interpretability	Inspection / reasoning in the:	
	physical part of the architecture (e.g. location of machines and software in the data warehouse)	number of undocumented machines/pieces of software
	logical part of the architecture (e.g. data layout for legacy systems and external data, table description for relational databases, primary and foreign keys, aliases, defaults, domains, explanation of coded values, etc.)	number of pieces of information not fully described
	conceptual part of the architecture (e.g. ER diagram)	number of undocumented pieces of information
	mapping of conceptual to logical and from logical to physical entities	number of undocumented mappings between conceptual, logical and physical entities
Metadata evolution quality	metadata evolution (versioning/timestamping of the metadata)	number of not documented changes in the metadata

4.2.2 Software Implementation/Evaluation Quality

Software implementation and evaluation are not tasks with specific data warehouse characteristics. We are not actually going to propose a new model for this task, rather than adopt the ISO 9126 standard [ISO97]. Yet, for reasons of completeness we will present here the quality factors of ISO 9126, enhanced with just one factor, namely *implementation efficiency*.
Software implementation is a hard and time-consuming task. The quality factors defined in the previous section can be used as guidelines for ensuring quality of the produced software. The best measurement methods we can propose in this context -apart from standard software developing techniques- is the testing of each module of software developed, with respect to the relative quality factors. The measurement of the quality of each module can be done, in most cases, by tracking down and enumerating the number of occasions where an attribute is not fully covered. As for software evaluation, standard benchmarks have been developed for this cause, covering both the technical and the measurement part.

Factor	**Methods of measurement**	**Formulae**
Functionality		

Suitability	testing of functions with regard to specified tasks	number of functions not appropriate for specified tasks
Accuracy	testing of software with regard to agreed results or effects	number of modules not providing the right or agreed results or effects
Interoperability	testing of software on its ability to interact with specified systems	number of modules unable to interact with specified systems
Compliance	testing of software on its compliance to application related standards or conventions or regulations in laws and similar prescriptions	number of modules not compliant with application related standards or conventions or regulations in laws and similar prescriptions
Security	testing of software on its ability to prevent unauthorized access, whether accidental or deliberate, to programs and data	number of modules unable to prevent unauthorized access, whether accidental or deliberate, to programs and data
Reliability		
Maturity	testing of software on the frequency of failure by faults in it.	frequency of failure by faults in the software
Fault tolerance	testing of software on its ability to maintain a specified level of performance in cases of software faults or of infringement of its specified interface	number of occasions where the software was unable to maintain a specified level of performance (in cases of software faults or of infringement of its specified interface)
Recoverability	testing of software on the capability to re-establish its level of performance and recover the data affected in the case of a failure and on the time and effort needed for it	number of occasions where the software was unable to re-establish its level of performance and recover the data affected in the case of a failure. The time and effort needed for this reestablishment
Usability		
Understandability	documentation of the users' opinion on the effort for recognizing the logical concept and the applicability of the software	percentage of acceptance by the users
Learnability	documentation of the users' opinion on the effort for learning the application of the software	percentage of acceptance by the users
Operability	documentation of the users' opinion on the effort for operation and operation control of the software	percentage of acceptance by the users
Software Efficiency		
Time behavior	testing of software on response and processing times and on throughput rates in performing its function	response times, processing times and throughput rates
Resource Behavior	testing of software on the amount of resources used and the duration of such use in performing its function	amount of resources used and the duration of such use in the performance of the function of the software
Maintainability		
Analyzability	documentation of the developers' opinion and the man-hours spent on the effort needed for diagnosis of deficiencies or causes of failures, or for identifications of	man-hours needed for diagnosis of deficiencies or causes of failures, or for identifications of parts to be modified

	parts to be modified	
Changeability	documentation of the developers' opinion and the man-hours spent on the effort needed for modification, fault removal or for environmental change	man-hours needed for modification, fault removal or for environmental change and percentage of acceptance from the developers
Stability	documentation of developers' opinion on the risk of unexpected effect of modifications	percentage of acceptance from the developers
Testability	documentation of developers' opinion and the man-hours spent on the effort needed for validating the modified software	man-hours needed for validating the modified software
Portability		
Adaptability	testing of software on the opportunity of its adaptation to different specified environments without applying other actions or means than those provided for this purpose for the software considered	number of occasions where the software failed to adapt to different specified environments without applying other actions or means than those provided for this purpose for the software considered
Installability	testing of software on the effort needed to install it in a specified environment	man-hours needed to install the software in a specified environment
Conformance	testing of software on its compliance to standards or conventions relating to portability	number of modules adhering to standards or conventions relating to portability
Replaceability	testing of software on the opportunity and effort of using it in the place of specified other software in the environment of that software	number of occasions and man-hours spent to use the software in the place of specified other software in the environment of that software
Implementation Efficiency	measurement and documentation of resource use	amount of resources used for the development of software and percentage of this recourses with respect to the originally expected ones
	measurement and documentation of completion rate	amount of completed modules with respect to predefined deadlines and milestones

4.2.3 Data Loading Quality

The loading of the data in the warehouse is a crucial task for the efficient operation of the warehouse. The usual technique is to load the data in a batch mode (e.g. at night). In the literature there is a great deal of work regarding alternative ways of updating the information of the warehouse. In [Wido95] four different modes for the propagation of updates at the sources are described: batch, immediate, periodic and on demand. The quality factors which we will present and which are related to data loading should be considered in this broader context; furthermore they are independent of the update policy which is (or can be) chosen for the data warehouse.

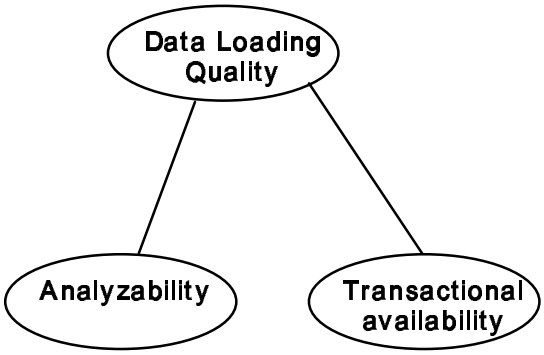


Figure 2.11 Data Loading Quality Factors

Factor	Methods of measurement	Formulae
Analyzability	testing the data warehouse processes for self-reporting in erroneous situations	number of processes which do not self-report
	testing the data warehouse processes for error-handling	number of processes which do not perform error-handling
Transactional availability	performance of statistic checks and logging	the percentage of time, when relevant information (in the sources or the warehouse) is not available due to update operations

The *analyzability* factor is concerned with the validation of each process and its interaction with the user (i.e. to report on the result of each operation, especially in the erroneous situations) and should be tested for self-reporting and error handling. *Transactional availability* should be measured in order to track down the cases where update operations, whether in the warehouse or the sources, make information unavailable. The transactional availability factor can be measured as the percentage of time the information in the warehouse or the source is available due to the absence of update processes (which write-lock the data).

4.2.4 Data Usage Quality

Since databases and -in our case- data warehouses are built in order to be queried, the most basic process of the warehouse is the usage and querying of its data. The *accessibility* factor is related to the possibility of accessing the data for querying. The *security* factor describes the authorization policy and the privileges each user has for the querying of the data. The *system availability* factor describes the percentage of time the source or data warehouse system is actually available to the users. The *transactional availability* factor, as already mentioned, describes the percentage of time the information in the warehouse or the source is available due to the absence of update processes which write-lock the data.

The *usefulness* factor describes the temporal characteristics (*timeliness*) of the data as well as the *responsiveness* of the system. The *responsiveness* is concerned with the interaction of a process with the user (e.g. a query tool which is self reporting on the time a query might take to be answered). The *currency* factor describes when the information was entered in the sources or/and the data warehouse (in a temporal database, currency should be represented by the transaction time factor). The *volatility* factor describes the time period for which the information is valid in the real world (in a temporal database, volatility should be represented by the valid time factor). The *interpretability* factor, as already mentioned, describes the extent to which the data warehouse is modeled efficiently in the information repository. The better the explanation is, the easier the queries can be posed.

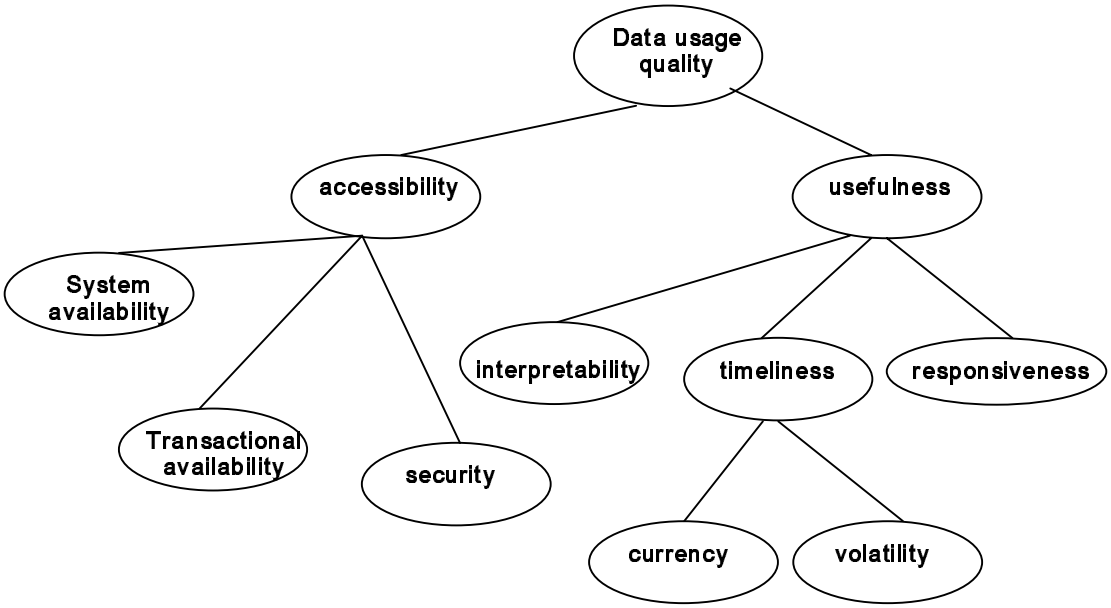


Figure 2.12 Data Usage Quality Factor

The *accessibility* of the system should be measured in order to track down the cases where failures, update operations, or other operations, whether in the warehouse or the sources, make information unavailable. The *timeliness* of data has to do with the history of the operations in the sources and the warehouse and with the relevancy of the stored information to the real world facts. Consequently, the data entry and purging processes as well as the validity of the information with respect to time should be tracked and measured for their quality. The *interpretability* has already been presented in the design and administration task and is described again for reasons of completeness.

Factor	**Methods of measurement**	**Formulae**
Accessibility		
System availability	performance of statistic checks and logging	the percentage of time, when relevant information (in the sources or the warehouse) is not available due to system failures or network, backups, etc.
Transactional availability	performance of statistic checks and logging	the percentage of time, when relevant information (in the sources or the warehouse) is not available due to update operations
Security	authorization procedures (user privileges, logging etc.) and their documentation	number of undocumented authorization procedures
Usefulness		
Responsiveness	testing of processes in order to determine whether they inform the user on their progress	number of processes that do not self-report to the user
Timeliness		
Currency	keeping track of the date when the data were entered in the sources (internal and external) and the warehouse	number of pieces of information where transaction time is not present, although needed
Volatility	keeping track of the time period during which the information is valid in the real world	number of pieces of information where valid time is not present, although needed
Interpretability	Inspection / reasoning in the:	
	physical part of the architecture (e.g. location of machines and software in the data warehouse)	number of undocumented machines/pieces of software
	logical part of the architecture (e.g. data layout for legacy systems and external data, table description for relational databases, primary and foreign keys, aliases, defaults, domains, explanation of coded values, etc.)	number of pieces of information not fully described
	conceptual part of the architecture (e.g. ER diagram)	number of undocumented pieces of information
	mapping of conceptual to logical and from logical to physical entities	number of undocumented mappings between conceptual, logical and physical entities

4.2.5 Data Quality

The quality of the data that are stored in the warehouse, is obviously not a process by itself; yet it is influenced by all the processes which take place in the warehouse environment. As already mentioned, there has been quite a lot of research on the field of data quality, in the past. We define data quality as a small subset of the factors proposed in other models. For example, in [WaRK95] our notion of data quality, in its greater part, is treated as a second level factor, namely *believability*. Yet, in our model, the rest of the factors proposed elsewhere, are treated as process quality factors.

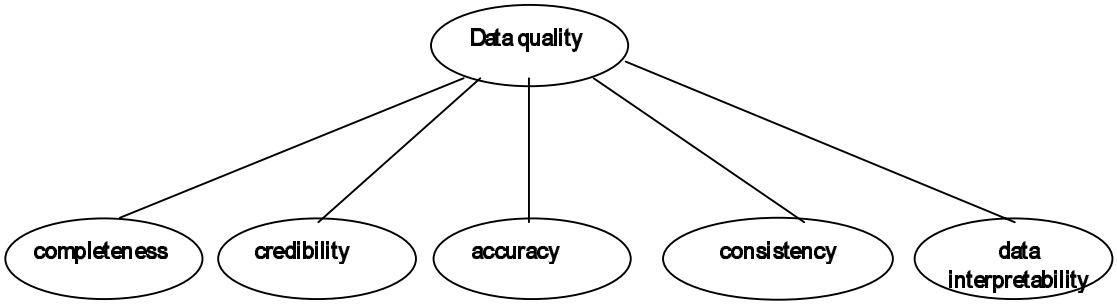


Figure 2.12 Data Quality Factors

The *completeness* factor describes the percentage of the interesting real-world information entered in the sources and/or the warehouse. For example, completeness could rate the extent to which a string describing an address did actually fit in the size of the attribute which represents the address. The *credibility* factor describes the credibility of the source that provided the information. The *accuracy* factor describes the accuracy of the data entry process which happened at the sources. The *consistency* factor describes the logical coherence of the information with respect to logical rules and constraints. The *data interpretability* factor is concerned with data description (i.e. data layout for legacy systems and external data, table description for relational databases, primary and foreign keys, aliases, defaults, domains, explanation of coded values, etc.)

Factor	**Methods of measurement**	**Formulae**
Completeness	performance of statistic checks	the percentage of stored information detected to be incomplete with respect to the real world values
Credibility	documentation of the source which provided the information	percentage of inaccurate information provided by each specific source
Accuracy	documentation of the person or machine which entered the information and performance of statistical checks	the percentage of stored information detected to be inaccurate with respect to the real world values, due to data entry reasons
Consistency	performance of statistic checks	the percentage of stored information detected to be inconsistent
Data Interpretability	data description (i.e. data layout for legacy systems and external data, table description for relational databases, primary and foreign keys, aliases, defaults, domains, explanation of coded values, etc.)	number of pieces of information not fully described

5. CONCLUSIONS

In this chapter we presented a general framework for the treatment of data warehouse metadata in a metadata repository. The framework requires the classification of metadata in at least two instantiation layers and three perspectives. The metamodel layer constitutes the schema of the metadata repository and the metadata layer the actual meta-information for a particular data warehouse. We linked this framework to a well-defined approach for the architecture of the data warehouse [JJQV99]. Then, we presented our proposal for a quality metamodel, which builds on the widely accepted Goal-Question-Metric approach for the quality management of information systems. Moreover, we enriched the generic metamodel layer with patterns concerning the linkage of (a) quality metrics to data warehouse objects and (b) of data warehouse stakeholders to template quality goals.

The exploitation of the quality model can be performed in versatile ways. It is important to note that as far as the lifecycle of the data warehouse is concerned, this usage can be done in a dual fashion. *Ex-*

post, the metadata repository can be used as a log for the management of quality. All the steps of the GQM process are, thus, traced in the metadata repository and can be re-used for further evaluation. Notice, that not only do we provide an initial specialization of the quality metamodel for common data warehouse processes, but the data warehouse stakeholder can further detail this provision with his own templates for the quality management of his specific data warehouse, in a similar fashion. Secondly, the use of the ConceptBase metadata repository can be exploited, due to its querying facilities. Following [JeQJ98] we give a small example of a query upon the metadata repository.
The following two queries detect objects with trace quality problems, i.e. measurements that are outside the expected interval, to their causes. Actually, the first query is just a view returning ranges in the Telos language, and the second is a query (or equivalently in Telos, a view) with all the objects whose measurement at a certain point of time was out of the acceptable range.

```
QualityQuery Range isA Integer
with parameter
  q: EstimatedMeasurement
constraint
  c: $ exists n/Interval, u/Integer, l/Integer
       (n upper u) and (n lower l) and (q hasValue n) and
       (this le u) and (this ge l) $
end

QualityQuery ObjectsWithQualityProblems isA DW_Object
with constraint
  c: $ exists q1 ActualMeasurement , q2/EstimatedMeasurement,
       m/Metric, t/Timestamp
       (m actual q1) and (m expescted q2) and
       (q1 timestamp t) and (q2 timestamp t) and
       not(q1 in range[q2]) and
       (q1 for this)$
end
```

Third, the quality metamodel is coherent with the generic metadata framework for data warehouses that we introduced in Section 1. Thus, every new data warehouse object can be linked to metrics and measurements for its quality gracefully, without any change to the schema of the repository. This, *ex ante* treatment of the metadata repository is complemented by a full set of *steps*, i.e., quality question, which constitute our methodology for data warehouse quality management, which will be presented in Chapter 4.

Chapter 3
Data Warehouse Processes

1. INTRODUCTION

In Chapter 2, we have presented the proposal of [JJQV99] for the *static* architecture and quality metamodels, in the context of a metadata repository. In this chapter, we complement these metamodels with their *dynamic* counterpart: the processes of a data warehouse environment.
This chapter is organized as follows: in Section 2 we give a description of the types of data warehouse processes. In Section 3 we present related work and in Section 4 we describe the proposed process metamodel. In Section 5 we present the linkage of the metamodel to the quality model and in Section 6 we present how the metadata repository can be exploited, when enriched with this kind of information. Finally, in Section 7 we conclude our results. The model was developed in cooperation with RWTH Aachen and it is also presented in [VQVJ00].

2. TYPES OF DATA WAREHOUSES PROCESS

Data Warehouses are complex and data-intensive systems that integrate data from multiple heterogeneous information sources and ultimately transform them into a multidimensional representation, which is useful for decision, support applications. Apart from a complex architecture, involving data sources, the global data warehouse, the operational data store (ODS), the client data marts, etc., a data warehouse is also characterized from a complex lifecycle. The data warehouse involves an everlasting *design phase*, where the designer has to produce various modeling constructs (a conceptual model, possibly not in a standard ER formalism and a usually voluminous logical schema), accompanied by a detailed physical design for efficiency reasons (involving indexing, clustering etc.) To top all these, the designer must deal with the data warehouse processes too, which are complex in structure, large in number and hard to code at the same time. Dealing with the data warehouse as set of layered, materialized views is, thus, a very simplistic view. As it has been indicated, the data warehouse refreshment process can already consist of many different subprocesses like *data cleaning, archiving, transformations, aggregations* interconnected through a complex schedule [BoFM99]. The *administration* of the data warehouse is also a complex task, where deadlines must be met for the population of the data warehouse and contingency actions taken in the case of errors. To make the picture complete, we must add the *evolution* phase, which is a

combination of design and administration: as time passes, new data are requested by the end users, new sources of information become available, and the data warehouse architecture must evolve to meet these challenges.
Such systems require a careful design, which is often supported by a repository that tracks all metadata that is relevant for the data warehouse. In Chapter 2, we have presented the metadata modeling approach of [JJQV99] that enables the capturing of the *static* parts of the architecture of a data warehouse, along with information over different quality dimensions of these components. The linkage of the architecture model to quality parameters (quality model) and its implementation in ConceptBase has been formally described in Chapter 2, too. In this chapter, we complement these approaches with the meta-modeling for the *dynamic* part of the data warehouse environment: the *processes*.
The meta-information concerning data warehouse operational processes can be used to support the design and the evolution of the data warehouse. In these phases of the data warehouse lifecycle, the designers of a data warehouse need different kinds of information about processes: what are they supposed to do, why are they necessary, how are they implemented and how they affect other processes in the data warehouse. To this end, we present a model for data warehouse processes, influenced by ideas developed in the area of workflow management systems [CCPP95, SaOr99, GeHS95, WfMC98] as well as the Actor-Dependency model [YuMy94] (and its last version, the Strategic Dependency model [Yu99]). We have identified several requirements for a data warehouse process model, which are close to the specific nature of such an environment. Specifically, these requirements are:

(a) Complexity of structure: the data warehouse processes are quite complex in their nature, in terms of tasks executed within a single process, execution coherence, contingency treatment, etc. The complexity of structure is also confusing for the zooming in and out the repository, which is a well-known requirement in the field of repository management.
(b) Relationship of processes with involved data: the data warehouse processes are data intensive by nature. To handle the data flow in a repository, it is important that the repository is able to capture the interrelationship between processes and relevant data.
(c) Information on process traces: not only the structure of a process is important; the specific traces of executed processes should be tracked down too. The repository, thus, gains added value since, *ex ante* the data warehouse stakeholders can use it for design purposes (e.g., to select the data warehouse objects necessary for the performance of a task) and *ex post*, people can relate the data warehouse objects to decisions, tools and the facts which have happened in the real world [JaJR90].
(d) Respect of the metadata framework. In Chapter 2 we have defined a metadata framework which introduces a clear separation of perspectives and instantiation layers. We demand that both these requirements are respected. As far as the separation of perspectives is concerned, we can discriminate between *what* components an information systems consists of (logical perspective), *how* they actually perform (physical perspective) and *why* these components exist (conceptual perspective). The data warehouse process model should reflect this separation of viewpoints. As far as the separation of instantiation layer is concerned, the *templates* for generic data warehouse processes should be clearly distinct from the description of the *specific processes of a particular data warehouse*, and of course, from the *traces* of these processes.
(e) Linkage to architecture and quality metamodels: Apart from respecting the overall metadata framework, the process model should be in harmony with the adopted metamodels for data warehouse architecture and quality, which have been presented also in Chapter 2.

The contribution of the proposed metamodel is towards the fulfillment of all the aforementioned requirements. We do not advocate that our approach is suitable for any kind of process, but rather we focus our attention to the internals of data warehouse systems. Our model has been implemented in the metadata repository ConceptBase, using the Telos language. The usefulness of our approach is demonstrated by the fact that the proposed model enables data warehouse management, design and evolution, as we will show in section 4. First, the design of the data warehouse is supported by extended use of consistency checks, to ensure the correctness of the representation. Moreover, instead of treating the data warehouse as a set of layers of materialized views, we enrich this viewpoint by deriving the description of the data warehouse materialized views from the process definitions used

for their population. Third, the model facilitates the administration of the data warehouse, by enabling the measurement of quality of data warehouse processes and the spotting of inefficiencies. Finally, a specific task of data warehouse administration, namely evolution, is supported by the exploitation of the information on the interdependencies of data warehouse components, to forecast any possible impact by the change of any of these components.

3. RELATED WORK

In the field of workflow management, [WfMC98] is a standard proposed by the Workflow Management Coalition (WfMC). The standard includes a metamodel for the description of a workflow process specification and a textual grammar for the interchange of process definitions. A *workflow process* comprises of a network of *activities*, their interrelationships, criteria for staring/ending a process and other information about *participants*, invoked *applications* and relevant *data*. Also, several other kinds of entities that are external to the workflow, such as system and environmental data or the organizational model are roughly described. A widely used web server for workflow literature is maintained by [Klam99].

[SaOr99] use a simplified workflow model, based on [WfMC98], using *tasks* and *control flows* as its building elements. The authors present an algorithm for identifying structural conflicts in a control flow specification. The algorithm uses a set of graph reduction rules to test the correctness criteria of *deadlock freedom* and *lack-of-synchronization freedom*. In [MaOr99] the model is enriched with modeling constructs and algorithms for checking the consistency of workflow temporal constraints. In [DaRe99] several interesting research results on workflow management are presented in the field of electronic commerce, distributed execution and adaptive workflows.

In [CCPP95], the authors propose a conceptual model and language for workflows. The model gives the basic entities of a workflow engine and semantics about the execution of a workflow. The proposed model captures the mapping from workflow specification to workflow execution (in particular concerning exception handling). Importance is paid to the inter-task interaction, the relationship of workflows to external agents and the access to databases. Other aspects of workflow management are explored in [CaFM99, CCPP98].

In [KLCO96] a general model for transactional workflows is presented. A transactional workflow is defined to consist of several tasks, composed by constructs like ordering, contingency, alternative, conditional and iteration. Nested workflows are also introduced. Furthermore, correctness and acceptable termination schedules are defined over the proposed model.

Process and workflow modeling have been applied in numerous disciplines. A recent overview on process modeling is given in [Roll98], where a categorization of the different issues involved in the process engineering field is provided. The proposed framework consists of four different but complementary viewpoints (expressed as "worlds"): the *subject* world, concerning the definition of the process with respect to the real world objects, the *usage* world, concerning the rationale for the process with respect to the way the system is used, the *system* world, concerning the representation of the processes and the capturing of the specifications of the system functionality and finally, the *development* world, capturing the engineering meta-process of constructing process models. Each world is characterized by a set of *facets*, i.e., attributes describing the properties of a process belonging to it.

In [JaJR90] the authors propose a software process data model to support software information systems with emphasis on the control, documentation and support of decision making for software design and tool integration. Among other features, the model captures the representation of design objects ("what"), design decisions ("why") and design tools ("how").

The *MetaData Coalition (MDC)*, is an industrial, non-profitable consortium with aim to provide a standard definition for enterprise metadata shared between databases, CASE tools and similar applications. The *Open Information Model (OIM)* [MeDC99] is a proposal (led by MicroSoft) for the core metadata types found in the operational and data warehousing environment of enterprises. The MDC OIM uses UML both as a modeling language and as the basis for its core model. The OIM is divided in sub-models, or *packages*, which extend UML in order to address different areas of information management. The *Data Transformations Elements* package covers basic transformations for relational-to-relational translations. The package is not a data warehouse process modeling package (covering data propagation, cleaning rules, or the querying process), but covers in detail the sequence of steps, the functions and mappings employed and the execution traces of data transformations in a data warehouse environment.

4. METAMODEL FOR DATA WAREHOUSE OPERATIONAL PROCESSES

The basis of all our references will be the metadata framework, already presented in Chapter 2. A condensed graphical overview of the architecture metamodel is given in Figure 3.1. The framework describes a data warehouse in three *perspectives*: a *conceptual*, a *logical* and a *physical* perspective. Each perspective is partitioned into the three traditional data warehouse *levels*: *source*, *data warehouse* and *client* level. On the *metamodel* layer, the framework gives a notation for data warehouse architectures by specifying meta classes for the usual data warehouse objects like data store, relation, view, etc. On the *metadata* layer, the metamodel is instantiated with the concrete architecture of a data warehouse, involving its schema definition, indexes, tablespaces, etc. The lowest layer in Figure 3.1 represents the real world where the actual processes and data reside.
The *static* description of the architecture parts of the data warehouse (left part of Figure 3.1) is complemented, in this chapter, with a metamodel of the *dynamic* parts of the data warehouse, i.e. the data warehouse operational processes. As one can notice on the right side of Figure 3.1, we follow again a three level instantiation: a *Process Metamodel* deals with generic entities involved in all data warehouse processes (operating on entities found at the data warehouse metamodel level), the *Process Model* covers the processes of a specific data warehouse by employing instances of the metamodel entities and the *Process Traces* capture the execution of the actual data warehouse processes happening in the real world.

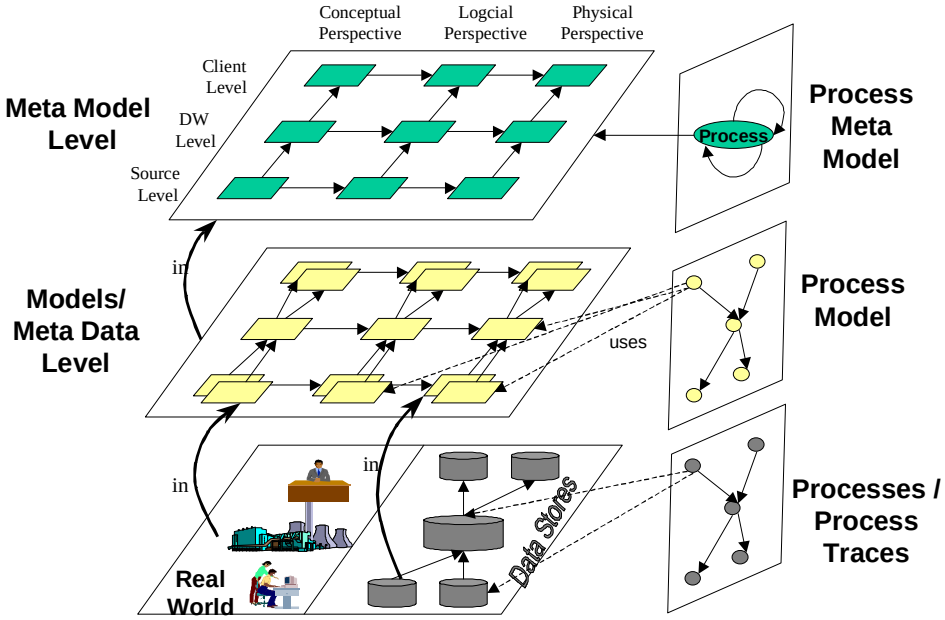


Figure 3.1: Framework for Data Warehouse Architecture [JJQV99].

Our approach has been influenced by ideas on dependency and workflow modeling stemming from [CCPP95, SaOr99, JaJR90, RaDh92, YuMy94, GeHS95] and on the Workflow Reference Model, presented in [WfMC98] by the Workflow Management Coalition (WfMC). We found the Workflow Reference Model too abstract for the purpose of a repository serving a specific kind of processes, the data warehouse operational processes. First, the relationship of an activity with the data it involves is not really covered, although this would provide extensive information of the data flow in the data warehouse. Second, the separation of perspectives is not clear, since the WfMC proposal focuses only on the structure of the workflows. To compensate this shortcoming, we employ the basic idea of the Actor-Dependency model to add a conceptual perspective to the definition of a process, capturing the reasons behind its structure.

4.1 The 3 Perspectives for the Process Model.

In Figure 3.2 we offer an intuitive view for the categorization of the entities in the process metamodel. The model has three different perspectives covering distinct aspects of a process: the *conceptual*,

logical and *physical* perspective. The categorization is following the separation of the framework proposed in Chapter 2, and fits naturally with the adopted architecture model of [JJQV99], since the perspectives of the process model operate on objects of the respective perspective of the architecture model. As mentioned in [YuMy94] there are different ways to view a process: *what* steps it consists of (logical perspective), *how* they are to be performed (physical perspective) and *why* these steps exist (conceptual perspective). Thus, we view a data warehouse process from three perspectives: a central *logical* part of the model, which captures the basic structure of a process, its *physical* counterpart which provides specific details over the actual components that execute the activity and the *conceptual* perspective which abstractly represents the basic interrelationships between data warehouse stakeholders and processes in a formal way.
Typically, the information about how a process is executed concerns stakeholders who are involved in the everyday use of the process. The information about the structure of the process concern stakeholders that manage it and the information relevant to the reasons behind this structure concern process engineers who are involved in the monitoring or evolution of the process environment. In the case of data warehouses it is expected that all these roles are covered by the data warehouse administration team, although one could also envision different schemes. Another important issue shown in Figure 3.2 is that there is also a data flow at each of the perspectives: a type description of the incoming and outcoming data at the logical level, where the process acts as an input/output function, a physical description of the details of the physical execution for the data involved in the activity and a relationship to the conceptual entities related to these data, connected through a corresponding role.

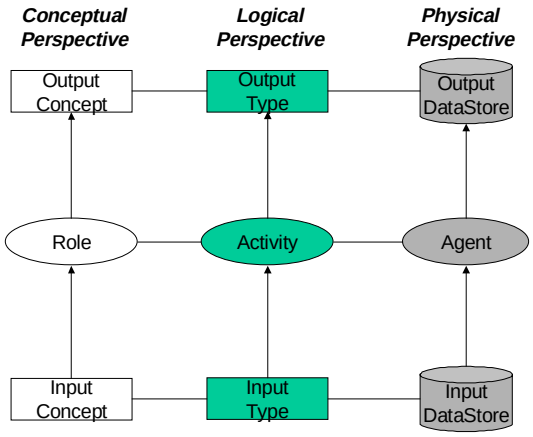


Figure 3.2 The reasoning behind the 3 perspectives of the process metamodel.

Once again, we have implemented the process metamodel in the Telos language and specifically in the ConceptBase metadata repository. The implementation of the process metamodel in ConceptBase is straightforward, thus we choose to follow an informal, bird's-eye view of the model, for reasons of presentation and lack of space. Wherever definitions, constraints, or queries of ConceptBase are used in the chapter, they will be explained properly in natural language, too.
In the sequel, when we present the entities of the metamodel, this will be done in the context of a specific perspective. In the Telos implementation of the metamodel, this is captured by specializing the generic classes C*onceptualObject, LogicalObject* and *PhysicalObject* with ISA relationships, accordingly. We will start the presentation of the metamodel from the logical perspective. First, we will show how it deals with the requirements of structure complexity and capturing of data semantics in the next two sections. Then, in subsections 2.4 and 2.5 we will present the physical and the conceptual perspectives. In the former, the requirement of trace logging will be fulfilled too. The full metamodel is presented in Figure 3.3.

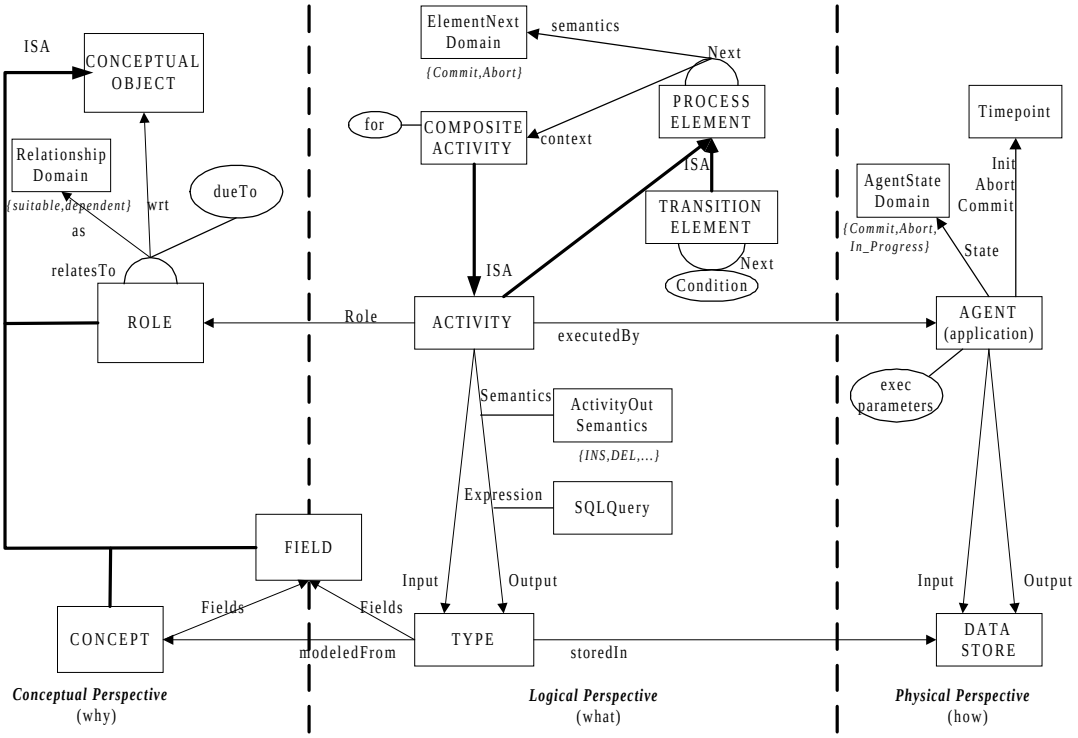


Figure 3.3 The Data Warehouse Operational Process Metamodel.

