Document Number:	DWQ -- RWTH -- 002
Project Name:	Foundations of Data Warehouse Quality (DWQ)
Project Number:	EP 22469
Title:	**Data Warehouse Architecture and Quality Model**
Author:	Christoph Quix, Matthias Jarke, Manfred A. Jeusfeld, Panos Vassiliadis, Maurizio Lenzerini, Diego Calvanese, Mokrane Bouzeghoub
Workpackage:	WP 2
Document Type:	Deliverable
Classification:	Public
Distribution:	All
Status:	Final
Document file:	del21-final.doc
Version:	1.0
Date:	26 October, 1997
Number of pages:	50

Data Warehouse Architecture and Quality Model

Abstract

This report provides a mid-term overview of the DWQ work done on data warehouse architectures and quality modeling. Based on a critical analysis of present architectural frameworks for data warehouses from the viewpoint of business analysis, we develop a new architectural framework which can be linked to the full range of data quality questions addressed in literature and practice. Using the Telos modeling language and its implementation in ConceptBase, we represent this architecture and quality model as a repository schema in which particular data warehouse tools and applications can be modeled and analyzed for quality. An example illustrates the usefulness of the approach.

1 Introduction

A data warehouse is a collection of technologies aimed at enabling the knowledge worker (executive, manager, analyst) to make better and faster decisions. It is expected to present the right information in the right place at the right time with the right cost in order to support the right decision. The practice proved that the traditional *Online Transaction Processing* (OLTP) systems are not appropriate for decision support and the high speed networks can not by themselves solve the information accessibility problem.

The data warehouse movement is 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, *On-Line Analytical Processing*) cannot coexist efficiently in the same database environment, for two main reasons, both having to do with trade-offs in data quality:

- **data characteristics**: OLTP databases maintain *current data* in great detail locally to their immediate operational usage, whereas OLAP deals with lightly aggregated and often globally reconciled *historical data*, covering much more than just the current ones; mixing both causes complex compromises between different degrees of detail, and varying needs for history information.
- **transaction characteristics**: OLTP emphasizes efficiency of *short update* transactions each covering a small part of the database, whereas OLAP requires *long queries* each surveying a large part of the database; mixing the two causes concurrency control problems.

A data warehouse therefore caches selected data of interest to a customer group, so that access becomes faster, cheaper and more effective. As the long-term buffer between OLTP and OLAP, data warehouses have been said to face two essential questions:

- How to reconcile the stream of incoming data from multiple heterogeneous legacy sources? and
- How to customize the derived data storage to specific OLAP applications?

The trade-offs driving the design decisions concerning these two issues change continuously with business needs, therefore design support and change management are of greatest importance if we do not want to run data warehouse projects into dead ends. This is a recognized problem in industry.

Furthermore, in DWQ, we are interested in modeling quality aspects of data warehouses. The practice has shown that neither the accumulation nor the storage process of the information seem to be completely credible. Errors in databases have been reported to be up to ten percent range and even higher in a variety of applications [WRK95]. 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 undertakes the task of implementing it, it is at least accepted that there can be an equilibrium in the cost-quality tradeoff. [WSF95] report that more than $2 billion of U.S. federal loan money had been lost because of poor data quality at a single agency; manufacturing companies spent over 25% of their sales on wasteful practices. The number came up to 40% for service companies.

Our central claim in this paper is that the traditional DW architectures are insufficient to even talk about these problems because they do not contain any explicit business knowledge. They therefore capture only part of the problem and necessarily only part of the solution. Besides elaborating this critique, we present a first version of the DWQ architecture which makes these issues explicit by a multi-perspective approach. This richer architecture model also allows us to express a much broader range of quality goals, and to link them to the architecture via methods for quality evaluation and (in the second phase of the DWQ project) quality-oriented design and evolution. The integrated architecture and quality model is captured formally as a meta database schema, in the Telos meta modeling language, and prototypically implemented in the ConceptBase system.

Work status and paper overview: This report has been written in the middle of the work package "DW Architecture and Quality Model". Section 2 presents the basic idea and structure of the model, followed by a brief discussion of related work in section 3 (more can be found in the deliverable of WP 1). The bulk of the report is made up by a detailed elaboration and formalization of the architecture (sec. 4) and quality model (sec. 5) that represent the schema of the meta database. Section 5 provides a fairly detailed example, whereas section 6 summarizes the defined Telos classes. .

In the second half of the work package, several other steps have to be done. In particular, the query classes described in this report offer only a shallow analysis which assumes the existence of specific reasoning or measurement engines for individual quality dimensions such as performance, accuracy, completeness, and the like. Such tools are under development in other workpackages of the project and need to be integrated formally and technically with the basic schema reported here. Moreover, the further development of the prototype and its experimental application to DW vendor products such as SourcePoint and DW application projects will undoubtedly lead to some revisions of this work. Despite this disclaimer, we believe that the new architecture is a major step forward in providing a framework that improves the semantic foundations of data warehouse quality in a basic and yet industrially highly relevant manner.

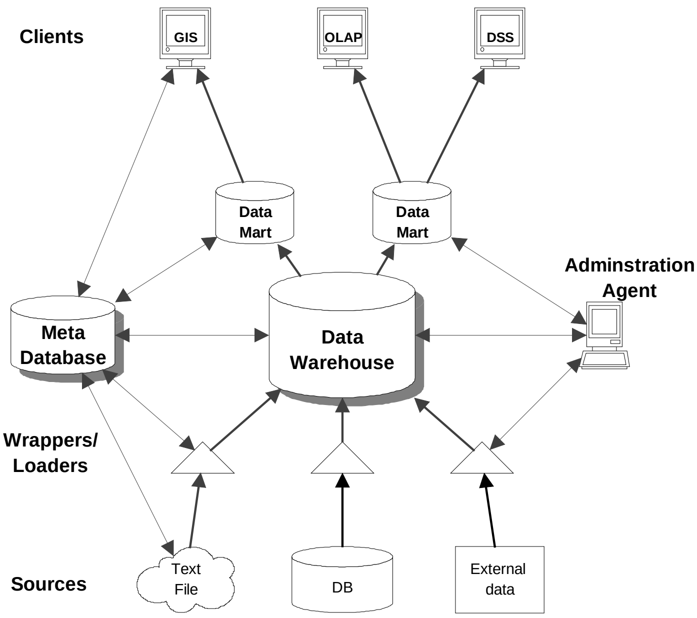

Figure 1 Current Understanding of a Data Warehouse

2 Rationale of the DWQ Architecture and Quality Model

The DWQ project will provide a neutral architectural reference model covering the design, the setting-up, the operation, the maintenance, and the evolution of data warehouses. In this section, we shall first review the presently popular standard view of what a DW is, and point out the shortcomings of this view (2.1). We then propose an extended architecture (2.2) and demonstrate how it can be linked to an explicit quality model (2.3). Finally, we discuss how all of this can be managed in an extended repository system (2.4).

2.1 Shortcomings of Current DW Architectures

As mentioned in the introduction and illustrated in Figure 1, almost all current research and practice understand a data warehouse architecture as a stepwise information flow from information sources through materialized views towards analyst clients. The basic components and relationships in this figure can be briefly explained as follows:

- *sources*: any data store whose content is subject to be materialized in a data warehouse,
- *wrappers* to load the source data into the warehouse,
- *destination databases*: data warehouses and data marts,
- *meta database*: repository for information about other components, e.g. the schema of the source data,
- *agents* for administration (data warehouse design, scheduler for initiating updates, etc.) and
- *clients* to display the data, for example statistics packages.

The detailed elaboration of these components and relationships can be filled in according to many different user requirements and modeling techniques.

However, the main point we want to make here is that the above architecture only covers half of the tasks faced in data warehousing and is therefore unable to even express, let alone support, a large number of important quality problems and management strategies.

Figure 2 clarifies this point by adding an explicit consideration of the business perspective to the picture of figure 1.

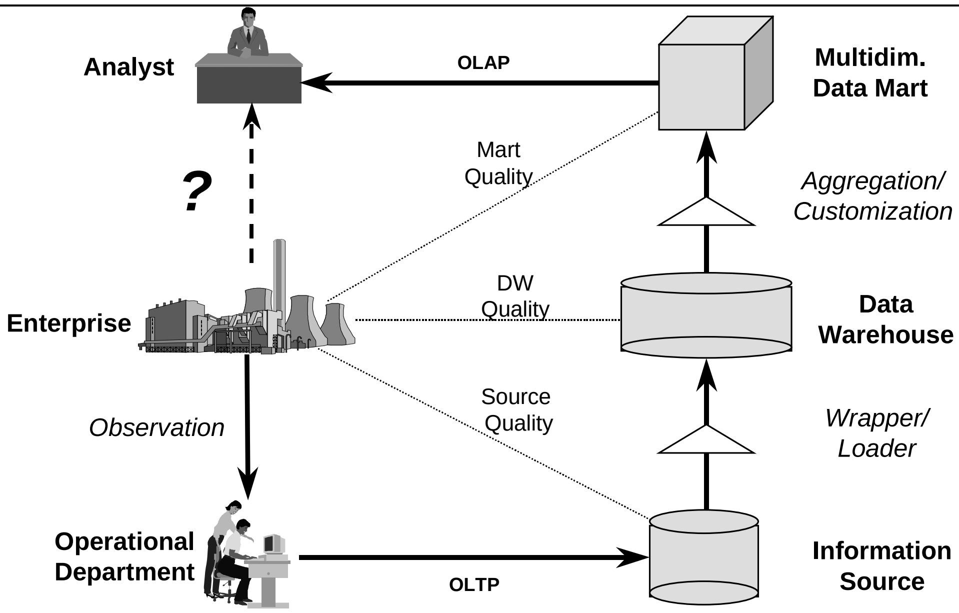

Figure 2 Business Perspective of a Data Warehouse

Suppose an analyst wants to know something about the business but does not have the time to observe the business directly but rather wants to rely on existing information. In the figure, this is illustrated by the broken line with a question mark, leading from the analyst to the factory icon. The figure also shows that five steps have to be made to answer this question via a data warehousing approach; these are indicated by the bold arrows.

Firstly, operational departments have made certain observations on the business which, through OLTP systems, they secondly, may or may not enter into operational information sources; analogous considerations can be made for external information sources concerning observations of the enterprise environment which we do not discuss here for simplicity.

Only now, the information flows of figure 1 come into play. The third step is the extraction, cleaning, and reconciliation of certain data from the information sources to the data warehouse, and the fourth step is the selective customization and aggregation of data warehouse data into the analyst's local data mart.