4.2 Complexity in the process structure

Following [WfMC98], the main entity of the logical perspective is an *Activity*. An activity represents an amount of "work which is processed by a combination of resource and computer applications". Activities are rather complex in nature and this complexity is captured by the specialization of *Activity*, namely *CompositeActivity*. We follow here the lessons coming both from repository and workflow management: there must be the possibility of zooming in and out the repository. Composite activities are composed by *ProcessElements*, which is a generalization of the entities *Activity* and *TransitionElement.* A transition element is the "bridge" between two activities: it is employed for the interconnection of activities participating in a complex activity. The attribute *Next* of the process elements captures the sequence of events.

Formally, a *Process Element* is characterized by the following attributes:

- *Next*: a *ProcessElement* that is next in the sequence of a composite activity. The attribute *Next* has itself two attributes, that characterize it:
  - *Context*: A *CompositeActivity*. Since two activities can be interrelated in more than one complex data warehouse processes, the context of this interrelationship is captured by the relevant *CompositeActivity* instance.
  - *Semantics*: This attribute denotes whether the next activity in a schedule happens on successful termination of the previous activity (*COMMIT)* or in the case where a contingency action is required (*ABORT*). Thus, the class *ElementNextDomain* has two instances {*COMMIT*, *ABORT*}.

A *TransitionElement* inherits the attributes of *ProcessElement*, but most important, is used to add more information on the control flow in a composite activity. This is captured by two mechanisms. First, we enrich the *Next* link with more meaning, by adding a *Condition* attribute to it. A *Condition* is a logical expression in Telos denoting that the firing of the next activity is performed when the *Condition* is met.

Second, we specialize the class Transition Element to four prominent subclasses, capturing the basic connectives of activities, as suggested by [WfMC98]: *Split_AND, Split_XOR, Join_AND, Join_XOR.* This specialization is not depicted in Figure 3.3 to avoid overloading the figure. The same happens with several attributes of the other entities, too. Their semantics are obviously the same with the ones of the WfMC proposal. For example, the *Next* activity of a *Join_XOR* instance is fired when (a) the

Join_XOR has at least two activities "pointing" to it through the *Next* attribute and only one *Next* Activity (well-formedness constraint), (b) the *Condition* of the *Join_XOR* is met and (c) only one of the "incoming" Activities has *COMMIT* semantics in the *Next* attribute. This behavior can be expressed in Telos with the appropriate rules.
The WfMC proposes two more ways of transition between Activities. The *dummy activities*, which perform routing based on condition checking, are modeled in our approach, as simple Transition Elements. The *LOOP* activities are captured as instances of *CompositeActivity*, with an extra attribute: the *for* condition, expressed as a string.
In Figure 3.4, we depict how two transitions are captured in our model. In case (a), we model a composite activity, composed of two sub-activities, where the second is fired when the first activity commits and a boolean condition is fulfilled. *P1_Commited?* is a transition element. In case (b) we give two alternatives for the capturing of two concurrent activities, fired together. *Dummy* is an activity with no real operations performed (e.g., like the *dummy* activity of WfMC).

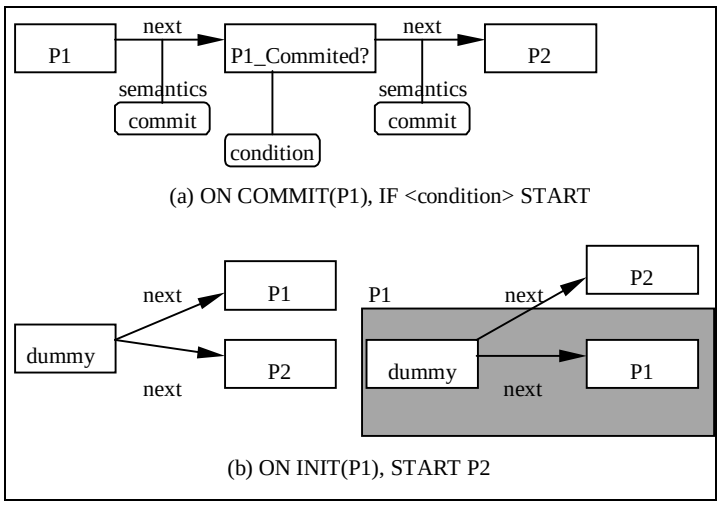


Figure 3.4 Examples of complex activities.

4.3 Relationship with data

We introduce the entity *Type* to capture the logical representation of a data store. A *Type* can serve as a wrapper for all kinds of data stores, denoting their schema. Formally, a *Type* is defined as a specialization of *LogicalObject* with the following attributes:

- *Name*: a single name denoting a unique *Type* instance.
- *Fields*: a multi-value attribute. In other words, each *Type* has a name and a set of *Fields*, exactly like a relation in the relational model.
- *Stored*: a *DataStore*, i.e., a physical object representing any application used to manipulate stored data (e.g., a DBMS). This attribute will be detailed later, in the description of the physical perspective.

Any kind of data store can be represented by a Type in the logical perspective. For example, the schema of multidimensional cubes is of the form $[D_1,...,D_n,M_1,...,M_m]$ where the D_i represent dimensions (forming the primary key of the cube) and the M_j measures [Vass98]. Cobol files, as another example, are in general records with fields having two peculiarities (a) nested records and (b) alternative representations. One can easily unfold the nested records and choose only one of the alternative representations, reducing, thus, the intentional description of the Cobol file to a *Type*.
Each activity in a data warehouse environment is linked to a set of incoming types as well as to a set of outcoming types, too. The data warehouse activities are of data intensive nature in their attempt to push data from the data sources to the data warehouse materialized views or client data marts. We can justify this claim by listing the most common operational processes:

(a) *data transfer processes*, used for the instantiation of higher levels of aggregation in the ODS and the data warehouse with data coming from the sources or lower levels of aggregation;

(b) *data transformation processes*, used for the transformation of the propagated data to the desired format;
(c) *data cleaning processes*, used to ensure the consistency of the data in the warehouse (i.e. the fact that these data respect the database constraints and the business rules);
(d) *computation processes*, which are used for the derivation of new information from the stored data (e.g., further aggregation, querying , business logic, etc.).

Thus, the relationship between data and processes is apparent in such an environment and we capture it by expressing the outcome of a data warehouse process as a function over the inputs. These semantics are captured through SQL queries, extended with functions – wherever richer semantics than SQL are required.

Therefore, an *Activity* is formally characterized by the following attributes:

- *Next*: inherited by *Process Element.*
- *Name*: a name to uniquely identify the activity among the rest of the data warehouse activities.
- *Input*: a multi-value *Type* attribute. This attribute models all the data stores used by the activity to acquire data.
- *Output*: a multi-value *Type* attribute. This attribute models all the data stores or reports, where the activity outputs data (even a report for the decision-maker can have a schema, and be thus represented by a *Type*). The *Output* attribute is further explained by two attributes:
  - *Semantics*: a single value belonging to the set {*Insert, Update, Delete, Select*} (captured as the domain of class *ActivityOutSemantics*). A process can either add (i.e., append), or delete, or update the data in a data store. Also it can output some messages to the user (captured by using a "Message" Type and Select semantics).
  - *Expression*: a single SQL query, (instance of class *SQLQuery*) to denote the relationship of the output and the input types. We adopt the SQL language extended with functions, to capture this relationship.
- *ExecutedBy*: a physical *Agent* (i.e., an application program) executing the Activity. More information on agents will be provided in the sequel.
- *Role*: a conceptual description of the activity. This attribute will be properly explained in the description of the conceptual perspective.

The following table shows how the various types of processes are modeled in our approach.

Activity	**Semantics**	**Expression**
transfer	INS/UPD	SELECT * FROM <In>
transformation	INS/UPD	SELECT * FROM <In> WHERE $attr_i$ = f($attr_1$, ..., $attr_n$)
cleaning	DEL/UPD	- Primary Key: SELECT <P.K.> FROM <IN> GROUP BY <P.K.> HAVING COUNT(*) > 1 - Foreign Key: SELECT <P.K.> FROM <IN> WHERE <F.K.> NOT IN (SELECT <F.K> FROM <TARGET>) - Any other kind of query
computation	INS/UPD	Any kind of query
messaging	SEL	Any kind of query

Figure 3.5 Examples of *Output* attribute for particular kinds of activities

To motivate the discussion, we will use a part of a case study, enriched with extra requirements, to capture the complexity of the model that we want to express. The role of the discussed organization is to collect various data about the yearly activities of all the hospitals of a particular region. The system relies on operational data coming from COBOL files. The source of data, for our example, is a COBOL file, dealing with the yearly information by class of beds and hospital (here we use only three classes, namely A, B and C). The COBOL file yields a specific attribute for each type of class of beds. Each year, the COBOL file is transferred from the production system to the data warehouse and stored in a "buffer" table of the data warehouse, acting as mirror of the file inside the DBMS. Then, the tuples of the buffer table are used by computation procedures to further populate a «fact» table inside the data warehouse. Several materialized views are then populated with aggregate information and used by client tools for querying.

We assume the following four *Types*: *CBL*, *Buffer*, *Class_info* and *V1*. The schemata of these types are depicted in Figure 3.6. There are four *Activities* in the data warehouse: *Loading, Cleaning, Computation* and *Aggregation*. The *Loading* activity simply copies the data from the *CBL* Cobol file to the *Buffer* type. *H_ID* is an identifier for the hospital and the three last attributes hold the number

of beds per class. The *Cleaning* activity deletes all the entries violating the primary key constraint. The *Computation* activity transforms the imported data into a different schema. Suppose that the date is converted from American to European format and the rest of the attributes are converted to a combination (*Class_id*, *#Beds*). For example, if (`03,12/31/1999,30,0,50`) is a tuple in the *Buffer* table, the respective tuples in the *Class_Info* table are {(`03,31/Dec/1999,A,30`), (`03,31/Dec/1999,50`)}. The *Aggregation* activity simply produces the sum of beds by hospital and year.

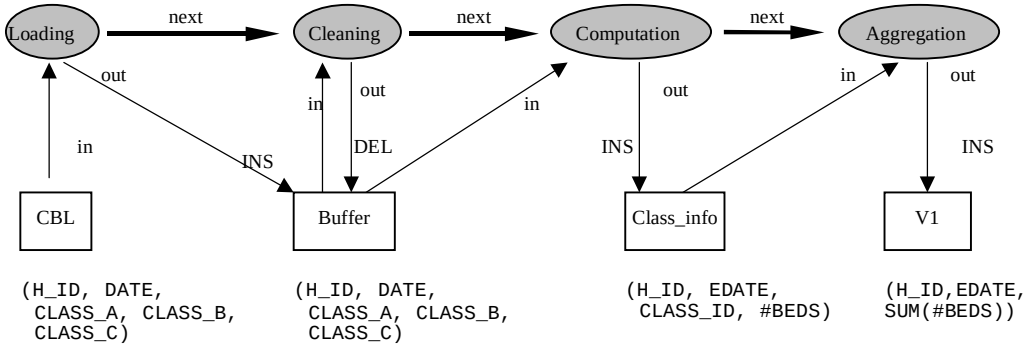


Figure 3.6 Motivating Example

The expressions and semantics for each activity are listed in Figure 3.7.

Attribute name	**Expression**	**Semantics**
Loading.Out:	`SELECT * FROM CBL`	INS

Cleaning.Out:	`SELECT * FROM BUFFER B1`
	`WHERE EXISTS (SELECT B2.H_ID, B2.DATE FROM BUFFER B2`
	`WHERE B1.H_ID = B2.H_ID AND B1.DATE = B2.DATE`
	`GROUP BY H_ID,DATE HAVING COUNT(*)> 1)`
		DEL

Computation.Out:	`SELECT H_ID, EUROPEAN(DATE) AS EDATE, 'A',`
	`CLASS_A AS #BEDS`
	`FROM BUFFER WHERE CLASS_A <> 0`
	`UNION`
	`SELECT H_ID, EUROPEAN(DATE) AS EDATE, 'B',`
	`CLASS_B AS #BEDS`
	`FROM BUFFER WHERE CLASS_B <> 0`
	`UNION`
	`SELECT H_ID, EUROPEAN(DATE) AS EDATE, 'C',`
	`CLASS_C AS #BEDS`
	`FROM BUFFER WHERE CLASS_C <> 0`
		INS

Aggregation.Out:	`SELECT H_ID, EDATE, SUM(#BEDS) AS SUM_BEDS`
	`FROM CLASS_INFO`
	`GROUP BY H_ID, EDATE`
		INS

Figure 3.7 Expressions and semantics for the motivating example

All activities are appending data to the involved types, so they have *INS* semantics, except for the cleaning process, which deletes data, and thus has *DEL* semantics. We want to stress, also, that the use of SQL as a mapping function is done for the *logical* perspective. We do not imply that everything should actually be implemented through the use of the employed queries, but rather that the relationship of the input and the output of an activity is expressed as a function, through a declarative language such as SQL.

4.4 The Physical Perspective

Whereas the logical perspective covers the structure of a process ("what" in [YuMy94] terminology), the *physical perspective* covers the details of its execution ("how"). Each process is executed by an *Agent* (i.e. an application program). Each *Type* is physically stored, by a *DataStore* (providing information for issues like tablespaces, indexes, etc.). An *Agent* can be formalized as follows:

- *State*: a single value belonging to the domain of class *AgentStateDomain* = {*In_Progress*, *Commit*, *Abort*}.

- *Init_time, Abort_time, Commit_time*: timestamps denoting the *Timepoints* when the respective events have occurred.
- *Execution_parameters*: a multivalue string attribute capturing any extra information about an execution of an agent.
- *In, Out*: physical *DataStores* communicating with the *Agent*. The types used by the respective logical activity must be stored within these data stores.

Activity Name	Loading	Cleaning	Computation	Aggregation
Agent Name	sqlldr80.exe	Clean.pql	Comp.pql	Aggr.pql
State	COMMIT	COMMIT	COMMIT	COMMIT
Init time	31/12/99:00:00	31/12/99:01:00	31/12/99:01:15	31/12/99:03:00
Abort time	--	--	--	--
Commit time	31/12/99:01:00	31/12/99:01:15	31/12/99:03:00	31/12/99:03:30
Execution parameters	Parfile=param.par	--	--	--

Figure 3.8 Trace information after a successful execution of the process of the motivating example

In Figure 3.8 we present the trace information after a successful execution of the process described in Figure 3.6. We show the relationship between the logical and the physical perspective by linking each logical activity to a specific (physical) application program.

4.5 The Conceptual Perspective

One of our major purposes behind the introduction of the conceptual perspective is to help the interested stakeholder understand the reasoning behind any decisions on the architecture and characteristics of the data warehouse processes. First of all, each *Type* (i.e. *Relation, Cube*, etc.) in the logical perspective is a representation of a *Concept* in the conceptual perspective. A concept is an abstract entity representing a real world class of objects, in terms of a conceptual metamodel -e.g., the ER model. Both *Types* and *Concepts* are constructed from *Fields* (representing their attributes), through the attribute *fields*. We consider *Field* to be a subtype both of *LogicalObject* and *ConceptualObject.*

The central entity in the conceptual perspective is the *Role*, which is the conceptual counterpart both of activities and concepts. The *Role* is basically used to express the interdependencies of these entities, through the attribute *RelatesTo.* Formally, a *Role* is defined as follows:

- *RelatesTo*: another Role.
  - *As*: a single value belonging to the domain of class *RelationshipDomain* = {*suitable*, *dependent*}.
  - *Wrt*: a multi-value attribute including instances of class *ConceptualObject*.
  - *dueTo:* a string attribute, textually documenting any extra information on the relationship of two roles.

A role represents any program or data store participating in the environment of a process, charged with a specific task and/or responsibility. The interrelationship between roles is modeled through the *RelatesTo* relationship. An instance of this relationship is a statement about the interrelationship of two roles in the real world, such as 'View V1 *relates to* table Class_Info *with respect to* the attributes Id, Date and number of beds *as* dependent due to loading reasons'. Since both data and processes can be characterized by SQL statements, their interrelationship can be traced in terms of attributes. In Figure 3.9 two examples are also used to clarify this instantiation, based on the example of Figure 3.6.

Attribute	**Example 1**	**Example 2**
Role 1	*Buffer*	*Aggregation*
Role 2	*CBL*	*Class_Info*
As	*Dependent*	*Dependent*
Wrt	*CBL.**	*H_ID, EDate, #Beds*

Figure 3.9 Examples of role interrelationships.

The conceptual perspective is influenced by the Actor Dependency model [YuMy94]. In this model, the actors *depend* on each other for the accomplishment of goals and the delivery of products. The *dependency* notion is powerful enough to capture the relationship of processes where the outcome of the preceding process in the input for the following one. Still, our approach is more powerful since it can capture *suitability* too (e.g., in the case where more than one concepts can apply for the population of an aggregation).
In the process of understanding the occurring errors or the design decisions over the architecture of a data warehouse, the conceptual perspective can be used as a reasoning aid in the task of discovering the interdependencies of the actors (possibly in a transitive fashion) and the possible alternatives for different solutions through a set of suitable candidates. For example the previous statement for the relationship of view V1 and table Class_Info could be interpreted as 'View V1 is affected from any changes to table Class_Info and especially the attributes Id, Date and no. of beds'. Moreover, the provided links to the logical perspective can enable the user to pass from the abstract relationships of roles to the structure of the system. On top of these, as we shall show in the last section, data warehouse evolution can be designed and influenced by the interdependencies tracked by the *Role* entities. We also show that these interdependencies do not have to be directly stored, but can also be incrementally computed.

5. ISSUES ON PROCESS QUALITY

In this section we present how the process model is linked to the metamodel for data warehouse quality, along with specific quality dimensions and factors of data warehouse operational processes.

5.1 Terminology for Quality Management

As already explained in Chapter 2, we can make good use of a quality metamodel capturing the relationship between quality metrics (or factors) and data warehouse objects. In Chapter 2, we made a detailed proposal for such a model and its use. In this chapter, for reasons of clarity, we will use only a part of this model to show how data warehouse operational processes can be linked to quality management. This subset of the metamodel is also quite close to other, similar, proposal such the one of [JeQJ98]. The minimal subset of a quality metamodel, which is required here, is composed from *quality goals* and *quality factors* (which are related to objects of the architecture metamodel) interconnected through *quality questions*. The terminology for the quality goals is provided from a set of *quality dimensions*, which can also be considered as high-level groupings of quality factors. The semantics of all these entities are not outside the scope of the definitions of Chapter 2.

5.2 Quality Dimensions of Data Warehouse Processes

For the case of processes, we will first define a set of generic quality dimensions. We do not wish to detail the whole set of possible dimensions for all the data warehouse processes, rather, we intend to come up with a minimal characteristic set, which is generic enough to cover the most basic characteristics of data warehouse processes. Then, this set of quality dimensions can be refined and enriched with customized dimensions by the data warehouse stakeholders.
The set of generic quality dimensions is influenced mainly from the quality criteria for workflows defined in [GeRu97] and the quality dimensions for software evaluation presented in [ISO91].

1. *Correctness*: a specification exists, describing the conditions under which, the process has achieved its aim (e.g. provides the correct results).
2. *Functionality*: the process satisfies specific needs of data warehouse stakeholders.

3. *Efficiency*: the process has a good balance between level of performance and amount of used resources. Efficiency can be decomposed to sub-dimensions like *timeliness* and *resource consumption*.
4. *Maintenability*: the degree of easiness with which the process can be modified. It can also be decomposed to sub-dimensions like *analyzability* (effort required to detect deficiencies and propose changes) and *changeability* (effort required to implement these changes).

Moreover, the quality of a process is also determined by the quality of its output (which is basically a relation). We refer the reader to the previous chapter and [JJQV99] for further probing on this issue.

5.3 Relationships between Processes and Quality

Quality goals are high-level abstractions of the data warehouse quality: they describe *intentions* or *plans* of the data warehouse users with respect to the status of the data warehouse. Moreover, we use the repository to track how quality goals are evaluated by specific questions and measurements. In contrast, our process model presented in section 2 describes *facts* about the current status of the data warehouse and what activities are performed in the data warehouse.

However, the reason behind the execution of a process is a quality goal, which should be achieved or improved by this process. For example, a data cleaning process is executed on the ODS in order to improve the accuracy of the data warehouse. We have represented this interdependency between processes and quality goals by extending the relationship between roles and data warehouse objects in the conceptual perspective of the process model (relationship *Expressed For*). This is shown in the upper part of Figure 3.10.

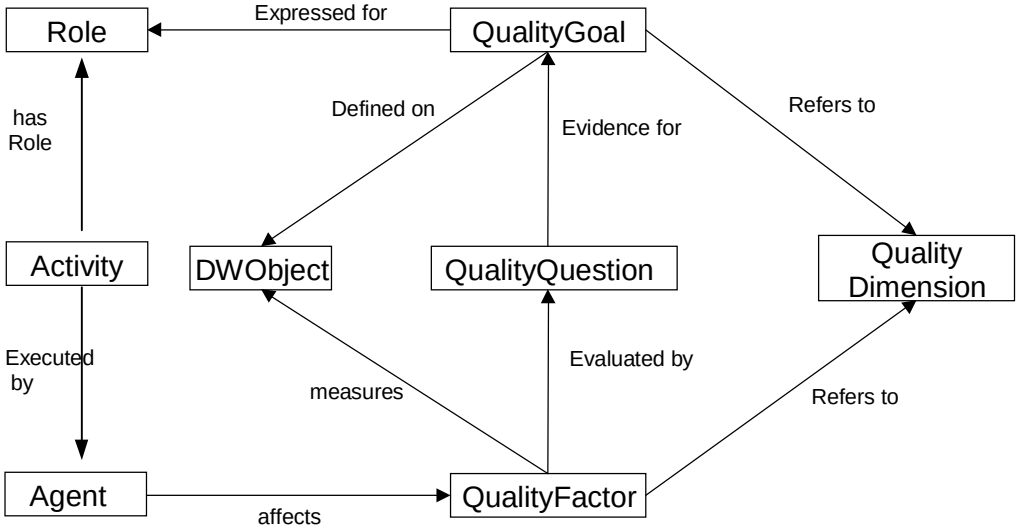


Figure 3.10 Relationships between processes and quality

Our model is capable of capturing all the kinds of dependencies, mentioned in [YuMy94]. *Task dependencies*, where the dependum is an *activity*, are captured by assigning the appropriate role to the attribute *Wrt* of the relationship. *Resource dependencies*, where the dependum is the availability of a *resource*, are modeled when fields or concepts populate this attribute. The relationship *ExpressedFor* relates a role to a high-level quality goal; thus *Goal dependencies*, dealing with the possibility of making a condition true in the real world, are captured from the model, too. *Soft-goal dependencies*, are a specialization of goal dependencies, where their evaluation cannot be done in terms of concrete quality factors: the presence or absence of this feature, determines the nature of the quality goal.

The lower part of Figure 3.10 represents the relationship between processes and quality on a more operational level. The operation of an agent in the data warehouse will have an impact on quality factors of data warehouse objects. The relationship *Affects* represents both the measured and expected effect of a data warehouse process on data warehouse quality. The achieved effect of a data warehouse process must always be confirmed by new measurements of the quality factors. Unexpected effects of

data warehouse processes can be detected by comparing the measurements with the expected behavior of the process.
Between the high-level, user oriented, subjective quality goals and the low-level, objective, component-oriented quality factors, we use the notion of *Quality Question*. This provides the methodological bridge to link the aforementioned entities. The vocabulary / domain of the quality questions, with respect to the process model is anticipated to be the set of data warehouse activities, which, of course, can be mapped to reasons (roles) and conditions (agents) for a specific situation. Thus, the discrimination of logical, conceptual and physical perspectives is verified once more, in the quality management of the data warehouse: the quality goals can express «why» things have happened (or should happen) in the data warehouse, the quality questions try to discover «what» actually happens and finally, the quality factors express «how» this reality is measured.

6. EXPLOITATION OF THE METADATA REPOSITORY

We exploit the metadata repository in all the phases of the data warehouse lifecycle. During the *design* phase, the user can check the consistency of his/her design, to determine any violations of the business logic of the data warehouse, or the respect of simple rules over the structure of the data warehouse schema. During the *administration* phase (i.e., in the everyday usage of the data warehouse) we can use the repository to discover quality problems. A particular task in the data warehouse lifecycle, *data warehouse evolution*, is examined separately, to determine possible impacts, when the schema of a particular object changes. The metadata repository ConceptBase is powerful enough to allow the developer define constraints and views on the stored objects. However, our approach can be, of course, exploited over any other metadata repository with the respective facilities.

6.1 Consistency checking in the metadata repository

The consistency of the metadata repository should be checked to ensure the validity of the representation of the real world in the repository. The following two views find out whether the types used as inputs (or outputs) of an activity are stored in the respective data stores used as inputs (or outputs respectively) of the agent, executed by the activity. The check is performed as a view and not as a constraint, since composite activities may not directly correspond to a single agent.

```
QueryClass InconsistentInTypes isA Type with
constraint
       c : $ exists d/DataStore ac/Activity ag/Agent
              (ac input this) and (ac executedBy ag) and
              (ag input d) and not(this storedIn d) $
end

QueryClass InconsistentOutTypes isA Type with
constraint
       c : $ exists d/DataStore ac/Activity ag/Agent
              (ac output this) and (ac executedBy ag) and
              (ag output d) and not(this storedIn d) $
end
```

Other simple checks performed through constraints involve the local structure of the process elements. For example, split transition elements should obligatorily have at least one incoming edge and more that one outcoming (the opposite holding for join transition elements). The timestamps of an agent should also be consistent with its state. The repository can also be used by external programs to support the execution of consistency checking algorithms like the ones proposed in [SaOr99, MaOr99].

6.2 Interdependencies of types and processes

Our modeling enables us to construct a partially ordered graph: for each *Type* instance, say *t*, there is a set of types and activities, used for the population of *t* ("before" *t*), denoted as **B**(*t*). Also, there is another set of objects using *t* for their population ("after" *t*), denoted as **A**(*t*). We can recursively compute the two sets from the following queries in ConceptBase:

```
GenericQueryClass Previous isA Activity with
parameter
       t : Type
constraint
       c : $ (this output t) $
end

GenericQueryClass Before isA DW_Object with
parameter
       t : Type
constraint
       c : $ (this in Previous[t/t]) or
       ( exists t1/Type (ac in Previous[t/t]) and (ac input t1) or
       ( exists d/DW_Object (ac in Previous[t/t]) and (ac input t1)
     and (d in Before[t1/t]))) $
end
```

Queries for the successor and after relationships can be defined in a similar way. Using the previous and successor objects of a type in the metadata repository, we can derive an SQL definition for this type, based on the data flow that populates it. We suppose that there is a set of types belonging to the set *SourceSchema*, denoting all the types found in the data sources. We treat the types of *SourceSchema* as source nodes of a graph: we do not consider any processes affecting them. For the rest of the types, we can derive an SQL expression by using existing view reduction algorithms. Several complementary proposals exist such as [Kim82 corrected with the results of GaWo87, Mura89, Mura92 (which we will mention as Kim82+ in the sequel)], [Daya87], [ChSh96], [MFPR90], [PiHH92], [LeMS94]. The proposed algorithm is applicable to graphs of activities that do not involve updates. In most cases, an update operation can be considered as the combination of insertions and deletions or as the application of the appropriate function to the relevant attributes.

Algorithm `Extract_Type_Definitions`
Input: a list of processes **P**=[P_1, P_2, …, P_n], a set of types **T**={T_1, T_2, …, T_m}. Each process `P[i]` has a type `P[i].out`, belonging to **T**, and an expression `P[i].expr`. Each type of **T**, say `t`, has an SQL expression `t.expr` comprised of a set of "inserted data" (`t.i_expr`) and "deleted" data (`t.d_expr`). Also there is a subset of **T**, **S**, with the source types.
Output: A set of SQL definitions for each type of **T**.

```
Begin
   Initialize all the expressions of T-S to {}.
   For i := 1 to n
      Case
         P[i].semantics = 'INS'
            P[i].out.i_expr := P[i].out.i_expr UNION Reduce(P[i].expr)
         P[i].semantics = 'DEL'
            P[i].out.d_expr := P[i].out.d_expr UNION Reduce(P[i].expr)
      End_case
      P[i].out.expr := P[i].out.i_expr MINUS P[i].out.d_expr
   End_for
End
```

Where *Reduce(expr)*:
1. Use the technique of [MFPR90] to represent SQL queries; if self-references exist (e.g. in the case of DEL statements) discriminate between multiple occurrences of the same table.
2. Use the reduction techniques of [PiHH92], [Kim82+], [LeMS94] wherever applicable to reduce the query definition to a compact form.

Figure 3.11. Algorithm for extracting the definition of a type in the repository

We have not implemented this algorithm in ConceptBase yet, although external programs could easily do this.
Suppose that we apply the algorithm to our motivating example. Then the results are as shown in the following table. For reasons of convenience, we break composite definitions of table expressions into the different lines of Fig. 3.12. For example, when the 3rd iteration refers to the definition of table *Buffer*, it does so with respect to the definition of line 2.

The local computation of the expression of a single type can also be done. It should take into consideration the list of processes affecting this particular type. Replace the total number of involved process n, with the serial number of the last process that affects this particular type. Note, though, that the intermediate definitions for any other involved type might be different from the one of algorithm `Extract_Type_Definitions`.

i	Expression
1	Buffer.expr = Buffer.i_expr:= SELECT * FROM CBL
2	Buffer.d_expr:= (SELECT * FROM CBL C1 WHERE EXISTS (SELECT C2.H_ID, C2.DATE FROM CBL C2 WHERE C1.H_ID=C2.H_ID AND C1.DATE=C2.DATE GROUP BY H_ID,DATE HAVING COUNT(*)>1) Buffer.expr:= (SELECT * FROM CBL) MINUS (SELECT * FROM CBL C1 WHERE EXISTS (SELECT C2.H_ID, C2.DATE FROM CBL C2 WHERE C1.H_ID=C2.H_ID AND C1.DATE=C2.DATE GROUP BY H_ID,DATE HAVING COUNT(*)>1)
3	Class_info.expr = Class_info.i_expr:= SELECT H_ID, EUROPEAN(DATE) AS EDATE, 'A', CLASS_A AS #BEDS FROM BUFFER WHERE CLASS_A <> 0 UNION SELECT H_ID, EUROPEAN(DATE) AS EDATE, 'B', CLASS_B AS #BEDS FROM BUFFER WHERE CLASS_B <> 0 UNION SELECT H_ID, EUROPEAN(DATE) AS EDATE, 'C', CLASS_C AS #BEDS FROM BUFFER WHERE CLASS_C <> 0;
4	V1.expr = V1.i_expr:= SELECT H_ID, EDATE, SUM(#BEDS) AS SUM_BEDS FROM CLASS_INFO GROUP BY H_ID, EDATE

Figure 3.12. Type reduction for the motivating example.

6.3 Deriving Role Interdependencies automatically

The *Roles* of the conceptual perspective can be directly assigned by the data warehouse administrator, or other interested stakeholders. Nevertheless, we can derive dependency relationships by exploiting the structure of the logical perspective of the metadata repository. Simple rules can derive the production of these interdependencies:

Depender Role	Dependee Role	As	Wrt
Activity	input types	dependent	schema of input types
Type	populating activity	dependent	schema of type
ConceptualObject	ConceptualObject	dependent	the set of fields wrt. which the dependee depends (transitivity of dependency)

Figure 3.13. Role Interdependencies

The following rule defines the first rule in ConceptBase. The other rules are coded similarly.

```
Class Activity with
rule
       activityDependentOnOutput :
              $ forall ac/Activity t/Type r,r1/Role f/field
              exists c/Concept
              ((ac input t) and (ac role r) and
              (t modeledFrom c) and (c field f)) ==>
              ((r relates c) and (r1 in r!relates) and
              (r1 As 'dependent') and (r1 wrt f)) $
    end
```

In our motivating example, this rule would produce the following result:

Depender Role	**Dependee Role**	**As**	**Wrt**
Load	CBL	dependent	CBL.*
Clean	Buffer	dependent	Buffer.*
Compute	Buffer	dependent	Buffer.*
Aggregate	Class_Info	dependent	H_ID, EDATE, #BEDS

Figure 3.14. Role interdependencies for the motivating example.

The rest of the interdependencies can be produced similarly. One could use external algorithms to produce also the possibility of reusing a type for the computation of a materialized view (*suitability* relationships).

6.4 Exploitation of the quality modeling in the repository for the administration of the data warehouse

The information stored in the repository may be used to find deficiencies in data warehouse. To show how the quality model is exploited, we take the following query. It returns all data cleaning activities related to data stores whose availability has decreased, according to the stored measurements. The significance of the query is that it can show that the implementation of the data cleaning process has become inefficient.

```
GenericQueryClass DecreasedAccuracy isA DWCleaningAgent with
parameter
       ds : DataStore
constraint
       c : $ exists qf1,qf2/DataStoreAccuracy t1,t2,t3/Commit_Time
                     v1,v2/Integer
              (qf1 onObject ds) and (qf2 onObject ds) and
              (this affects qf1) and (this affects qf2) and
              (this executedOn t3) and (qf1 when t1) and (qf2 when t2)
              and (t1 < t2) and (t1 < t3) and (t3 < t2) and
              (qf1 achieved v1) and (qf2 achieved v2) and (v1 > v2) $
   end
```

The query has a data store as parameter, i.e. the query will return only cleaning processes that are related to the specified data store. The query returns the agents which have worked on the specified data store and which were executed between the measurements of quality factors *qf1* and *qf2*, and the measured value of the newer quality factor is lower than the value of the older quality factor.

6.5 Repository support for data warehouse evolution

The data warehouse is a constantly evolving environment. A data warehouse is usually built incrementally. New sources are integrated in the overall architecture from time to time. New enterprise and client data stores are built in order to cover novel user requests for information. As time passes by, users seem more demanding for extra detailed information. Due to these reasons, not only the structure but also the processes of the data warehouse evolve.

Thus, the problem that arises is to keep all the data warehouse objects and processes consistent to each other, in the presence of changes. For example, suppose that the definition of a materialized view in the data warehouse changes. This change triggers a chain reaction in the data warehouse: the update process must evolve (both at the refreshment and the cleaning steps), the old, historical data must be migrated to the new schema (possibly with respect to the new selection conditions too) and all the data stores of the data warehouse and client level which are populated from this particular view must be examined with respect to their schema, content and population processes.

In our approach, we distinguish two kinds of impact:

- *Direct impact*: the change in the data warehouse object imposes that some action must be taken against an affected object. For example when an attribute is deleted from a materialized view, then the activity which populates it must also be changed accordingly.

- *Implicit impact:* the change in the data warehouse object might change the semantics of another object, without obligatorily changing the structure of the latter.

We have shown in a previous subsection, that we can produce a set of previous and successor objects for each materialized view (Type) in a data warehouse. Suppose that a type t is characterized by an expression e which is supported by a set of auxiliary SQL expressions producing, thus the set $\mathbf{e}=\{e_1,e_2,\ldots,e\}$. Obviously some of the expressions belonging to **e** belong also to $\mathbf{B}(t)$. Thus, we extend $\mathbf{B}(t)$ as $\mathbf{B}(t)\cup\mathbf{e}$ (with set semantics). Suppose, then, that the final SQL expression of a type t, say e, changes into e'. Following the spirit of [GuMR95], we can use the following rules for schema evolution in a DW environment (we consider that the changes abide by the SQL syntax and the new expression is valid):

- If the *select clause* of e' has an extra attribute from e, then propagate the extra attribute down the line to the base relations: there must be at least one path from one type belonging to a *SourceSchema* to an activity whose out expression involves the extra attribute. In the case where we delete an attribute from the select clause of a Type, then it must not appear in the select clause at least of the processes that directly populate the respective type, as well as in the following Types and the processes that use this particular Type. In the case of addition of an attribute, the impact is direct for the previous objects $\mathbf{B}(t)$ and implicit for the successor objects $\mathbf{A}(t)$. In the case of deletion the impact is direct for both categories.
- If the *where clause* of e' is more strict than the one of e, then the *where clause* of at least one process belonging to $\mathbf{B}(t)$ must change identically. If this is not possible, a new process can be added just before t simply deleting the respective tuples through the expression e'-e. If the *where clause* of e' is less strict than the one of e, then we can use well known subsumption techniques [SDJL96, LMSS95, GuHQ95, NuSS98] to determine which types can be (re)used to calculate the new expression e' of t. The *having clause* is treated in the same fashion. The impact is direct for the previous and implicit for the successor objects.
- If an attribute is deleted from the *group by clause* of e, then at least the last activity performing a *group-by* query should be adjusted accordingly. All the consequent activities in the population chain of t must change too (as if an attribute has been deleted). If this is not feasible we can add an aggregating process performing this task exactly before t. If an extra attribute is added to the *group by clause* of e, then at least the last activity performing a *group by* query should be adjusted accordingly. The check is performed recursively for the types populating this particular type, too. If this fails, the subsumption techniques mentioned for the *where-clause* can be used for the same purpose again. The impact is direct both for previous and successor objects. Only in the case of attribute addition it is implicit for the successor objects.

We do not claim that we provide a concrete algorithmic solution to the problem, but rather, we sketch a methodological set of steps, in the form of suggested actions to perform this kind of evolution. Similar algorithms for the evolution of views in data warehouses can be found in [GuMR95, Bell98]. A tool could easily visualize this evolution plan and allow the user to react to it.

7. CONCLUSIONS

This chapter describes a metamodel for data warehouse operational processes. This metamodel enables data warehouse management, design and evolution based on a high level conceptual perspective, which can be linked to the actual structural and physical aspects of the data warehouse architecture. The proposed metamodel is capable of modeling complex activities, their interrelationships, the relationship of activities with data sources and execution details. Finally, the metamodel complements proposed architecture and quality models in a coherent fashion, resulting in a full framework for data warehouse metamodeling. We have implemented this metamodel using the language Telos and the metadata repository system ConceptBase.

In this chapter, we have dealt only with the operational processes of a data warehouse environment. Yet, there are also design processes in such an environment, which do not seem to fit this model so smoothly. It is in our future plans to investigate the modeling of design processes and to capture the trace of their evolution in a data warehouse. Also, we have used the *global-as-view* approach for the data warehouse definition, i.e., we reduce the definition of the data warehouse materialized views to the data sources. We plan to investigate the possibility of using the *local-as-view* approach (which means reducing both the view definitions and the data sources to a global enterprise model), as it appears to provide several benefits that the global-as-view approach lacks [CDL*99].

In the next chapter we will complement our approach by providing a methodology for the actual exploitation of the information found in the metadata repository and the quality-oriented evolution of a data warehouse based on the architecture and quality model.

Chapter 4
Data Warehouse Repository Exploitation

1. INTRODUCTION

In Chapter 2, a metadata modeling approach has been 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 chapter, we refine the quality metamodel with a more detailed linkage between objective quality factors and user-dependent quality goals. Moreover, we extend the Goal-Question-Metric (GQM) methodology [BaCR94] 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. The chapter is based on the work presented in [VaBQ99].

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 to achieve reasonable quality level which fits the best user requirements.

We want to stress out that we do not follow the ISO 900x paradigm [ISO92] 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 chapter is to show how subjective quality goals can be evaluated using more objective quality factors, following an extended GQM approach.

The chapter is organized as follows: Section 2 gives some general definitions, using an example for the instantiation of the architecture and quality metamodel. In Section 3, we detail the proposed 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 contrasts our approach to related work and finally, in section 7 we summarize our results.

2. LINKAGE TO THE ARCHITECTURE AND QUALITY MODEL

2.1 Architecture and Quality Model

In Chapter 2, we have presented an architecture and quality model, acting as a framework for the metadata of the data warehouse repository. The proposed *architecture metamodel* (i.e., the topmost layer in Figure 2.1) provides a notation for data warehouse generic entities, such as schema, or agent, including the business perspective. Each box shown in Figure 2.1 is decomposed into more detailed data warehouse objects in the metamodel of [JJQV99]. This metamodel is instantiated with the *metadata* of the data warehouse (i.e., the second layer in Figure 2.1), e.g. relational schema definitions or the description of the conceptual data warehouse model. The lowest layer in Figure 2.1 represents the real world where the actual processes and data reside: in this level the metadata are instantiated with data instances, e.g., the tuples of a relation or the objects of the real world which are represented by the entities of the conceptual model.

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 meta-modeling approach. Thus, the *quality metamodel* is part of the metadata repository, and quality information is explicitly linked with architectural objects. This way, stakeholders can represent their quality goals explicitly in the metadata repository, while, at the same time, the relationship between the measurable architecture objects and the quality values is retained.

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

Based on the meta-model of data warehouse architecture, we have developed a set of quality factor templates, which can be used as an initial set for data warehouse quality management. The exhaustive list of these templates can be found in [QJJ*98]. The following section gives an intuition of some of them, which are associated to the data warehouse refreshment process.

We want to stress that our approach exploits the fact that data warehouse objects are related to quality factors and measurements. Thus it is orthogonal to any quality metamodel, as far as it fulfills this requirement. Examples of such a metamodel are the metamodel of Chapter 2, or the metamodel presented in [JeQJ98].

2.2 Quality metamodel instantiation: the refreshment case

The refreshment process is one of the main data warehouse processes for which the quality is an important issue. In this subsection, we will exploit the work performed in [BFMS98] to give an example of how quality factors interplay with each other. Moreover, we will provide a categorization for different kinds of quality factors, based on this kind of interplay.

The quality template associated with the refreshment process 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 to do 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. However, the quality factors are not necessarily independent of each other, e.g., completeness and coherence may induce a certain accuracy of data. We discriminate between *primary* and *derived* quality factors as well as *design choices*. A primary quality factor is a

simple estimation of a stakeholder or a direct measurement. For 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 subjective value directly assigned by the data warehouse administrator or the source administrator. On the other hand, derived quality factors are computed as formulae over some other quality factors: for example, the completeness of the operational data store content can be defined as a formula over the completeness of the sources. The design choices are a special kind of quality factors, expressed as parameter values and control strategies which aim to regulate or tune the algorithm followed for the performance of each task in the data warehouse.

Quality Dim.	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
Complete-ness	• Sources • ODS	• Availability window of each source • History duration for each DW store	• 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 4.1. Different levels of abstraction for the management of quality for the refreshment of the data warehouse.

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. In Table 4.1, 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, others are derived. The deriving procedures can be either 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.

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 [Bacr94,OiBa92]: 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 proposed 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 4.1, 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 [Wang98]. 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.

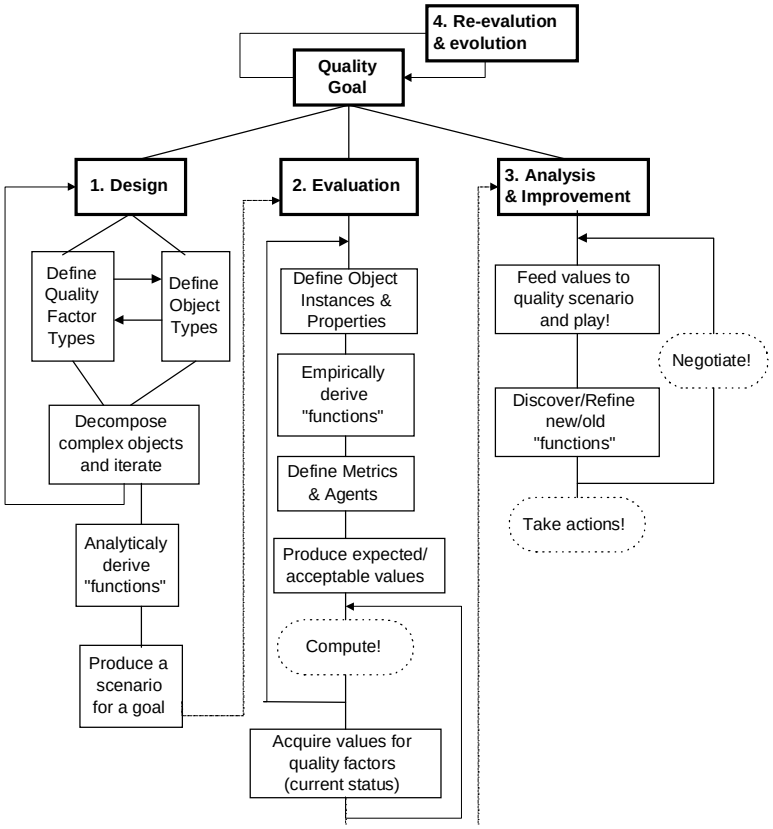


Figure 4.1. The proposed methodology for data warehouse quality management

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 its 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 metamodel 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 4.1). 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, 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. It is not our intention to detail these results here, but rather we refer the interested reader to [QJJ*98]. Second, these interrelationships, found at 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 of 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 figure (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 [BBBB95], 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.
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. We also refer the interested reader to results concerning the data warehouse refreshment [ThBo99] and design [LSTV99,ThLS99,ThSe97,ThSe99] problems that present such algorithms.
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. In the example of Figure 4.2, 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.

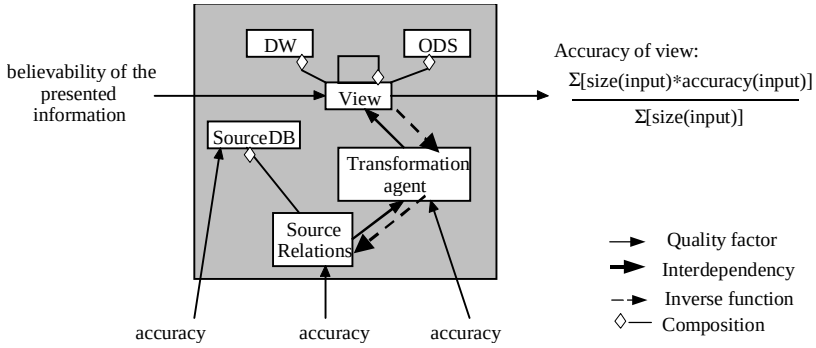


Figure 4.2. The scenario of a quality goal

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 4.2, 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.

First, we must *determine the specific object instances* of the specific evaluation through a query to the metadata repository. In the example of Figure 4.2, 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), the accuracy of which we have to quantify (Figure 4.3).

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. [BBBB95]. 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.

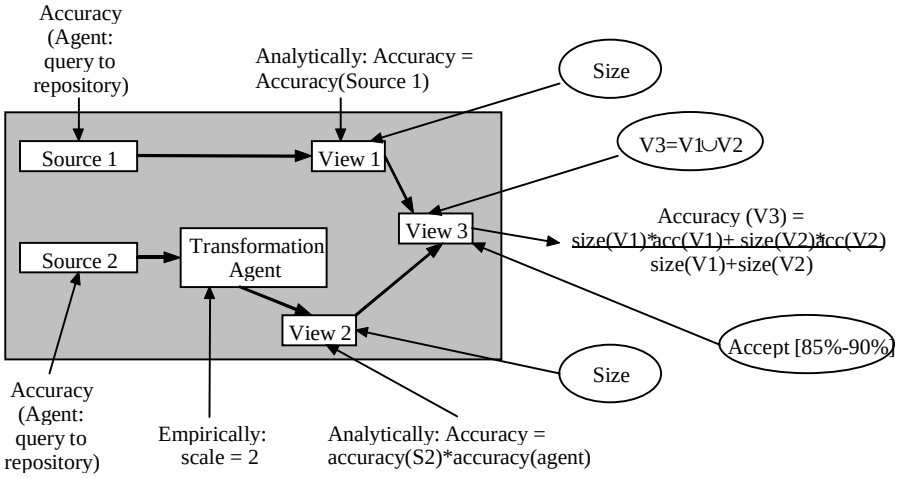


Figure 4.3. The map of a quality goal

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 4.3).

With respect to the scenario of Figure 4.2, 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 3).

After that, the only thing left is the acquisition/calculation of the specific values of the selected quality factors, though the necessary computation. In Figure 4.4, 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.

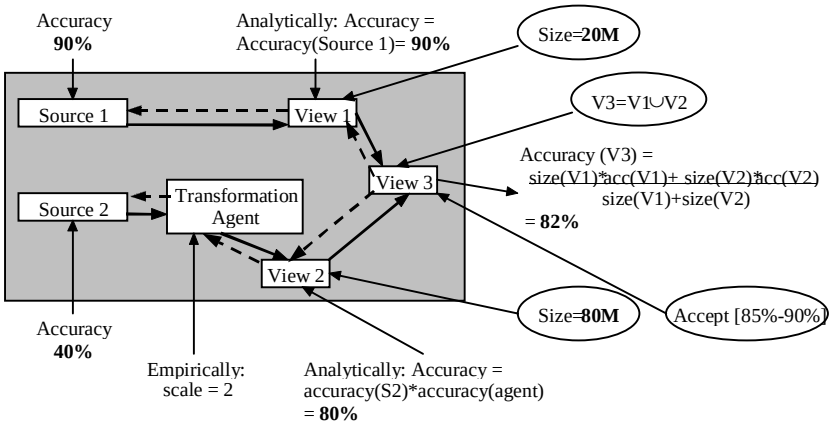


Figure 4.4. Instance of a quality map

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.

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 4.4). 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 4.4, where the computed accuracy for view 3 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 3 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 4.5, the final map instance of the motivating example is depicted, according to the second proposed solution (b).

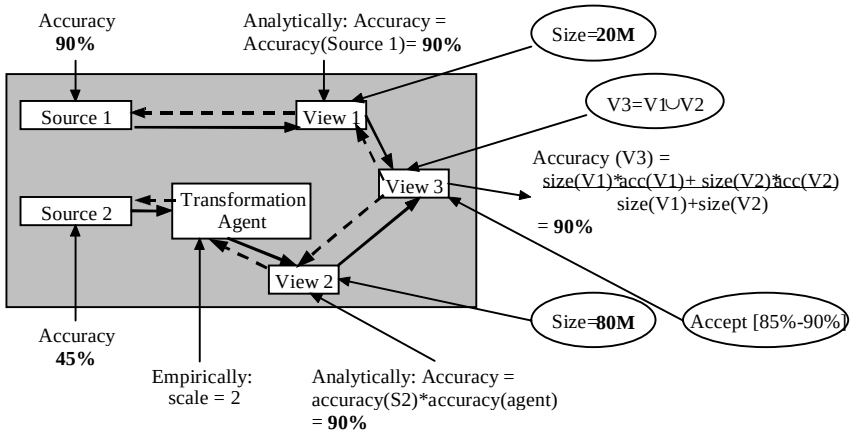


Figure 4.5. Final instance of a quality map

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 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. [BBBB95].

4. DATA WAREHOUSE EVOLUTION

Complementing the three previous steps of the proposed 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*. We adopt the approach of [JeQJ98], where *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 subsection 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 subsection 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 a data store, or specific reasons due to their nature (e.g., in the sources, the time window for extraction or the data entry process can change). The software components can be upgraded, completed, debugged, etc. The propagation agents of all types (e.g., loaders, refreshers, wrappers, mediators and source integrators) can obtain new schedules, new algorithms, rules, physical properties, etc. 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 [ThLS99], 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 such an evolving context of the data warehouse, the re-evaluation of a goal and of the strategy to achieve it is a strict contingency in a data warehouse environment. 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 fresh 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, 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 [JaJR90].

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.

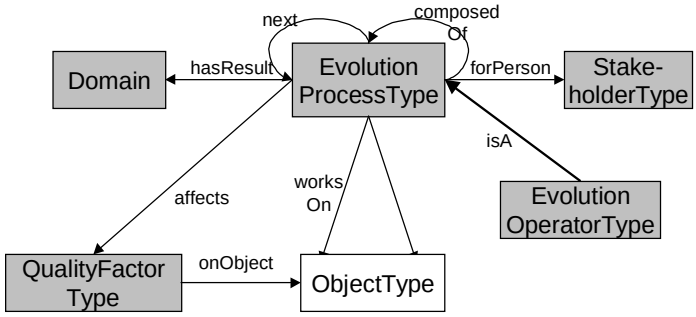


Figure 4.6. A meta model for data warehouse evolution [Quix99].

In this subsection we show how we can combine our ex ante approach with the metamodel for data warehouse evolution, proposed in [Quix99]. The metamodel of [Quix99] tries to document the semantics of evolution operations and to their effect on data warehouse quality in an ex post fashion. In this proposal, 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 [ClNR99], 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 4.2 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 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.

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 4.2. Evolution operators for base relations and views in data warehouses and their effect on data warehouse quality [Quix99].

5. CASE STUDY

To further demonstrate the power 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 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. We focus on two COBOL files, the first dealing with the yearly information for the hospitals by department of a hospital and the second with the yearly information by class of beds. 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.

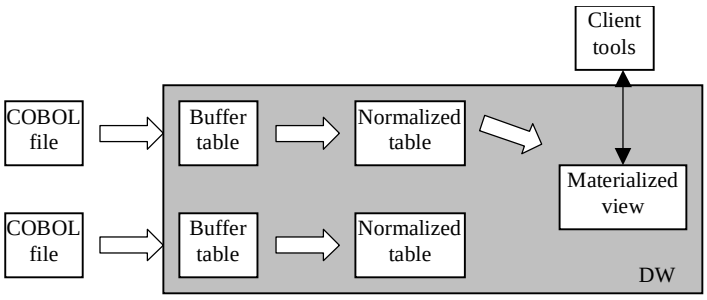


Figure 4.7. The data flow in the architecture of the case study

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 presentation:

- *Data Completeness.* All the information for all the hospitals should be in place after the yearly refreshment process.
- *Data Consistency.* Any information presented to the users should be identical to the one that would be presented 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: *"Achieve 100% data consistency for the data warehouse materialized views"*. In the sequel, we will show the results and the actions undertook in the context of each step of the methodology.

5.3 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). Note that the loading process that we previously described followed 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. The *loading*, *cleaning* and *computing* applications that were used were also of great importance. The identified quality factors which were of importance are mainly the *correctness* of the developed software applications and the *completeness* and *consistency* of the involved data stores.

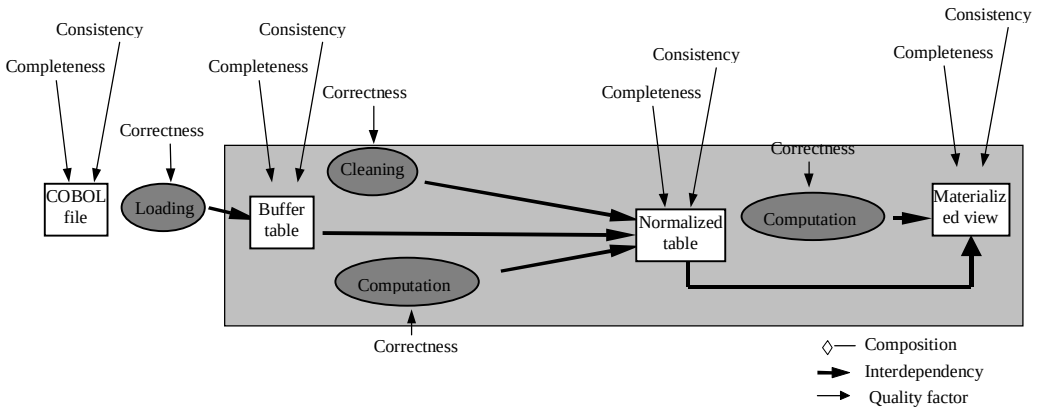


Figure 4.8. The scenario of the quality goal of the case study

The analytical functions we employed were rather naive: 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*

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 Map of the quality goal and analysis of the situation

The construction of the map of the quality goal was rather easy, due to the simplicity of the processes involved. We had to define metrics and agents to test the assigned quality factors. The task was dealt with naive techniques: white box testing would examine the correctness of deployed applications. 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. As one can notice in Figure 4.9, 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". Initially, we performed the computation of the hospital beds from the tables involving the "class of beds". We used the "inverse functions" to test the application: we started testing from the application performing the load / update of the materialized view, then tested the consistency of the "normalized" tables, etc. 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. The mystery was solved when the source administrators verified – or better, admitted – 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.

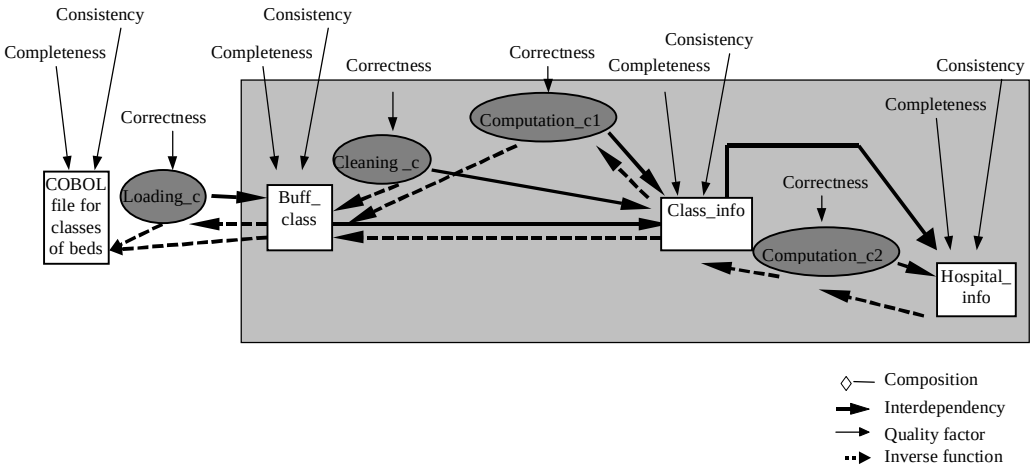


Figure 4.9. The map of the case study.

The final map of the problem is depicted in Figure 4.10. In Figure 4.10 one can notice that the new computation process "Computation_d2" performs the computation of the materialized view "Hospital_Info".

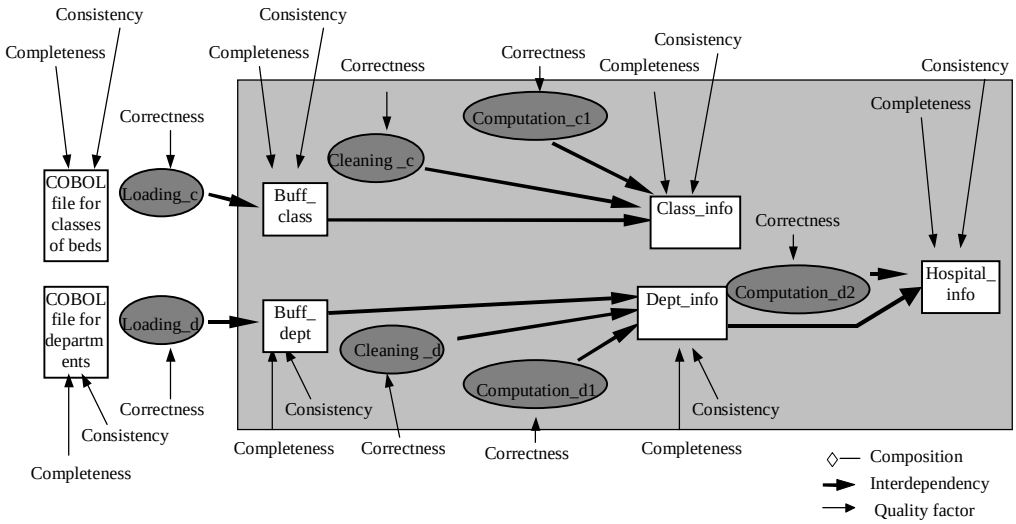


Figure 4.10. The final map instance of the case study.

6. RELATED WORK

The *TDQM* methodology [Wang98] 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. We will make a quick summary in the rest of this subsection and refer the interested reader to Chapter 1 for a detailed description. *Statistical Process Control* (SPC) is one of the best tools for monitoring and improving product and service quality [BBBB95]. 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) [Dean97,BBBB95] 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 [LSTV99,ThLS99,ThSe97,ThSe99] and (b) the tuning of the refreshment process based on the quality factors of section 2.3 [ThBo99].

7. SUMMARY

In this chapter, we deal with the problem of quality-oriented design, usage and evolution of data warehouses. Following the approach of previous work [JeQJ98,JJQV99], 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.
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, in one hand, to map a high-level subjective quality goal into the measurement of a set of interrelated quality factors, and, in 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 4.3 we can clearly depict this fact.

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 4.3. Different levels of abstraction for the management of quality in a data warehouse

The *subjective*, *user-oriented* GQM *Goal*, shown in the first row of Table 4.3, 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 4.3. 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 chapter 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.

Chapter 5
Modeling Multidimensional Databases

1. INTRODUCTION

The data warehouse metadata framework, which is presented in Chapter 2, divides the space of data warehouse metamodels in nine areas, produced by the combination of three *levels* and three *perspectives*. The levels deal with the location of data: the *source* level covers the production systems that provide data to the warehouse; the *primary data warehouse* level acts as a uniform container of meta-information and data; and finally, the *client* level deals with customized, aggregated information (along with the respective schemata), directly accessible to the user. The perspectives, on the other hand, follow a more traditional separation: the *conceptual* perspective deals with the high level representation of the world; the *physical* perspective deals with the details of the representation of the information in the hardware; and the *logical* perspective, which acts as an intermediate between the two aforementioned extremes, tries to balance a storage-independent paradigm and a natural representation of the information in terms of computer-oriented concepts.

On-Line Analytical Processing (OLAP) is a trend in database technology, based on the multidimensional view of data, which is employed at the client level. While several conceptual (e.g. [BaSa98],[Kimb96]) and physical (e.g. [Sara97]) models exist, it has been argued that traditional logical data models (e.g., the relational one) are in principle not powerful enough for data warehouse applications, and that data cubes provide the functionality needed for summarizing, viewing, and consolidating the information available in data warehouses [JLVV99]. Despite this consensus on the central role of multidimensional data cubes, and the variety of the proposals made by researchers, there is little agreement on finding a common terminology and semantic foundations for a logical data model.

Several industrial standards already exist [OLAP97a, TPC98, Meta97, Micr98], yet, apart for the last one, none of them seems to propose a well-founded model for OLAP databases. In academia, several proposals on the modelling of cubes also exist [AgGS95, LiWa96, GyLa97, BaPT97, CaTo97, Lehn98, Vass98]. Despite all these efforts, we feel that several key characteristics of a cube model have not been stressed, neither by the academia nor the industry. To this end, we present a *logical* model for cubes. This model extends the proposal of [Vass98] in a more formal and systematic way. It deals with all the commonly encountered entities of a multidimensional model (dimension hierarchies, data cubes and cube operations) without being restricted from their physical implementation (e.g., ROLAP or

MOLAP architectures). One of our key observations is that *a cube is not a self-existing entity, but rather a view* (materialised or not) *over an underlying data set*. This property allows us to develop complex operations, not dealt by other models so far (e.g., the drill-down operation and the change of aggregation function).
To our knowledge, existing OLAP tools behave in an "extensional" fashion. Cubes are treated simply as sets of tuples, ignoring the fact that they are produced as queries over an underlying detailed data set (e.g., the fact table of a data warehouse). Our framework, instead, suggests a different strategy: we keep the "history" of performed selections and thus, we are able to compute a new cube taking into account its "intentional" description. Therefore, we can define more complex operations (such as drill-down) and sequences of operations, which are not covered by other models. Our model is accompanied by an algebra powerful enough to capture the usual OLAP operations such as (a) *selection* over a cube, (b) *roll-up*, which means aggregation over a cube to coarser granularities of information and (c) *drill-down*, which involves de-aggregation of a specific cube and presentation of more detailed information.
The contribution of this chapter lies not only in terms of expressiveness, but also we present results on optimization issues for multidimensional databases. We investigate the *cube usability problem*, a variant of the relational view usability problem, for multidimensional cubes. We accompany our framework with optimization techniques for the cube usability problem that enable the exploitation of existing cubes in order to compute new cubes. To handle the cube usability problem, we extend well-known techniques already found in the relational context on the containment of selection conditions [Ullm89]. We have observed that although quite a lot of work has been performed in the field of query containment and view usability in the context of relational databases [DJLS96, GuHQ95, NuSS98, CKPS95], there exist no results to exploit the information about dimension hierarchies in the context of multidimensional databases. We present results on two major topics. First, we tackle the problem of containment of two selections, taking into account their marginal conditions in the presence of dimension hierarchies. Secondly, we come up with a set of axioms to characterize containment for expressions involving functionally dependent attributes. Although several results already exist to characterize query containment between expressions involving one domain [Ullm89], to our knowledge, no results exist for expressions involving different functionally dependent levels. The results that we present are based on [VaSk99, VaSe99] and were achieved in cooperation with my colleague Spiros Skiadopoulos in the National Technical University of Athens.
This chapter is organised as follows. In Section 2 we make an overview of the related work. Then, in Section 3 we present the logical cube model. Finally, Section 4 presents optimisation issues.

2. RELATED WORK

In this section we will focus on the presentation of different proposals for *multidimensional data cubes*, which are the basic *logical* model for OLAP applications.
We have proceeded in the following categorization of the work in the field: on the one hand there are the commercial tools -which actually initiated the work in the field; we present them first, along with terminology and standards, in subsection 2.2. On the other hand there are academic efforts, which are mainly divided in two classes: the relational model extensions and the cube-oriented approaches. We present the former in subsection 2.3 and the latter in subsection 2.4. Finally, in subsection 2.5, we attempt a comparative analysis of the various efforts.

2.1 Terminology, Products and Standards

2.1.1 Terminology

A good definition of the term OLAP is found in [OLAP97]: "…On-Line Analytical Processing (OLAP) is a category of software technology that enables analysts, managers and executives to gain insight into data through fast, consistent, interactive access to a wide variety of possible views of information that has been transformed from raw data to reflect the real dimensionality of the enterprise as understood by the user. OLAP functionality is characterized by dynamic multidimensional analysis of consolidated enterprise data supporting end user analytical and navigational activities including calculations and modeling applied across dimensions, through hierarchies and/or across members, trend analysis over sequential time periods, slicing subsets for on-screen viewing, drill-down to deeper levels of consolidation, rotation to new dimensional comparisons in the viewing area etc. …". A standard terminology for OLAP is provided by the OLAP Council [OLAP97].

The focus of OLAP tools is to provide multidimensional analysis to the underlying information. To achieve this goal, these tools employ multidimensional models for the storage and presentation of data. Data are organized in *cubes* (or *hypercubes*), which are defined over a multidimensional space, consisting of several dimensions. Each dimension comprises of a set of aggregation levels. Typical OLAP operations include the aggregation or de-aggregation of information (*roll-up* and *drill-down*) along a dimension, the *selection* of specific parts of a cube and the re-orientation of the multidimensional view of the data on the screen (*pivoting).*

2.1.2 Products and Technologies

The debate on the underlying physical model, supporting OLAP, is centered around two major views. Whereas some vendors, especially vendors of traditional relational database systems (RDBMS), propose the *ROLAP architecture* (Relational On-Line Analytical Processing) [MStr95, MStr97, Info97, RBSI97], others support the *MOLAP architecture* (Multidimensional On-Line Analytical Processing) [Arbo96]. The advantage of the MOLAP architecture is, that it provides a direct multidimensional view of the data whereas the ROLAP architecture is just a multidimensional interface to relational data. On the other hand, the ROLAP architecture has two advantages: (a) it can be easily integrated into other existing relational database systems, and (b) relational data can be stored more efficiently than multidimensional data.

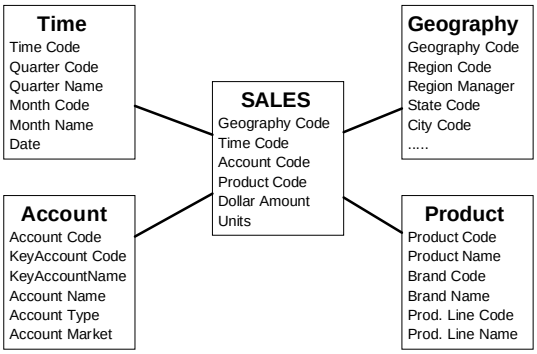


Figure 5.1. Star schema [STGI96]

In a ROLAP architecture, data are organized in a *star* (Figure 5.1) or *snowflake* schema. A star schema consists of one central *fact* table and several denormalized *dimension* tables. The *measures* of interest for OLAP are stored in the fact table (e.g. Dollar Amount, Units in the table SALES). For each dimension of the multidimensional model there exists a dimension table (e.g. Geography, Product, Time, Account) with all the levels of aggregation and the extra properties of these levels. The normalized version of a star schema is a snowflake schema, where each level of aggregation has its own dimension table.
Multidimensional database systems (MDBMS) store data in multidimensional arrays. Each dimension of the array represents the respective dimension of the cube. The contents of the array are the measure(s) of the cube. MDBMS require the precomputation of all possible aggregations: thus they are often more performant than traditional RDBMS [Coll96], but more difficult to update and administer.

2.1.3 Benchmarks and Standards

The OLAP Council has come up with the *APB-1 benchmark* [OLAP97a] for OLAP databases. The APB-1 benchmark simulates a realistic OLAP business situation that exercises server-based software. The standard defines a set of dimensions with respect to their logical perspective. The logical database structure is made up of six dimensions: *time*, *scenario*, *measure*, *product*, *customer*, and *channel*. The benchmark does not assume a specific underlying physical model: the input data are provided in the form of ASCII files. The operations nicely simulate the standard OLAP operations and include bulk and incremental loading of data from internal or external data sources, aggregation or drill-down of data along hierarchies, calculation of new data based on business models, etc.
The *TPC-H* and *TPC-R benchmarks* [TPC99] (replacing the older TPC-D benchmark) model a decision support environment in which complex, ad hoc, (pre-canned respectively) business-oriented queries are

submitted against a large database. TPC-H and TPC-R use the same hybrid star and snowflake schema, involving several dimension and fact tables. The benchmarks are definitely relational-oriented: there is no explicit treatment of cubes and dimension hierarchies. Of course, one can always deduce them implicitly from the underlying schema; nevertheless, the dimensions seem too simple in their structure and depth. Each benchmark is accompanied by a set of queries which seem to be close to the usual queries in a DSS environment. These queries do not seem to fit the pattern of typical OLAP operations, which are sequential and interactive in their nature.

The *OLEDB for OLAP* [Micr98] standard has been developed by Microsoft as a set of COM objects and interfaces, destined to provide access to multidimensional data sources through OLEDB. OLEDB for OLAP employs a model for cubes and dimensions, that supports the logical notions already explained in section 2.1. Moreover, it provides a language of *MultiDimensional eXpressions* (MDX) for the calculation and presentation of cubes. OLEDB for OLAP provides a good intuition on the entities comprising a multidimensional database; nevertheless it has several disadvantages: it lacks a solid theoretical background (e.g. there is no definition of the *schema* of a multicube) and combines *presentational* with *computational* issues. The result is a complex and, to some extent, hard to use (although powerful enough) language.

The Metadata Coalition, an open group of companies such as IBM, Sybase, Informix, etc, proposed the *Metadata Interchange Specification* [Meta97]. The Metadata Interchange Specification (MDIS) provides a standard access mechanism and a standard application-programming interface to control and manage metadata with interchange specification-compliant tools. MDIS tries to present a metadata metamodel for a wide set of database models (relational, object-oriented, entity-relationship, etc.), with a model for multidimensional databases belonging to this set. The model proposed by MDIS supports the notion of dimension, which just comprises a set of levels. Cubes are not directly modeled in the MDIS model.

2.2 Relational Extensions

2.2.1 Models for OLAP

The *data cube* operator was introduced in [GBLP96]. The *data cube* operator expands a relational table, by computing the aggregations over all the possible subspaces created from the combinations of the attributes of such a relation. Practically, the introduced *CUBE* operator calculates all the marginal aggregations of the detailed data set. The value '*ALL*' is used for any attribute, which does not participate in the aggregation, meaning that the result is expressed with respect to all the values of this attribute.