However, it is also important to consider the last step, back from the technical world of stored data to the conceptual world of the analyst: through OLAP interfaces, the analyst has to interpret the answer he gets from the data mart with respect to the initial analysis problem on the business.

Returning to the question of DW quality, it is clear that things can go wrong at each of these five steps, and that certain problems that are unexplainable in figure 1 now have obvious explanations. Since quality is a subjective factor ("beauty is in the eyes of the beholder"), it can be defined from the viewpoints of any of the stakeholders shown in the left part of figure 2: the individual analyst, the operational departments, or the management of the enterprise. In our opinion, the strategic aim of data warehousing is to impose an enterprise perspective on the analysts. Therefore, the dotted lines in figure 2 indicate the quality of information sources, the data warehouse itself, and the various data marts from the viewpoint of the enterprise.

To illustrate the power of this approach, we give just two examples of quality problems that can be uncovered by an architecture and quality model based on it:

1. Information sources are captured in different operational departments which may have different perspectives from the enterprise model on which the schema of the DW may be constructed. As a consequence, we face the paradox that we can have bad data quality in the DW despite the fact that each information source has perfect data quality individually – a problem that has baffled consultants in many DW projects.

2. From the viewpoint of the analyst, there is a trade-off between two sub-dimensions of data accessability. In terms of query performance, it is best to stay in the data mart and on the short range at best permissible to go back to the DW. Accessing other information about the enterprise may require DW re-design (uploading new views into the DW with all the problems that entails), or even setting up new sensors in the operational departments. In terms of information precision and timeliness, the goal hierarchy is exactly the opposite – perfect and current information is best obtained close to the business operation. However, if the enterprise view is made explicit, our approach allows at least to ask which information could be available and to evaluate the trade-off.

Some further examples are given below when we discuss how the DWQ architecture and quality model actually provide this enterprise information.

2.2 Overview of the DWQ Architecture

In addition to the lack of conceptual enterprise information, architectures such as the one shown in figure 1 tend to mix two concerns, namely the ones of logical and physical data flow. As a consequence, the DWQ architecture considers not just two, but actually three perspectives which are, of course, closely interrelated. We call these the conceptual (or business) perspective, the logical (or data modeling) perspective, and the physical (or distributed information flow) perspective. Any given data warehouse component or substructure can be analyzed from all three perspectives. Moreover, in the design, operation, and especially evolution of data warehouses it is crucial that these three perspectives are maintained consistent with each other. Finally, quality factors are often associated with specific perspectives, or with specific relationships between perspectives.

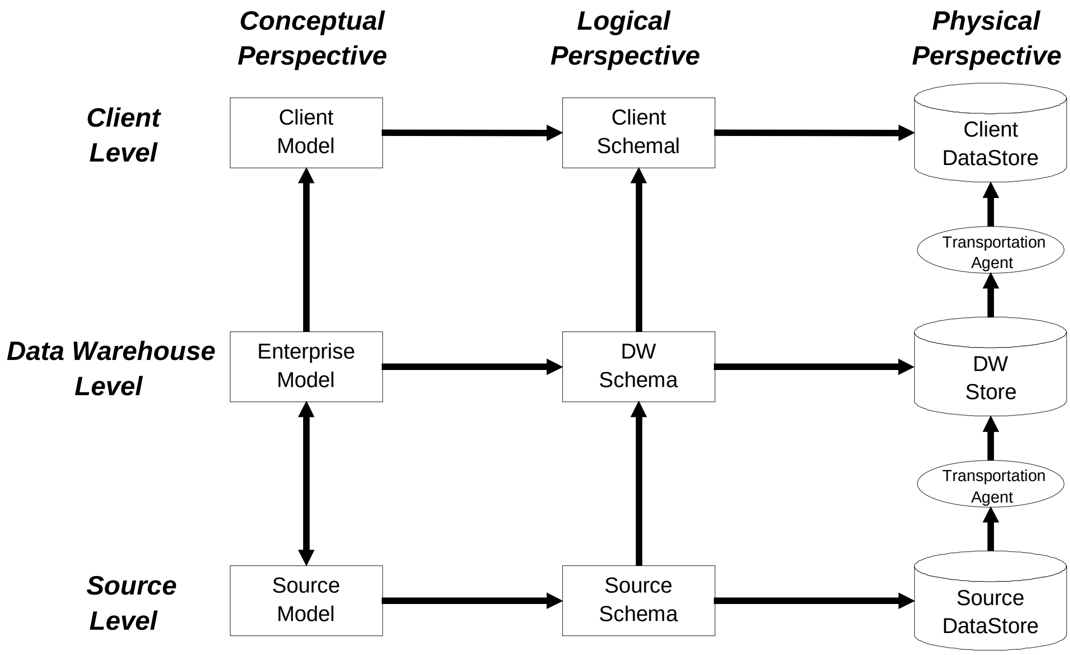

Figure 3 The DWQ Framework: Three Aspects of Data Warehouse Architecture

In figure 3, we find correspondingly that there are 9 component types, linked by at least 12 kinds of relationships maintained by what we call data warehouse agents in our meta model. We discuss each perspective in turn. Note that the discussion is purely at the schema level. Each data warehouse component is an instance of all three perspectives.

The Conceptual Perspective

For strategic management, a data warehouse can serve the purpose to impose an overall business perspective – a conceptual enterprise model – on the information resources and analysis tasks of an organization. If this enterprise model is understood as a class structure, its instance is the organization the data warehouse maintains information about, and thus not part of the data warehouse itself. Instead, the data warehouse architecture is a network of direct and derived views (i.e., recorded observations) on this reality. The schema of information sources is defined as a collection of pre-materialized conceptual views on the enterprise model. This important observation has already been made in the Information Manifold project at AT&T [LMSS95]. Furthermore, the concepts in the enterprise model can be defined in terms of the concepts of information sources, e.g. stating that a concept in the enterprise model is completely defined by the contents of several information sources.

On the client side, the interests of user groups (such as analysts) can also be described as conceptual views on the enterprise models. However, these client views cannot be satisfied by direct observation of reality; instead, they must rely on existing materialized views, i.e. the information sources. As a consequence, we have the problem of mapping conceptual client views on conceptual source views – a theorem proving problem, where the proofs correspond to the queries to be asked by client views to information sources.

Furthermore, given the quality factors of performance and security, client views are often not allowed to access the sources directly. A set of intermediate conceptual views on the enterprise model is therefore often constructed – the conceptual model of the data warehouse. This set must have the formal property that it allows all expected client views to get answers to their queries, preferably the same they would get if they would access the source views directly.

In terms of data warehouse quality, the enterprise model defines a theory of the organization. Actual observations (i.e. information sources) can be evaluated for quality factors such as accuracy, timeliness, completeness with respect to this theory. Moreover, the fact that data warehouse views intervene between client views and source views can have both a positive and a negative impact on information quality. Positive impact is achieved by using the enterprise model for data cleaning and evidence integration – probably the single most important contributions of the conceptual level in data warehouse practice. Potential negative impact to be taken care of is given by delays in updating the materialized data warehouse views (timeliness of information usually deteriorates in comparison with OLAP data), and by limitations of expressiveness in the conceptual model which may, for reasons of decidability in reasoning and simplicity of usage, not include all linguistic facilities of all sources.

The Logical Perspective

The logical perspective conceives a data warehouse from the viewpoint of the actual data models involved. Researchers and practitioners following this perspective are the ones that consider a data warehouse simply a collection of materialized views on top of each other, based on existing information sources. Examples of projects focusing on the logical perspective include TSIMMIS project at Stanford University [CGMH+94] [Ull97], Squirrel at the University of Colorado [ZHK96] and the WHIPS project at Stanford University [HGMW+95, WGL+96].

On the source side, the main problems addressed in this perspective include the wrapping of heterogeneously represented data sources, and the integration of data from multiple sources into particular views. These problems are interpreted both in terms of initial query processing (also called bulk loading in this context) and in terms of incremental maintenance. Particular attention has been paid to so-called self-maintainable warehouse views [QGMW96] where the data warehouse itself contains sufficient information to maintain all its views, given just the changes in the base data (but no additional back requests to the OLTP system to see how these changes propagate into the views).

On the client side, the two questions of view-based query optimization and incremental update propagation remain central but the focus shifts because user data models are different from the source data models. In particular, the introduction of multi-dimensional data models with aggregation and summarization have a strong impact on algorithms and materialization strategies [DJLS96, HRU96, LM96, SDJL96].

Data warehouse products differ in where they introduce these complexities. While *MOLAP* (Multidimensional OLAP) products already keep the data warehouse views themselves as multi-dimensional data structures, *ROLAP* (Relational OLAP) products use relational databases to store the data warehouse views and make the transformation to multi-dimensional data models only in the mapping to client views.

Quality factors impacted by the logical perspective concern mainly the coverage of data models at the source and client side, and thus the factor of accessibility of information as well as the interpretability on the client side. In addition, logic-level optimization of queries as well as updates have a strong impact on the amount of data transported and thus on the performance of the system. For example, a good incremental view maintenance algorithm can reduce the daily data transfer from giga bytes to small numbers of mega bytes in many large organizations, thus also reducing the time window in which OLTP systems are unavailable for transaction processing due to data warehouse loading.

The Physical Perspective

The physical perspective interprets the data warehouse architecture as a network of data stores, data transformers, and communication channels, aiming at the quality factors of reliability and performance in the presence of very large amounts of slowly changing data.

On the source side, a major topic is to minimize interference with OLTP systems while maintaining a reasonable degree of actuality in the data. Typical research questions therefore include replication policies, recovery from broken bulk transfers, and decisions to what components to allocate certain functions. For example, when reconciling data differences, this could be done in the data warehouse itself, or it could be outsourced to mediator agents with local caches. The SourcePoint tool marketed by Software AG [SAG96] is an example of a commercial loading environment that already allows a lot of customization at the physical level, but could benefit from a more declarative approach in linking to the logical and conceptual perspectives.

On the client side, a central question is how to configure a network of pre-materialized views as to minimize average response times for a given mix of updates and queries. As shown in figure 1, this can involve the usage of tailored data marts as well as a central data warehouse. It can also involve client-side caching of intermediate results to minimize data transfer.

Relationships between the Perspectives

Generally speaking, the physical perspective has been mostly explored by industry, whereas the conceptual perspective is mainly approached by research. Research and practice meet in the logical level, but with different emphasis, depending on where they come from. Given the quest for quality in data warehousing, these streams of interest are bound to merge.