In [LiWa96] a multidimensional data model is introduced based on relational elements. Dimensions are modeled as *dimension relations*, practically annotating attributes with dimension names. Cubes are modeled as functions from the cartesian product of the dimensions to the measure and are mapped to *grouping relations* through an applicability definition. A grouping algebra is presented, extending existing relational operators and introducing new ones, such as ordering and grouping to prepare cubes for aggregations. Furthermore, a multidimensional algebra is presented, dealing with the construction and modification of cubes as well as with aggregations and joins. For example, the operator *roll* is almost a monotone roll-up. Finally, a relation can be grouped by intervals of values; the values of the "dimensions" are ordered and then "grouped by", using an auxiliary table.

In [BaPT97], multidimensional databases are considered to be composed from sets of tables forming denormalized star schemata. Attribute hierarchies are modeled through the introduction of functional dependencies in the attributes of the dimension tables. Nevertheless, this work is focused on the data warehouse design optimization problem and not on the modeling of cubes or cube operations.

In [GyLa97] *n-dimensional* tables are defined and a relational mapping is provided through the notion of *completion*. An algebra (and an equivalent calculus) is defined with classical relational operators as well as restructuring, classification and summarization operators. The expressive power of the algebra is demonstrated through the modeling of the data cube and monotone roll-up operators.

In [GiLa98] a new extension of the relational model and a new language are proposed. The underlying model is an extension of the relational model to handle *federated names*. A complex name is a pair, comprising of a name (or concept) and a finite set of associated criteria set, relating the concept to a common, global set of criteria. An extension of SQL, *nD-SQL* is also provided, along with its mapping to an extension of the relational algebra. The applicability of the language to OLAP operations is shown through a set of examples, practically modeling the *CUBE* operator of [GBLP96]. The authors give different semantics to the *ROLLUP* and *DRILLDOWN* operators than the ones we give here. Moreover, results on the optimization of the execution of queries are also provided.

2.2.2 Relationship with Statistical Databases

A lot of relevant work has been done in the past in the area of *statistical databases* [Shos97]. In [Shos97] a comparison of work done in statistical and multidimensional databases is presented. The comparison is made with respect to application areas, conceptual modeling, data structure representation, operations, physical organization aspects and authorization/security issues. The basic conclusion of this comparison is that the two areas have a lot of overlap, with statistical databases emphasizing on conceptual modeling and OLAP emphasizing on physical organization and efficient access.
In [OzOM85, OzOM87] a data model for statistical databases is introduced. The model is based on *summary tables* and operators defined on them such as construction/destruction, concatenation/extraction, attribute splitting/merging and aggregation operators. Furthermore, physical organization and implementation issues are discussed. [OzOM85] is very close to practical OLAP operations, although discussed in the context of summary tables.
In [RaRi91] a functional model ("Mefisto") is presented. Mefisto is based on the definition of a data structure, called "statistical entity" and on operations defined on it like summarization, classification, restriction and enlargement.

2.3 Cube-Oriented Models

There have been efforts to model directly and more naturally multidimensional databases; we call these efforts *cube-oriented*. This does not mean that they are far from the relational paradigm – in fact all of them have mappings to it – but rather that their main entities are *cubes* and *dimensions*.
In [AgGS95], a model for multidimensional databases is introduced. The model is characterized from its symmetric treatment of *dimensions* and *measures*. A set of minimal (but rather complicated) operators is also introduced dealing with the construction and destruction of cubes, join and restriction of cubes, and merging of cubes through direct dimensions. Furthermore, an SQL mapping is presented.
In [CaTo97], a multidimensional database is modeled through the notions of *dimensions* and *f-tables*. Dimensions are constructed from hierarchies of *dimension levels*, whereas f-tables are repositories for the factual data. Data are characterized from a set of *roll-up* functions, mapping the instances of a dimension level to instances of another dimension level. A query language is the focus of this work: a calculus for f-tables along with scalar and aggregate functions is presented, basically oriented to the formulation of aggregate queries. In [CaTo98a] the focus is on the modeling of multidimensional databases: the basic model remains practically the same, whereas ER modeling techniques are given for the conceptual modeling of the multidimensional database. A mapping to physical entities such as relations and multidimensional arrays is provided. In [CaTo98b] a graphical query language as well as an equivalent algebra is presented. The algebra is a small extension to the relational algebra, including a roll-up operator, yet no equivalence to the calculus is provided.
In [Vass98] dimensions and dimension hierarchies are explicitly modeled. Furthermore, an algebra representing the most common OLAP operations is provided. The model is based on the concept of the *basic cube* representing the cube with the most detailed information (i.e. the information at the lowest levels of the dimension hierarchies). All other cubes are calculated as expressions over the basic cubes. The algebra allows for the execution of sequences of operations as well as for drill-down operations. A relational mapping is also provided for the model, as well as a mapping to multidimensional arrays.
In [Lehn98] another model is presented, based on primary and secondary multidimensional objects. A *Primary Multidimensional Object* (PMO), which represents a cube, consists of a cell identifier, a schema definition, a set of selections, an aggregation type (e.g. sum, avg, no-operator) and a result type. A *Secondary Multidimensional Object* (SMO) consists of all the dimension levels (also called "dimensional attributes") to which one can roll-up or drill-down for a specific schema. Operations like Roll-up, Drill-down, Slice, Dice etc. are also presented; yet not all of them are defined at the instance level. In [LeAW98], which is a sequel to the previous paper, two multidimensional normal forms are proposed, defining (a) modeling constraints for summary attributes and (b) constraints to model complex dimensional structures.
In [GeJJ97] the CoDecide model is – informally – presented. The so-called *tape model* consists of structured hierarchies called *tapes* (corresponding to dimensions). Each tape consists of a set of hierarchically interrelated *tracks* (corresponding to *levels*). The intersection of tracks defines a *multidimensional matrix*. Operations like roll-up and drill-down are defined for the tape model. It is important to note that the tape model can combine several matrices, defined as networks of crossing tapes. Moreover, the tape model is the lower part of a layered set of models, representing the logical perspective. On top of it, the *transformation*, *visualization* and *control* models are defined (*presentational* perspective).

2.4 Comparison

In this subsection, we present a comparison of the various models. The first list of requirements for logical cube models is found in [BSHD98]. In our approach we followed the discrimination between entities and operations and came up with three big categories of attributes for cube models. The first group of attributes deals with the representation of the multidimensional space: as usual, we check whether entities are modeled as cubes or tables (denoted by *C* or *T* respectively) and whether level hierarchies are modeled, or not. The second group of attributes deals with language issues: the character of the query language (procedural, declarative, visual), the direct support of sequences of operations and a subjective characterization of how naturally the classical OLAP operations are modeled. The third group is concerned with the existence of physical mappings to relations and/or multidimensional arrays.
In Table 1, 'SQL ext.' indicates extension of SQL, and N/A means that the information is not directly available in the material examined (papers).

		Multidimensional space		**Language aspects**					**Physical representation**	
		Cubes	*level hierarchies*	*Procedural QL*	*Declarative QL*	*Visual QL*	*Seq. of operations*	*natural repr.*	*relational mapping*	*m/d mapping*
Relational-Oriented	GBLP96	T			SQL ext.				✓	
	LiWa96	T	implicitly	Algebra				✓	✓	
	GyLa97	T	✓	Algebra	calculus				✓	
	GiLa98	T		✓	✓				✓	
	BaPT97	T	✓						✓	
Cube-Oriented	AgGS95	C		Algebra					✓	✓
	CaTo97, 98, 98a	C	✓	Algebra	calculus	✓		✓	✓	✓
	Vass98	C	✓	Algebra			✓	✓	✓	✓
	Lehn98, LeAW98	C	✓	Algebra				✓	✓	implic.
	GeJJ97	C	implicitly	N/A	N/A	✓		✓	N/A	N/A
Standards	APB-1	C	✓	Natural lang.				✓		
	TPC H/R	T			SQL				✓	
	OLEDB	C	✓	VB, C++ calls	SQL-like	MS Excel, Access		✓	✓	implic.
	MDIS	T	✓							
Statistical	OzOM85	T	implicitly	Algebra				✓	✓	
	RaRi91	T	implicitly	Algebra			✓	✓	✓	

Table 5.1. Comparison of the various cube models.

Clearly, a lot of interesting work can be expected in the area. The issue of reaching a consensus on the modeling issues is still open, both in the logical and the conceptual perspective. Devising a common standard declarative language is also of high importance. Moreover, there is potential for useful results, in the area of logical optimization and caching rules (in order to exploit the possibility of reusing existing cubes for the computation of new ones), through the use of a generic logical multidimensional model (independently from the underlying physical model).

3. CUBES FOR MULTIDIMENSIONAL DATABASES

In this section we present the basic entities and operations of the proposed model. Entities involve dimensions, data sets and cubes. Operations involve selections and change in the granularity of data. This model extents previous proposals of [Vass98, CaTo97, Lehn98].
One of the main characteristics of OLAP applications is the *multidimensional view of data* in the perception of the user, which considers that information is stored in a *multi-dimensional array*, called *Cube* or *HyperCube*. Thus, a *Cube* is a group of data cells. Each cell is uniquely defined by the corresponding values of the dimensions of the cube. The contents of the cell are named *measures* and

represent the measured values of the real world. Measures are functionally dependent, in the relational sense, on the dimensions of the cube.

A *dimension* is defined in [OLAP97] as "a structural attribute of a cube that is a list of members, all of which are of a similar type in the user's perception of the data". Informally, a dimension models all the possible ways in which the user can group the detailed information stored in the multidimensional database with respect to a specific context. Each dimension has an associated *hierarchy of levels* of aggregated data i.e., it can be viewed from different levels of detail. Formally, a *dimension* D is a lattice $(\mathbf{L}, \prec)$: $\mathbf{L}=(L_1, \ldots, L_n, ALL)$. We require that the upper bound of the lattice is always the level ALL, so that we can group all the values of the dimension into the single value 'all'. The lower bound of the lattice is called the *detailed level* of the dimension. For instance, let us consider the dimension Date of Figure 5.6. Levels of dimension Date are Day, Week, Month, Year and All. Day is the most detailed level. Level All is the most coarse level for all the dimensions. Aggregating to the level All of a dimension ignores the respective dimension in the grouping (i.e., practically groups the data with respect to all the other dimensions of the cube, except for this particular one).

The relationship between the values of the dimension levels is achieved through the use of the set of $anc_{L_1}^{L_2}$ functions. A function $anc_{L_1}^{L_2}$ assigns a value of the domain of L_2 to a value of the domain of L_1. For instance $anc_{Month}^{Year}(Feb\text{-}97)=1997$.

The major multi-dimensional operations are *selection* and *navigation*. Selection is used whereby a criterion is evaluated against the data or levels of a dimension in order to restrict the set of retrieved data. Navigation is a term used to describe the processes employed by users to explore a cube interactively by changing the granularity of the multi-dimensionally viewed data [JLVV99, OLAP97]. Possible navigation operations, which can be applied to a cube, are: (a) *Roll-up* which corresponds to the aggregation of data from a lower to a higher level of granularity within a dimension's hierarchy, (b) *Drill-Down* which is the inverse of roll-up and allows the de-aggregation of information moving from higher to lower levels of granularity and (c) *Slice* which corresponds to the grouping of data with respect to a subset of the dimensions of a cube. For instance, let us consider the dimension Date; aggregating from Month to Year is a roll-up operation and de-aggregating from Month to Day is a drill-down operation. In our model, the slice operation is modelled as a roll-up to level All.

In our model, we denote sets of tuples under a specific schema by the term *data set*. Moreover, we assume the existence of a *detailed data set*, i.e., a data set that is defined at the finest levels of granularity for all its dimensions. This detailed data set is the central source of data, which will populate any cubes produced during an OLAP session (e.g., a fact table in a data warehouse).

One of our key observations is that a *cube* is not a self-existing entity (as commonly encountered in the literature), but rather a view over an underlying detailed data set. As usual, a view (and thus a cube) can be either materialised or not. Therefore, a cube can be seen either as a data set or simply a query. In our model, we retain this dual nature formally; a cube is not only a set of tuples, but also has a definition. This definition is a query that reduces the computation of the cube to a set of operations over the initial materialised detailed data set.

Formally, a *cube* c over the schema $[L_1, \ldots, L_n, M_1, \ldots, M_m]$, is an expression of the form: $c=(DS^0, \varphi, [L_1, \ldots, L_n, M_1, \ldots, M_m], [agg_1(M_1^0), \ldots, agg_m(M_m^0)])$, where DS^0 is a detailed data set over the schema $S=[L_1^0, \ldots, L_n^0, M_1^0, \ldots, M_k^0]$, $m \leq k$, φ is a detailed selection condition, $M_1^0, \ldots, M_m^0$ are detailed measures, $M_1, \ldots, M_m$ are aggregated measures, L_i^0 and L_i are levels such that $L_i^0 \prec L_i$, $1 \leq i \leq n$ and agg_i, $1 \leq i \leq m$ are aggregated functions from the set {sum, min, max, count}.

Intuitively, to compute a cube, first we apply the selection condition to the detailed data set. Then, we replace the values of the levels for the tuples of the result, with their respective ancestor values (at the levels of the schema of and group them into a single value for each measure, through the application of the appropriate aggregate function. Note that a data set can be trivially expressed as a cube, having a true selection condition. For instance, the cube of the detailed data set DS^0 of Figure 5.2 is expressed as: $c^0=(DS^0, true, [day, day, item, salesman, city, sales], sum(sales))$.

This approach introduces a powerful expression mechanism, able to directly capture operations like drill-down and change of aggregate function and thus, aimed towards the modelling of sequences of operations, as normally encountered in OLAP systems. To our knowledge, no other model can capture these operations directly. The reduction of a cube's definition to a normalised form seems to be the only alternative that directly achieves this kind of functionality.

Formally, the model consists of the following elements:

– Each *dimension* D is a lattice $(\mathbf{L}, \prec)$ such that: $\mathbf{L}=(L_1, \ldots, L_n, ALL)$ is a finite subset of *Levels* and $\prec$ is a partial order defined among the levels of $\mathbf{L}$, such that $L_1 \prec L_i \prec ALL$ for every $1 \leq i \leq n$.

- A family of functions $anc_{L_1}^{L_2}$ satisfying the following conditions (extending [CaTo97]):
  1. For each pair of levels L_1 and L_2 such that $L_1 \prec L_2$ the function $anc_{L_1}^{L_2}$ maps each element of $dom(L_1)$ to an element of $dom(L_2)$.
  2. Given levels L_1, L_2 and L_3 such that $L_1 \prec L_2 \prec L_3$, the function $anc_{L_1}^{L_3}$ equals to the composition $anc_{L_1}^{L_2} \circ anc_{L_2}^{L_3}$.
  3. For each pair of levels L_1 and L_2 such that $L_1 \prec L_2$ the function $anc_{L_1}^{L_2}$ is monotone, i.e., $\forall x, y \in dom(L_1), L_1 \prec L_2: x<y \Rightarrow anc_{L_1}^{L_2}(x) \leq anc_{L_1}^{L_2}(y)$.
  4. For each pair of levels L_1 and L_2 the $anc_{L_1}^{L_2}$ function determines a set of finite equivalence classes X_i such that: $\forall x, y \in dom(L_1), L_1 \prec L_2: anc_{L_1}^{L_2}(x)=anc_{L_1}^{L_2}(y) \Rightarrow x, y$ belongs to the same X_i.
  5. The relationship $desc_{L_1}^{L_2}$ is the inverse of the $anc_{L_1}^{L_2}$ function -i.e., $desc_{L_1}^{L_2}(l)= \{x \in dom(L): anc_{L_1}^{L_2}(x)=l\}$.
- Each *data set* DS over a schema $S=[L_1, \ldots, L_n, M_1, \ldots, M_m]$ is a finite set of tuples over S such that: comprise a primary key (in the usual sense).
- Each *selection condition* φ is a formula in disjunctive normal form. An *atom* of a selection condition is `true`, `false` or an expression of the form $x \ \theta \ y$, where θ is an operator from the set $(>, <, =, \geq, \leq, \neq)$ and each of x and y can be one of the following: (a) a level L, (b) a value l, (c) an expression of the form $anc_{L_1}^{L_2}(L_1)$ where $L_1 \prec L_2$ and (d) an expression of the form $anc_{L_1}^{L_2}(l)$ where $L_1 \prec L_2$ and $l \in dom(L_1)$. The *detailed equivalent of* φ, denoted by φ^0, is a selection condition obtained through the following procedure: for each occurrence of a level name L in φ, we substitute it with the equivalent expression $anc_{L^0}^{L}(L^0)$, where L^0 is the detailed level of the dimension to which L belongs.
- Each *cube* c over the schema $[L_1, \ldots, L_n, M_1, \ldots, M_m]$, is an expression of the form: $c=(DS^0, \varphi, [L_1, \ldots, L_n, M_1, \ldots, M_m], [agg_1(M_1^0), \ldots, agg_m(M_m^0)])$, where DS^0 is a detailed data set over the schema $S=[L_1^0, \ldots, L_n^0, M_1^0, \ldots, M_k^0]$, $m \leq k$, φ is a detailed selection condition, $M_1^0, \ldots, M_m^0$ are detailed measures, $M_1, \ldots, M_m$ are aggregated measures, L_i^0 and L_i are levels such that $L_i^0 \prec L_i$, $1 \leq i \leq n$ and agg_i, $1 \leq i \leq m$ are aggregated functions from the set $\{sum, min, max, count\}$. The expression characterising a cube has the following formal semantics:

$$c=\{x \in Tup(L_1, \ldots, L_n, M_1, \ldots, M_m) \mid \exists y \in \varphi(DS^0), x[L_i]=anc_{L_i^0}^{L_i}(y[L_i^0]),\ 1 \leq i \leq n,$$
$$x[M_j]=agg_j(\{q \mid \exists z \in \varphi(DS^0), x[L_i]=anc_{L_i^0}^{L_i}(z[L_i^0]), 1 \leq i \leq n, q=z[M_j^0]\}), 1 \leq j \leq m\}.$$

The *Cube Algebra* (CA) is composed of three operations:

1. *Navigate:* Let $S=[L_1, \ldots, L_n, M_1, \ldots, M_m]$ be a schema and agg_1, …, agg_m be aggregate functions. If L_i^a and L_i belong to the same dimension D_i and $agg_i \in \{sum, min, max, count\}$ then, navigation is defined as follows:
   $$nav(c^a, S, agg_1, \ldots, agg_m)=(DS^0, \varphi^a, S, [agg_1(M_1^0), \ldots, agg_m(M_m^0)]).$$
2. *Selection:* Let φ be a selection condition applicable to c^a. Then, we define the *selection* operation as:
   $$\sigma_\varphi(c^a)=(DS^0, \varphi^a \wedge \varphi^0, [L_1^a, \ldots, L_n^a, M_1^a, \ldots, M_m^a], [agg_1(M_1^0), \ldots, agg_m(M_m^0)])$$
   where φ^0 is the detailed equivalent of the selection condition φ.
3. *Split measure:* Let M be a measure of the schema of the cube c. Without loss of generality, let us assume that M is M_m. Then *split measure* is defined as follows:
   $$\Pi_{M_m}(c^a)=(DS^0, \varphi^a, [L_1^a, \ldots, L_n^a, M_1^a, \ldots, M_{m-1}^a], [agg_1(M_1^0), \ldots, agg_m(M_{m-1}^0)]).$$

Example 3.1. To motivate the discussion we customise the example presented in [Micr98] to an international publishing company with travelling salesmen selling books and CD's to stores all over the world. The database (Figure 5.2) stores information about the sales of a title that a salesman achieved on a particular date and city. The dimensions of our example are `Person` (Figure 5.3), `Location` (Figure 5.4), `Product` (Figure 5.5) and `Date` (Figure 5.6). Measure `Sales` is functionally dependent on dimensions `Date`, `Product`, `Person` and `Location`.

Day	Title	Salesman	Store	Sales
6-Feb-97	Symposium	Netz	Paris	7
18-Feb-97	Karamazof brothers	Netz	Seattle	5
11-May-97	Ace of Spades	Netz	Los Angeles	20
3-Sep-97	Zarathustra	Netz	Nagasaki	50
3-Sep-97	Report to El Greco	Netz	Nagasaki	30
1-Jul-97	Ace of Spades	Venk	Athens	13
1-Jul-97	Piece of Mind	Venk	Athens	34

Figure 5.2: Detailed Data Set DS^0 .

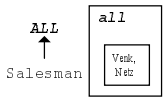


Figure 5.3: Dimension `Person`.

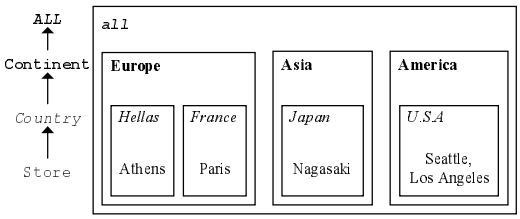


Figure 5.4: Dimension `Location`.

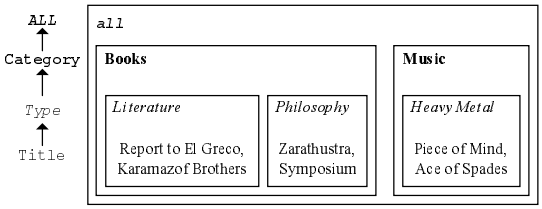


Figure 5.5: Dimension `Product`.

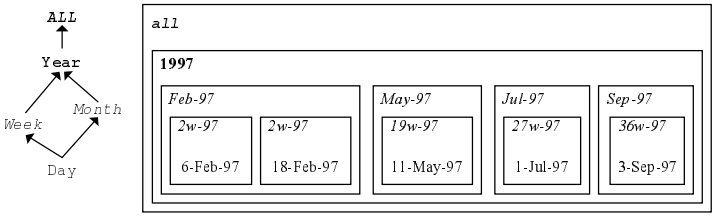


Figure 5.6: Dimension `Date`.

The organisation of information in different levels of aggregation (i.e., dimensions) is in hand because OLAP users are unlikely to directly ask questions about the detailed data that are stored in the database. Instead, they are more interested in aggregated information according to the categorisation groupings.

Following, we present three queries and the respective algebraic representation that could have been a typical sequence of operations during an OLAP session. These queries result in the data of Figures 5.7, 5.8 and 5.9, respectively.

Query 1. Find the maximum sales by month, category of item, salesman and country.

$$c^1=nav(DS^0,[Month,Category,Salesman,Country,Max\_val],max(sales))=$$
$$(DS^0,true,[Month,Category,Salesman,Country,Max\_val],max(sales)).$$

Query 2. Find the maximum sales outside the American continent by month, category of item, salesman and country.

$$c^2=\sigma_{anc^{continent}_{country}(country)\neq 'America'}(c^1)=(DS^0,anc^{continent}_{city}(City)\neq 'America',$$
$$[Month,Category,Salesman,Country,Max\_val],max(sales)).$$

Query 3. Find the summary of sales outside the continent of America by month, type of title and country of store.

$$c^3=nav(c^2,[Month,Type,All,Country,Sum\_val],sum(Sales))= (DS^0,$$
$$anc^{continent}_{city}$$
$$(City)\neq 'America',[Month,Type,All,Country,Sum\_val],sum(sales)).$$

During this particular OLAP session the user has performed:

1. a roll-up from the detailed data set.
2. a selection.
3. a slicing (of dimension `Person`) combined with a drill down (from `Category` to `Type` level) and a change in the aggregation function (from `max` to `sum`).

In the first operation, one can notice that the semantics of the navigation operation allow us to use an arbitrary name (e.g., `Max_val`) for the measure that computes the maximum value per group of aggregation.

In the second operation, notice that the expression $anc^{continent}_{country}(Country)$ which is directly applicable to the schema (and data) of the cube c^1 is transformed to its equivalent $anc^{continent}_{city}(City)$, that directly applies to the detailed data set DS^0, through the use of the definition of the detailed selection condition.

The presented model stresses the fact that *a cube we can treated both as a query and as a set of tuples* . We believe that this aspect of OLAP was neglected in the previous approaches. In this example, the contribution of treating cubes as views over the detailed data set is eminent. Actually, the fact that we have retained the history of selections permits us to be able to drill-down and change the aggregation function. Otherwise, to perform the drill-down operations we should employ a join operation of c^2 with DS^0. The same also holds for the change in the aggregation function. Using the history of selections we can (a) avoid to perform a costly join operation and (b) possibly further optimise the execution of the operation through the use of already computed cubes. The second possibility will be investigated in Section 4.

Month	Category	Salesman	Country	Max_val
Feb 97	Books	Netz	France	7
Feb 97	Books	Netz	USA	5
May 97	Music	Netz	USA	20
Sept 97	Books	Netz	Japan	50
July 97	Music	Venk	Greece	34

Figure 5.7: Cube c^1 - navigation as roll-up

Month	Category	Salesman	Country	Max_val
Feb 97	Books	Netz	France	7
Sept 97	Books	Netz	Japan	50
July 97	Music	Venk	Greece	34

Figure 5.8: Cube c^2 - Selection

Day	Type	ALL	Country	Sum_val
Feb 97	Philosophy	All	France	7
Sep 97	Philosophy	All	Japan	50
Sep 97	Literature	All	Japan	30
Jul 97	Heavy Metal	All	Greece	47

Figure 5.9: Cube c^3 - Complex sequence of operations

Theorem 3.1. The Cube Algebra CA is *sound* (i.e., the result of all the operations is always a cube) and *complete* (i.e., any valid cube can be computed as the combination of a finite set of CA operations).
Proof . We will prove the soundness first and the completeness next.
Soundness. If we consider a detailed data set DS^0 under the schema $S^0=[L_1^0,\ldots,L_n^0,M_1^0,\ldots,M_k^0]$, the valid cubes that can be defined over DS^0 must respect the following requirements:

C1. Their selection condition must be a detailed selection condition applicable to DS^0.
C2. All their levels must belong to the same dimensions with the respective levels of DS^0 (which also implies that they have the same number of dimensions).
C3. The number of their measures must be less or equal to the number of the measures of DS^0.
C4. All their measures must be produced from the application of {sum,max,min,count} to the respective measures of DS^0.

One can easily see that the results of all operations respect the constraints (C1) - (C4).
Completeness. Consider a detailed data set DS^0 under the schema $S^0=[L_1^0,\ldots,L_n^0,M_1^0,\ldots,M_k^0]$. Let also c be an arbitrary cube which can be defined over DS^0:

$$c=(DS^0,\varphi,[L_1,\ldots,L_n,M_1,\ldots,M_m],[agg_1(M_1^0),\ldots,agg_m(M_m^0)]).$$

We can compute c as follows:

1. First, we construct the detailed cube. Although we use the aggregate function sum, one can use any other aggregation function at this stage:
$$c^0=(DS^0,true,[L_1^0,\ldots,L_n^0,M_l^0,\ldots,M_k^0]],[sum(M_l^0),\ldots,sum(M_k^0)])$$
2. Secondly, we split the non-relevant measures (which without loss of generality can be considered to be the last k-m measures).
$$c^1=\Pi_{M_{m+1}}(\Pi_{M_{m+2}}(\ldots(\Pi_{M_k}(c^0))))=(DS^0,true,[L_1^0,\ldots,L_n^0,M_1^0,\ldots,M_m^0],[sum(M_l^0),\ldots,sum(M_m^0)])$$
3. Then, we apply a selection operator to c^1 and we obtain a primary cube c^2:
$$c^2=\sigma_\varphi(c^1)=(DS^0,\varphi,[L_1^0,\ldots,L_n^0,M_1^0,\ldots,M_m^0],[sum(M_l^0),\ldots,sum(M_m^0)])$$
4. Finally, we navigate to the levels of the schema of c.
$$nav(c^2,[L_1,\ldots,L_n,M_1,\ldots,M_m],[agg_1(M_1^0),\ldots,agg_m(M_m^0)]]=$$
$$(DS^0,\varphi,[L_1,\ldots,L_n,M_1,\ldots,M_m],[agg_1(M_1^0),\ldots,agg_m(M_m^0)])\equiv c.\ \blacksquare$$

As we have already stressed, this is a *logical* model for cubes. We do not advocate that the *physical* computation of the results of an operation should actually be computed all the way back from the detailed data set. Actually, although drill-down and change of aggregation function can be performed directly, only through the use of the semantics of our model, can the selection and roll-up operations be performed over the original cube, without referring to the detailed data set. In the case of selection, it suffices to simply pass all the tuples of the cube from the filter of the applied selection condition. In the case of roll-up to coarser levels of granularity, it also suffices to group the tuples of the cube and apply the appropriate aggregate function. These simple optimisation strategies are generalised in Section 4 with a more powerful approach, capable of detecting whether any cube can be computed from the data of another cube, simply by comparing their definitions.

For the moment, we restrict our selves to give a «passive», declarative mapping of cubes to relations and multidimensional arrays. The mapping of the logical cube model to the physical models is straightforward and is similar to mappings already proposed in the literature [Vass98, CaTo98]. Our approach is powerful enough to capture both the ROLAP and the MOLAP case.

First we will present the mapping to the relational model for ROLAP databases. Suppose a dimension D, composed from levels $L_1, \ldots, L_n, ALL$. These levels form a lattice, where L_1 is the detailed level (i.e. the lower bound of the lattice) and ALL the highest level. We map the dimension D to a homonymous table under the schema $D=(\underline{D\_ID}, L_1, \ldots, L_n, ALL)$. Each tuple t in table D is defined as follows: $t[L_i]=anc_{L_1}^{L_i}(t[L_1])$. Note that since L_1 is the common lower bound of the dimension and all the ancestor functions are defined with respect to it, this representation is feasible. The artificial key D_ID could be omitted, although this is not the standard practice in star and snowflake schemata, used in data warehousing. The single-valued attribute ALL, could also be omitted. Moreover, it is easy to guarantee all the constraints of the ancestor functions in such a context.

A data set DS, under the schema $S=[L_1, \ldots, L_n, A_1, \ldots, A_k]$ is mapped to a table $DS=[ID_1, \ldots, ID_n, A_1, \ldots, A_k]$. A primary cube c over a detailed data set F^0 is expressed through the following view C:

$$c=(F^0, \varphi, [L_1, \ldots, L_n, M_1, \ldots, M_m], [agg_1(M_1^0), \ldots, agg_m(M_m^0)])$$

```
DEFINE VIEW C AS
       SELECT      L1,…,Ln,agg1(M1^0) AS M1,…,aggm(Mm^0) AS Mm
       FROM        F^0,D1,D2,…,Dn
       WHERE       F.ID1=D1.ID1 AND F.ID2=D2.ID2 AND … AND F.IDn=Dn.IDn
                   AND σ
       GROUP BY    L1,…,Ln
```

where

- $D_1, \ldots, D_n$ are the dimension tables.
- σ is produced from the primary selection condition of c, if we keep the values l and the level names L unchanged and replace expressions of the form $anc_L^{L'}$ with L'.

The mapping to multidimensional arrays (MOLAP) is performed exactly as in [Vass98, CaTo98]. We assume that there exists a mapping function $enum(d)$ between a value l of a dimension level L and the set of integers. In other words, for each dimension level, we assign a unique integer to each one of its values. The assignment is done in a contiguous fashion. As a result, each value $x=[l_1, l_2, \ldots, l_n, m_1, m_2, \ldots, m_m]$, belonging to the cell data of a cube can be considered to be as the conjunction of co-ordinates $[enum(l_1), enum(l_2), \ldots, enum(l_n)]$ having $[m_1, m_2, \ldots, m_m]$ as values. We consider that the multidimensional array takes a $NULL$ value in the conjunction of co-ordinates when the corresponding tuple of its conjunction does not exist in the cell data of the original cube. In the case where the multidimensional array does not support multiple measures, the primary cube can be split into many single measured primary cubes and reconstructed as requested, on demand. The same can be considered also for secondary cubes. The dimensions can be stored in an arbitrary fashion (i.e. as cubes, indexes, relational tables or any other format the MOLAP engine supports).

4. THE CUBE USABILITY PROBLEM

Problem description
There are several cases where there is the need to decide whether a view can be recomputed from another view. To name two prominent examples, (a) the OLAP users perform interactive navigations over their data and (b) the DW designer has to choose, among many candidates, which views to materialise. In the first case, the problem is as follows: the OLAP user selects some data and performs an operation over them. The result of the new query can be computed, of course, from the detailed data. Nevertheless, it is possible that previously computed and cached results, or existing materialised views could also allow the computation of the requested information. In the second case, the designer of the DW needs algorithmic aid in order to decide if he/she is going to materialise any extra (possibly redundant) views in the DW, so that user queries are answered more efficiently. Sometimes, the redundancy can aid in the refreshment of the DW data, too [ThLS99, LSTV99, Gupt97]. Part of the design algorithm, then, is a method that determines whether a view can be used to compute a query (or another view). As a general statement, one could say that the problem lies in whether the computation of a new cube can be performed from an intermediate level of aggregation, than from the detailed data set.
Formally, let DS^0 be a detailed data set. Let also c^{old} and c^{new} be two cubes defined over DS^0. By definition, cubes c^{old} and c^{new} can be calculated from DS^0. The *cube usability problem* lies on determining whether the tuples of c^{old} can be used to compute cube c^{new}. It is clear that the cube usability problem is a variant of the *view subsumption* problem, already investigated in the field of relational databases [Ullm97].

Shortcomings of current approaches
Too much effort has been spent, in the past, to tackle the problem of view subsumption and query rewriting in the presence of views [NuSS98, CoNS99, DJLS96, CKP95, GuHQ95, LMSS95, ChSh96, LaYa85]. Nevertheless, the previous approaches were relational-oriented and lack to deal with specific characteristics of the multidimensional modelling. We will use two examples to demonstrate these shortcomings.

Example 4.1. Intuitively, someone would expect, that in order to solve the cube usability problem, the new cube c^{new} should:
1. be defined over the same dimensions with c^{old} and at a higher or equal level;
2. be defined over the same measure of DS^0. Moreover, the aggregation functions agg^{new} and agg^{old} should be the same;
3. have a more restrictive selection condition than c^{old}, i.e., φ^{new} is contained in φ^{old} in the usual relational sense.

Checking conditions 1 and 2 is an easy task. To perform the comparison of Condition 3, we need to transform the selection conditions of the two cubes in order to treat them as conjunctive queries [Ullm89]. One could argue that existing relational techniques are adequate to handle this problem. Unfortunately, as we will show, there are cases where those techniques are not sufficient.

Cube c^1

Month	Sales
Feb	5
May	20
Sep	80

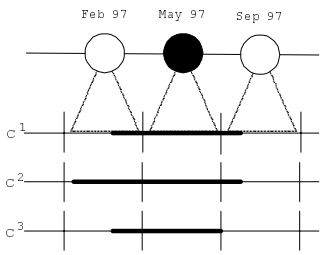
Cube c^2

Month	Sales
Feb	**12**
May	20
Sep	80

Cube c^3

Month	Sales
Feb	5
May	20

φ_1=18-Feb-97≤day≤3-Sep-97 ∧Salesman=Netz
φ_2=6-Feb-97≤day≤3-Sep-97∧Salesman=Netz
φ_3=18-Feb-97≤day≤31-May-97∧Salesman=Netz
(a)



(b)

Figure 5.10: Cube usability problems

Let us consider the detailed data set DS^0 of Figure 5.2. Let c^i, $1 \le i \le 3$ be cubes defined as $c^i=[DS^0, \varphi_i, [Month, ALL, ALL, ALL, ALL, ALL, ALL, Sales], sum(sales)]$. Figure 5.10a presents the `Month` level, the `Sales` measure and the selection conditions for each of the cubes. The problem is whether a new cube c^3 can be computed using the tuples of one of the existing cubes c^1 and c^2. Since Conditions 1, 2 and 3 hold, one could argue that this is feasible. Yet, as we can see in Figure 5.10a,

only c^1 can be used to compute c^3. The intuitive explanation of the problem is depicted in Figure 5.10b. There are three horizontal axes defined at the `day` level, each for one of the cubes c^1, c^2 and c^3. Each bold line denotes the set of days participating in the computation of the respective cube. Cube c^3 is defined at the `month` level; consequently, we partition the three axes with respect to the function anc_{day}^{month}. As we can see, we have three partitions: `Feb'97`, `May'97` and `Sep'97`. Cube c^3 can be computed from c^1 because for all the partitions of c^3 (i.e., `Feb'97`, `May'97`), cubes c^1 and c^3 cover exactly the same days. This does not hold for c^1 and c^2.