The steps from conceptual to logical and then to physical design are common to all database design tasks, however, they have not yet been thoroughly investigated for the special case of data warehousing. Basically, when going from conceptual to logical, we face two problems: (a) the move from a rich conceptual model to an impoverished data model such as the relational one; (b) the fragmentation of a coherent conceptual schema into individual logical data structures. In DWQ, we are exploring the hypothesis that the link between logical and conceptual can be defined in a very similar manner as the link between conceptual source views and the conceptual data warehouse schema, namely through special kinds of views on the conceptual schema. Therefore, the link between the logical sources and logical data warehouse schema can be derived from the links between the logical and conceptual layer and the link from the conceptual source views to the enterprise model.

Going from logical to physical requires the allocation of the identified data fragments and transformation tasks to a network of distributed, cooperating software agents. In this step, considerations of reliability and performance must be added, while semantic considerations must be indirectly expressed through mechanisms that enforce them. When studying the specific tasks that need to be addressed in adding quality to data warehousing, techniques from all three perspectives are often needed.

Current solutions for data warehouse architecture mainly concentrate on partial aspects, such as source integration or modeling of OLAP applications. The goal of the data warehouse architecture model is to have a commonly agreed architectural framework for data warehouse applications. A data warehouse application consists of many components, such as databases, wrappers, client applications etc. The relationships between the components are most times data flows: a component delivers data to another component. Metadata is also transferred between components. Another type of relationship is controlling of other components. For example, a data warehouse administration tool controls other components and invokes them at a certain time. Furthermore, the components report back information about their current status and notify the administration tool about errors or exceptions occurred during their operation.

In DWQ, we use a meta database to store an abstract representation of the data warehouse application in terms of the three-perspective scheme discussed in the previous subsection. The meta database is used as a repository for meta information of the data warehouse components. To fill the repository with information, it is necessary to extract meta data from the components. This is done either automatically by accessing directly the meta information of a component or done manually with the assistance of a data warehouse designer. It is very important, that the meta data abstract from physical details of the components which are not necessary to represent a data warehouse architecture.

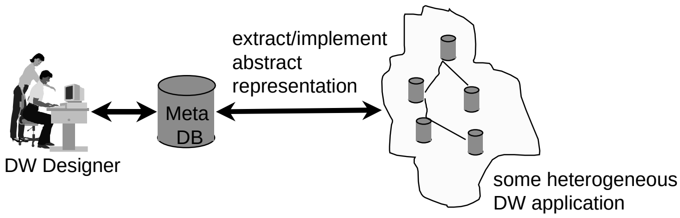

Figure 4 Data Warehouse Design Scenario

Figure 4 shows a scenario in a data warehousing design process. The data warehouse designer develops the data warehouse application conceptually with the meta database before it is implemented physically. From a practical viewpoint, the relationships within and between perspectives will be captured in our approach according to process models of DW design and operation, as indicated in figure 5. Principles underlying the representation and management of such process models on top of conceptual product models have been studied in depth in projects such as NATURE [JPD+94].

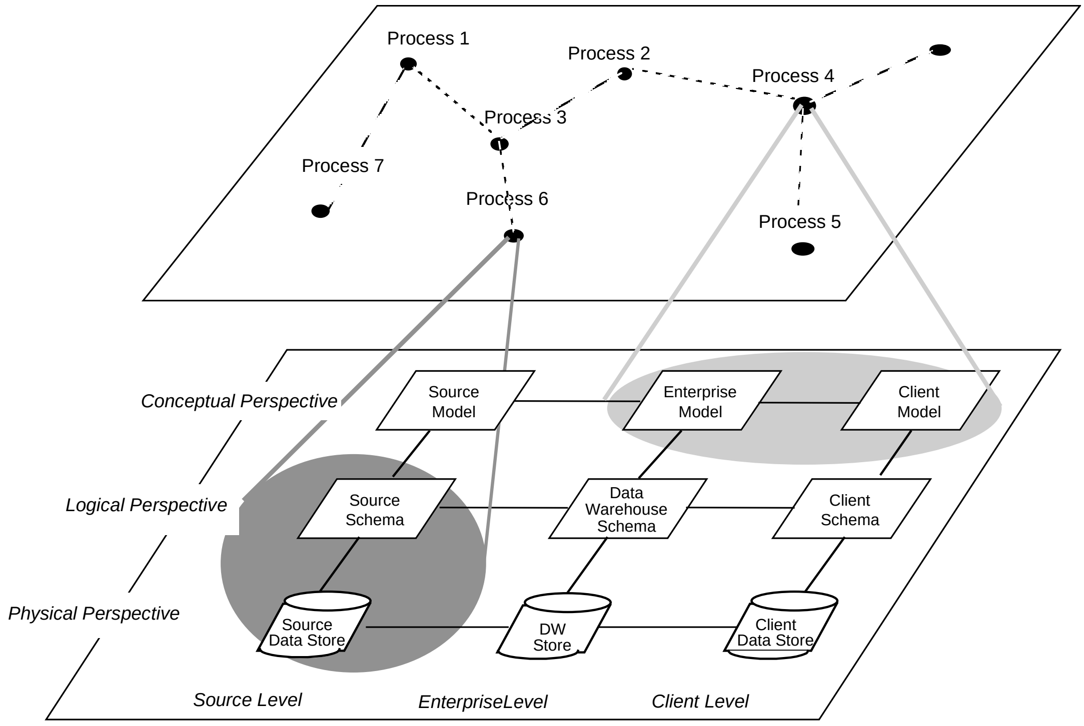

Figure 5 Process Model of Data Warehouse Design and Operation

2.3 Linking the DW Architecture to a Formal Quality Model

For the design and analysis of a data warehouse application, it is necessary to add quality factors to components of the data warehouse architecture. For the analysis of bottlenecks, e.g., it is necessary to know how much data a component is able to transfer in a certain time interval. This value is a quality factor of the component. Data quality comes into play, when the analyst reports bad data quality in the OLAP views of the data warehouse. In such a case, the data warehouse designer must be able to find out the reason for this problem, e.g. bad data quality in one of the source information systems.

As data warehouses are represented in a common language, it is possible to compare data warehouses. The data warehouse designer is able to create alternative data warehouse architectures and to check the improvement against the old architecture. Moreover, an architecture model standardize the development of a data warehouse application, because specific tasks are required to complete the architecture and quality model. The model can also serve as a documentation of the warehouse.

Furthermore, the data warehouse architecture and quality model support the evolution of the data warehouse architecture. New architectures or components can be tested on the model before the components are implemented in the data warehouse application. Another issue is that external tools can retrieve information about the data warehouse application from the repository and use this information to reason on properties of the application. These tools can also store their information into the repository making this information available to other client programs.

Our formal approach in linking architecture (both static and process aspects) and quality can be seen as an adaptation of the Goal-Question-Metric (GQM) approach that is widely adopted in software quality management [OB92, JP93]. At the highest level of this approach, a goal hierarchy of quality factors is elaborated; our approach here extends one suggested in the Total Data Quality Programme of MIT. Each goal can then be associated with certain questions that must be answered to evaluate the goal; finally, each question is associated with a set of metrics that need to be evaluated in order to answer the question. Note that due to the subjective nature the set of questions may not fully cover satisfaction of a goal, nor can a question necessarily be fully answered by metrics. Nevertheless, the use of advanced repository technology (at the question level) in combination with special-purpose reasoners, simulators, and measurement tools (at the metric level) will significantly improve the current situation. In this report, we assume that the metrics have been evaluated and look at the question level only. The approach then looks as shown in figure 6.

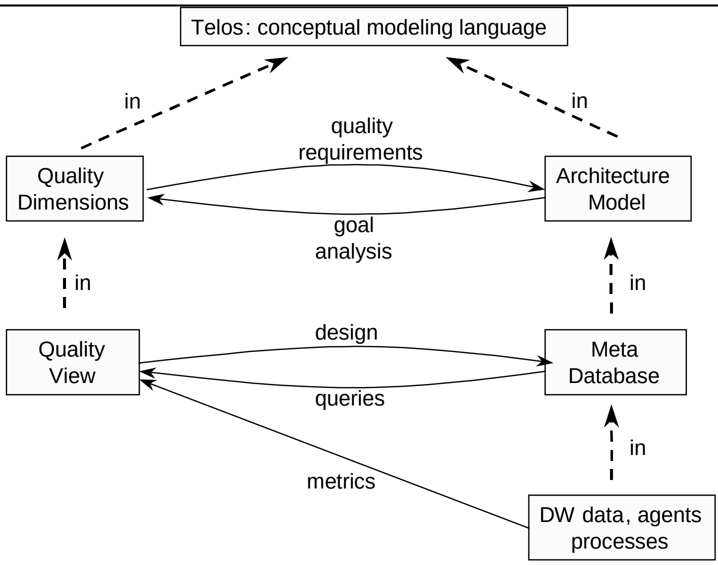

Figure 6 Framework for Architecture and Quality Model

Figure 6 shows how DWQ links the quality model to the architecture model when implementing the above approach in a repository. The architecture and quality models are meta models, described in the conceptual modeling language Telos [MBJK90]. A data warehouse meta database is an instance of the architecture model. It contains, for example, meta data of programs and database systems, and schema information.

Following the GQM idea, the quality model is supported by quality views which are technically represented by queries on the meta data, or by metrics measured on the actual systems. To abstract from the specific metrics and measurement tools which are still under development, we currently assume that the results of such measurements are stored as materialized views in the meta databases, such that we only need to deal with queries on meta data in the remainder of this report. Incidentally, the quality views themselves can also be materialized as quality annotations in the meta database – an approach often used in practice (see section 3).

In our model, quality factors can be attached either to a specific data warehouse architecture or to the data warehouse data, agents or processes. For example, a schema has quality factors like relevance and interpretability, whereas a process, which is an instance of program in the architectural model, has quality factors like efficiency and correctness.

The architecture and quality model can be used in two ways. On the one hand, the data warehouse administrator can model an existing data warehouse application and quality requirements and test if the data warehouse fulfils the requirements. Furthermore, the administrator can use the model to find some weaknesses in the architecture. On the other hand, a data warehouse designer can start with modeling the quality requirements for a data warehouse. The requirements constrain the architecture, so that the development process of the data warehouse is driven by the quality requirements.

The current state of the architecture and quality model supports only the first way, analyzing an existing architecture for weaknesses and errors. The design and evolution of an data warehouse with respect to quality requirements will be a task in the second phase of the DWQ project.