Example 4.2. Suppose the case, where a cube c^1 has a selection condition φ_1=`arr.year<dep.year` (where `arr` denotes dimension `arrival date` and `dep` denotes the dimension `departure date`). Suppose also that a cube c^2 is defined at the month level and has a selection condition φ_2=`arr.month<dep.month`. We can see that cube c^1 can be computed from c^2. This means that if c^2 is materialised we can use its tuples to compute c^1. We are able to perform this kind of reasoning because we take advantage of the relationship between months and years, expressed through the dimension hierarchies, and the family of `anc` functions. To our knowledge, there is no effort in the view subsumption literature that uses this kind of knowledge.
In Figure 5.11, we depict the problem graphically. As one can see, we represent the tuples of the detailed as cells in a 2-dimensional space. The horizontal axis represents the dimension `departure date` and the vertical axis represents the dimension `arrival date` (we focus only on these dimensions taking part in the selection conditions of the example). As one can notice, the set of tuples of the detailed data set, fulfilling the condition `arr.month<dep.month` is a strict superset of the set of tuples fulfilling condition `arr.month<dep.month`.

01Jan96							
...							
01Feb96							
...							
01Jan97							
...							
01Feb97							
arr / dep	01Jan96	...	01Feb96	...	01Jan97	...	01Feb97

Legend

Denotes the tuples of the detailed data set fulfilling only the condition `arr.month<dep.month`
Denotes the tuples of the detailed data set fulfilling only the condition `arr.year<dep.year`
Denotes the tuples of the detailed data set fulfilling both the conditions

Figure 5.11. Graphical description of the fitness of data for the cube usability problem

Contribution
In this section, we will show that the cube usability problem is reduced to simple tests and operations. Different tests apply for different classes of queries. We divide the selection conditions in two categories: (a) selection conditions with atoms involving values (i.e., of the form $L\theta l$, $L\theta anc_{L_1}^{L_2}(l)$, etc.) and (b) selection conditions with atoms involving only levels (i.e., of the form $L_1\theta L_2$, $L\theta anc_{L_1}^{L_2}(L_1)$, etc.). We will examine the optimisation issues for the former in subsection 4.1 and for the latter in subsection 4.2. Finally, subsection 4.3 presents a theorem with sufficient criteria and the corresponding rewriting algorithm for both cases of the cube usability problem under consideration.

In the rest of the chapter, for reasons of simplicity, we will deal with cubes having only one measure. All our results can be easily extended to cubes having an arbitrary number of measures [NuSS98]. Let $c^{new}=(DS^0, \varphi^{new}, [\mathbf{L}^{new}, M^{new}], agg^{new}(M))$ be the new cube and $c^{old}=(DS^0, \varphi^{old}, [\mathbf{L}^{old}, M^{old}], agg^{old}(M))$ be the candidate cube, where $\mathbf{L}^{new}$ and $\mathbf{L}^{old}$ are sets of levels coming from dimension sets $\mathbf{D}^{new}$ and $\mathbf{D}^{old}$ respectively, M^{new} and M^{old} are measures, and finally, agg^{new} and agg^{old} are aggregate functions.

4.1 Equivalent Transformations for Atoms Involving Values

Suppose two level L^{old} and L^{new}, such that $L^{old} \prec L^{new}$. Function $anc_{L^{old}}^{L^{new}}$ defines a partition over the values of L^{old} with respect to the values of L^{new} (e.g., the partition of `year` to `month`). Suppose now, two atoms a_1 and a_2 over L^{old}, as in the case of Figure 5.10. To perform an aggregation to L^{new}, the two atoms must hold the same ranges of values for each and every partition that L^{new} defines over L^{old}. Generalising this observation, in the case where two selection conditions involve a larger conjunction of atoms, we must:
(a) transform the selection conditions to concrete ranges for each dimension;

(b) reduce the atoms to the same level, using appropriate transformations (so that they can be compared);
(c) check whether the broader selection condition is defined identically for the marginal constraints of the other selection condition.

The following auxiliary definition introduces the notion of *dimension interval*, which is a concrete range over the domain of a certain dimension level.

Definition 4.1: A *dimension interval* (DI) is one of the following (a) `true`, (b) `false` and (c) an expression of the form $l_1 \leq L \leq l_2$, where `L` is a variable ranging over the level of a dimension and l_1 and l_2 are values. ■

Atom	Dimension Interval	Atom	Dimension Interval
`True`	`true`	$anc_L^{L'}(L)<l$	$-\infty<L\leq max(desc_L^{L'}(prev(l)))$
`False`	`false`	$l\leq anc_L^{L'}(L)$	$min(desc_L^{L'}(l))\leq L<+\infty$
$anc_L^{L'}(L)=l$	$min(desc_L^{L'}(l))\leq L\leq max(desc_L^{L'}(l))$	$l<anc_L^{L'}(L)$	$min(desc_L^{L'}(next(l)))\leq L<+\infty$
$anc_L^{L'}(L)\leq l$	$-\infty<L\leq max(desc_L^{L'}(l))$		

Figure 5.12: Transformation from atoms to dimension intervals

Figure 5.12 shows how single atoms can be transformed to DI's. Values $-\infty$ and $+\infty$ have the obvious semantics. Moreover, functions `prev` and `next` result in the previous and the following value of `l` in the domain of `L` respectively.

In general, to determine whether a cube c^{old} can be used for the computation of c^{new}, we need to partition the detailed level of each dimension according to the respective level of c^{new}. If for each partition of c^{new}, if there exists an identical partition of c^{old}, then c^{old} can be used to compute c^{new}. We formalise this relationship between two cubes, through Definition 4.2.

Definition 4.2. L-containment: Let **D** be a set of dimensions and φ^{old}, φ^{new} be two selection conditions involving levels only from **D**. Let **L** be a set of levels, each belonging to a different dimension of **D**. Let also the two cubes $c^{new}=(DS^0, \varphi^{new}, [L,M], agg(M))$ and $c^{old}=(DS^0, \varphi^{old}, [L,M], agg(M))$, defined over an arbitrary detailed data set DS^0. Selection condition φ^{new} is **L**-*contained* in φ^{old} (denoted by $\varphi^{new} \subseteq_L \varphi^{old}$) if $c^{new} \subseteq c^{old}$ for any data set DS^0. ■

To tackle the problem of cube usability between cubes of different aggregation granularities, we start by checking the containment of conjunctions of atoms that involve values. Notice that our analysis does not include $\neq$. This case will be handled in subsection 4.3.

Algorithm `Check_Atoms_Usability` of Figure 5.13 takes as inputs two conjunctions of atoms, **a** and **b**, involving only values. It yields `true` if **a** **L**-contains **b** with respect to a specific set of levels **L'**, and `false` otherwise. The algorithm proceeds as follows. Initially, Algorithm `Check_Atoms_Usability` rewrites all atoms of **α** and **β** to DI's using the transformations of Figure 5.13 (Line 1). Then, it groups all dimension intervals of **α** and **β** by dimension level and produces, for every set, a single DI having the most restrictive boundaries. The result is stored in the sets of DI's **α'** and **β'** respectively (Line 2). Lines 3-6 check whether there exists a dimension level $D_i.L$ of a in **α'** that does not exist in any DI of **β'**. In that case, the algorithm introduces the dimension interval $-\infty \leq D_i.L^0 \leq +\infty$ to **b'** (Line 5). Finally, Lines 7-24 check if for every DI β of **β** there exists a DI α in **α** such that $\beta \subseteq_{L'} \alpha$ for a certain level $L' \in$ **L'**. More specifically, Lines 12-21 check whether the DI α is **L**-contained with respect to DI β. Lines 12-13 check whether DI β is broader than α. Then, the algorithm checks whether the marginal partitions are identical (Lines 14-17). If all these conditions are fulfilled, then the algorithm returns `true`.

For Example 4.1 using Algorithm `Check_Atoms_Usability` we can deduce that φ_1 **L**-contains φ_3 (with respect to level `Month`), while φ_2 does not. Moreover, it is interesting to see that if one considers the `year` level, neither φ_1 nor φ_2 **L**-contains φ_3.

Algorithm Check_Atoms_Usability.
Input: Two conjunctions of atoms **a** and **b** involving only values, and a set of levels **L'**.
Output: true if $\mathbf{a} \subseteq_{\mathbf{L'}} \mathbf{b}$, false otherwise.

1. Write all atoms of **a** and **b** as DI's using the transformations of Figure 5.12.
2. Group all DI's of **a** and **b** by dimension level and produce for every set a single DI' having the most restrictive boundaries. Let **a'** and **b'** be the result, respectively.
3. For every DI a of **a'**
4. If a is defined over dimension level $D_i.L^0$ that does not exist in any DI of **b'** Then
5. Introduce DI $-\infty \le D_i.L^0 \le \infty$ to **b'**.
6. EndFor
7. flag = false
8. For every DI a of **a'**
9. flag = false
10. For every DI b of **b'**
11. For every dimension level L' of **L'** involved in b
12. Case $A_s < B_s$ or $B_e < A_e$ or b=false
13. flag = true
14. Case $L \neq L'$ and $A_s \neq \min(desc_L^{L'}(anc_L^{L'}(A_s)))$ and $A_s \neq B_s$
15. flag = false
16. Case $L \neq L'$ and $A_e \neq \max(desc_L^{L'}(anc_L^{L'}(A_e)))$ and $A_e \neq B_e$
17. flag = false
18. Default
19. flag = true
20. EndFor
21. EndFor
22. If flag = false Then
23. Return false
24. EndFor
25. Return true

Figure 5.13: Algorithm Check_Atoms_Usability

4.2 Equivalent Transformations for Atoms Involving only Levels

Following [Ullm89], we assume the existence of two infinite, totally ordered domains, **L** and **L'** isomorphic to the integers. Let also f be a total, monotone function over **L**, mapping the values of domain **L** to the values of domain **L'**. The family of anc functions fulfils these requirements.

We assume that we are given a collection of inequalities of the form $X<Y$, $X \le Y$, $X \neq Y$, $f(X)<f(Y)$, $f(X) \le f(Y)$, $f(X) \neq f(Y)$ and equalities of the form $f(X)=f(Y)$. We do not allow equalities of the form $X=Y$. If such a subgoal is found in a query, we substitute every occurrence of X with Y. We also eliminate any pair of inequalities $f(X) \le f(Y)$ and $f(Y) \le f(X)$, where X,Y are distinct variables, with $f(X)=f(Y)$.

We will use the following set of axioms for these inequalities:

A1	$X \le X$	A8	$X \le Z$, $Z \le Y$, $X \le W$, $W \le Y$ and $W \neq Z$ imply $X \neq Y$
A2	$X<Y$ implies $X \le Y$	A9	$X \le Y$ implies $f(X) \le f(Y)$
A3	$X<Y$ implies $X \neq Y$	A10	$F(X)<f(Y)$ implies $X<Y$
A4	$X \le Y$ and $X \neq Y$ imply $X<Y$	A11	$F(X) \neq f(Y)$ implies $X \neq Y$
A5	$X \neq Y$ implies $Y \neq X$	A12	$F(X) \le f(Y)$ and $f(Y) \le f(X)$ implies $f(X)=f(Y)$
A6	$X<Y$ and $Y<Z$ imply $X<Z$	A13	$F(X)=f(Y)$ and $f(Y) \le f(Z)$ implies $f(X) \le f(Z)$
A7	$X \le Y$ and $Y \le Z$ imply $X \le Z$	A14	$F(X)=f(Y)$ and $f(Y) \neq f(Z)$ implies $f(X) \neq f(Z)$

A15 $f(X)=f(Y)$ implies $f(X) \le f(Y)$

Figure 5.14: Axioms for L-containment checking.

We assume that our models are assignments of integers to variables. Expressions of the form $f(X)$ are also treated as variables. For variables of the form X we apply axioms A1 to A9 and for variables of the form $f(X)$ we apply axioms A1 to A15.

Theorem 4.1: The axioms are sound and complete.

Proof. We will prove the soundness first and then we will proceed to prove the completeness claim.

Soundness. To show the axioms sound, we must show that they infer only true statements. In other words, we must show that they hold in each model. A1-A8 are already proved to be sound in [Ullm89]. A9-A15 are simple, well-known, properties of the monotone function f.

Completeness. We will extend the proof of [Ullm89]. Let **U** be a finite set of variables. Let **S** be a set of inequalities, involving a subset of **U**, namely **V**, and a finite set of variables **V'**, such that all the variables participating in **V'**, are of the form **V'**={f(X),X∈**U**}. Let **U'** also be a finite set of "dummy" variables, having the same size with **U**. We restrict **V** and **V'**, so that we cannot derive from S, using A1 through A15, inequalities of the form X<X, X≤Y∧Y≤X, f(X)<f(X), f(X)θY, Xθf(Y), f(X)=f(Y) and f(X)≤f(Y) for every pair of distinct variables X and Y.

Suppose also that $\mathbf{S}^+$ is the set of inequalities that follow logically, using the axioms A1 to A15. Will we refer to $\mathbf{S}^+$ as the *closure* of **S**. If $\mathbf{S}^+$ is complete, every inequality XθY (respectively f(X)θf(Y)) not in $\mathbf{S}^+$, has some assignment of integers to the variables of **U** that makes every inequality in $\mathbf{S}^+$ true, but XθY (respectively f(X)θf(Y)) false.

We initialise $\mathbf{S}^+$ to be identical to **S**. As a first step, we search the set $\mathbf{S}^+$ for pairs of inequalities of the form f(X)≤f(Y) and f(Y)≤f(X). We replace any occurrence of any of these inequalities in $\mathbf{S}^+$, with the single statement f(X)=f(Y). After this step, one can be certain that if any inequality of the form f(X)≤f(Y) is found in $\mathbf{S}^+$, then no inequality of the form f(Y)≤f(X) is also found in $\mathbf{S}^+$.

Suppose also that **G** is a set, comprising of sets of variables. For each set of variables {f(X_1),f(X_2),…,f(X_k)} such that f(X_1)=f(X_2)=…=f(X_k), we introduce a new set in **G**. Each set of this kind will be called *group* in the sequel. All the aforementioned transformations and set instantiations can be done in finite time. Using A13 and A15 we can easily prove that the sets of **G** are disjoint. Moreover, if **G**={g_1, g_2,…} and A∈g_1, B∈g_2, then:

- if A<B then all members of g_1 are smaller than all member of g_2 (by A14 and A3);
- if A≤B then all members of g_1 are smaller or equal than all member of g_2 (by A15) and there does not exist A'∈g_1, B'∈g_2, such that B'≤A', because then the two sets would not be disjoint;
- if A≠B, then all members of g_1 are different from all the members of g_2 (by A14).

In the sequel, we will often need to group and/or order the variables of **U**. The result of this process will be referred to as **G**.

In order to prove the completeness of $\mathbf{S}^+$, we must show that every inequality XθY (respectively f(X)θf(Y)) not in $\mathbf{S}^+$, has some assignment of integers to the variables of **U** that makes every inequality in $\mathbf{S}^+$ true, but XθY (respectively f(X)θf(Y)) false. Depending on the type of the inequality we consider the following cases:

Case 1: X≤Y. We need to construct an assignment that satisfies $\mathbf{S}^+$ but makes Y<X true. Let **A** be a set of variables A such that X≤A∈$\mathbf{S}^+$ and **B** a set of variables B such that B≤Y∈$\mathbf{S}^+$. Let also **C**=**U**-**A**-**B**. We will refer to **A**, **B** and **C** as *blocks*. In general, if A∈**A**, B∈**B**, and C∈**C**, we can be sure that neither C≤B nor A≤C holds. Also, the sets **A**,**B** and **C** are disjoint. (**C** is disjoint from the other two sets by definition. **A** and **B** are also disjoint or else ∃K∈**A**,**B**, s.t. X≤K,K≤Y∈$\mathbf{S}^+$, which implies X≤Y∈$\mathbf{S}^+$, contrary to our initial assumption). Note also that all variables of the form f(X) are found in **C**, since we do not allow inequalities of the form f(X)θY.

Now, we topologically sort **A**, **B** and **C** with respect to the order ≤. Remember that after the initial transformation of $\mathbf{S}^+$ we cannot derive both $A_1 \le A_2$ and $A_2 \le A_1$ for two distinct variables. For **A** and **B**, which do not involve variables of the form f(X), the sorting is straightforward. For **C** we use the following trick: all the variables f(U) and f(V)such that f(U)=f(V) (i.e. belonging in the same group in **G**, are replaced from a single dummy variable, just for the sake of the topological ordering). After this transformation the topological ordering is feasible, since we cannot derive both f(U)≤f(V) and f(V)≤f(U) for two distinct variables. After the topological ordering is performed we replace the dummy variable with the respective variables, placed in consecutive positions. Next, we order all the variables in **U**, as follows: First we list all the elements of **B** in topological order, then all the elements of **C** in order and finally all the elements of **A** in order. After that, we assign distinct integers 1, 2, ... to the variables in this order, except for variables belonging to the same group, which are assigned the same integer. We call the integer assigned to a variable U, through this assignment, a(U).

So forth, we have managed that X is given a larger value than Y, so a(X)≤a(Y) does not hold. Now, we must show that all the inequalities in $\mathbf{S}^+$ hold in this assignment. We will use the variable names U,

V for the members of **V** and f(U), f(V) for the members of **V'**. Depending on the form of inequalities we consider the following cases:

a. $U \neq V \in \mathbf{S^+}$. Since no variables of this form are assigned to the same integer, all ≠ inequalities hold.

b. $U \leq V \in \mathbf{S^+}$. We have the following possible combinations for U and V:

- b_1. U, V are the same variable. Then, $a(U) \leq a(V)$ holds.
- b_2. U, V are in the same block. Then, $a(U) \leq a(V)$ holds, since each block is topologically ordered.
- b_3. $U \in \mathbf{B}$, $V \in \mathbf{C}$ (or **A**). Then, $a(U) \leq a(V)$ holds, since in the final order, everything in **B** precedes everything else in the other blocks.
- b_4. $U \in \mathbf{C}$, $V \in \mathbf{A}$. Then, $a(U) \leq a(V)$ holds for the same reason.
- b_5. $U \in \mathbf{A}$ then V must also be in **A** (or else A7 is violated) so b_1 holds. Moreover, if $U \in \mathbf{C}$ then V must be in **A** so b_4 holds.

All the above are summarised in the following table.

U↓ V→	B	C	A
B	b_2	b_3	b_3
C	b_5	b_2	b_4
A	b_5	b_5	b_2

c. $U < V \in \mathbf{S^+}$. Obviously, U, V are not the same variable. The argument given for $U \leq V$ can also be given in this case (b_2-b_5). Thus $U < V$ is satisfied by the proposed assignment.

d. $f(U) \neq f(V) \in \mathbf{S^+}$. All these variables are in **C**. Due to the nature of the assignment and the fact that only equal variables are assigned the same integer, the inequality holds.

e. $f(U) \leq f(V) \in \mathbf{S^+}$. The assignment respects the order, due to its nature. Remember that due to the "cleaning" transformations, inequalities of the form $f(V) \leq f(U)$ or $f(V) = f(U)$ do not exist in $\mathbf{S^+}$.

f. $f(U) < f(V) \in \mathbf{S^+}$. Obviously, $f(U) \neq f(V)$ (i.e., they are not the same variable). The argument for case **e** can be reused.

g. $f(U) = f(V) \in \mathbf{S^+}$. Due to the nature of the assignment, the variables which are equal to each other are assigned the same integer.

Since all the possible combinations of expressions that can be found in $\mathbf{S^+}$ respect the assignment and $a(Y) < a(X)$ then we conclude that the assignment satisfies $\mathbf{S^+}$ but not $X \leq Y$.

Case 2: Suppose that $X \neq Y$ is not in $\mathbf{S^+}$. We must find an assignment such that $a(X) \neq a(Y)$ is false and all the other assignments in $\mathbf{S^+}$ are true. [Ullm89] observes that there cannot be two distinct variables Z, W such that $X \leq Z$ and $Z \leq Y$ and $Y \leq W$ and $W \leq X$. Consequently, there is an order between X, Y and without loss of generality we can assume that $X \leq Y$. We define the following blocks: Let **Z** be a set containing all variables Z such that $X \leq Z \leq Y \in \mathbf{S^+}$, **D** be the set containing X, Y and all the variables from set **Z**, **A** be the set of variables A such that $D \leq A \in \mathbf{S^+}$, for any $D \in \mathbf{D}$, but A is not itself in **D**, **B** is the set of variables B such that $B \leq D$ is in $\mathbf{S^+}$, for any $D \in \mathbf{D}$, but B is not itself in **D** and **C** be the set of variables that remain. Sets **B, C, D, A** are disjoint. (**C** is disjoint from the rest of the sets by definition. **A** and **B** are disjoint from **D**, by definition too. **A** and **B** are disjoint, or else $\exists K \in \mathbf{A,B}$ and $D_1, D_2 \in \mathbf{D}$ such that, $D_1 \leq K, K \leq D_2 \in \mathbf{S^+}$, which implies $X \leq D_1 \leq K \leq D_2 \leq Y \in \mathbf{S^+}$, meaning that $K \in \mathbf{Z}$ and consequently $K \in \mathbf{D}$, contrary to our initial assumption). If $A \in \mathbf{A}$, $B \in \mathbf{B}$ and $C \in \mathbf{C}$ then it is possible that $B \leq C \leq A$ and $B \leq D \leq A$. Moreover, the following do not hold: $A \leq C, C \leq B, A \leq D, D \leq B$. For example, $A \leq C$ cannot hold, because if it did, then $\exists D \in \mathbf{D}$ such that, $D \leq A \leq C \in \mathbf{S^+}$, implying that $C \in \mathbf{A}$, which cannot hold. The rest of the cases can be proved similarly. Again all the variables of the form f(U) are found in **C**.

We topologically order all these sets of variables with respect to the order ≤, using the methodology of **Case 1**. We combine the orders in the sequence **B**, **C**, **D**, **A**. We assign distinct integers to all variables, except (a) for the variables of **D** and (b) for the variables belonging to the same set in **G**. In both these cases, each set of variables is given the same integer. Surely, $X \neq Y$ is not satisfied by this assignment, since $a(X) = a(Y)$. We will prove that all the inequalities in $\mathbf{S^+}$ are satisfied. We consider again the following seven cases for inequalities in $\mathbf{S^+}$:

a. $U \neq V \in \mathbf{S^+}$. Since no variables of this form are assigned to the same integer, except for variables belonging to **D**, all these inequalities hold. For the case where both U, V belong to **D**, we can prove that it is not possible that $U \neq V \in \mathbf{S^+}$. Suppose that $\exists Z, W \in \mathbf{D}$, such that $X \leq Z \leq Y, X \leq W \leq Y, Z \neq W \in \mathbf{S^+}$. Then $X \neq Y$ is implied (by A8), which contradicts our hypothesis. Consequently, there are no ≠ inequalities between members of **D**.

b. $U \leq V \in \mathbf{S^+}$. We have the following possible combinations for U and V:

b_1. U, V are the same variable. Then, $a(U) \leq a(V)$ holds.

b_2. U, V are in the same block. Then, $a(U) \leq a(V)$ holds, since each block is topologically ordered and the topological order respects the inequality for this kind of variables.

b_3. $U \in \mathbf{B}$ and $V \in \mathbf{C}, \mathbf{D}$ or $\mathbf{A}$. Then, $a(U) \leq a(V)$ holds, since in the final order, everything in **B** precedes everything else in the other blocks.

b_4. $U \in \mathbf{C}$ and $V \in \mathbf{D}$ or $\mathbf{A}$. For the same reason, $a(U) \leq a(V)$ holds.

b_5. $U \in \mathbf{D}$ and $V \in \mathbf{A}$. For the same reason, $a(U) \leq a(V)$ holds.

b_6. $U \in \mathbf{A}$ then V must be in **A** (or else A7 is violated). Moreover, if $U \in \mathbf{C}$ then V can not be in **B**.

b_7. $U \in \mathbf{D}$ then $V \in \mathbf{D}, \mathbf{A}$ and not in **B** or **C**. This holds because if $V \in \mathbf{B}$ then $D \leq B$ which does not hold. If, on the other hand, $V \in \mathbf{C}$ then $X \leq U \leq Y$ and $U \leq V$, implying that $\exists D \in \mathbf{D}$, such that $D \leq V$, i.e., $V \in \mathbf{A}$, which does not hold either.

c. $U < V \in \mathbf{S}^+$. Obviously, U, V are not the same variable. The argument given for $U \leq V$ can also be given in this case too (b_2-b_7). Thus $U < V$ is satisfied by the proposed assignment.

d. $f(U) \neq f(V) \in \mathbf{S}^+$. All these variables are in **C**. Due to the nature of the assignment and the fact that only equal variables are assigned the same integer, the inequality holds.

e. $f(U) \leq f(V) \in \mathbf{S}^+$. The assignment respects the order, due to its nature. Remember that due to the "cleaning" transformations, inequalities of the form $f(V) \leq f(U)$ or $f(V) = f(U)$ do not exist in $\mathbf{S}^+$.

f. $f(U) < f(V) \in \mathbf{S}^+$. Obviously, $f(U) \neq f(V)$ (i.e., they are not the same variable). The argument for case **e** can be reused.

g. $f(U) = f(V) \in \mathbf{S}^+$. Due to the nature of the assignment, the variables which are equal to each other are assigned the same integer.

Since all the possible combinations of expressions that can be found in $\mathbf{S}^+$, respect the assignment and $a(Y) = a(X)$ then we conclude that the assignment satisfies $\mathbf{S}^+$ but not $X \neq Y$.

Case 3: Suppose that $X < Y$ is not in $\mathbf{S}^+$. We must find an assignment such that $X < Y$ is false and all the other assignments in $\mathbf{S}^+$ are true. If both $X \leq Y$ and $X \neq Y$ are in $\mathbf{S}^+$, then, by A4, $X < Y$ is also in $\mathbf{S}^+$, contrary to the initial assumption for **Case 3**. Consequently, for $X < Y$ not to be in $\mathbf{S}^+$, at least one of the following two cases must hold:

a. $X \leq Y \notin \mathbf{S}^+$. Then we can use the assignment of **Case 1**; it satisfies $\mathbf{S}^+$ and proves that $X < Y$ is false.

b. $X \neq Y \notin \mathbf{S}^+$. Then we can use the assignment of **Case 2**; it also satisfies $\mathbf{S}^+$ and proves that $X < Y$ is false.

In both cases, $a(X) < a(Y)$ is false and all the other assignments in $\mathbf{S}^+$ are true.

Case 4: Suppose that $f(X) \leq f(Y)$ is not in $\mathbf{S}^+$. We need to construct an assignment that satisfies $\mathbf{S}^+$ but makes $a(f(Y)) < a(f(X))$ true. Let **A** be a set of variables A such that $f(X) \leq f(A)$ is in $\mathbf{S}^+$ and **B** a set of variables B such that $f(B) \leq f(Y)$ is also in $\mathbf{S}^+$. Let also $\mathbf{C} = \mathbf{U} - \mathbf{A} - \mathbf{B}$. In general, if $f(A) \in \mathbf{A}$, $f(B) \in \mathbf{B}$, and $f(C) \in \mathbf{C}$ we can be sure that neither $f(C) \leq f(B)$, nor $f(A) \leq f(C)$ hold. Also, the sets **A**, **B** and **C** are disjoint (similarly with **Case 1**). Note that all variables belonging to **V** (i.e., of the form U) are found in **C**, since we do not allow inequalities of the form $f(X) \theta Y$. Moreover, one can show that if a variable $f(U)$ belongs to a block and there exists a variable $f(V)$, such that $f(U) = f(V)$, then both variables are found in the same block. This holds for all the three blocks **A**, **B** and **C**. Moreover, $f(X)$, $f(Y)$ are allowed to be found in different blocks, since if $f(U) = f(V)$ is found in $\mathbf{S}^+$, this would imply that $f(X) \leq f(Y)$ (A15) is also found in $\mathbf{S}^+$, which we have already assumed that is not true.

Now, we topologically sort **A**, **B** and **C** with respect to the order ≤. We use the following trick: all the variables $f(U)$ and $f(V)$such that $f(U) = f(V)$, i.e., belonging in the same group in **G** are replaced from a single dummy variable, just for the sake of the topological ordering. Now, the topological ordering is feasible, since we cannot derive both $f(U) \leq f(V)$ and $f(V) \leq f(U)$ for two distinct variables. After the topological ordering is performed we replace the dummy variable with the respective variables, placed in consecutive positions. Next, we order all the variables in **U**, as follows: First we list all the elements of **B** in order, then all the elements of **C** in order and finally all the elements of **A** in order. After that, we assign distinct integers 1, 2, ... to the variables in this order, except for variables belonging to the same group, which are assigned the same integer. We call the integer assigned to a variable U, through this assignment, $a(U)$.

So forth, we have managed that $f(X)$ is given a larger value than $f(Y)$, so $f(X) \leq f(Y)$ does not hold (remember that $f(X)$, $f(Y)$ cannot be related with equality, so they are assigned different

numbers). Now, we must show that all the other inequalities in $\mathbf{S}^+$ hold in this assignment. We consider the following cases.

a. $U \neq V \in \mathbf{S}^+$. Since no variables of this form are assigned to the same integer, all $\neq$ inequalities hold.

b. $U \leq V \in \mathbf{S}^+$. All these variable belong to **C**, there are no equalities between them and the topological ordering assigns different integers to different variables: consequently, the assignment holds.

c. $U < V \in \mathbf{S}^+$. Obviously, U and V are not the same variable. The argument given for $U \leq V$ can also be given in this case. Thus $U<V$ is satisfied by the proposed assignment.

d. $f(U) \leq f(V) \in \mathbf{S}^+$. All variables of the same group in **G**, belong to the same block. We consider the following cases:

- b_1. $f(U)$ and $f(V)$ are in the same block. Remember that $f(X)=f(Y)$ is not in $\mathbf{S}^+$ and that only equal variables are assigned the same integer. Then, $a(f(U)) \leq a(f(V))$ holds, since each block is topologically ordered and the topological order respects the inequality for this kind of variables.
- b_2. $f(U) \in \mathbf{B}$ and $f(V) \in \mathbf{C}$ or **A**. Then, $a(f(U)) \leq a(f(V))$ holds, since in the final order, everything in **B** precedes everything else in the other blocks.
- b_3. $f(U) \in \mathbf{C}$ and $f(V) \in \mathbf{A}$. Then, for the same reason, $a(f(U)) \leq a(f(V))$ holds.
- b_4. $f(U) \in \mathbf{A}$. Then, $f(V) \in \mathbf{A}$ (or else A7 is violated), so $f(V)$ cannot be in **B**, **C**. The same happens if $f(U) \in \mathbf{C}$ and $f(V) \in \mathbf{B}$.

e. $f(U) \neq f(V) \in \mathbf{S}^+$. These two variables do not belong to the same group in **G**, thus they are assigned different integers.

f. $f(U) < f(V) \in \mathbf{S}^+$. Obviously, $f(U) \neq f(V)$ (i.e., they are not the same variable). The argument for **d** can be reused.

g. $f(U) = f(V) \in \mathbf{S}^+$. Due to the nature of the assignment, the variables which are equal to each other are assigned the same integer.

Since all the possible combinations of expressions that can be found in $\mathbf{S}^+$, respect the assignment and $a(f(Y)) < a(f(X))$ then we conclude that the assignment satisfies $\mathbf{S}^+$ but not $f(X) \leq f(Y)$.

Case 5: Suppose that $f(X) \neq f(Y) \notin \mathbf{S}^+$. We need to construct an assignment that satisfies $\mathbf{S}^+$ but makes $a(f(Y)) \neq a(f(X))$ false. First, note that since $f(X) \neq f(Y)$ is not in $\mathbf{S}^+$, then (by A5 and A3) also $f(X) < f(Y)$ and $f(Y) < f(X)$ are not in $\mathbf{S}^+$. Since $f(Y) < f(X)$ is not in $\mathbf{S}^+$, we can construct the following set **D**:

$$\mathbf{D} = \{f(D) : \{f(X) \leq f(D) \leq f(Y)\} \cup \{f(D) = f(X)\} \cup \{f(D) = f(Y)\}\}$$

Note that the definition of **D** is legitimate. There are three cases: (a) $f(X) \leq f(Y) \in \mathbf{S}^+$, (b) $f(Y) \leq f(X) \in \mathbf{S}^+$, (c) none of the previous two. In the first case the definition of **D** is as previously mentioned. In the second case, obviously $f(X), f(Y)$ are used in reverse order. In the third case, the set $\{f(X) \leq f(D) \leq f(Y)\}$ is empty. Without loss of generality, we choose that the first case holds.

We construct the following sets: Let **A** be the set of variables $f(A)$ such that $f(D) \leq f(A) \in \mathbf{S}^+$, for some $f(D) \in \mathbf{D}$, but $f(A)$ is not itself in **D**, let **B** is the set of variables $f(B)$ such that $f(B) \leq f(D) \in \mathbf{S}^+$, for some $f(D) \in \mathbf{D}$, but $f(B)$ is not itself in **D** and let **C** be the set of remaining variables. The sets **B**, **C**, **D** and **A** are disjoint. **C** is disjoint from the rest of the sets by definition. **A** and **B** are disjoint from **D**, by definition too. **A** and **B** are disjoint, or else $\exists f(K) \in \mathbf{A,B}$, $f(D) \in \mathbf{D}$ such that, $f(D) \leq f(K), f(K) \leq f(D) \in \mathbf{S}^+$, which implies $f(K)=f(D)$, meaning that $f(K) \in \mathbf{D}$, which cannot hold. Moreover if $f(B) \in \mathbf{A}$, $f(D) \in \mathbf{D}$, $f(B) \in \mathbf{B}$ and $f(C) \in \mathbf{C}$ then $f(B) \leq f(C) \leq f(A)$ and $f(B) \leq f(D) \leq f(A)$ can possibly hold and $f(A) \leq f(C)$, $f(C) \leq f(B)$, $f(A) \leq f(D)$ and $f(D) \leq f(B)$ do not hold. Again all the variables of the form U are found in **C**.

We topologically order these sets of variables with respect to the order $\leq$, using the methodology of **Case 1**. We combine the orders in the sequence **B**, **C**, **D** and **A**. We assign distinct integers to all variables, except (a) for the variables of **D** and (b) for the variables belonging to the same set in **G**. In both these cases, each set of variables is given the same integer.