2.4 The DWQ Repository ConceptBase

The DWQ architecture and quality model is represented in Telos, a conceptual modeling language for representing knowledge about information systems. The structurally object-oriented framework of Telos generalizes earlier data models and knowledge representation formalisms, such as entity-relationship diagrams or semantic networks, and integrates them with predicative assertions and temporal information. A formal description of Telos can be found in [MBJK90]. The object-oriented deductive database system ConceptBase [JGJ+95] is a repository for meta data based on Telos which the project uses for prototype implementations.

The query language of ConceptBase can be used to analyze a data warehouse architecture and to find out weaknesses and errors in the design of a data warehouse. As we will show in section 7.3, we can formulate queries to find components in a data warehouse which do not fulfil the quality requirements of the users. Constraints are used to find contradictions during the design phase of a data warehouse. Deductive rules infer implicit relationships between data warehouse components. Although the query language is powerful compared with traditional database systems, it will have to be linked to special-purpose reasoners, statistical analysis tools, quantitative analytical models and simulators in order to support not just the question level, but also the metrics level of our approach. To support such integration, ConceptBase offers various application program interfaces for programming languages such as Java, C, C++ or ProLog. It can be easily integrated in other applications, and serve as a central repository in the DWQ project.

To separate the different models and schemas in the meta database, we use different modules in the ConceptBase repository [NJ97]. Modules can be seen as a small object base inside another object base. They allow the use of the same object names within one object base, a feature which is especially useful for modeling the contents of several similar information systems. The modeling of the data warehouse components in the three perspectives can be done independently from other modeling tasks inside a module in the data warehouse architecture. This allows the cooperative design of a data warehouse among a group of data warehouse developers.

3 Related Work

This section gives an overview of the current state of the art in the context of data warehouse architectures and data warehouse quality design. Section 3.1 presents related work in other data warehouse research projects, in data warehouse industry and in standardization of metadata. In section 3.2, we discuss work done in quality management covering industry quality management as well as data and software quality management.

3.1 Data Warehouse Architectures

Data warehouses attract the interest of many of the database researchers. Most researchers and developers share the same general vision of what a data warehouse is [Wid95], [CD96]. Data are extracted from several data sources, cleansed, customized and inserted into the data warehouse. Initially, we can think of the data warehouse as a collection of *materialized views*. The logical structure and semantics of the data, or else *business model*, is stored in an *Information directory*. Next, the data warehouse data can be filtered, aggregated and stored in smaller specialized data stores, usually called *data marts*.

The research in data warehousing focuses on source integration and update propagation. The *Information Manifold* (IM) is an implemented system, developed at AT&T, for information gathering from disparate sources such as databases, SGML documents, unstructured files [LSK95, KLSS95, LRO96]. It is based on a rich domain model, expressed in a knowledge base, that allows for describing various properties of the information sources, from the topics they are about to the physical characteristics they have. This enables the user to pose high-level queries to extract the information from different sources in a unified way. Observe that the ability of modeling both conceptual and physical characteristics increases believability and accessibility of data. The architecture of IM suits the dynamic nature of the information sources.

TSIMMIS (*The Stanford-IBM Manager of Multiple Information Sources*) is a project that shares with IM the goal of providing tools for the integrated access to multiple and diverse information sources and repositories [CGMH+94, Ull97]. Each information source is equipped with a *wrapper* that encapsulates the source, converting the underlying data objects to a common data model. In TSIMMIS a simple object oriented data model called *Object Exchange Model* (OEM) is used. On top of wrappers TSIMMIS has another kind of system components: the *mediators* [Wie92]. Mediators can be conceptually seen as views of data found in one or more sources which are suitably integrated and processed.

The *Squirrel* Project is a system under development at the University of Colorado [ZHKF95a, ZHKF95b, HZ96, ZHK96] which provides a framework for data integration based on the notion of *integration mediator*. Integration mediators are active modules that support integrated views over multiple databases. The mediators are generated from high-level specifications and a view of mediator can be fully materialized, partially materialized or fully virtual. A key feature of these mediators is their ability to incrementally maintain the integrated views by relying on the active capabilities of sources. In [HZ96], emphasis is posed on the fact that, in several applications, materializing only matching information is a suitable design choice. Moreover, data quality is considered by defining formal properties of consistency and *freshness* for integrated views, and showing that automatically generated Squirrel mediators satisfy such properties, when the sources and the network respect some ″natural″ assumptions.

The WHIPS (*WareHouse Information Prototype at Stanford*) system [HGMW+95, WGL+96] is a data warehouse system currently under development at Stanford University which is being used as a test-bed for various integration schemes. The goal is to develop algorithms for the collection, integration and maintenance of information from heterogeneous and autonomous sources. The architecture of the WHIPS prototype system is described in [WGL+96]. The WHIPS architecture consists of a set of independent modules implemented as CORBA objects. This ensures both modularity and scalability. The central component of the system is the *integrator*, to which all other modules report. Different data models may be used both for each source and the data warehouse data. The relational model is used as a unifying model, and for each source and the data warehouse the underlying data are converted to the relational model by a specific *wrapper*.

Data warehouse industry focus the discussion of data warehouse architecture on the data model of the central data warehouse. Whereas some vendors, especially vendors of traditional relational database systems, propose a ROLAP architecture (Relational On-Line Analytical Processing) [MStr95, MStr97, Info97, RedB97], vendors of multi-dimensional database support the MOLAP architecture (Multi-dimensional On-Line Analytical Processing) [Arbo96]. The advantage of a MOLAP architecture is, that it provides a direct multi-dimensional view of the data whereas a ROLAP architecture is just a multi-dimensional interface to relational data. A ROLAP architecture has the advantage, that it can be easily integrated into other existing relational information systems and relational data can be stored more efficiently than multi-dimensional data. In practice, multi-dimensional database systems are used to complement a data warehouse architecture, they are implemented as a data retailer [Sahi96].

The simplest architecture consists only of source databases, a central data warehouse and several clients. As data warehouse applications are becoming more complex, warehouses are being built using multi-tier architectures to increase performance, i.e. there is not only one central data warehouse but also *data marts* which allow to place the data closer to the end-user [Weld97]. These architectures are called a *federated* or a *tiered* data warehouse architecture.

Some effort in data warehouse research has been spent on the standardization of meta data. The most promising approach is the *Metadata Interchange Specification* [Meta96], proposed by the Metadata Coalition, an open group of companies such as IBM, Sybase, Informix, etc. The Metadata Interchange Specification (MDIS) should provide a standard access mechanism and a standard application programming interface to control and manage metadata with interchange specification-compliant tools. The MDIS is character-based and platform-independent. Furthermore, it is extensible because the contents of what is considered as metadata will be evolved. As one knows from the past, many standards proposed by open groups have been overtaken by solutions of strong industry concerns. The repository system of Microsoft [BHS+97] is extensible and evolvable, so that customers and independent software vendors can store any kind of (meta-) data in the repository. It also contains a meta model (so called InformationModel) for data warehouse applications, which is currently under development in cooperation with data warehouse companies.

3.2 Data Warehouse Quality Design

In this section we will present the related work to quality issues, in three main areas. First, we will define the notion of quality and then present some ideas from the field of Total Quality Management. Next, we will present related work in the field of data quality dimensions and design. Finally, we will also present related work in the field of software quality and information systems quality management.

3.2.1 Quality Definition

In [BBM+95], a definition and quantification of *quality* is given:

$$Quality = \frac{Performance}{expectation}$$

Taguchi defined quality as the loss imparted to society from the time a product is shipped [BBM+95]. 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).

3.2.2 Total Quality Management

In our days, one of the most popular trend 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. TQM involves a strategic planning for the achievement of quality. In [BBM+95] it is suggested that several steps have to take place in order to complete the strategic planning successfully, such as discovery of the customers needs, positioning of the organization towards its customer base, future prediction, analysis of the gaps between current and future states of the organization, closing of these gaps through the construction of an action plan, alignment of this plan to the mission, vision and core values of the organization, and finally, implementation of the plan.

Despite the effort spent, the original design is never sufficient enough. Continuous control and improvement of processes must take place. In [Jura88], a specific approach based on the planning, control and improvement of processes is presented. This approach is called the *Juran Trilogy*.

To be useful, quality has to be measured. As reported in [Boed87], several criteria are defined for the evaluation of the measures employed, such as simplicity, number, development by the users, relevance to the customer, focus on the improvement of processes, prevention of errors and long-term planning, cost and knowledge of the measures from the side of the users.

3.2.3 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 more detailed presentation can be found in [DWQ97] and [WSF95].

Wang et al. [WSF95] present 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* (DMSs) that incorporate data quality aspects and definition of data quality dimensions.

In [WSF95] 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 former group, one can find efforts such as the ones presented in [BP85], [BP87] and [BWPT93]. In [BP85] and [BP87] 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 the field of the design of system technologies to ensure the quality of data, one can find efforts like [WKM93], [WRG93], [WRK95], [WM90], where an attribute-based model that can be used to incorporate quality aspects of data products is presented. 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.

Finally, there is a great deal of work related to the definition of data quality dimensions. In [BP82], [BP85], [BP87], [BT89], [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 [SLW94] data quality is modeled through the definition of intrinsic, contextual, representation, and accessibility aspects of data. In [Jans88], [LU90], [WRG93] *validation*, *availability*, *traceability* and *credibility* are introduced.

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 [Hall78] various factors are proposed, such as *usability, reliability, independence*, etc. [Krie79] proposes attributes such as *accuracy, timeliness, precision, reliability, completeness* and *relevancy*.

In [AA87] 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 [WSF95] 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, e.g. based on an ontological-based definition [Wand89], [WW96], [WW90], information theory [DM92], marketing research [WSG94]. 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 or defined by a data quality standard technical committee [WSF95].

3.2.4 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 [MRW78], suggests 11 criteria of software quality. The model developed by B. Boehm [Boeh89] suggests 19 quality factors for software. Finally, ISO 9126 [ISO97] suggests six basic quality factors which are further analyzed to an overall of 21 quality factors. In [HR96] 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, 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).

3.2.5 Information Systems Quality Management

So far, this review has mostly focused mostly on research and standards concerning the definition of quality factors. Starting from the initiative of the Software Engineering Institute, adapted to the European situation by the European Software Institute, there has also been significant work concerning the classification of organizational maturity with respect to software process quality as well as operational approaches to capture software quality or to design from quality goals. A typical example of the former is the GQM approach described earlier in this report, while methods such as QFD or FMEA are typical of the latter. A structured overview of the issues and strategies, embedded in a repository framework, can be found in [JP92].

4 Data Warehouse Architecture Model

We will present in this chapter the architecture model for data warehouses. The model is a methodology for describing a data warehouse in the three perspectives, which were shown in chapter 1. In this section, we use graphs to present the model, the formal presentation in the Telos language is given in chapter 7. The constraints of the architecture model are explained in chapter 7, too.