Surely, $a(X) \neq a(Y)$ is not satisfied by this assignment. We will prove that all the inequalities in $\mathbf{S}^+$ are satisfied. We consider again the following cases:

a. $U \neq V \in \mathbf{S}^+$. Since no variables of this form are assigned to the same integer, all such $\neq$ inequalities hold.

b. $U \leq V \in \mathbf{S}^+$. All these variable belong to **C**, there are no equalities between them and the topological ordering assigns different integers to different variables: consequently, the assignment holds.

c. $U<V \in \mathbf{S}^+$. Obviously, U, V are not the same variable. The argument given for $U \le V$ can also be given in this case. Thus $U<V$ is satisfied by the proposed assignment.
d. $f(U) \neq f(V) \in \mathbf{S}^+$. By default any variables that do not belong to the same group in **G**, are assigned different integers. The only problem could happen with variables belonging to **D**. Similarly with **Case 2**, we can prove that there are no variables $f(U) \neq f(V)$ such that $f(X) \le f(U) \le f(Y), f(X) \le f(V) \le f(Y) \in \mathbf{S}^+$. Moreover, if there is a variable belonging to the same group with either $f(X)$ or $f(Y)$, then there is no possibility for a $\neq$ inequality to hold for it, since then $f(X) \neq f(Y)$ would belong to $\mathbf{S}^+$.
e. $f(U) \le f(V) \in \mathbf{S}^+$. For any pair of variables the inequality for the assignment holds (as shown in previous cases). For the case of two variables belonging to **D**, then $a(f(U)) \le a(f(V))$ holds, since $a(f(U))=a(f(V))$ holds.
f. $f(U)<f(V) \in \mathbf{S}^+$. As with the case of **e**, for any pair of variables the inequality for the assignment holds (as shown in previous cases). For the case of two variables belonging to **D**, it is not possible that such an inequality holds (or else $f(U)<f(V)$ would imply $f(U) \neq f(V) \in \mathbf{S}^+$).
g. $f(U)=f(V) \in \mathbf{S}^+$. Due to the nature of the assignment, the variables which are equal to each other are assigned the same integer.
Case 6: Suppose that $f(X)<f(Y)$ is not in $\mathbf{S}^+$. Similarly with **Case 3**, we must find an assignment such that $a(f(X))<a(f(Y))$ is false and all the other assignments in $\mathbf{S}^+$ are true. If both $f(X) \le f(Y)$ and $f(X) \neq f(Y)$ are in $\mathbf{S}^+$, then, by A4, $f(X)<f(Y)$ is also in $\mathbf{S}^+$, contrary to the initial assumption for **Case 3**. Consequently, for $f(X)<f(Y)$ not to be in $\mathbf{S}^+$, at least one of the following two cases must hold:
a. $f(X) \le f(Y) \notin \mathbf{S}^+$. Then we can use the assignment of **Case 4**; it satisfies $\mathbf{S}^+$ and proves that $f(X)<f(Y)$ is false.
b. $f(X) \neq f(Y) \notin \mathbf{S}^+$. Then we can use the assignment of **Case 5**; it also satisfies $\mathbf{S}^+$ and proves that $f(X)<f(Y)$ is false.
In both cases, $a(f(X))<a(f(Y))$ is false and all the other assignments in $\mathbf{S}^+$ are true.
Case 7: Suppose that $f(X)=f(Y) \notin \mathbf{S}^+$. We need to construct an assignment that satisfies $\mathbf{S}^+$ but makes $a(f(Y))=a(f(X))$ true. Since $f(X)=f(Y) \notin \mathbf{S}^+$, we can deduce that only one of the following three cases can hold: (a) $f(X) \le f(Y) \in \mathbf{S}^+$, (b) $f(Y) \le f(X) \in \mathbf{S}^+$, (c) none of the previous two. For the second and the third case we can use directly the assignment of **Case 4**, which produces $a(f(Y))<a(f(X))$ (implying that $(f(Y)) \neq a(f(X))$) and respects all the inequalities found in $\mathbf{S}^+$, at the same time. For the first case, we can use the assignment of **Case 4** with reverse roles for $f(Y)$ and $f(X)$. ■
In order to check whether one set of inequalities T follows from another set of inequalities S we compute the closure S^+ by applying the axioms A1-A15 until they no longer generate any new inequalities. Then, we check whether T is a subset of S^+.

4.3 Testing Cube Usability

In this section, we combine the results of subsections 4.1 and 4.2 to provide a test for several cases of cube usability. One can transform any kind of formula using logical transformations [Ende72] to an equivalent formula consisting of disjunctions of conjunctions which do not involve $\neq$ and $\neg$. Theorem 4.1 provides sufficient criteria for a cube c^{old} to be used for the computation of another cube c^{new}. Algorithm Cube_Usability describes the specific steps to be followed for this computation.
Theorem 4.2: Suppose a detailed data set $DS^0=[L_1^0, \ldots, L_n^0, M^0]$ and two cubes $c^{old}=(DS^0, \varphi_{old}, [L_1^{old}, \ldots, L_n^{old}, M_{old}], agg_{old}(M^0))$ and $c^{new}=(DS^0, \varphi_{new}, [L_1^{new}, \ldots, L_n^{new}, M_{new}], agg_{new}(M^0))$. If
(a) $agg_{old}=agg_{new}$,
(b) $L_i^{old} \prec L_i^{new}$, $1 \le i \le n$, and
(c) one of the following two cases holds for φ_{old} and φ_{new}:

- φ_{old} and φ_{new} involve conjunctions of atoms only of the form $L_i \theta L_j$, all the levels L_i, L_j are higher from the respective levels of the schema of c^{old} (i.e. $L_{i,j}^{old} \prec L_{i,j}$) and φ_{old} belongs to the closure of φ_{new}, or,
- φ_{old} and φ_{new} involve conjunctions of atoms of the form $L \theta l$ and $\varphi_{new} \subseteq_{[L_1^{new}, \ldots, L_n^{new}]} \varphi_{old}$,

then Algorithm Cube_Usability correctly computes c^{new} from the tuples of c^{old}.

Algorithm Cube_Usability.
Input: A detailed data set $DS^0=[L_1^0,\ldots,L_n^0,M^0]$ and two cubes $c^{old}=(DS^0,\varphi_{old},[L_1^{old},\ldots,L_n^{old},M_{old}],agg_{old}(M^0))$ and $c^{new}=(DS^0,\varphi_{new},[L_1^{new},\ldots,L_n^{new},M_{new}],agg_{new}(M^0))$ such that φ_{old} and φ_{new} involve either (a) conjunctions of atoms of the form $L\theta l$ or (b) conjunctions of atoms of the form $L\theta L'$ where L and L' are levels and l is a value.
Output: A rewriting that calculates cube c^{new} from the tuples of c^{old}.
1. If all atoms of φ_{old} and φ_{new} involve conjunctions of atoms of the form $L\theta l$ Then
2. For every atom $a=anc_{L^0}^{L}(L^0)\theta l$ in φ_{new} (or equivalent to this form)
3. If L^{old} is the respective level in the schema of c^{old} and $L^{old}\prec L$ Then
4. Transform a to $anc_{L^{old}}^{L}(L^{old})\theta l$
5. EndIf
6. ElseIf L^{old} is the respective level in the schema of c^{old} and $L\prec L^{old}$ Then
7. Transform a to $L^{old}\theta' anc_{L}^{L^{old}}(l)$ where $\theta'=\theta$ except for two cases:
(a) $a=anc_{L^0}^{L}(L^0)<l$ and $l\neq min(desc_{L}^{L^{old}}(anc_{L}^{L^{old}}(l)))$ where $\theta'=\leq$,
(b) $a=anc_{L^0}^{L}(L^0)>l$ and $l\neq max(desc_{L}^{L^{old}}(anc_{L}^{L^{old}}(l)))$ where $\theta'=\geq$
8. EndIf
9. EndFor
10. EndIf
11. If all atoms of φ_{old} and φ_{new} involve conjunctions of atoms of the form $a=anc_{L^0}^{L}(L^0)\theta anc_{L^{0'}}^{L'}(L^{0'})$ (or equivalent to this form), where both L and L' are higher than the respective levels of c^{old} Then
12. For every atom $a=anc_{L^0}^{L}(L^0)\theta anc_{L^{0'}}^{L'}(L^{0'})$ in φ_{new}
15. Transform a to $anc_{L^{old}}^{L}(L^{old})\theta anc_{L^{old'}}^{L'}(L^{old'})$
16. EndFor
17. EndIf
18. Apply the transformed selection condition to c^{old} and derive a new data set DS^1.
19. Replace all the values of DS^1 with their ancestor values at the levels of c^{new}, resulting in a new data set DS^2.
20. Aggregate ("group by" in the relational semantics) on the tuples of DS^2, so that we produce c^{new}.

Figure 5.15: Algorithm Cube_Usability

Proof. We distinguish two cases for the theorem: (a) selection conditions involving only levels and (b) selection conditions involving atoms of the form $L_i\theta L_j$.
Case 1: atoms involving only levels, i.e., of the form $L_i\theta L_j$, $L_i^{old}\prec L_i$, $L_j^{old}\prec L_j$ or their equivalent ones. In **Lines 11-17** of Algorithm Cube_Usability, we transform the expression φ_{new} at the levels of the schema of c^{old}, and obtain the expression $\varphi_{n@o}$. To do so, we modify all the atoms of φ_{new} which are of the form $L_i\theta L_j$, $L_i^{old}\prec L_i$, $L_j^{old}\prec L_j$, to the equivalent form $anc_{L_i^0}^{L_i^{old}}(L_i^{old})\theta anc_{L_j^0}^{L_j^{old}}(L_j^{old})$. The equivalence holds because all the levels occurring in the expression are higher than the respective levels in the schema of c^{old}.
Line 18. The new selection condition $\varphi_{n@o}$ is equivalent to φ_{new}. Moreover since $\varphi_{new}\subseteq_S\varphi_{old}$, we can deduce (a) $\varphi_{n@o}\subseteq_S\varphi_{old}$, (b) $\varphi_{n@o}\wedge\varphi_{old}\equiv\varphi_{n@o}\equiv\varphi_{new}$. Thus, if we apply $\varphi_{n@o}$ to c^{old}, treating c^{old} as a data set, we have
$\sigma_{\varphi_{n@o}}(c^{old})=DS^1=(DS^0,\varphi_{n@o}\wedge\varphi_{old},[L_1^{old},\ldots,L_n^{old},M_{old}],agg_{old}(M^0))\equiv(DS^0,\varphi_{new},[L_1^{old},\ldots,L_n^{old},M_{old}],agg_{old}(M^0))=c^1$
Line 19-20. Next, we aggregate on the tuples of c^1, so that we produce c^{new}. Note that we use c^1 and c^{old} as data sets and not as expressions over the detailed data set. The transformation is done in the following manner:

$$DS^2=\{x'\in Tup(L_1^{new},\ldots,L_n^{new},M_{new})\mid \exists x\in c^1:\ x'[L_i^{new}]=anc_{L_i^{old}}^{L_i^{new}}(x[L_i^{old}]),1\leq i\leq n,$$
$$x'[M_{new}]=agg(\{f\mid\exists g\in c^1,x'[L_i^{new}]=anc_{L_i^{old}}^{L_i^{new}}(g[L_i^{old}]),1\leq i\leq n,f=g[M_{old}]\})\Leftrightarrow(1)$$
$$DS^2=\{x'\in Tup(L_1^{new},\ldots,L_n^{new},M_{new})\mid \exists y\in\varphi_{new}(DS^0),x\in c^1,\ (Note:\varphi_{n@o}\wedge\varphi_{old}\equiv\varphi_{new})$$
$$x[L_i^{old}]=anc_{L_i^0}^{L_i^{old}}(y[L_i^0]),1\leq i\leq n,\ x'[L_i^{new}]=anc_{L_i^{old}}^{L_i^{new}}(x[L_i^{old}]),1\leq i\leq n,$$
$$x'[M_{new}]=agg(\{f\mid\exists g\in c^1,x'[L_i^{new}]=anc_{L_i^{old}}^{L_i^{new}}(g[L_i^{old}]),1\leq i\leq n,$$

$f=agg_{old}(\{q|\exists z\in\varphi_{new}(DS^0), x[L_i^{old}]=anc_{L_i^0}^{L_i^{old}}(z[L_i^0]), 1\le i\le n, q=z[M^0]\})\}\Leftrightarrow(2)$

$DS^2=\{x'\in Tup(L_1^{new},\ldots,L_n^{new},M_{new})|\ \exists y\in\varphi_{new}(DS^0), x\in c^1,$

$x'[L_i^{new}]=anc_{L_i^{old}}^{L_i^{new}}(anc_{L_i^0}^{L_i^{old}}(y[L_i^0])), 1\le i\le n,$

$x'[M_{new}]=agg(\{f|\exists g\in c^1, z\in\varphi_{new}(DS^0),$

$x'[L_i^{new}]=anc_{L_i^{old}}^{L_i^{new}}(anc_{L_i^0}^{L_i^{old}}(z[L_i^0])), 1\le i\le n, f=agg_{old}(z[M^0])\})\}\ \Leftrightarrow\ (3)$

$DS^2=\{x'\in Tup(L_1^{new},\ldots,L_n^{new},M_{new})|\ \exists y\in\varphi_{new}(DS^0),$

$x'[L_i^{new}]=anc_{L_i^0}^{L_i^{new}}(y[L_i^0])), 1\le i\le n, x'[M_{new}]=agg(\{agg_{old}(z[M^0])|\exists z\in\varphi_{new}(DS^0),$

$x'[L_i^{new}]=anc_{L_i^0}^{L_i^{new}}(z[L_i^0])), 1\le i\le n\})\}\ \Leftrightarrow\ (4)$

$DS^2 = c^{new}$.

Discussion. The first equivalence (1) holds, since we have simply expanded the semantics for c^2, by incorporating the definition of c^1 into the formula. The second equivalence (2) obviously holds, since we just combine the atoms in the formula. The third equivalence (3) holds due to the transitivity of the ancestor function. The fourth equivalence (4) holds since we can always find a function agg, which applied over the function agg_{old} will produce the function agg_{new}. More specifically,

$$sum\circ sum=sum, min\circ min=min, max\circ max=max, sum\circ count=count.$$

Consequently, the rewriting operations do indeed produce the cube c^{new}.

Case 2: atoms involving levels and values, i.e., of the form $L_i\theta l$, $anc_{L^0}^{L}(L^0)\theta l$ or their equivalent ones. Let $\mathbf{L}$ be the schema of c^{new}.

Since $\varphi_{new}\subseteq_{\mathbf{L}}\varphi_{old}$ then there exist $\varphi'_{new},\varphi'_{old}$ produced from Algorithm $Check\_Atoms\_Usability$ as conjunctions of dimension intervals, such that:

- $\varphi'_{new},\varphi'_{old}$ are conjunctions of dimension intervals. Let a_i^{new} be a DI covering dimension D_i and belonging to φ'_{new}. The respective notation is also used for the atoms of φ'_{old}.
- $\varphi'_{new}\subseteq_{\mathbf{L}}\varphi'_{old}$, i.e., for each atom a_i^{new} there exists an atom a_i^{old} such that $a_i^{new}\subseteq_L a_i^{old}$ and of course $a_i^{new}\subseteq a_i^{old}$.
- $\varphi'_{new}\equiv\varphi_{new}$ and $\varphi'_{old}\equiv\varphi_{old}$.

The fact that $\varphi'_{new}\subseteq_{\mathbf{L}}\varphi'_{old}$ means that for each dimension D_i the domain of L_i^0 is partitioned with respect to the level L_i^{new}. If we call p_{ij}^{new} each partition created by φ'_{new}, with respect to level L_i^{new}, we have:

- $\bigcup_j p_{ij}^{new}\subseteq dom(L_i^0)$, $\bigcup_k p_{ik}^{old}\subseteq dom(L_i^0)$,
- for each p_{ij}^{new} there exists a p_{ik}^{old} such that $p_{ij}^{new}=p_{ik}^{old}$ and
- finally, if x_1,x_2 belong to the same p_{ij} then $anc_{L_i^0}^{L_i^{new}}(x_1)=anc_{L_i^0}^{L_i^{new}}(x_2)$. In other words, each partition is defined by the fact that its elements have a common ancestor at the level of L_i^{new}.

Due to the properties of the family of anc functions (i.e., disjointness of descendants and monotonicity) and the fact that $L_{old}\prec L_{new}$ we can claim that each partition defined with respect to the levels of c^{new} is the union of all the partitions that can be defined for the levels of c^{old}. **Notation:** We will use the letter p to denote partitions defined from levels of c^{new} and q for partitions defined from levels of c^{old}. The superscript in these symbols denotes the selection condition from which the partitions are derived.

For each p_{ij}^{new}, there exist a set of q_{il}^{new}such that

- $p_{ij}^{new}=\bigcup_l q_{il}^{new}$ and
- for each x_1,x_2 belonging to the same q_{ij} then $anc_{L_i^0}^{L_i^{old}}(x_1)=anc_{L_i^0}^{L_i^{old}}(x_2)$.

Exactly the same relationships hold for all the p_{ik}^{old}.

For example, the partition of a particular year is the union of all the partitions for its months, if we consider c^{old} to be at the month level).

The question arises, then, about the relationship between partitions and dimension intervals. We introduce the term covers between a DI a and a partition p if

Check_DI_Usability(p,a,L_{new})=true, i.e., p is a subset of a and they have the proper marginal conditions. Consequently, for each a_i^{new} the following hold

- there exists a set of partitions p_{ij}^{new}, such that $a_i^{new}=\bigcup_j p_{ij}^{new}$,
- there exists an atom a_i^{old} such that $a_i^{new} \subseteq_L a_i^{old}$,
- for each p_{ij}^{new} covered by a_i^{new} there exists a p_{ij}^{old} covered by a_i^{old} such that $p_{ij}^{new}=p_{ij}^{old}$,
- for each q_{ij}^{new} covered by a_i^{new} there exists a q_{ij}^{old} covered by a_i^{old} such that $q_{ij}^{new}=q_{ij}^{old}$.

For each dimension interval a_i over L_i^0 we can construct a corresponding interval b_i over L_i^{old}, such that if $a_i=(A_s,A_e)$, then $b_i=(B_s,B_e)=(anc_{L_i}^{L_i^{old}}(A_s), anc_{L_i}^{L_i^{old}}(A_e))$. Algorithm Cube_Usability produces exactly these DI's (remember that the domains of all the levels are isomorphic to the integers, which means that if $x \le l$ then x<succ(l) --transformation employed for <,> when the edge of a partition is involved). Moreover, if we consider an arbitrary a_i^{new} and the respective b_i^{new} produced by Algorithm Cube_Usability, and their respective a_i^{old}, b_i^{old} such that $a_i^{new} \subseteq_L a_i^{old}$ then, $b_i^{new} \subseteq_L b_i^{old}$.

Also, the sets of partitions $\mathbf{p}^{new}$,$\mathbf{q}^{new}$ covered by b_i^{new} are a subset of the respective partitions $\mathbf{p}^{old}$,$\mathbf{q}^{old}$ covered by b_i^{old}. This means that if we transform $\varphi'_{new}, \varphi'_{old}$ at the levels of c^{old}, as suggested by Algorithm Cube_Usability, then for each DI of $\varphi'_{new}, \varphi'_{old}$ there exists the respective DI at the level of c^{old} and $b_i^{new} \subseteq_L b_i^{old}$. We will call the result of the transformation of the expressions $\varphi'_{new}, \varphi'_{old}$, y'_{new}, y'_{old} respectively. Remember that although $\varphi'_{new}, \varphi'_{old}$ were defined at the detailed levels, y'_{new}, y'_{old} are defined at the levels of c^{old}. Obviously, since for all the DI's of y'_{new}, y'_{old}, $b_i^{new} \subseteq_L b_i^{old}$, we can claim that $y'_{new} \subseteq y'_{old}$ and thus $y'_{new} \subseteq_{[L_1^{old},\ldots,L_n^{old}]} y'_{old}$. Using the definition of **L**-containment, and based on the fact that the partitions covered are identical, we can deduce that for an arbitrary DS^0

$$\sigma_{y'_{new}}(c^{old}) \equiv (DS^0, \varphi'_{new} \wedge \varphi'_{old}, [L_1^{old},\ldots,L_n^{old}, M_{old}], agg_{old}(M^0))$$

(although, is φ'_{new} a subset the detailed equivalent of y'_{new} and the reverse does not hold)

Then, all we have to show is that what holds for $\varphi'_{new}, \varphi'_{old}$ holds also for $\varphi_{new}, \varphi_{old}$. Yet, this is trivial, since $\varphi'_{new}, \varphi'_{old}$ are equivalent to $\varphi_{new}, \varphi_{old}$. Consequently, if we transform φ_{old} at the levels of c^{old}, we obtain the expression $\varphi_{n@o}$ and the following hold

$$\sigma_{\varphi_{n@o}}(c^{old}) \equiv (DS^0, \varphi'_{new} \wedge \varphi'_{old}, [L_1^{old},\ldots,L_n^{old}, M_{old}], agg_{old}(M^0))$$
$$\equiv (DS^0, \varphi_{new} \wedge \varphi_{old}, [L_1^{old},\ldots,L_n^{old}, M_{old}], agg_{old}(M^0))$$
$$\equiv (DS^0, \varphi_{new}, [L_1^{old},\ldots,L_n^{old}, M_{old}], agg_{old}(M^0)).$$

The effect of Lines 19-20 of Algorithm Cube_Usability has been covered for **Case 1**. ■

Theorem 4.2 tests for usability, pairs of cubes involving conjunctive selection conditions which do not involve $\neq$ and $\neg$. Cubes involving disjunctive selection conditions can be treated in the usual way [Ullm89].

Note also, that the inverse ('and only if') of the theorem does not hold. Suppose the case of a particular dimension D, involving two levels low and high, where the desc relationship is a function (meaning that anc_{low}^{high} has an inverse function and the mapping from detailed to ancestor values is 1:1). Then, although condition (b) of Theorem 4.2 is violated, a cube at the level high can be used to compute a cube at the level low. Moreover, it is easy to construct an example which shows that the above techniques cannot be applied to a class of queries containing both atoms involving only levels (i.e., $L_1 \theta L_2$) and atoms involving levels and values (i.e., $L \theta l$).

Example 4.1. Let c^{new} and c^{old} be the cubes over DS^0 of Figure 5.16 defined as follows.

$c^{old}=(DS^0, \varphi_{old}, [Month,Country,Type,Salesman,Sum\_old], sum(Sales))$ and
$c^{new}=(DS^0, \varphi_{new}, [Month,Country,Categoty,Salesman,Sum\_new], sum(Sales))$

where φ_{old}=18-Feb-97≤day ∧ day≤3-Sep-97 ∧ $anc_{Item}^{Category}$(Item)="Books" and φ_{new}=1-Mar-97≤day ∧ day≤3-Sep-97 ∧ "Literature"≤anc_{Item}^{Type}(Item) ∧ anc_{Item}^{Type}(Item)≤"Philosophy".

To check whether c^{new} can be computed from c^{old} we apply Theorem 4.2. The schemata and aggregation functions of the two cubes are compatible (conditions (a), (b) of Theorem 4.2. Moreover,

φ_{new} is **L**-contained from φ_{old} with respect to the levels of c^{new}. Following Lines 2-10 of Algorithm Cube_Usability, we transform φ_{new} so that it can be applied to the schema of cube c^{old}. The transformations of lines 3-8 result in

$$\varphi_{n@o}=\text{Mar-97}\leq\text{Month} \wedge \text{Month}\leq\text{Sep-97} \wedge \text{“Literature”}\leq\text{Type} \wedge \text{Type}\leq\text{”Philosophy”}.$$

We apply the transformed selection condition to c^{old} (depicted in Figure 5.16a) and derive a new data set DS^1 (depicted in Figure 5.16b). Then, we replace all the values of DS^1 with their ancestor values at the levels of c^{new} (Line 19), resulting in a new data set DS^2 (depicted in Figure 5.16c). Finally, we aggregate the tuples of DS^2 and we produce c^{new} (depicted in Figure 5.16d).

Month	Type	Salesman	Country	Sum_old
Feb-97	Literature	Netz	USA	5
Sep-97	Philosophy	Netz	Japan	50
Sep-97	Literature	Netz	Japan	30

(a)

Month	Type	Salesman	Country	Sum_1
Sep-97	Philosophy	Netz	Japan	50
Sep-97	Literature	Netz	Japan	30

(b)

Month	Category	Salesman	Country	Sum_2
Sep-97	Book	Netz	Japan	50
Sep-97	Book	Netz	Japan	30

(c)

Month	Category	Salesman	Country	Sum_new
Sep-97	Book	Netz	Japan	80

(d)

Figure 5.16: Calculating c^{new} from c^{old}.

5. SUMMARY

In this chapter we provided a categorization of the work in the area of OLAP logical models by surveying some major efforts, including commercial tools, benchmarks and standards, and academic efforts. We have also attempted a comparison of the various models along several dimensions, including representation and querying aspects. Then, we presented a *logical* model for cubes based on the key observation that a cube is not a self-existing entity, but rather a view over an underlying data set. The proposed model is powerful enough to capture all the commonly encountered OLAP operations such as selection, roll-up and drill-down, through a sound and complete algebra. We have showed how this model can be used as the basis for processing cube operations and have provided syntactic characterisations for the problems of cube usability. Theorem 3.2, which provides these syntactic characterisations, is very important for the usual operations of the model. Two of the most eminent cases are: (a) navigation from a certain cube c to a cube having all its levels higher (or equal) than the respective levels of c and (b) selection over a certain cube c where all the levels acting as variables are higher (or equal) than the levels of c.

Of course, the applicability of Theorem 4.2 is not restricted in these two simple cases. Normally, an OLAP screen contains more than one cubes [Micr98]. Thus, an interactive OLAP session produces many cubes which possibly overlap. Computing a new set of cubes can possibly be achieved by using already computed and cached cubes (provided that they fulfil the criteria of Theorem 4.2. Consequently, the results on the problem of cube usability can be used both for the query optimisation and the caching processes. The cube usability results can also be applied in the problem of data warehouse / data mart design [ThLS99], where the optimal set of views (with respect to query and maintenance cost) has to be derived. Testing for cube usability can avoid redundancy in the final data warehouse schema and optimise the run-time of the design algorithm [LSTV99].

As future work, we plan to incorporate our results in a system under construction in NTUA. The modelling parts could be extended to take into account aspects of the hierarchy structure (partial ancestor functions, hierarchies that are not well captured as lattices [LeSh97], etc.). The theoretical results over query processing can be extended to handle optimisation issues for a broader set of selection conditions, partial rewritings and optimisation of the physical execution for cube operations. Finally, a challenging issue is how to devise smarter algorithms for the cube usability problems.

Chapter 6
Iktinos: an OLAP tool

1. INTRODUCTION

Iktinos is a prototype OLAP tool developed in the Knowledge and Database Systems Laboratory (KDBSL) of the National Technical University of Athens (NTUA). Iktinos implements the logical cube model of [Vass98], which is a direct ancestor of the cube model presented in Chapter 5. The basic differences of the two models are more in terms of formalism rather than in terms of functionality.

Figure 1. A screenshot from Iktinos

The logical model of Iktinos is based on the idea of *cubes*, i.e., aggregations/summarizations of data with respect to specific attributes. In this model, cubes are views defined over a detailed data set, called *basic cube*. Each *dimension* of the data is organized as a hierarchy of aggregation *levels*, modeled as a lattice. All the cube operations aim to track the sequence of operations that the user has performed in the past –i.e., they retain the history of the user's navigations. The model captures the most common OLAP operations (*roll-up, drill-down, selection, slicing*).

Architecture: Iktinos uses a relational mapping of the model, implemented in Informix Universal Server. The Iktinos server listens to requests performed by the client software, through a perl program that implements this communcation. The server maps the generic OLAP (like roll-up, drill-down etc.) requests to relational operations (group-by's, selections, etc.) and returns the result to the user.

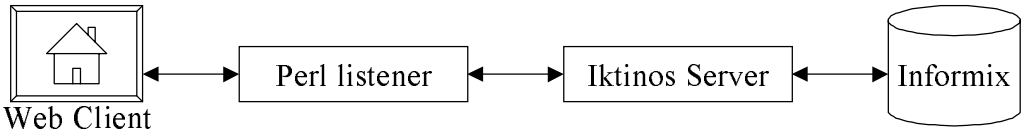


Figure 2. The architecture Iktinos

Iktinos can be accessed at http://www.dbnet.ece.ntua.gr/~dwq/iktinos/ The requirements from a web client are Netscape 4.x or Internet Explorer 4.x, with the Java plug-in installed (which is available from http://java.sun.com/cgi-bin/plugin-download.sh?).

2. IKTINOS' MODEL OF MULTIDIMENSIONAL SPACE AND CUBES

In the model of [Vass98], the existence of a cube at the lowest level of detail is assumed. In the terminology of [Vass98], this cube is called the *basic cube*. The application of any of the algebraic operations of the model upon the basic cube produces a new cube, possibly aggregated. The operations can continue over any of the cubes (basic or not) producing new cubes. Each of these cubes, apart from a schema and an instance, is also characterized by a mapping of its tuples to the tuples of the detailed basic cube, which ultimately produce them. The set of these detailed tuples is called the *base cube* of the aggregated cube.

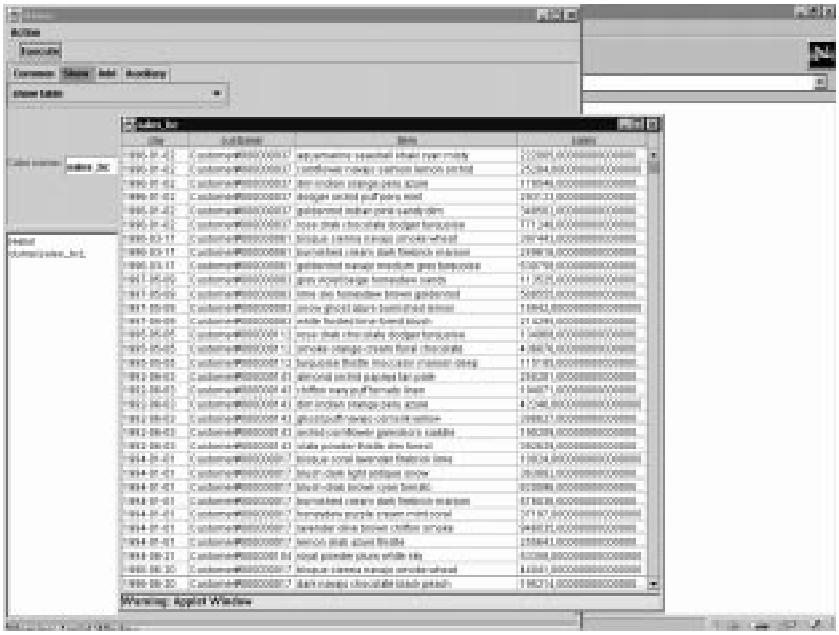


Figure 3. A basic cube.

Of course, the multidimensional view of data, imposes the existence of *dimensions* in the model. Each dimension is characterized by a set of *levels* of possible aggregation of the information. These levels form a lattice, i.e., we impose the existence of a most detailed level for each dimension. This is, of course, the reason that enables the definition of basic cubes. Each path in the lattice of levels is called a *dimension path.* The values belonging to the domain of levels of the same dimension path are related through an *ancestor function*, which is a mapping from the lowest level of granularity to the higher (here: ancestor) one. The *measures* of the cubes are also considered to be dimension, comprising only one level.

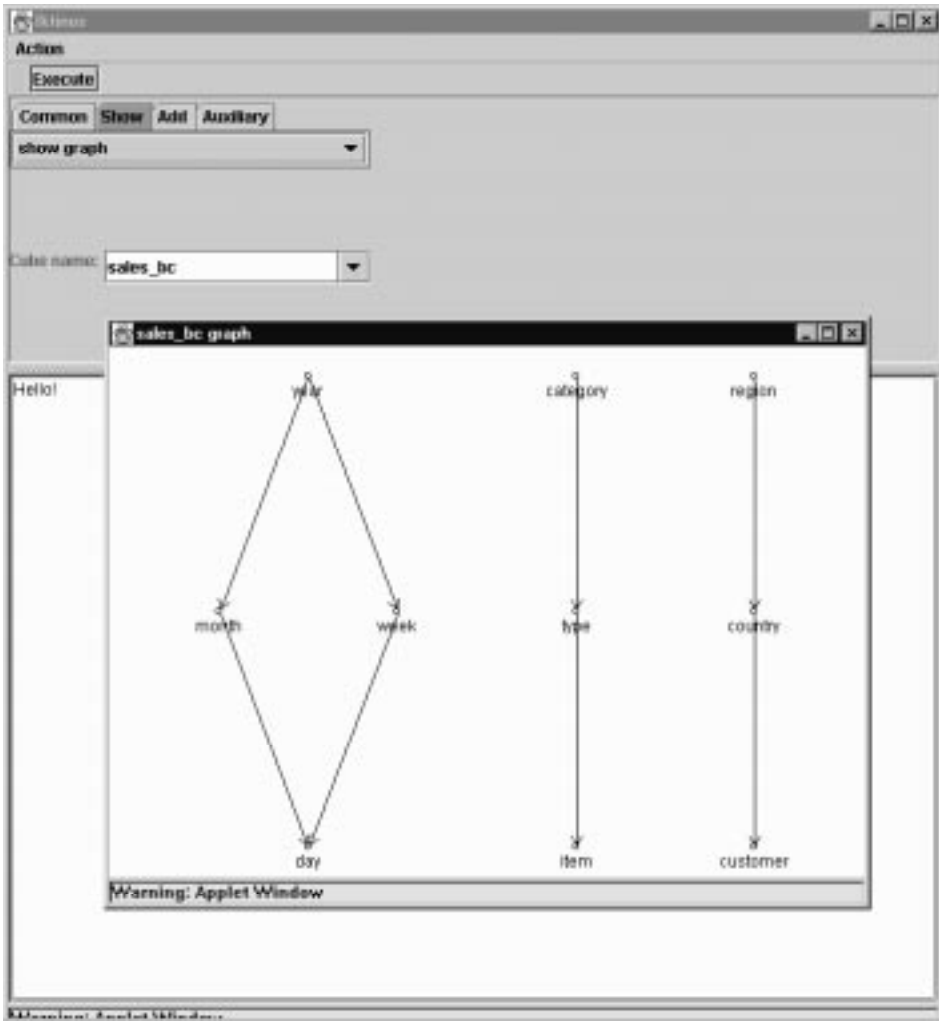


Figure 4.Each *dimension* of the data is organized as a hierarchy of aggregation *levels*, modeled as a lattice.

Formally, we can define a *Cube* C as a 4-tuple <***D***, ***L***, C_b, ***R***>, where

- ***D*** = <D_1, D_2, ... D_n, M> is a list of dimensions. M is a dimension that represents the measure of the cube.
- ***L*** = <DL_1, DL_2, ... DL_n, *ML> is a list of dimension levels. *ML is the dimension level of the measure of the cube.
- C_b is a *basic_cube*. We will call C_b, the *base_cube* of C (C_b = *base_cube(C)*). The data of C_b can be used for the calculation of the contents of C
- ***R*** is a set of cell data -i.e. a set of tuples of the form as a tuple $x = [x_1, x_2, ..., x_n, *m]$, where $\forall\, i \text{ in } [1, ..n], x_i \in dom(DL_i)$ and $*m \in dom(*ML)$.

We can define several operations to the cube model, which are implemented all by Iktinos. *Roll-up* is a basic operation in OLAP, performing the summarization of data to a higher level of aggregation. In the Figure 5 we roll-up the data from day to year level, through the use of the *navigation* algebraic operator

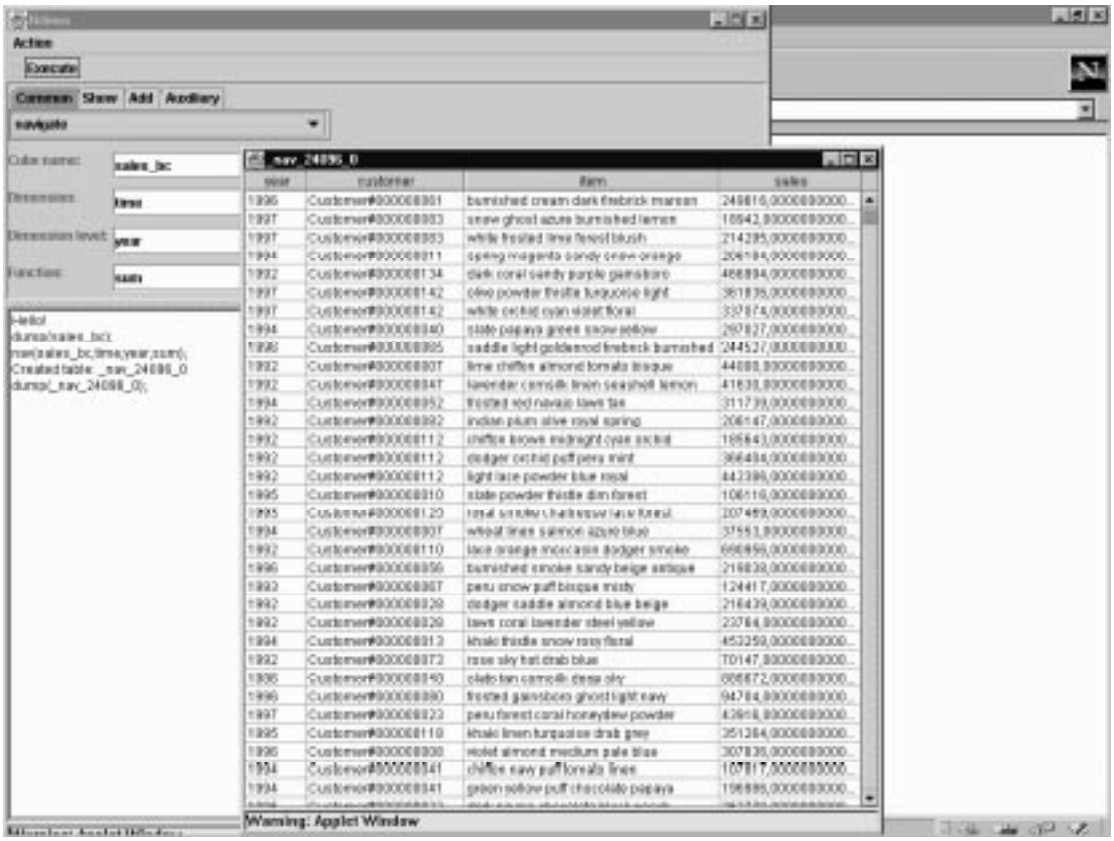


Figure 5.Roll-up from day to year level.

Another example of the roll-up operation is shown in the following figure, where we roll-up from the item to the category level, of dimension product.

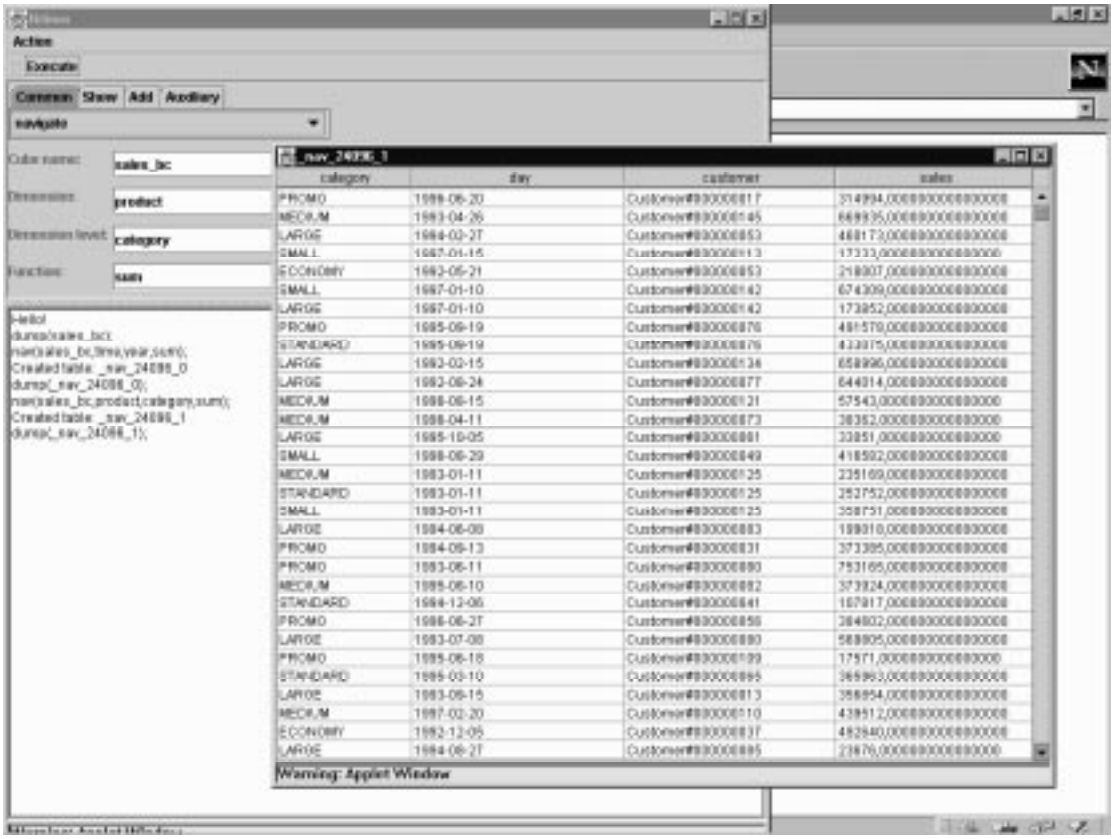


Figure 6.Roll-up from item to catogory dimension

Defining the cubes as views over the detailed data set, allows us to track the sequence of operations that the user has performed. For example, observe Figure 7:

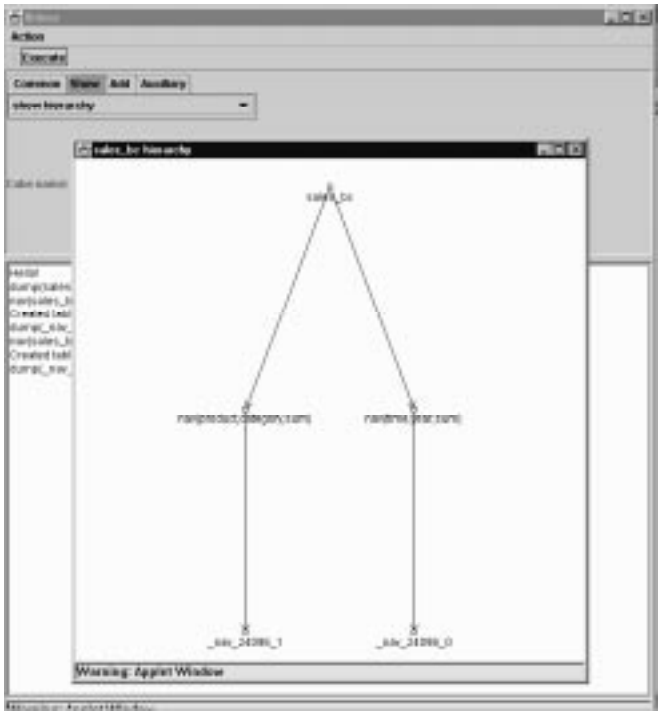


Figure 7.Tracking of user operations

We can also select subsets of the information, through the use of the *dice* operator. Observe Figure 8:

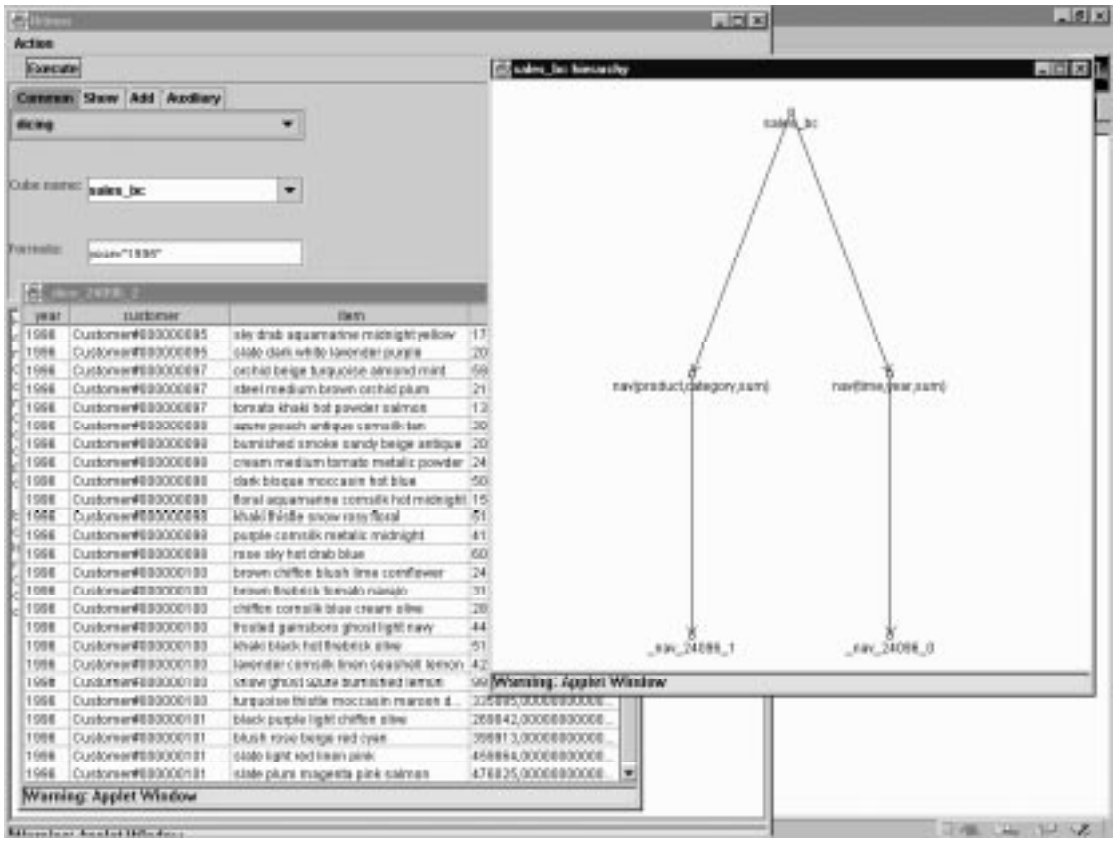


Figure 8. Selection (dicing) of information.

Another algebraic operation is *slicing*, used to ignore one or more dimensions in the aggregation. Notice that in this example we also change the aggregation function from sum to count, the new cube retains the previous selection.

Figure 9. Slicing, i.e., aggregation over a whole dimension

Drilling-down, which is the deaggregation of the information, is also performed efficiently using the view definition of the cube over the basic cube.

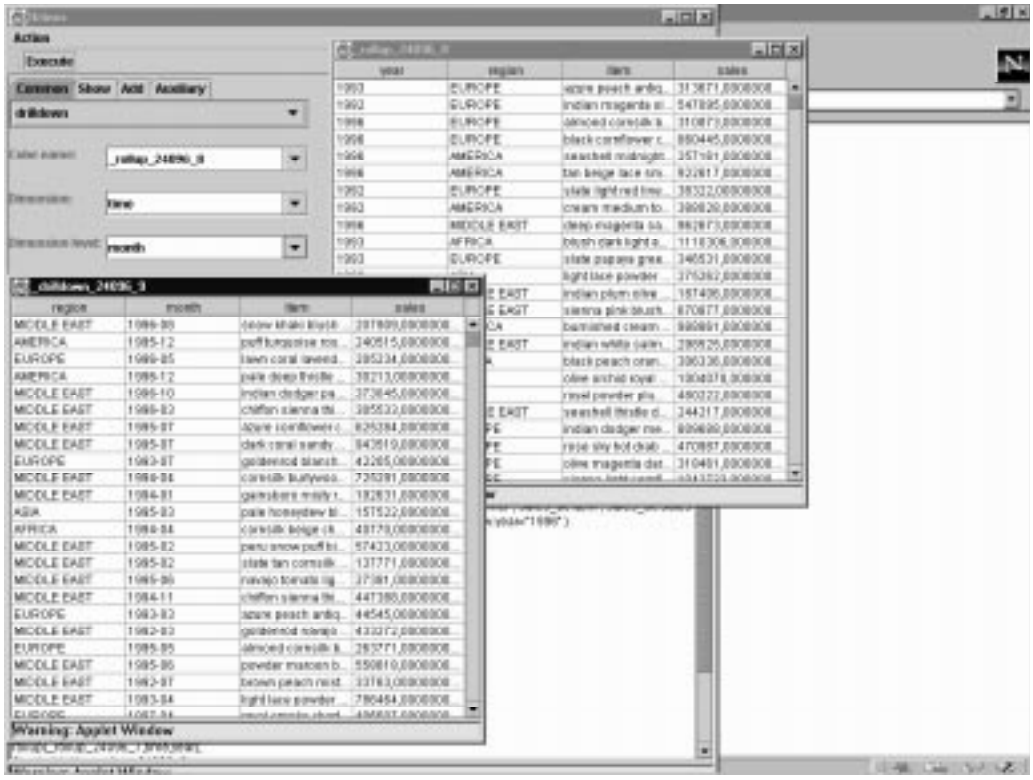


Figure 10. Drill-down, i.e., de-aggregating a cube

Formally, there are two kinds of operations: first-class operations, like the ones which have already been mentioned, and auxiliary, which enable the execution of the first-class operations. We proceed to give a formal definition of these operations.

Level_Climbing. Let $\underline{d}$ be a set of dimensions belonging to C and $\underline{dl}$ the set of the corresponding dimension levels of C. Without loss of generality we assume that $\underline{d}$ consists of the last k dimensions of D. Let also $\underline{dl}_{old}$ be the original dimension levels of C, belonging to $\underline{d}$: $\underline{dl}_{old} = \{DL_{n-k+1}, \ldots, DL_n\}$. Then, *C' = Level_Climbing(C, d, dl) = LC(C, d, dl)* is defined as follows:

$\boldsymbol{D}' = \boldsymbol{D}$, $\boldsymbol{L}' = \boldsymbol{L} - \underline{dl}_{old} \cup \underline{dl}$, $C_b' = C_b$ and

$\boldsymbol{R}' = \{x \mid \exists\ y \in \boldsymbol{R}$: *dimensions*$(x)(D_i)$ = *dimensions*$(y)(D_i)$ $\forall\ D_i \notin \underline{d} \wedge$ *dimensions*$(x)(D_i)$ = *ancestor*(*dimensions*$(y)(D_i)$, dl_j), $\forall\ D_i \in \underline{d}$, $dl_j \in \underline{dl}$, $dl_j \in$ *levels*$(D_j) \wedge$ *measure*(x) = *measure*(y), if $M \notin \underline{d}\}$

We impose the restrictions that $\underline{d}$, $\underline{dl}$ are consistent with each other and that for all the dimension levels of $\underline{dl}$, the respective dimension levels of $\underline{dl}_{old}$ belong to the same dimension path and are of lower or equal level (for example, one cannot perform *Level_Climbing* between months and weeks). Intuitively, *Level_Climbing* is the replacement of all values of a set of dimensions with values of dimension levels of higher level.

Packing. We define *C' = Packing(C) = P(C)* as follows:

$\boldsymbol{D}' = \boldsymbol{D}$, $\boldsymbol{L}' = \boldsymbol{L}$, $C_b' = C_b$ and

$\boldsymbol{R}' = \{x \mid \exists\ y \in \boldsymbol{R}$: *dimensions*$(x)(D_i)$ = *dimensions*$(y)(D_i)$ $\forall\ i \in 1, \ldots, n \wedge$ *measure*(x) = $\{l \mid \exists\ t \in \boldsymbol{R}$, *dimensions*$(y)$ = *dimensions*$(t) \wedge l$ = *measure*$(t)\}\}$

Intuitively, *packing* is the consolidation of the cube, through the merging of multiple instances having the same dimension values into one. *Packing* has bag semantics.

Function_Application. Let *f* be a function belonging to {*sum, avg, count, min, rank(n), no-operation*}. Then, *C' = Function_Application(C, f) = F(C, f)* is defined as follows:

$\boldsymbol{D}' = \boldsymbol{D}$, $\boldsymbol{L}' = \boldsymbol{L}$, $C_b' = C_b$ and

$\boldsymbol{R}' = \{x \mid \exists\ y \in \boldsymbol{R}$: *dimensions*$(x)$ = *dimensions*$(y) \wedge$ *measure*(x) = f(*measure*(y)) $\}$

Intuitively, *Function_application* is the application of a specific function to the measure of a cube.

Projection. Let d be a projected dimension. $C' = Projection(C, d) = \pi(C, d)$ is then defined, as follows:

$\boldsymbol{D}' = \boldsymbol{D}$ - d, $\boldsymbol{L}' = \boldsymbol{L}$ - DL, DL $\in$ *levels*(d), DL $\in \boldsymbol{L}$,

$C_b' = \langle \boldsymbol{D_b}', \boldsymbol{L_b}', C_b', \boldsymbol{R_b}' \rangle$, where,

$\boldsymbol{D_b}' = \boldsymbol{D_b}$ - d,

$\boldsymbol{L_b}' = \boldsymbol{L_b}$ - *levels*(d)(1), and

$\boldsymbol{R_b}' = \{x \mid \forall\ y \in \boldsymbol{R_b}$, *dimensions*$(x)(D_i)$ = *dimensions*$(y)(D_i)$, $\forall\ D_i \neq d$, $i \in 1, \ldots, n \wedge$ *measure*(x) = *measure*$(y)\}$

$\boldsymbol{R}' = \{x \mid \exists\ y \in \boldsymbol{R}$: *dimensions*$(x)(D_i)$ = *dimensions*$(y)(D_i)$, $\forall\ D_i \neq d$, $i \in 1, \ldots, n \wedge$ *measure*(x) = *measure*$(y)\}$

Intuitively, *projection* is the deletion of a dimension both from the cube and its *base_cube*.

Navigation. Let *d* be the dimension over which we navigate, *dl* the target level of the navigation and *f* the applied aggregate function. Suppose that the dimension d is the *i-th* element of D. Then, we define *C' = Navigation(C, d, dl, f)* as follows:

C' = Navigation(C, d, dl, f) = F(P(LC(C_b, {$D_1, D_2, \ldots, d, \ldots, D_n$}, {$DL_1, DL_2, \ldots, dl, \ldots, DL_n$} *)),f)*

The purpose of the navigation operator is to take a cube from a specific state, change the level of a specific dimension, pack the result and produce a new cube with a new state, through the use of an aggregate function. The dimensions of the new cube are the dimensions of the old one. The dimension levels are also the same, except for the one of the dimension where we change level. Notice that the restrictions imposed by *Level_Climbing*, regarding the position of the respective dimension levels in the dimension lattice, still hold. Furthermore, the *base_cube* remains the same. The *Navigation* is performed at the level of the *base_cube*. *Roll-up* and *dril-down* are uniformely expressed as navigations over the base cube of a cube.

Dicing. Let *d* be the dimension over which we perform the dicing, σ a formula consisting of a dimension, an operator and a value *v*. We assume that v belongs to the values of the dimension level of *d* in C and that σ is applicable to *d* (in the sense presented in [15]) -i.e. that {<, =} are applied to atomic dimension levels and {≡, ⊂, ∈} to multi-valued ones). Let $\sigma(v)$ be of the form *d op v*. Then, *C' = Dicing(C, d, σ(v))* is defined as follows:

$\boldsymbol{D}' = \boldsymbol{D}$, $\boldsymbol{L} = \boldsymbol{L}'$,
$C_b' = <\boldsymbol{D_b}', \boldsymbol{L_b}', C_b', \boldsymbol{R_b}'>$, where
$\boldsymbol{D_b}' = C_b.\boldsymbol{D_b}$, $\boldsymbol{L_b}' = C_b.\boldsymbol{L_b}$, and
$\boldsymbol{R_b}' = \{x \mid x \in C_b.\boldsymbol{R_b}, x[d]$ op $y =$ true, $y \in$ *descendants*$(v,$ *levels(d)*$(1))\}$
$\boldsymbol{R}' = \{x \mid \exists\, x \in \boldsymbol{R}, x[d]$ op $v =$ true$\}$
Intuitively, *dicing* is a simple form of selection. Yet, it has its impact both on the cube itself and its *base_cube*. We are allowed to check for descendants of *v* in the *base_cube*, since each dimension path ends at a dimension level of the lowest granularity and the *base_cube* is in the lowest possible granularity for all levels.

Slicing. Let *d* be the dimension which we slice and *f* the applied aggregate function. We define *Slicing* as follows:
$C' = Slicing(C, d, f) = F(P(\pi(LC(C_b, \{D_1, D_2, ..., d, ..., D_n\}, \{DL_1, DL_2, ..., dl, ..., DL_n\}\ ), d)),f)$
The purpose of the slicing operator is to take a cube from a specific state, cut out a specified dimension and aggregate over the rest of the dimensions, using an aggregation function. Notice that all the restrictions of *Level_Climbing* implicitly hold, without realy affecting the *Slicing* operation.

3. SQL MAPPING OF THE MULTIDIMENSIONAL MODEL

In this section we map multidimensional cubes, defined in Section 3, to relational SQL statements. The motivation for the relational mapping is double: on the one hand, Iktinos, being and engine performing Relational OLAP must be able to map multidimensional to relational entities and on the other hand, the data warehouse administrator can be helped to check out whether a relational database fulfills the requirements to model a cube (and vice versa -what kind of database one needs to construct in order to be able to map a cube to relational tables). At the end of the section a mapping of our multidimensional model to multidimensional arrays (used as logical structures in engines performing MOLAP) is also presented. A detailed presentation of the formal aspects of these mappings is found in [AnPa98, Vass98a] and is outside the scope of this document.

SQL mapping for level_climbing(C, d, dl). Let (a) d be the list of dimensions; (b) dl the list of the levels, given as arguments at the algebraic operation and old_dl be the list of the respective levels at which C is already found; (c) n be the size of this list; (d) rem_dl be the list of rest of the levels of C; (e) _temp[i] be a temporary table created by projecting the useful columns from the i-th dimension table (for each of the elements of d).We suppose the schema of _temp[i] to be (_temp[i].from_dl, _temp[i].to_dl), where the former belongs to old_dl and the latter to dl. Then, the SQL-command performing **level_climbing(C, d, dl)** is as follows:

SELECT <**dl**>, <**rem_dl**>
FROM **C**, $\forall$ i $\in$ {0,…,n-1} <**_temp[i]**>
WHERE $\forall$ i $\in$ {0,…,n-1} <**C.old_dl[i]=_temp[i].from_dl**>

The terms of the WHERE clause, are, of course, connected by "AND" connectives.

SQL mapping for aggregate(C,*f*). Let (a) m_dim be the dimension of the measure of the cube and m_dl the respective level; (b) d be the list of dimensions; (b) dl the list of the levels at which C is already found; (c) n be the size of this list and (d) C_b the base cube of C. Then the SQL program performing **aggregate(C, f)** is as follows:
1. `level_climbing(`C_b`, d, dl);`
2. one of the following SQL statements, based on the aggregate function:
if agg in {sum, avg, count, min}

```
SELECT  <dl>, agg(<m_dl>)
FROM    C_L
GROUP BY <dl>
```

else, if the aggregate function is rank(n) (i.e. the first n rows, according to an ordering criterion)

```
SELECT <dl>, <m_dl>
FROM   C_L a
WHERE  ranking > (SELECT count(*)
                  FROM   C_L b
                  WHERE  ∀ i ∈ {0,…n-1} a.dl[i]=b.dl[i]
                  AND    b.m_dl in (SELECT c.m_dl
                                    FROM   C_L c
                                    WHERE ∀ i ∈ {0,…n-1}
                                              c.dl[i]=b.dl[i]
                                          AND
                                      c.m_dl > a.m_dl))
```

SQL mapping for navigation(C, d, dl, *f*), rollup(C, d, dl) and drilldown(C, d, dl). Let (a) m_dim be the dimension of the measure of the cube and m_dl the respective level; (b) dl the target level of the operation and d the respective dimension; (c) d_array is the list of the dimensions of the cube, except for d, and dl_array the respective array for levels; (d) n be the size of this array. Then the SQL program performing **navigation(C, d, dl, f)** is as follows:

1. C_L=level_climbing(C, d, dl);
2. aggregate(C_L,f).

Rollup and drill-down implicitly use the sum function. Their difference, is, of course, the direction in the level hierarchy: the former moves to coarser and the latter to finer granularities.

SQL mapping for projection(C, d). Let (a) d be the projected dimension of the cube, (b) rem_d be the remaining dimensions and rem_dl be the respective levels of the cube; (c) rem_dl be the same list for the base cube of C, C_b. Then the SQL-command for **projection(C,d)**, with respect to the new cube C' is

```
SELECT  <rem_dl>
FROM    C
```

and with respect to the new base cube C_b':

```
SELECT  <rem_dl_bc>
FROM    C_b
```

SQL mapping for slicing(C, d, f). The program for **slicing(C,d,f)** is simple:

1. C_p=projection(C,d), resulting also to new base cube C_{pb}
2. Aggregate(C_p,f).

SQL mapping for dicing(C, φ). The SQL-command for **dicing(C,φ)**, with respect to the new cube C' is

```
SELECT  *
FROM    C
WHERE φ
```

and with respect to the new base cube C_b':

```
SELECT  <rem_dl_bc>
FROM    C_b
WHERE φ'
```

where φ' is the equivalent of φ, reduced at the base cube level.

SQL mapping for union(C1, C2) and difference(C1, C2). The respective SQL statements for the new cube C' and its base cube C'$_b$ are:

SQL statement for C'	**SQL statement for C'$_b$**
SELECT *	SELECT *
FROM C1	FROM $C1_b$
UNION [MINUS]	**UNION [MINUS]**
SELECT *	SELECT *
FROM C2	FROM $C2_b$

A mapping of the multidimensional model to multidimensional arrays. The multidimensional model can trivially be mapped to multidimensional arrays, practically in the same way it is done in [CaTo98]. We assume that there exists a mapping function *enum(d)* between a value *d* of a dimension level *l* and the set of integers. In other words, for each dimension level, we assign a unique integer to each one of its values. The assignment is done in a contiguous fashion. As a result, each value $x = [x_1, x_2, ..., x_n, *m]$, belonging to the cell data of a cube can be considered to be as the conjunction of coordinates $[enum(x_1), enum(x_2), ..., enum(x_n)]$ with value $*m$.

4. CONCLUSIONS

In this chapter we have presented *Iktinos*, a prototype OLAP tool developed in the Knowledge and Database Systems Laboratory (KDBSL) of the National Technical University of Athens (NTUA). Iktinos implements the logical cube model of [Vass98], which is a direct ancestor of the cube model presented in Chapter 5. The logical model of Iktinos is based on the idea of cubes, i.e., aggregations/summarizations of data with respect to specific attributes. In this model, cubes are views defined over a detailed data set, called basic cube. In this chapter we have presented the functionality of the tool, its theoretical background and the mapping of thetheoretical background to physical data structures. Several people have worked for the implementation of Iktinos, namely A. Papagianni, P. Andritsos, V. Papadimos and M. Hadjieleftheriou.

Chapter 7
Conclusions

This document presented a set of results towards the effective modeling and management of data warehouse metadata with special treatment to data warehouse quality. The first major result that we presented was a *general framework for the treatment of data warehouse metadata* in a metadata repository. The framework requires the classification of metadata in at least two instantiation layers and three perspectives. The metamodel layer constitutes the schema of the metadata repository and the metadata layer the actual meta-information for a particular data warehouse.

We linked this framework to a well-defined approach for the architecture of the data warehouse [JJQV99]. Then, we presented our proposal for a *quality metamodel*, which builds on the widely accepted Goal-Question-Metric approach for the quality management of information systems. Moreover, we enriched the generic metamodel layer with patterns concerning the linkage of (a) quality metrics to data warehouse objects and (b) of data warehouse stakeholders to template quality goals. The exploitation of the quality model can be performed in versatile ways. It is important to note that as far as the lifecycle of the data warehouse is concerned, this usage can be done in a dual fashion. ***Ex-post***, the metadata repository can be used as a log for the management of quality. All the steps of the GQM process are, thus, traced in the metadata repository and can be re-used for further evaluation. Notice, that not only do we provide an initial specialization of the quality metamodel for common data warehouse processes, but the data warehouse stakeholder can further detail this provision with his own templates for the quality management of his specific data warehouse, in a similar fashion. Secondly, the use of the ConceptBase metadata repository can be exploited, due to its querying facilities. Third, the quality metamodel is coherent with the generic metadata framework for data warehouses that we previously introduced. Thus, every new data warehouse object can be linked to metrics and measurements for its quality gracefully, without any change to the schema of the repository.

Then, we went on to describe *a metamodel for data warehouse operational processes*. This metamodel enables data warehouse management, design and evolution based on a high level conceptual perspective, which can be linked to the actual structural and physical aspects of the data warehouse architecture. This metamodel is capable of modeling complex activities, their interrelationships, the relationship of activities with data sources and execution details. Finally, the metamodel complements proposed architecture and quality models in a coherent fashion, resulting in a full framework for data warehouse metamodeling. We have implemented this metamodel using the language Telos and the metadata repository system ConceptBase.

The ***ex ante*** treatment of the metadata repository is enabled by a full set of *steps*, i.e., quality question, which constitute our *methodology for data warehouse quality management and the quality-oriented evolution of a data warehouse* based on the architecture, process and quality metamodels. 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, in one hand, to map a high-level subjective quality goal into the measurement of a set of interrelated quality factors, and, in 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. 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 requirement of a stakeholder. We have verified our methodology in a set of case studies, one of which has also been presented in this chapter as an example of the partial application of the methodology.

Special attention was paid to a particular part of the architecture metamodel, the *modeling of OLAP databases*. To this end, we first provided a categorization of the work in the area of OLAP logical models by surveying some major efforts, including commercial tools, benchmarks and standards, and academic efforts. We have also attempted a comparison of the various models along several dimensions, including representation and querying aspects. Our contribution lies in the introduction a *logical* model for cubes based on the key observation that a cube is not a self-existing entity, but rather a view over an underlying data set. The proposed model is powerful enough to capture all the commonly encountered OLAP operations such as selection, roll-up and drill-down, through a sound and complete algebra. We have also showed how this model can be used as the basis for processing cube operations and have provided *syntactic characterisations for the problems of cube usability*. These syntactic characterisations, are very important for the usual operations of the model. Two of the most eminent cases are: (a) navigation from a certain cube `c` to a cube having all its levels higher (or equal) than the respective levels of `c` and (b) selection over a certain cube `c` where all the levels acting as variables are higher (or equal) than the levels of `c`. Of course, the applicability of our results is not restricted in these two simple cases. Since an OLAP screen contains more than one cubes, an interactive OLAP session produces many cubes which possibly overlap. Computing a new set of cubes can possibly be achieved by using already computed and cached cubes. Consequently, the results on the problem of cube usability can be used both for the query optimisation and the caching processes. The cube usability results can also be applied in the problem of data warehouse / data mart design, where the optimal set of views (with respect to query and maintenance cost) has to be derived. Testing for cube usability can avoid redundancy in the final data warehouse schema and optimise the run-time of the design algorithm. *An implementation prototype, based on an earlier version of our metamodel has also been demonstrated*.

Research can continue in several ways using our results. As far as the process metamodel is concerned, we have dealt only with the operational processes of a data warehouse environment. Yet, there are also design processes in such an environment, which do not seem to fit this model so smoothly. It is in our future plans to investigate the modeling of design processes and to capture the trace of evolution in a data warehouse. Also, we have used the *global-as-view* approach for the data warehouse definition, i.e., we reduce the definition of the data warehouse materialized views to the data sources. We plan to investigate the possibility of using the *local-as-view* approach (which means reducing both the view definitions and the data sources to a global enterprise model), as it appears to provide several benefits

that the global-as-view approach lacks. As far as the proposed methodology is concerned, we believe that its full application in a wider extent in the future will provide the academic community with the insight for further tuning.
As far as the modeling and optimization of OLAP operations is concerned, several extensions could be persued. The modelling parts could be extended to take into account aspects of the hierarchy structure (partial ancestor functions, hierarchies that are not well captured as lattices, etc.). The theoretical results over query processing can be extended to handle optimisation issues for a broader set of selection conditions, partial rewritings and optimisation of the physical execution for cube operations. Finally, a challenging issue is how to devise smarter algorithms for the cube usability problems. We plan to explore all these open issues in the context of the European IST EDITH.

Acknowledgments

Thanks are due to several persons. First, I must thank Prof. Timos Sellis and Prof. Yannis Vassiliou, for supervising my thesis and giving me the opportunity to indulge in this particular area of knowledge. Their versatile support throughout these years was a basic factor for the completion of my thesis. Christoph Quix and Spiros Skiadopoulos were my colleagues who contributed the most in the successful completion of my thesis. I would never make it without their help; for this I am grateful to them. I also wish to thank Prof. M. Jarke, Prof. M. Bouzeghoub, and Prof. M. Jeusfeld for a long, productive cooperation during the past years. I am much obliged to Prof. E. Skordalakis, Prof. M. Lenzerini, Prof. E. Zachos and Dr. D. Theodoratos for comments, discussions and help any time it was requested. Of course, I must mention all the members of the Knowledge and Database Systems Laboratory (KDBSL) at the National Technical University of Athens (NTUA) for their support (and tolerance!) all these years. Out of them, A. Papagianni, P. Andritsos, V. Papadimos, M. Hadjieleftheriou, N. Polyzotis, A. Tsois, P. Xeros, G. Adamopoulos and Dr. S. Ligoudistianos have contributed with one way or another to this thesis; I thank them too. My family has played a big part in the completion of this thesis: I would like to publicly thank them too. Finally, it should be noted that all the work presented in this document, was part of the European Esprit project DWQ [JaVa97].

References

[AdCo97] AdCom Services. Data Warehouse - Project Proposal. http://pulsar.adcom.uci.edu:80/dwh/dwpro4/prohome.html, 1997.

[AgAh87] N. Agmon and N.Ahituv. Assesing data reliability in an information system. J. Management Information Systems, vol. 4, no. 2, 1987.

[AgGS95] R. Agrawal, A. Gupta, and S. Sarawagi. Modelling multidimensional databases. Technical report, IBM Almaden Research Center, San Jose, California, 1995.

[Akao90] Akao, Y., ed. Quality Function Deployment. Productivity Press, Cambridge MA. , 1990.

[AnPa98] P. Andritsos, A. Papagianni. Development of an OLAP tool. Technical Report KDBSL-DIPL-98-06. Knowledge and Database Systems Laboratory, NTUA. July 1998

[Arbo96] Arbor Software Corporation. *Arbor Essbase*.http://www.arborsoft.com/essbase.html, 1996.

[ASII91] American Supplier Institute, Inc.. Taguchi Methods: Introduction to Quality Engineering. Allen Park, Mich. 1991.

[BaCR94] V. R. Basili, G.Caldiera, H. D. Rombach. The Goal Question Metric Approach. *Encyclopedia of Software Engineering - 2 Volume Set*, pp 528-532, John Wiley & Sons, Inc. (1994). Available at http://www.cs.umd.edu/users/basili/papers.html

[BaPa82] D.P. Ballou and H.L. Pazer. The impact of inspector failability on the inspection policy serial production system. Management Science, vol. 28, no.4, 1982.

[BaPa85] D.P. Ballou and H.L. Pazer. Modelling data and process quality multi-input, multi-output information systems. Management Science, vol. 31, no.2 1985.

[BaPa87] D.P. Ballou and H.L. Pazer. Cost/quality tradeoffs for control procedures information systems. OMEGA: Internatinal J. Management Science, vol. 15, no. 6, 1987.

[BaPT97] E. Baralis, S. Paraboschi and E. Teniente. Materialized View Selection in a Multidimensional Database. *Proceedings of the 23rd International Conference on Very Large Databases (VLDB)*, August 1997.

[BaSa98] F. Baader and U. Sattler. Description Logics with Concrete Domains and Aggregation. *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)*, pp. 336-340, 1998.

[BaTa89] D.P. Ballou and K.G. Tayi. Methodology for allocating resources for data quality enhancement. Communications of the ACM (CACM), vol. 32, no. 3, 1989.

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

[BBC*99] P.A. Bernstein, Th. Bergstraesser, J. Carlson, S. Pal, P. Sanders and D. Shutt. Microsoft Repository Version 2 and the Open Information Model. Information Systems, vol. 24, no. 2, 1999.

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

[Best94] D.H. Besterfield. Quality Control. Englewood Cliffs, N.J.: Prentice Hall, 1994.

[BFL*97] M. Bouzeghoub, F. Fabret, F. Llirbat, M. Matulovic and E. Simon. Designing data warehouse refreshment system. DWQ Technical report, October 1997.