For distinction of the three perspectives in the architecture model, we have introduced three base classes:

- *ConceptualObject* is the base class for all objects in the conceptual perspective
- *LogicalObject* is the base class for all objects in the logical perspective
- *PhysicalObject* is the base class for all objects in the physical perspective

This part of the model is shown in figure 7. In the following figures, we use bold arrows to mark specialization links between objects, thin arrows with a label are used for attribute links.

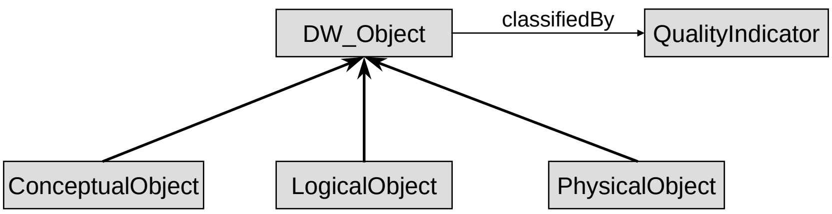

Figure 7 Base classes for architecture model

Every class in the architecture model is derived from the class *DW_Object*. The linkage to the quality model is established by the attribute link *classifiedBy* from DW_Object to *QualityIndicator*, associating any object of the data warehouse architecture with a quality indicator. Note, that DW_Object is a base class for all objects on any level of abstraction, e.g. in our model a schema is a DW_Object as well as the instances of the schema are DW_Objects, too. The classes ConceptualObject, LogicalObject and PhysicalObject are specializations of DW_Object, so that the attributes of DW_Object are inherited to any kind of object in the architectural model. The quality model is explained in chapter 5.

4.1 Conceptual Perspective

The conceptual perspective is a view on the business model of the information systems of an enterprise. A central role plays the enterprise model, which gives an integrative overview of the conceptual objects of an enterprise. The models of the client and source information systems are views on the enterprise model, i.e. their contents is described in terms of the enterprise model. One goal of the conceptual perspective is to have a model of the information independent from physical organization of the data, so that relationships between concepts can be analyzed by intelligent tools, e.g. to simplify the integration of the information sources.

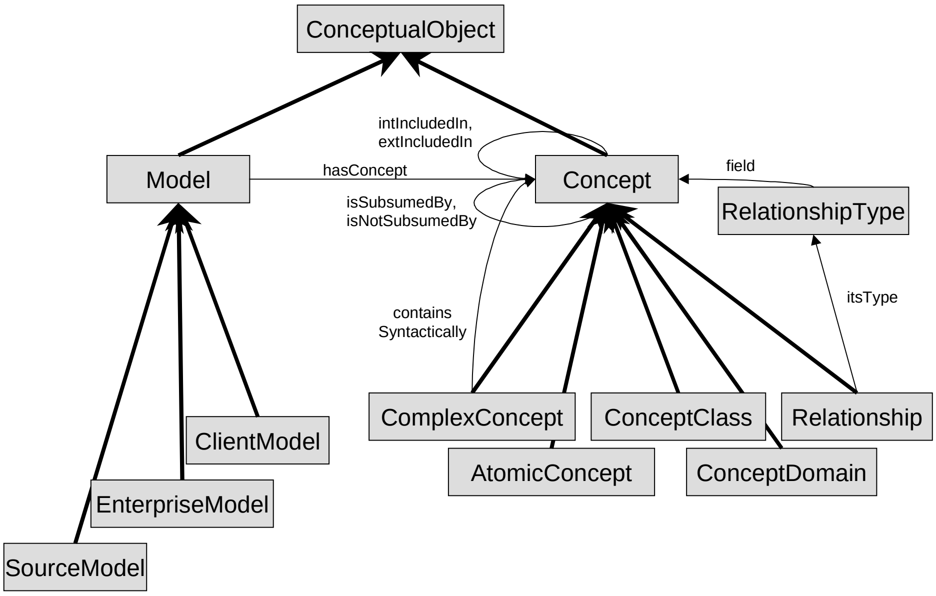

Figure 8 Conceptual Perspective of the architectural model

The central class of the conceptual perspective is *Model*. A model is related to a source, a client or the whole enterprise, and it represents the concepts which are available in the corresponding source, client or enterprise. The classes *ClientModel*, *SourceModel* and *EnterpriseModel* are needed, to distinct the models of several sources, clients and the enterprise itself. The enterprise model is a class with only one instance, because there should be only one enterprise model. A model consists of *Concepts*, each representing a concept of the real world, i.e. the business world. Concepts are defined in an extension of a description logic, which is described in deliverable 3.1. To store the definitions of the concepts in the architecture model, we need some specializations of the class *Concept:*

- *ComplexConcept* is a concept defined by a complex expression in the description logics language,
- *AtomicConcept* is a concept, which is not defined by a complex expression,
- *ConceptClass* is a name of concept,
- *ConceptDomain* is a domain of simple values, e.g. integer or string,
- *Relationship* is a n-ary relation between concepts.

Every concept is an instance of two classes, either an instance of ComplexConcept or AtomicConcept and an instance of ConceptClass, Relationship or ConceptDomain. A relationship has a *RelationshipType*, which describes the concepts participating in the relation. For simplicity, we store the expression defining a complex concept just as a string. The attribute *containsSyntactically* means that a concept appears syntactically in the expression of the corresponding complex concept. A reasoner for the description logic will be connected to the DWQ repository and retrieve the concept definitions from the repository. Furthermore, it stores the results of the reasoning process in the model as attributes *isSubsumedBy* or *isNotSubsumedBy* of the corresponding concepts. Essentially, the repository will serve as a cache for reasoning results. Any tool can ask the repository for containment of concepts. If the result was already computed, it is directly answered by the repository. Otherwise, the reasoner is invoked by the repository to compute the result.

4.2 Logical Perspective

Our approach to the design of a data warehouse is similar to traditional database design. After the design of a conceptual model, we will map this model to a logical schema. The data model of the logical schema is given by the corresponding physical component, which implements the logical schema. We use currently the relational model as an example data model, other data models such as the object-oriented or multi-dimensional data model can be easily integrated in this framework.

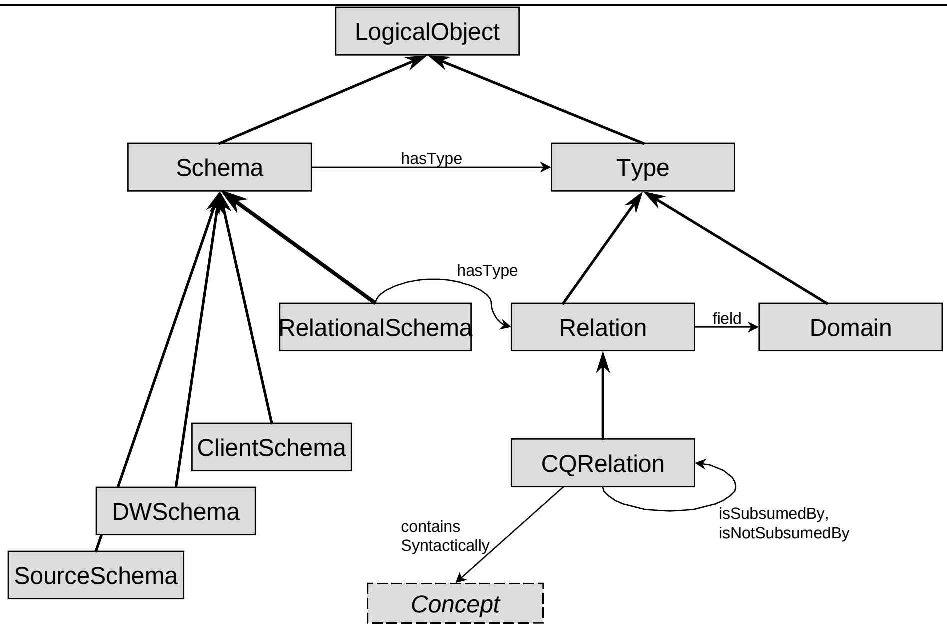

Figure 9 Logical Perspective of the architectural model

The central point in the logical perspective is *Schema.* As a model consists of concepts a schema consists of *Types*. An example for a logical schema is the *RelationalSchema*. A *Relation* is a type with fields, which contain values of a relational *Domain.* The connection to the conceptual perspective is established by the class *CQRelation,* a special relation which is defined as a conjunctive query over the concepts in the conceptual model. For relations defined by a conjunctive query, an external reasoner can make a subsumption check. The result is stored and used in the same way as in the conceptual model.

Like in the conceptual perspective, we distinguish in the logical perspective between *ClientSchema*, *DWSchema* and *SourceSchema* for the schemas of clients, the data warehouse and the sources. Again, there is only one data warehouse schema, because we model only one data warehouse application at a time. For each client or source model, there is one corresponding schema. The conjunctive query for a relation in a source schema can only be defined with concepts of the corresponding source model. The same holds for the relations in the client schema and the data warehouse schema. This restriction is guaranteed by a constraint in the architecture model. The attribute *containsSyntactically* of *CQRelation* contains the concepts, which appears syntactically in the conjunctive query expression.

4.3 *Physical Perspective*

Data warehouse industry has especially explored the physical perspective, so that many aspects in the physical perspective are taken from the analysis of commercial data warehouse solutions such as Software AG's SourcePoint tool [SAG96], the data warehouse system of RedBrick [RedB97], Informix's MetaCube[Info97], Essbase of Arbor Software [Arbo96] or the product suite of MicroStrategy [MStr95, MStr97]. We observed that the basic physical components in a data warehouse architecture are agents and data stores. Agents are programs that control other components or transport data from one physical location to another. Data stores are databases which store the data that is delivered by other components. An overview of the physical perspective is given in figure 10.