[BFMS98] M. Bouzeghoub, F. Fabret, M. Matulovic, E. Simon. Data Warehouse Refreshment: A Design Perspective from Quality Requirements. *Technical Report D8.5, DWQ Consortium* (1998). Available at http://www.dbnet.ece.ntua.gr/~dwq/

[Boed87] R.F. Boedecker. Eleven Conditions for Excellence. The IBM Total Quality Improvement Process. Quincy, Mass. American Institute of Management, 1987.

[Boeh89] B. Boehm. Software Risk Management. IEEE Computer Society Press, CA, 1989.

[BoFM99] M. Bouzeghoub, F. Fabret, M. Matulovic. Modeling Data Warehouse Refreshment Process as a Workflow Application. In Proc. Intl. Workshop on Design and Management of Data Warehouses (DMDW'99), Heidelberg, Germany, June 1999

[BSHD98] M. Blaschka, C. Sapia, G. Hofling and B. Dinter. Finding your way through multidimensional data models. *Proceedings of 9th Intl. Workshop On Database and Expert Systems Applications* (DEXA), Vienna, Austria, August 1998.

[BWPT93] D.P. Ballou, R.Y. Wang, H.L. Pazer and K.G. Tayi. Modelling Data Manufacturing Systems To Determine Data Product Quality. (No. TDQM-93-09) Cambridge Mass. Total Data Quality Management Research Program, MIT Sloan School of Management, 1993.

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

[CaFM99] Fabio Casati, Mariagrazia Fugini, Isabelle Mirbel. An Environment for Designing Exceptions in Workflows. *Information Systems*, **24**(3), 255-273 (1999).

[CaTo97] L. Cabbibo and R. Torlone. Querying Multidimesional Databases. *6th International Workshop on Database Programming Languages (DBPL6)*, 1997.

[CaTo98] L. Cabbibo and R. Torlone. A Logical Approach to Multidimensional Databases. *Proceedings of the 6th International Conference on Extending Database Technology (EDBT-98)*, Valencia, Spain, 1998.

[CaTo98a] L. Cabibbo, R. Torlone. From a Procedural to a Visual Query Language for OLAP. *Proceedings of 10th International Conference on Scientific and Statistical Database Management (SSDBM)*, Capri, Italy, July 1998.

[CCPP95] F. Casati, S. Ceri, B. Pernici, G. Pozzi. Conceptual Modeling of Workflows. In Proc. Of OO-ER Conference, Australia, 1995.

[CCPP98] Fabio Casati, Stefano Ceri, Barbara Pernici, Giuseppe Pozzi. Workflow Evolution. DKE 24(3), 211-238 (1998).

[CDL*98] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Information integration. Conceptual modeling and reasoning support. *Proceedings of the 6th International Conference on Cooperative Information Systems* (CoopIS-98), pp. 280-291, 1998.

[CDL*99] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, R. Rosati. A principled approach to data integration and reconciliation in data warehousing. In Proc. Intl. Workshop on Design and Management of Data Warehouses (DMDW'99), Heidelberg, Germany, June 1999

[ChDa97] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. SIGMOD Record, Vol. 26, No. 1, March 1997.

[ChSh96] S. Chaudhuri, K. Shim. Optimizing Queries with Aggregate Views. In Proc. of EDBT 1996. *Proceedings of the 5th International Conference on Extending Database Technology (EDBT-96)*, Avignon, France, March 25-29, 1996.

[CKPS95] S. Chaudhuri, S. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. *Proceedings of the 11th International Conference on Data Engineering (ICDE)*, IEEE Computer Society, pp. 190-200, Taipei, March 1995.

[ClNR99] K.T. Claypool, C. Natarajan, E.A. Rundensteiner. Optimizing the Performance of Schema Evolution Sequences. Technical Report WPI-CS-TR-99-06, Worcester Polytechnic Institute, Dept. of Computer Science, March 1999.

[Coll96] G. Colliat. OLAP, Relational, and Multidimensional Database Systems. *SIGMOD Record*, vol. 25, vo.3, September 1996.

[CoNS99] S. Cohen, W. Nutt, A. Serebrenik. Rewriting Aggregate Queries Using Views. *Proceedings of the 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, Philadelphia, Pennsylvania. ACM Press, 1999.

[DaRe99] P. Dadam, M. Reichert (eds.). Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications. GI Workshop Informatik'99, 1999. Available at http://www.informatik.uni-ulm.de/dbis/veranstaltungen/Workshop-Informatik99-Proceedings.pdf

[Day93] R.G. Day. Quality Function Deployment. Linking a Company with Its Customers. ASQC Quality Press, Milwaukee WI, 1993.

[Dayal87] U. Dayal. Of Nests and Trees: A unified approach to processing queries that contain nested subqueries, aggregates and quantifiers. In Proceedings of 13th International Conference on Very Large Data Bases, Brighton, England, September 1-4, 1987.

[Dean97] E.B. Dean. *Quality Functional Deployment from the Perspective of Competitive Advantage*. Available at http://mijuno.larc.nasa.gov/dfc/qfd.html, 1997.

[DeMc92] W.H. Delone, E.R. McLean. Information Systems Success. The quest for the dependent variable. Information Systems Research, vol. 3, no. 1, 1992

[DJLS96] Sh. Dar, H.V. Jagadish, A.Y. Levy, and D. Srivastava. Answering queries with aggregation using view. *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB)*, Bombay (India), September 1996.

[EDD98] Home Page of EDD Data Cleanser tool available at http://hood.npsa.com/edd.

[Ende72] H.B. Enderton, A Mathematical Introduction to Logic. Academic Press, 1972.

[ETI98] Home Page of ETI (Evolutionary Technologies International) available at http://www.evtech.com.

[Garv88] D.A. Garvin. Managing Quality. The Strategic and Competitive Edge. N.Y. Free Press, 1988.

[GaWo87] Richard A. Ganski, Harry K. T. Wong. Optimization of Nested SQL Queries Revisited. In Proc. of SIGMOD Conference 1987

[GBLP96] J. Gray, A. Bosworth, A. Layman, H. Pirahesh. Data Cube. A Relational Aggregation Operator Generalizing Group-By, Cross-Tabs, and Sub-Totals. *Proceedings of the 12th International Conference on Data Engineering (ICDE '96)*, New Orleans, February 1996. Also Microsoft Technical Report, MSR-TR-95-22, available at http://www.research.microsoft.com/~gray.

[GeHS95] D. Georgakopoulos, M. Hornick, A. Sheth. An overview of workflow management. From process modeling to workflow automation infrastructure. *Distributed and parallel databases*, 3,119-153, 1995.

[GeJJ97] M. Gebhardt, M Jarke and S. Jacobs. A toolkit for negotiation support on multi-dimensional data. Proceedings of ACM SIGMOD International Conference on Management of Data. Tucson, Arizona, 1997.

[GeRu97] D. Georgakopoulos, M. Rusinkiewicz. Workflow management: From business process automation to inter-organizational collaboration. *Tutorials of the 23rd International Conference on Very Large Data Bases*, Athens, Greece, 1997.

[GiLa98] F. Gingras, L. Lakshmanan. nD-SQL: A Multi-dimensional Language for Interoperability and OLAP. Proceedings of the 24th VLDB Conference, N. York, August 1998. *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB)*, New York City, New York, Morgan Kaufmann Publishers, August 1998.

[Gree97] L. Greenfield. Data Warehousing Information Center. http://pwp.starnetinc. com/larryg/index.html

[GuHQ95] A. Gupta, V. Harinarayan, and D. Quass. Aggregate query processing in data warehouses. *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB)*, Zurich, Switzerland, Morgan Kaufmann Publishers, August 1995.

[GuMR95] A. Gupta, I. Mumick, K. Ross. Adapting Materialized Views after Redefinitions. In Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995. SIGMOD Record 24(2), June 1995.

[Gupt97] H. Gupta. Selection of Views to Materialize in a Data Warehouse. *Proceedings of International Conference on Database Theory (ICDT)*, Delphi, Greece, 1997.

[GuSW96] S. Guo, W. Sun, M. Weiss. Solving Satisfiability and Implication Problems in Database Systems. ACM Transactions on Database Systems (TODS), **21**(2), June 1996.

[GyLa97] M. Gyssens, L.V.S. Lakshmanan. A Foundation for Multi-Dimensional Databases. *Proceedings of the 23rd International Conference on Very Large Databases (VLDB)*, Athens, August 1997.

[HaCl88] J.R. Hauser, D. Clausing. The House of Quality. The Harvard Business Review, May-June, No. 3, 1988.

[Hall*78] Haloran et al.. Systems development quality control. MIS Quarterly, vol. 2, no.4, 1978.

[HKRW90] Y.U. Huh, F.R. Keller, T.C. Redman and A.R. Watkins. Data Quality. Information and Software Technology, vol. 32, no. 8, 1990.

[Hurw97] Mike Hurwicz. Take Your Data to the Cleaners. BYTE, January, 1997.

[HyRo96] L. Hyatt, L. Rosenberg. A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality. 8th Annual Software Technology Conference, Utah, April, 1996.

[IEEE97] IEEE, Information Technology, http://standards.ieee.org/ catalog/it.html

[Info97] Informix Inc. The INFORMIX-MetaCube Product Suite. http://www.informix.com/informix/products/new_plo/metabro/metabro2.htm, 1997.

[Inmo96] W. H. Inmon. Building the Data Warehouse. John Wiley & Sons, second edition, 1996.

[InVa98] Home Page of Integrity tool available at http://www.vality.com/html/prod-int.html.

[ISO91] ISO/IEC 9126 International Standard. ISO, International Organization for Standardization. 1991

[ISO92] ISO,"ISO9000 International Standards for Quality Management. Geneva, International Organization for Standardization, 1992.

[ISO97] ISO, International Organization for Standardization, http://www.iso.ch

[JaJR90] M. Jarke, M.A. Jeusfeld, T. Rose. A software process data model for knowledge engineering in information systems. *Information Systems 15*, 1 (1990), 85-116.

[Jans88] M. Janson. Data quality: The Achilles heel of end-user computing. Omega J. Management Science, vol. 16, no. 5, 1988.

[JaPo92] M. Jarke, K. Pohl. Information systems quality and quality information systems. Proceedings IFIP 8.2 Working Conference, Minneapolis 1992.

[JaRo88] M. Jarke, T. Rose. Managing knowledge about information system evolution. Proceedings of the ACM SIGMOD International Conference of the Management of Data, pp. 303-311, 1998.

[JaVa97] M. Jarke, Y. Vassiliou. Foundations of data warehouse quality – a review of the DWQ project. In *Proc. 2nd Intl. Conference Information Quality (IQ-97)*, Cambridge, Mass. (1997). Available at http://www.dbnet.ece.ntua.gr/~dwq/

[JeQJ98] M.A. Jeusfeld, C. Quix, M. Jarke. Design and Analysis of Quality Information for Data Warehouses. Proceedings of the 17th International Conference on Conceptual Modeling (ER'98), Singapore, 1998.

[Jeus92] M.A. Jeusfeld. Update Control in Deductive Object Bases. PhD thesis, University of Passau, (in German), 1992.

[JGJ*95] M. Jarke, R. Gallersdörfer, M.A. Jeusfeld, M. Staudt, 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, No. 2, pp. 167-192, 1995.

[JJQV98] M. Jarke, M.A.Jeusfeld, C. Quix, P. Vassiliadis. Architecture and quality in data warehouses. Proceedings of the 10th International Conference CAiSE'98, Pisa, Italy, 1998.

[JJQV99] M. Jarke, M.A.Jeusfeld, C. Quix, P. Vassiliadis. Architecture and quality in data warehouses: An extended repository approach. *Information Systems*, **24**(3), pp. 229-253, 1999. (a previous version appeared in *Proc. 10th Conference of Advanced Information Systems Engineering (CAiSE '98)*, Pisa, Italy, 1998)

[JLVV99] M. Jarke, M. Lenzerini, Y. Vassiliou, P. Vassiliadis (eds.). *Fundamentals of Data Warehouses*. Springer-Verlag, 1999.

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

[Jura88] J. M. Juran, ed.. Quality Control Handbook. N.Y. McGraw-Hill, 1988.

[Jura92] J.M. Juran. Juran on Quality Design: The new steps for Planning Quality into Goods and Services" N.Y., Free Press, 1992.

[Kack86] R. Kackar. Taguchi's Quality Philosophy: Analysis and Commentary. Quality Progress, Dec. 1986,

[Kim82] Won Kim. On Optimizing an SQL-like Nested Query. TODS 7(3), 443-469 (1982)

[Kimb96] Ralph Kimball. The Data Warehouse Toolkit. John Wiley and Sons, 1996.

[Klam99] R. Klamma. Readings in workflow management; annotated and linked internet bibliography, RWTH Aachen, http://sunsite.informatik.rwth-aachen.de/WFBib/

[KLCO96] D. Kuo, M. Lawley, C. Liu, M. Orlowska. A General Model for Nested Transactional Workflows. In Proc. of Intern. Workshop on Advanced Transaction Models and

Architecture, India, 1996.

[Krie79] K. Kriebel. Evaluating the quality of information system. Design and Implementation of Computer Based Information Systems, N. Szysperski and E. Grochla ,eds. Germantown, Sijthoff and Noordhoff, 1979.

[LeAW98] W. Lehner, J. Albrect, H. Wedekind. Normal Forms for Multidimensional Databases. *Proceedings of 10th International Conference on Scientific and Statistical Database Management (SSDBM)*, Capri, Italy, July 1998.

[LeBo98] M. Lenzerini, M. Bouzeghoub. Personal communication.

[Lehn98] W. Lehner. Modeling Large Scale OLAP Scenarios. *Proceedings of the 6th International Conference of Extending Database Technology (EDBT-98)*, 1998.

[LeMS94] A. Levy, I. Mumick, Y. Sagiv. Query Optimization by Predicate Move-Around. In Proceedings of the 20th International Conference on Very Large Data Bases, (VLDB'94), Santiago de Chile, Chile, September 12-15, 1994.

[LeSh97] H. Lenz, A. Shoshani. Summarizability in OLAP and Statistical databases. *Proceedings of 9th International Conference on Scientific and Statistical Database Management (SSDBM)*, Olympia, Washington, 1998.

[LiUp90] G.E. Liepins and V.R.R. Uppuluri. Accuracy and Relevance and the Quality of Data. A.S. Loebl, ed., vol. 112, N.Y, Marcel Dekker, 1990.

[LiWa96] C. Li, X. Sean Wang. A Data Model for Supporting On-Line Analytical Processing. *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, 1996.

[LMSS95] A.Y. Levy, A.O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. *Proceedings of the 14th Symposium on Principles of Database Systems (PODS)*, pp. 95-104, San Jose (California, USA), May 1995. ACM Press.

[LSTV99] S.Ligoudistianos, T.Sellis, D.Theodoratos, and Y.Vassiliou. Heuristic Algorithms for Designing the Data Warehouse with SPJ Views. *Proceedings of the First International Conference on Data Warehousing and Knowledge Discovery, (DaWaK)*, Lecture Notes in Computer Science, Vol. 1676, Springer, 1999.

[MaOr99] O. Marjanovic, M. Orlowska. On Modeling and Verification of Temporal Constraints in Production Workflows. Knowledge and Information Systems 1(2), 1999

[MBJK90] J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis. Telos: a language for representing knowledge about information systems. ACM Transactions on Information Systems, vol. 8, no. 4, 1990.

[McRW78] J.A. McCall, P.K. Richards, and G.F. Walters. Factors in software quality. Technical Report, Rome Air Development Center, 1978.

[MeDC99] MetaData Coalition. Open Information Model, version 1.0, 1999. Available at www.MDCinfo.com

[Meta97] Metadata Coalition. Metadata Interchange Specification (MDIS v.1.1). http://www.metadata.org/standards/toc.html 1997.

[MFPR90] I. Mumick, S. Finkelstein, H. Pirahesh, R. Ramakrishnan. Magic is Relevant.In Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990.

[MiAk94] S. Mizuno, and Y. Akao, ed. "QFD: The Customer-Driven Approach to Quality Planning and Development. Asian Productivity Organization, Tokyo, Japan, available from Quality Resources, One Water Street, White Plains NY, 1994.

[Micr96] Microsoft. Microsoft SQL Server and Data Warehousing. Presentation, http://www.microsoft.com:80/sql/datawarehouse/ dw_pp/index.htm, 1996.

[Micr98] Microsoft Corp. OLEDB for OLAP February 1998. Available at http://www.microsoft.com/data/oledb/olap/

[More82] R.C. Morey. Estimating and improving the quality of information the MIS. Communications of the ACM (CACM), vol. 25, no. 5, 1982.

[MStr95] MicroStrategy, Inc. Relational OLAP: An Enterprise- Wide Data Delivery Architecture. White Paper, http://www.strategy.com/ wp_a_i1.htm, 1995.

[MStr97] MicroStrategy, Inc. MicroStrategyÆs 4.0 Product Line. http://www.strategy.com/launch/4_0_arc1.htm, 1997.

[Mura89] M. Muralikrishna. Optimization and Dataflow Algorithms for Nested Tree Queries. In Proc. 15th VLDB, 1989.

[Mura92] M. Muralikrishna. Improved Unnesting Algorithms for Join Aggregate SQL Queries. In Proc. of 18th VLDB, 1992.

[MyCN92 J. Mylopoulos, L. Chung, B. Nixon. Representing and using non-functional

] requrirements: a process-centered approach. IEEE Transactions on Software Engineering 18, 6 (1992).

[NiJa98] H.W. Nissen, and M. Jarke. Requirements Engineering Repositories: Formal Support for Informal Teamwork Methods. In M. Broy (ed.): Requirements Targeting Software and Systems, Springer LNCS, 1998 (extended version accepted for Information Systems, Special Issue on Meta Modeling and Method Engineering, to appear 1999).

[NuSS98] W. Nutt, Y. Sagiv, S. Shurin. Deciding Equivalences among Aggregate Queries. *Proceedings of the 17th Symposium on the Principles of Databases (PODS'98),* Seattle, June 1998.

[OiBa92] M. Oivo, V. Basili. Representing software engineering models: the TAME goal-oriented approach. *IEEE Transactions on Software Engineering*, **18**(10), 886-898, (1992).

[OLAP97] OLAP Council. OLAP AND OLAP Server Definitions. 1997 Available at http://www.olapcouncil.org/research/glossaryly.htm

[OLAP97a] OLAP Council. The APB-1 Benchmark. 1997. Available at http://www.olapcouncil.org/research/bmarkly.htm

[OrCo96a] Oracle7 Server Utilities User's Guide, SQL*Loader Concepts, available at http://marshal.uwyo.edu/Oracle/DOC/server/doc/ SUT73/ch3.htm#toc058, 1996.

[Orli97] R.J. Orli . Data Extraction, Transformation, and Migration Tools. Kismet Analytic Corp, http://www.kismeta.com/ex2.htm.

[Orr98] K. Orr. Data quality and systems theory. *Communications of the ACM*, **41**(2), 66-71, (1998).

[OzOM85] G. Ozsoyoglu, M. Ozsoyoglu and F. Mata. A Language and a Physical Organization Technique for Summary Tables. *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data*, Austin, Texas, May 1985.

[OzOM87] G. Ozsoyoglu, M. Ozsoyoglu and V. Matos. Extending Relational Algebra and Relational Calculus with Set-Valued Attributes and Aggregation Functions. *ACM Transactions on Database Systems*, vol. 12, no. 4, 1987.

[PaRe90] R.W. Pautke, and T.C. Redman. Techniques to control and improve quality of data large databases. Proceedings of Statistics Canada Symp. 90, Canada, 1990.

[PuIn98] Home Page of CarletonÆs Pure Integrator tool available at http://www.apertus.com/products/Integrate/index.htm.

[QJJ*97] C. Quix, M. Jarke, M.A. Jeusfeld, P. Vassiliadis, M. Lenzerini, D. Calvanese, M. Bouzeghoub. Data Warehouse Architecture and Quality Model. Technical Report D 2.1, DWQ Consortium, 1997.

[QJJ*98] C. Quix, M. Jarke, M. Jeusfeld, M. Bouzeghoub, D. Calvanese, E. Franconi, M. Lenzerini, U. Sattler, P. Vassiliadis. Quality Oriented Data Warehouse Evolution. *Technical Report D9.1, DWQ Consortium* (1998). Available at http://www.dbnet.ece.ntua.gr/~dwq/

[Quix99] 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 (1999). Available at http://www.dbnet.ece.ntua.gr/~dwq/

[RaDh92] B. Ramesh, V. Dhar. Supporting systems development by capturing deliberations during requirements engineering. *IEEE Trans. Software Engineering 18*, 6 (1992).

[RaRi91] M. Rafanelli, and F.L. Ricci. A functional model for macro-databases. *SIGMOD Record*, vol. 20, no. 1, March 1991.

[RBSI97] Red Brick Systems, Inc.. *Red Brick Warehouse 5.0*. http://www.redbrick.com/rbs-g/html/whouse50.html, 1997.

[Redm92] T.C. Redman. Data Quality: Management and Technology. N.Y. Bantam Books, 1992.

[Roll98] C. Rolland. A comprehensive view of process engineering. *Proc. 10th Intl. Conf. Advanced Information Systems Engineering*, (CAiSE'98), Pisa, Italy, 1-25.

[Sach97] S. Sachdeva. Metadata for Data Warehouse. White paper, Sybase, Inc. http://www.sybase.com/services/dwpractice/meta.html, 1997.

[SaOr99] W. Sadiq, M. Orlowska. Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models. In Proc. of 11th Int. Conference CAiSE'99, Heidelberg, Germany, June 1999.

[Sara97] Sunita Sarawagi. Indexing OLAP Data. Data Engineering Bulletin, **20**(1), 36-43 (1997).

[Schn90] N. Schneidewind. Standard for a Software Quality Metrics Methodology. Soft. Eng. Standards Subcommittee of the IEEE, 1990.

[SDJL96] Divesh Srivastava, Shaul Dar, H. V. Jagadish, Alon Y. Levy. Answering Queries with Aggregation Using Views. In Proc. of 22nd VLDB Conference, 1996.

[Shos97] A. Shoshani. OLAP and Statistical Databases: Similarities and Differences. *Tutorials of 16th Symposium on Principles Of Database Systems (PODS), 1997*.

[STGI96] Stanford Technology Group, Inc. Designing the Data Warehouse on Relational Databases. http://www.informix.com/informix/corpinfo/zines/whitpprs/stg/metacube.htm, 1996.

[StLW94] D.M. Strong, Y.W. Lee, and R.Y. Wang. Beyond Accuracy: How Organizations are Redefining Data Quality" (No. TDQM-94-07), Cambridge Mass. Total Data Quality Management Research Program, MIT Sloan School of Management, 1994.

[StQJ98] M. Staudt, C. Quix, M.A. Jeusfeld. View Maintenance and Change Notification for Application Program Views. In *Proc. ACM Symposium on Applied Computing*, (1998).

[Svan84] M.I. Svanks. Integrity Analysis: Methods for automating data quality assurance. EDP Auditors Foundation, vol. 30, no. 10, 1984.

[TaBa98] G.K. Tayi, D. P. Ballou. Examining Data Quality. In Communications of the ACM, **41**(2), 54-57 (1998).

[ThBo99] D. Theodoratos, 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 (1999). Available at http://www.dbnet.ece.ntua.gr/~dwq/

[ThLS99] D.Theodoratos, S.Ligoudistianos, and T.Sellis. Designing the Global DW with SPJ Queries. *Proceedings of the 11th Conference on Advanced Information Systems Engineering (CAiSE)*, pages 180--194, June 1999.

[ThSe97] D. Theodoratos, 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).

[ThSe99] D. Theodoratos, T. Sellis. Designing Data Warehouses. *Data and Knowledge Engineering (DKE)*, **31**(3), 279-301(1999). Available at http://www.dbnet.ece.ntua.gr/~dwq/

[TPC98] TPC. TPC Benchmark D. Transaction Processing Council. February 1998. Available at http://www.tpc.org/dspec.html

[TPC99] TPC. TPC Benchmark H and TPC Benchmark R. Transaction Processing Council. June 1999. Available at http://www.tpc.org/

[Ullm89] J. Ullman. Principles of Database and Knowledge-Base Systems. Volume II: The New Technologies. Computer Science Press. 1989.

[Ullm97] J.D. Ullman. Information integration using logical views. *Proceedings of the 6th International Conference on Database Theory (ICDT-97)*, Lecture Notes in Computer Science, pp. 19-40. Springer-Verlag, 1997.

[VaBQ99] P. Vassiliadis, M. Bouzeghoub, C. Quix. Towards Quality-Oriented Data Warehouse Usage and Evolution. In *Proc. 11th Conference of Advanced Information Systems Engineering (CAiSE '99)*, Heidelberg, Germany, 1999.

[VaSe99] P. Vassiliadis, T. Sellis. "A Survey on Logical Models for OLAP Databases". *SIGMOD Record*, vol. 28, no. 4, December 1999.

[VaSk00] P. Vassiliadis, S. Skiadopoulos. Modelling and Optimization Issues for Multidimensional Databases. In Proc. 12th Conference on Advanced Information Systems Engineering (CAiSE '00), Stockholm, Sweden, 5-9 June 2000.

[VaSk99] P. Vassiliadis, S. Skiadopoulos. (Not) Yet Another Model for Multidimensional Databases, Extended version, Technical Report, KDBSL 1999. Available at http://www.dblab.ece.ntua.gr/~pvassil/publications/cube99.ps.gz

[VaSL97] Y. Vassiliou, T. Sellis, S. Ligoudistianos. Data Warehouse Quality Requirements and Framework. *Technical Report D1.1, DWQ Consortium* (1997). Available at http://www.dbnet.ece.ntua.gr/~dwq/

[Vass98] P. Vassiliadis. Modeling Multidimensional Databases, Cubes and Cube Operations. *Proceedings of 10th International Conference on Scientific and Statistical Database Management (SSDBM)*, Capri, Italy, July 1998.

[Vass98a] P. Vassiliadis. *Formal Foundations for Multidimensional Databases"* (extended version of [Vass98]) - NTUA Technical Report, January 1998. Available at http://www.dblab.ece.ntua.gr/~pvassil/publications/ssdbm_tr.ps.gz

[VQVJ00] P. Vassiliadis, C. Quix, Y. Vassiliou, M. Jarke. A Model for Data Warehouse Operational Processes. In Proc. 12th Conference on Advanced Information Systems Engineering (CAiSE '00), Stockholm, Sweden, 5-9 June 2000.

[WaKM93 R.Y. Wang, H.B. Kon, and S.E. Madnick. Data quality requirements analysis and

]	modeling. Proceedings 9th International Conference on Data Engineering (ICDE), Vienna, 1993
[WaKM93]	R.Y. Wang, H.B. Kon, 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.
[WaKo93]	R.Y. Wang, and H.B. Kon. Towards total data quality management. Information Technology Action: Trends and Perspectives, R.Y. Wang, ed. Englewood Cliffs, N.J., Prentice Hall, 1993
[WaMa90]	R.Y. Wang, and S.E. Madnick. A Polygen model for heterogeneous database systems: The source tagging perspective. Proceedings of the 16th International Conference on Very Large Data Bases (VLDB), Brisbane, Australia, 1990
[Wand89]	Y. Wand. A proposal for a formal model of objects. Object-Oriented Concepts, Databases, and Applications, W. Kim and F. Lochovsky, eds. N.Y. ACM Press, 1989
[Wang98]	R.Y. Wang. A product perspective on total data quality management. Communications of the ACM (CACM), vol. 41, no. 2, February 1998.
[WaRG93]	R.Y. Wang, M.P. Reddy, and A. Gupta. An object-oriented implementation of quality data products. Proceedings Third Ann. Workshop Information Technologies and Systems, 1993.
[WaRK95]	R.Y. Wang, M.P. Reddy, and H.B. Kon. Towards Quality Data: An attribute-based Approach. Decision Support Systems, vol. 13, 1995.
[WaSF95]	R.Y. Wang, V.C. Storey, C.P. Firth. A Framework for Analysis of Data Quality Research. IEEE Transactions on Knowledge and Data Engineering, Vol. 7, No. 4, August 1995.
[WaSG94]	R.Y. Wang, D. Strong, 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).
[WaWa96]	Y. Wand, and R.Y. Wang. Anchoring data quality dimensions ontological foundations. Communications of the ACM (CACM), vol. 39, no. 11, November 1996.
[WaWe90]	Y. Wand, and R. Weber. An ontological model for an information system. IEEE Transactions on Software Engineering, vol. 16, no. 11, 1990.
[WfMC98]	Workflow Management Coalition. Interface 1: Process Definition Interchange Process Model. Document number WfMC TC-1016-P, 1998. Available at www.wfmc.org
[Wido95]	J. Widom. Research Problems in Data Warehousing. Proceedings of the 4th International Conference on Information and Knowledge Management (CIKM), November 1995.
[YaLa85]	P. Larson, H. Z. Yang. Computing Queries from Derived Relations. In Proc. of VLDB 1985. *Proceedings of the 11th International Conference on Very Large Data Bases (VLDB),* Stockholm, Sweden, Morgan Kaufmann Publishers, August 1985.
[Yu99]	E. Yu. Strategic Modelling for Enterprise Integration. In *Proc. 14th World Congress of Int. Federation of Automatic Control*, China 1999.
[YuMy94]	E. Yu, J. Mylopoulos. Understanding 'Why' in Software Process Modelling, Analysis and Design. In *Proc. 16th Int. Conference Software Engineering*, May 1994.

Appendix A- ISO Standards for Information Quality

The International Organization for Standardization (ISO) was founded in Geneva, Switzerland, in 1946. Its goal is to promote the development of international standards to facilitate the exchange of goods and services worldwide.

The original ISO 9000 [ISO92], [ISO97] standards were a series of international standards (ISO 9000, ISO 9001, ISO 9002, ISO 9003, ISO 9004), developed by ISO Technical Committee 176 (TC176) to provide guidance on the selection of an appropriate quality management program (system) for a supplier's operations. The series of standards serves the purpose of common terminology definition and demonstration of a supplier's capability of controlling its processes. The content of the 1994 edition of the ISO 9000 series is described in the following paragraphs.

ISO 9000 -1, Quality Management and Quality Assurance Standards - Guidelines for Selection and Use. This standard explains fundamental quality concepts, defines key terms and provides guidance on selecting, using and tailoring series. Furthermore, it helps in the selection and use of the standards in the ISO 9000 family.

ISO 9001-1, Quality Systems - Model for Quality Assurance in Design/ Development, Production, Installation and Servicing. This is the most comprehensive standard. It addresses all elements including design. The 1994 edition improved the consistency of the terminology and clarified or expanded the meaning of some of the clauses. Several new requirements, such as that for quality planning, were added. The standard contains 20 elements describing the quality parameters, from the receipt of a contract through the design/delivery stage, until the service required after delivery.

ISO 9002, Quality Systems - Model for Quality Assurance in Production and Installation and Servicing. Identical to ISO 9001 except for design requirements. Consequently, it addresses organizations not involved in the design process.

ISO 9003, Quality Systems - Model for Quality Assurance in Final Inspection and Test. This is the least comprehensive standard. It addresses the detection and control of problems during final inspection and testing. Thus, it is not a quality control system. The 1994 edition added additional requirements including: contract review, control of customer supplied product, corrective actions, and internal quality audits.

ISO 9004 -1, Quality Management and Quality System Elements - Guidelines. This standard provides guidance in developing and implementing an internal quality system and in determining the extent to which each quality system element is applicable. The guidance in ISO 9004-1 exceeds the requirements contained in ISO 9001, ISO 9002 and ISO 9003. ISO 9004-1 is intended to assist a supplier in improving internal quality management procedures and practices. Yet, it is not intended for use in contractual, regulatory or registration applications.

Out of them, there is just one, *"**ISO/DIS 9000-3** Quality management and quality assurance standards – Part 3: Guidelines for the application of ISO 9001:1994 to the development, supply, installation and maintenance of computer software (Revision of ISO 9000-3:1991)"* specifically intended for use in the computer software industry. Furthermore, there are several standards developed from ISO, concerned with the achievement of quality in the development and evaluation of software. Yet, these standards are not directly concerned with ISO 9000.
The interested reader can find a lot of other standards developed from ISO and IEEE in the field of software quality. A list of them is following. Note that standards are constantly being added and revised, so this list can quickly become out of date.

IEEE Standards on Information Technology [IEEE 97]

730-1989 IEEE Standard for Software Quality Assurance Plans (ANSI).
1061-1992 IEEE Standard for a Software Quality Metrics Methodology .
730.1-1995 IEEE Standard for Software Quality Assurance Plans. (Revision and redesignation of IEEE Std 938-1986).
1074-1995 IEEE Standard for Developing Software Life Cycle Processes; together with
1074.1-1995 IEEE Guide for Developing Software Life Cycle Processes.

ISO REFERENCES [ISO 97]

ISO/DIS 9000-3 Quality management and quality assurance standards – Part 3: Guidelines for the application of ISO 9001:1994 to the development, supply, installation and maintenance of computer software.
ISO/IEC 12119:1994 Information technology – Software packages – Quality requirements and testing.
ISO/IEC 9126:1991 Information technology – Software product evaluation – Quality characteristics and guidelines for their use.
ISO/IEC DIS 13236 Information technology – Quality of service – Framework.
ISO/IEC DTR 15504-2 Software Process Assessment – Part 2: A reference model for processes and process capability (normative).
ISO/IEC DTR 15504-3 Software Process Assessment – Part 3: Performing an assessment (normative).

ISO 9000 family

ISO 9000-1: 1994 Quality management and quality assurance standards-Part 1: Guidelines for selection and use.
ISO 9000-2: 1993 Quality management and quality assurance standards-Part 2: Generic guidelines for the application of ISO 9001, ISO 9002 and ISO 9003.
ISO/FDIS 9000-2 Quality management and quality assurance standards – Part 2: Generic guidelines for the application of ISO 9001, ISO 9002 and ISO 9003 (Revision of ISO 9000-2:1993).
ISO 9000-3: 1991 Quality management and quality assurance standards-Part 3: Guidelines for the application of ISO 9001 to the development, supply and maintenance of software.
ISO/DIS 9000-3 Quality management and quality assurance standards – Part 3: Guidelines for the application of ISO 9001:1994 to the development, supply, installation and maintenance of computer software (Revision of ISO 9000-3:1991).
ISO 9000-4: 1993 Quality management and quality assurance standards – Part 4: Guide to dependability program management.
ISO 9001: 1994 Quality system-model for quality assurance in design, development, production, installation and servicing.
ISO 9002: 1994 Quality system-model for quality assurance in production, installation and servicing.
ISO 9003: 1993 Quality Systems-Model for quality assurance in final inspection and test.
ISO 9004-1: 1994 Quality management and quality system elements – Part 1: Guidelines.
ISO 9004-2: 1991 Quality management and quality system elements – Part 2: Guidelines for services.
ISO 9004-3: 1993 Quality management and quality system elements – Part 3: Guidelines for processed materials.
ISO 9004-4: 1993 Quality management and quality system elements – Part 4: Guidelines for quality improvement.
ISO 10005: 1995 Quality management – Guidelines for quality plans (formerly ISO/DIS 9004-5).

ISO/FDIS 10006 Quality management – Guidelines to quality in project management (Formerly CD 9004-6).
ISO 10007: 1995 Quality management – Guidelines for configuration management.
ISO 10011-1: 1990 Guidelines for auditing quality systems. Part 1: Auditing.
ISO 10011-2: 1991 Guidelines for auditing quality systems. Part 2: Qualification criteria for quality systems auditors.
ISO 10011-3: 1991 Guidelines of auditing quality systems. Part 3: Management of audit programs.
ISO 10012-1: 1992 Quality assurance requirements for measuring equipment-Part 1: Metrological confirmation system for measuring equipment.
ISO 10013 Guidelines for developing quality manuals.
ISO/TR 13425 Guidelines for the selection of statistical methods in standardization and specification.
ISO 8402: 1994 Quality management and quality assurance-Vocabulary.