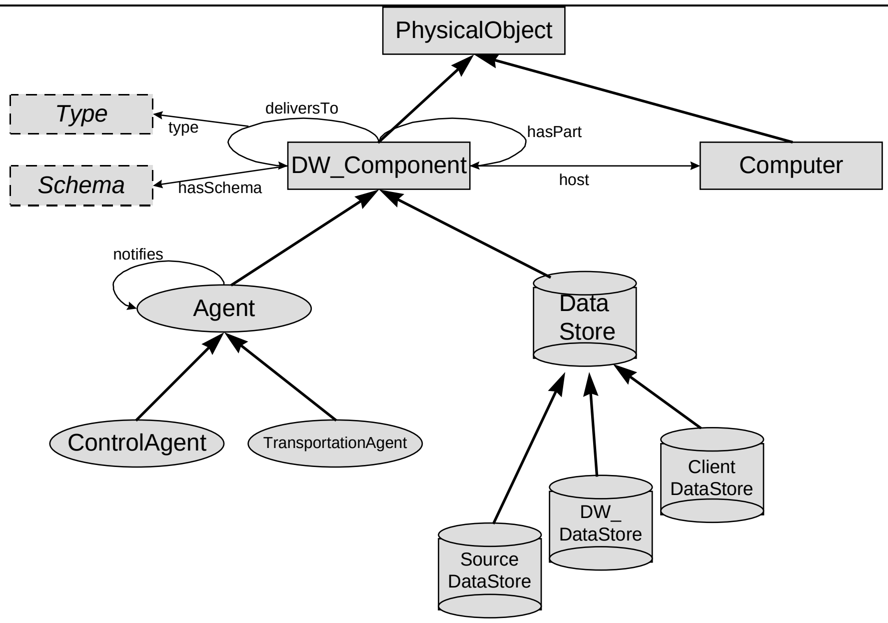

Figure 10 Physical Perspective of the architectural model

The basic class in the physical perspective is *DW_Component*. A data warehouse component may be composed out of other components. This fact is expressed by the attribute *hasPart*. Furthermore, a component *deliversTo* another component a *Type,* which is part of the logical perspective. Another link to the logical model is the attribute *hasSchema* of DW_Component. Note, that a component may have schema, i.e. a set of several types, but it can only deliver a type to another component. This is due to the observation that agents usually transport only „one tuple at a time" of a source relation. The attribute *host* of DW_Component shows that a component has a physical location, a *Computer* system which may have additional attributes such as the IP address, operating system, administrator, etc.

One type of component in a data warehousing environment is an *Agent.* There are two types of agents: *ControlAgent* which controls other components and agents, e.g. it *notifies* another agent to start the update process, and *TransportationAgent* which transports data from one component to another component. It may also notify other agents about errors or termination of its process.

Another type of component is a *DataStore*. It physically stores the data which is described by models and schemas in the conceptual and logical perspective. As in the other perspectives, we distinguish between *ClientDataStore*, *DW_DataStore* and *SourceDataStore* for data stores of clients, the data warehouse and the sources. Furthermore, the meta database is special data store.

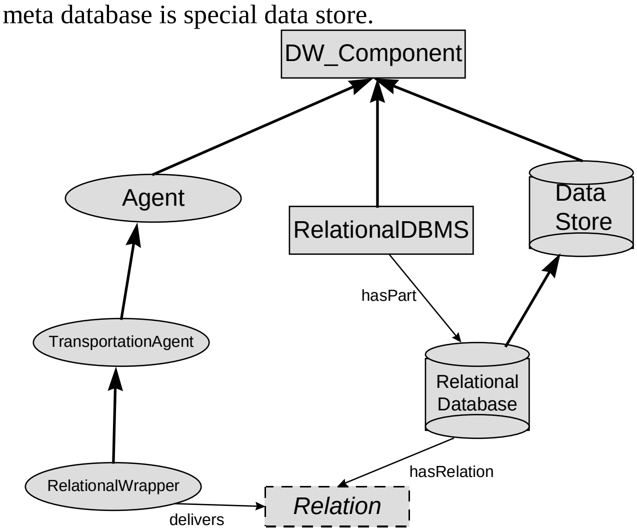

Figure 11 Physical components for the relational model

Figure 11 shows an example for a specialization of the physical components, the relational model which is implemented in a relational database management system (*RelationalDBMS*). One part of the relational DBMS is the storage management, namely a *RelationalDatabase* which stores the relations described in the logical model. A *RelationalWrapper* is a transportation agent, which wraps any kind of data source into the relational model and *delivers* a relation.

4.4 Integration of the three Perspectives

In addition to the already described relationships between the three perspectives in the previous sections, we have defined two classes to relate corresponding models, schemas and stores with each other.

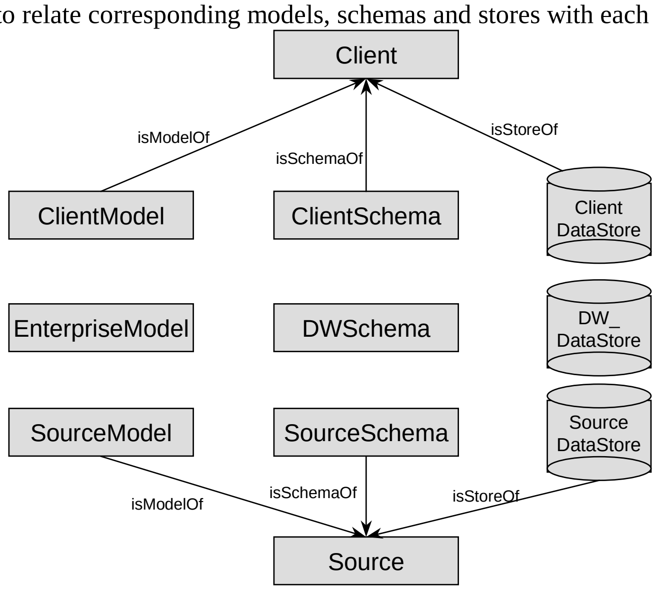

Figure 12 Integration of the three perspectives

The classes *Client* and *Source* represent independently from the three layers a client respectively a source. We add an attribute to each class in the client level and the source level to represent the relationship to a source or client object. There is no corresponding class for the data warehouse level, because the goal of the data warehouse approach is to have only one integrated enterprise model, which is implemented in one data warehouse schema and in one data warehouse store.

Furthermore, we can attach additional information to the instances of *Source* and *Client,* if we can not classify the information in one of the three perspectives. For example, we can add the person who is responsible for a source to the source object.

5 Data Warehouse Quality Model

In this section, we are going to define data warehouse quality dimensions, their mapping to specific components and/or processes of a data warehouse and metrics for their evaluation. Nevertheless, before we proceed with the rest of this section, there are two basic definitions regarding data warehouse quality:

- *Data warehouse quality dimensions* are groupings of properties of data warehouse components or processes, representing their state with respect to deficiencies (or efficiency) in their operation.
- *Data warehouse quality indicators* are data describing and measuring the state of a component or process of a data warehouse with respect to deficiencies (or efficiency) in their operation. In other words, quality indicators can be viewed as instantiations of quality dimensions.

A data warehouse quality dimension can be decomposed into more detailed dimensions. Consequently, *dimension graphs* (or trees) are formed out of this analysis. This is why quality dimensions are defined as *groupings* of properties. As we will see in the sequel, in several cases, two dimensions can be refined to non-disjoint sets of more “detailed” dimensions.

Since quality dimensions try to model states of the real world, their instantiations, quality indicators, (being data themselves) must have their own properties. As stated before, a quality indicator describes and measures the quality of a component or a process in the data warehouse. The data warehouse designer has several *expectations* on the quality of a component during the design of the data warehouse. Furthermore, we have the currently *achieved* quality of the component. The *expected* quality value may be an interval with an *upper* and a *lower* bound, whereas the *achieved* value is one measured value.

One important issue in modeling a data warehouse architecture and data warehouse quality model is the *traceability* of quality. On the one hand, the quality dimension refers to a process of the warehouse, there are certainly architecture components *related* to it. In addition, the computation of a quality indicator may include some other quality indicators, so the quality indicator *depends on* other quality indicators. We use these links to find the reason for unexpected quality measures or the set of affected data warehouse objects by a quality indicator. The *method of computation* of the achieved quality level is specified by another attribute of the quality indicator class. Furthermore, we attach a *priority* to a quality dimension. This fact reflects the different importance of quality dimensions in data warehouse applications.

All the aforementioned properties hold for all the quality dimensions and indicators. Consequently, in the Telos representation of the model, we define a super-class for all the classes representing quality dimensions. The definition of this class is as follows:

```
Class QualityIndicator with
attribute,single
       expected_level_upper : QualityMeasure;
       expected_level_lower : QualityMeasure;
       achieved_level : QualityMeasure
attribute
       related_objects : DW_Object;
       dependsOn : QualityIndicator;
       method_of_computation: String;
       priority : Real;
       comment : String
end
```

The first attributes are in the attribute category single, because only one value is allowed for this attribute, whereas the other attributes can have multiple values. We can pose queries to our information directory, to find out where the measured values of some quality indicators are not in the required interval of values. Other characteristic examples for queries are queries asking the quality indicators which are out of range and belong to a specific quality dimension, or queries discovering quality indicators with high priorities. In addition to the predefined queries which we will present in section 7.3, the data warehouse user or administrator can formulate his own queries to find some application specific properties of a data warehouse application.

5.1 Data Warehouse Quality Dimensions

In this paragraph we present the various data warehouse quality dimensions. We categorize the quality dimensions according to the *processes* which take place during the data warehouse operation. 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 section we will present the quality dimensions related to the aforementioned processes and data 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.

5.1.1 Design and Administration Quality

We consider the following basic quality dimensions concerning the design and administration of a data warehouse:

- schema quality
- metadata evolution

In figure 13 the overall hierarchy of design and administration quality is depicted.

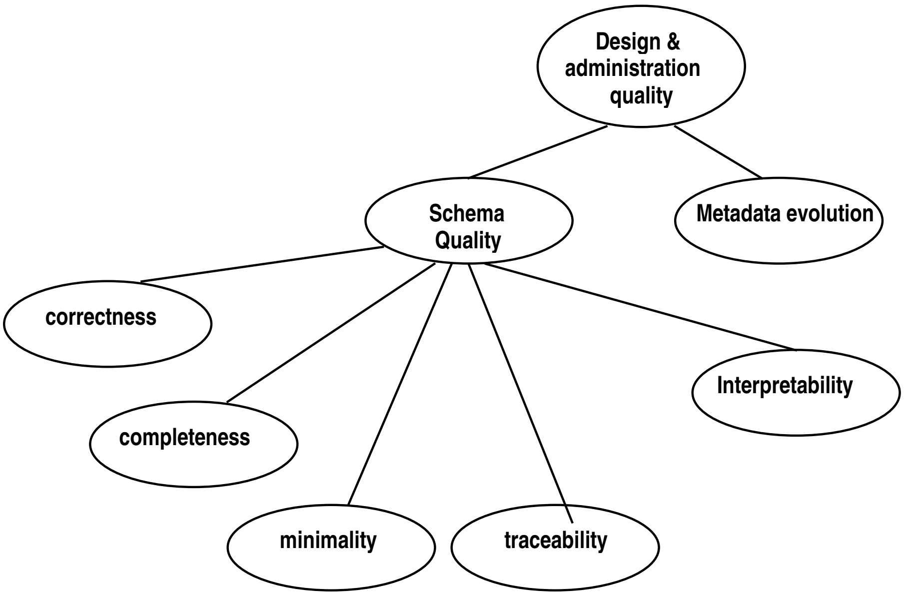

Figure 13 Design and Administration Quality Dimension

The *schema quality* refers to the ability of a schema or model to represent adequately and efficiently the information. The *correctness* dimension is concerned with the proper comprehension of the entities of the real-world entities, the schemata of the sources (models) as well as with the user needs. The *completeness* dimension is concerned with the preservation of all the crucial knowledge in the data warehouse schema (model). The *minimality* dimension describes the degree up to which undesired redundancy (due to cycles of relationships, transitivity of inclusion dependencies, or other forms of integrity constraints) is avoided during the source integration process. The *traceability* dimension is concerned with the fact that all kinds of requirements of users, designers, administrators and managers should be traceable to the data warehouse schema. The *interpretability* dimension ensures that all components of the data warehouse are well described, so as to be administered easily. The *metadata_evolution* is concerned with the way the schema evolves during the data warehouse operation.

As it is obvious from the above, the design and administration quality dimensions are related to models and schemata. Consequently they are linked to the respective objects (*LogicalObject* and *PhysicalObject*) 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. Dimensions like interpretability and metadata evolution deal mostly with the quality of the information directory and less with the data warehouse itself.

5.1.2 Software Implementation/Evaluation Quality

Software implementation and/or evaluation is not a task 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 dimensions of ISO 9126, enhanced with just one dimension, namely *implementation efficiency*.

In the three perspectives of the architecture model, software is modeled only in the physical level, since the conceptual and logical level deal mostly with the models and schemata of the data stores. It is present for all the levels of the physical model: source, data warehouse and client level, as well as for communication agents between these levels. Software is modeled through *DW_Components* in the Telos architecture module. Consequently, all the quality dimensions referring to software implementation and evaluation have *DW_Components* as their related architecture components.

In table 1 the first column describes the basic quality dimensions, the second analyzes them into more detailed dimensions and the third gives the definition of each one of them in ISO 9126.

Functionality		set of attributes that bear on the existence of a set of functions and their specific properties. The functions are those that satisfy stated or implied needs.
	Suitability	attribute of software that bears on the presence and appropriateness of a set of functions for specified tasks
	Accuracy	attributes of software that bear on the provision of right or agreed results or effects

	Interoperability	attributes of software that bear on its ability to interact with specified systems
	Compliance	attributes of software that make the software adhere to application related standards or conventions or regulations in laws and similar prescriptions
	Security	attributes of software that bear on its ability to prevent unauthorized access, whether accidental or deliberate, to programs and data
Reliability		set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time
	Maturity	attributes of software that bear on the frequency of failure by faults in the software
	Fault tolerance	attributes of software that bear on its ability to maintain a specified level of performance in cases of software faults or of infringement of its specified interface
	Recoverability	attributes of software that bear 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
Usability		set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users
	Understandability	attributes of software that bear on the user's effort for recognizing the logical concept and its applicability
	Learnability	attributes of software that bear on the user's effort for learning its application
	Operability	attributes of software that bear on the user's effort for operation and operation control
Software Efficiency		set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions
	Time behavior	attributes of software that bear on response and processing times and on throughput rates in performing its function
	Resource Behavior	attributes of software that bear on the amount of resources used and the duration of such use in performing its function
Maintainability		set of attributes that bear on the effort needed to make specified modifications
	Analyzability	attributes of software that bear on the effort needed for diagnosis of deficiencies or causes of failures, or for identifications of parts to be modified
	Changeability	attributes of software that bear on the effort needed for modification, fault removal or for environmental change
	Stability	attributes of software that bear on the risk of unexpected effect of modifications
	Testability	attributes of software that bear on the effort needed for validating the modified software
Portability		set of attributes that bear on the ability of software to be transferred from one environment to another
	Adaptability	attributes of software that bear 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
	Installability	attributes of software that bear on the effort needed to install the software in a specified environment
	Conformance	attributes of software that make the software adhere to standards or conventions relating to portability
	Replaceability	attributes of software that bear on the opportunity and effort of using it in the place of specified other software in the environment

		of that software
Implementation Efficiency		set of attributes that bear on the progress of the development of software with respect to specified milestones or deadlines.

Table 1

5.1.3 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). Yet, in the literature there is a great deal of work regarding alternative ways of updating the information of the warehouse. In [Wid95] four different modes for the propagation of updates at the sources are described: batch, immediate, periodic and on demand. The quality dimensions 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. Since data loading is a process happening at the *physical* level for data warehouse and client data stores, we can only relate the data loading quality dimensions to the respecting architectural components.

We consider the following basic quality dimensions concerning the loading of data in the data warehouse:

- analyzability
- transactional availability

In figure 14 the overall hierarchy of quality dimensions related to data loading is depicted.

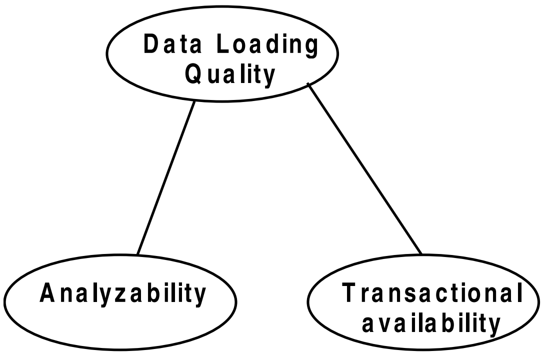

Figure 14 Data Loading Quality Dimension

The *analyzability* dimension 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). The *transactional availability* dimension 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).

5.1.4 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 process of querying is related with the following basic quality dimensions:

- accessibility
- usefulness

In figure 15 the full hierarchy of quality dimensions related to data usage is depicted.

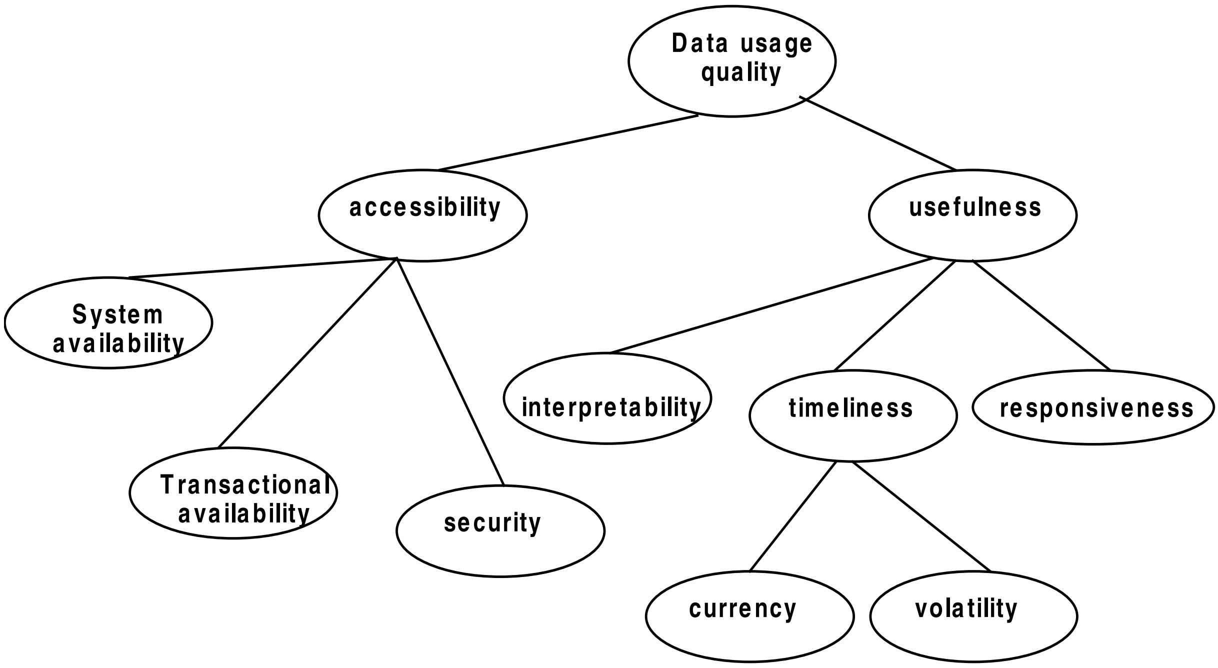

Figure 15 Data Usage Quality Dimension

The *accessibility* dimension is related to the possibility of accessing the data for querying. The *security* dimension describes the authorization policy and the privileges each user has for the querying of the data. The *system availability* dimension describes the percentage of time the source or data warehouse system is available (i.e. the system is up and no backups take place, etc.). The *transactional availability* dimension, 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* dimension 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* dimension 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 dimension). The *volatility* dimension 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 dimension). The *interpretability* dimension, 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.

The data usage quality is referring to the querying of the data. Since this is done by tools described in the physical layer of the architecture, it is related to components belonging to this part of the architecture. More specifically, the accessibility dimension (and its components) refers to data stores of all levels represented by instances of *ClientDataStore*, *DW_DataStore* and *SourceDataStore* in the architecture model. The timeliness refers to characteristics of the stored data and the responsiveness to querying tools.

5.1.5 Data Quality

The quality of the data which 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. 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 dimensions proposed in other models. For example, in [WRK95] our notion of data quality, in its greater part, is treated as a second level dimension, namely *believability*. Yet, in our model, the rest of the dimensions proposed elsewhere, are treated as process quality dimensions. The basic quality dimensions we introduce are:

- completeness
- credibility
- accuracy
- consistency
- data interpretability

In figure 16 the full hierarchy of data quality dimensions is depicted.

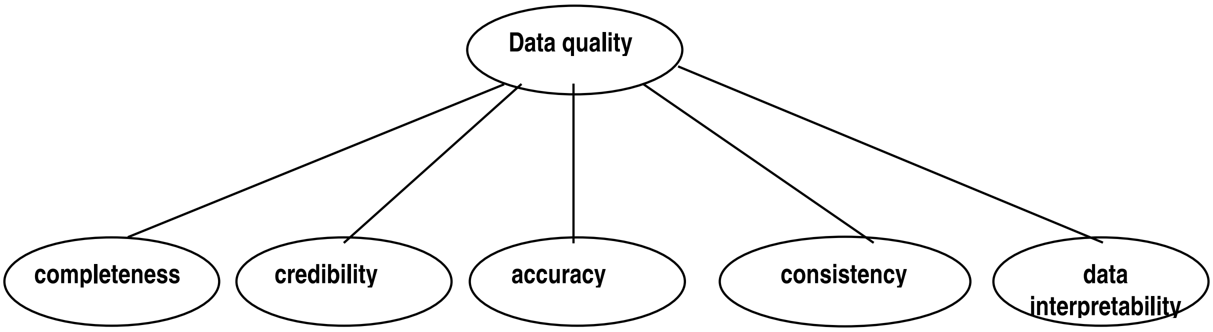

Figure 16 Data Quality Dimension

The data quality dimension does not cover a data warehouse process: it refers directly to properties of the stored data (i.e. not of the schemata or the models). Consequently, it is related to components of the the architecture representing *physical* data stores at all levels.

The *completeness* dimension describes the percentage of the 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* dimension describes the credibility of the source that provided the information. The *accuracy* dimension describes the accuracy of the data entry process which happened at the sources. The *consistency* dimension describes the logical coherence of the information. The *data interpretability* dimension 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.)

5.2 Methods of Measurement and Metrics for the Data Warehouse Quality Dimensions

In the previous paragraphs we defined the dimensions of quality in a data warehouse environment. We associated quality dimensions to their relevant processes and affected components. In the following paragraphs we will provide measurement techniques and metrics for data warehouse quality dimensions.

5.2.1 Methods of measurement and Metrics for the Design and Administration Quality Dimension

The design and administration major quality dimensions are schema quality and metadata evolution, which are in terms analyzed to other dimensions. The correctness, completeness, minimality and traceability dimensions should be measured after the source integration process in order to guarantee the absence of errors. The interpretability dimension 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 dimension is used to track down the way the schema and models of the data warehouse evolve.

In table 2 a full list of quality dimensions regarding design and administration, methods of measurement and metrics is given.

Dimension	**Methods of measurement**	**Metrics**
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		

	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	metadata evolution (versioning/timestamping of the metadata)	number of metadata evolutions not documented

Table 2

5.2.2 Methods of measurement and Metrics for the Software Implementation and Evaluation Quality Dimension

Software implementation is a hard and time-consuming task. The quality dimensions 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 dimensions. 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.

In table 3 a full list of quality dimensions regarding software implementation and evaluation, methods of measurement and metrics is given.

Dimension	**Methods of measurement**	**Metrics**
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	number of occasions where the software was unable to re-

	affected in the case of a failure and on the time and effort needed for it	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
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 parts to be modified	man-hours needed for diagnosis of deficiencies or causes of failures, or for identifications of 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

Table 3

5.2.3 Methods of measurement and Metrics for the Data Loading Quality Dimensions

The process of data loading, as already mentioned, is basically related to *analyzability* and *transactional availability*. The *analyzability* dimension, dealing with the performance of a process in erroneous situations 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.

In table 4 a full list of quality dimensions regarding data loading, methods of measurement and metrics is given.

Dimension	**Methods of measurement**	**Metrics**
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

Table 4

5.2.4 Data Usage Methods of measurement and Metrics

The data usage processes, as already mentioned, are basically related to *accessibility* and *usefulness*. 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.

In table 5 a full list of quality dimensions regarding data usage, methods of measurement and metrics is given.

Dimension	**Methods of measurement**	**Metrics**
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	documentation of the data purging rules	number of undocumented purging

		rules
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		
	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

Table 5

5.2.5 Data Quality Methods of measurement and Metrics

The quality of data is characterized by five major dimensions: completeness, credibility, accuracy, consistency and data interpretability. The methods of measurement, as well as the metrics for these dimensions are listed in table 6.

Dimension	**Methods of measurement**	**Metrics**
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

Table 6

5.3 Data Warehouse Users and Data Warehouse Quality

In the previous paragraphs, we described the way quality dimensions are structured, related to data warehouse components, and measured. In the rest of this chapter, we are going to deal with data warehouse quality issues from a different perspective: we will describe the way various groups of people are affected from the existence or not of a data warehouse. Furthermore, we will describe the relationship between the users of a data warehouse and the data warehouse quality, i.e. in what kind of quality a person related to a data warehouse system may be interested.

We have identified the following *roles* 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 an OLAP query tool to get answers interesting to him. A decision maker is usually concerned with the *quality of the stored data*, in the degree that they represent real world facts. The *timeliness* of the data is important to the decision maker, since he needs "fresh" data in order to come up with correct decisions. Furthermore, since he is a high level user, he must be aided in order to come up with a correct query with a friendly interface, consequently he is interested in *ease of querying* which is reflected in the quality dimensions schema quality and software quality.

The *Data Warehouse Administrator* is concerned with keeping the data warehouse properly operating. To do his job more easily, a data warehouse administrator needs facilities like *error reporting* (in the case of failure of operations, e.g. data loading). He also needs *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* is involved in a continuous process which is the data warehouse design (usually done incrementally). The data warehouse designer has to measure the *quality of the schemata* he produces, in order to ensure that the design was the proper one. This is why he makes most use of the information directory, modeling the data warehouse. As a consequence the data warehouse designer needs *metadata quality* as well. Furthermore, the data warehouse designer is also the evaluator of purchased components, so he needs *software evaluation standards* to test the software packages we considers purchasing.

The *Programmers of Data Warehouse Components* are the people who develop the actual data warehouse applications. As developers, they can make good use of *software implementation standards*. They also need the means - most probably through the same standards - to *evaluate* their work. *Metadata reporting* can also facilitate their job, since they can avoid mistakes related to schema information.

The *Source Data Users/Administrators/Designers* are affected from the data warehouse in the sense that they get *feedback for the quality of their data*, stored in the source data stores. Even if they decide to ignore it (as the usual practice seems to be) data quality reporting could be very useful to them. In addition, they have to take care of the *transactional availability* of their OLTP systems, so they are interested in efficient update programs and data loaders for the data warehouse. In some data warehouse applications, the source data administrators have to provide their data and schema information for the data warehouse.

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

6 Examples

In this chapter, we will present an example data warehouse architecture represented by the proposed reference architecture. This is not an full example, since modeling a complete data warehouse architecture is beyond the scope of this paper. We emphasize on the steps necessary to express a data warehouse in our data warehouse modeling language and we show which results can be taken from the analysis of the data warehouse architecture.

First, we show informally the architecture of the data warehouse. We take as an example the data warehouse of a supermarket company. The information about this kind of data warehouse was taken from the discussion with another data warehousing project (HP-Retail), which deals with data warehouses for medium-sized retail companies in France, Germany and the Netherlands.

The scenario is as follows: a retail company has many branches distributed over the whole country. The goal of the data warehousing project in the retail company is to have a integrated view of products, finances and employees in all branches. Because of the huge amount of data arising at the individual branches, it is not useful to integrate the detailed data in the central data warehouse. The data is first integrated and aggregated in each branch in a local data warehouse from two source database. The integration in the central data warehouse takes as input the result of this preprocessing step, so that the amount of data transferred between the branches and location of the central data warehouse is lower, and communication costs are reduced.

To have a reasonable performance for queries to the data warehouse, the warehouse delivers data to external data marts. These data marts summarize the data relevant for a specific department. For example, the financial department is only interested in the turnover, taxes and the debts and the deposit of the company. The marketing department is focused on product data, since they want to have the sale rates of products by category or brand, and week, month or year. Finally, the staff department is interested in data of employees, how much do they work, where do they work, etc. Figure 17 gives an overview of the architecture of this data warehouse. The figure contains for simplicity only two local data warehouse, but there can be more.

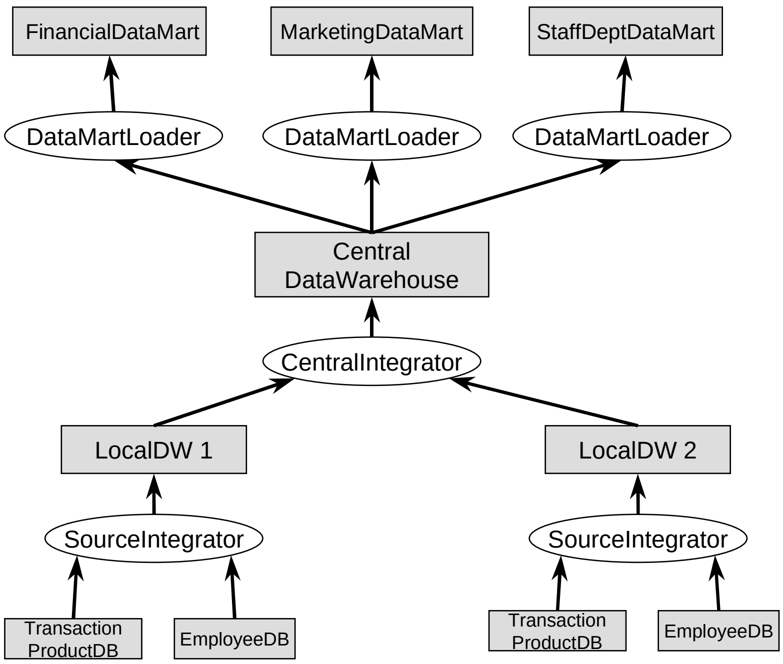

Figure 17 Informal architecture of example data warehouse for a retail company

When we start to describe the architecture of this data warehouse in our model, we must first distinguish between the source, data warehouse and client level. Because of the multi-tiered architecture of this data warehouse, this is not an obvious task. The data warehouse consists of four levels. Our approach is to decompose the data warehouse level in our reference model into two levels: the central data warehouse level and the level of the local data warehouses.

After the separation of the different levels of the data warehouse architecture, we model the conceptual perspective. The enterprise model builds the central point of the conceptual view. First, we have to identify the concepts in the enterprise model:

- *Employee* is a concept class and an atomic concept
- *Product* is a concept class and an atomic concept
- *Transaction* is a relationship between the concepts Date and Product
- *Date* is a concept domain and an atomic concept

The model of the source database *TransactionProductDB* has the concepts *Transaction*, *Product* and *Date*, whereas the database *EmployeeDB* has only the concept *Employee*. We will not go into details for problems of source integration here, so we assume that the union of the concepts in the two source databases correspond to the concepts in the enterprise model. The same holds for the client models. The model of the *FinancialDataMart* consists of the concepts *Transaction*, *Product* and *Date*. The data mart of the marketing department knows the same concepts, but as we will see later, the mapping to the logical perspective is different. The staff department is only interested in employees, therefore the model of the data mart contains only the concept *Employee*. Figure 18 shows an overview of the conceptual perspective.

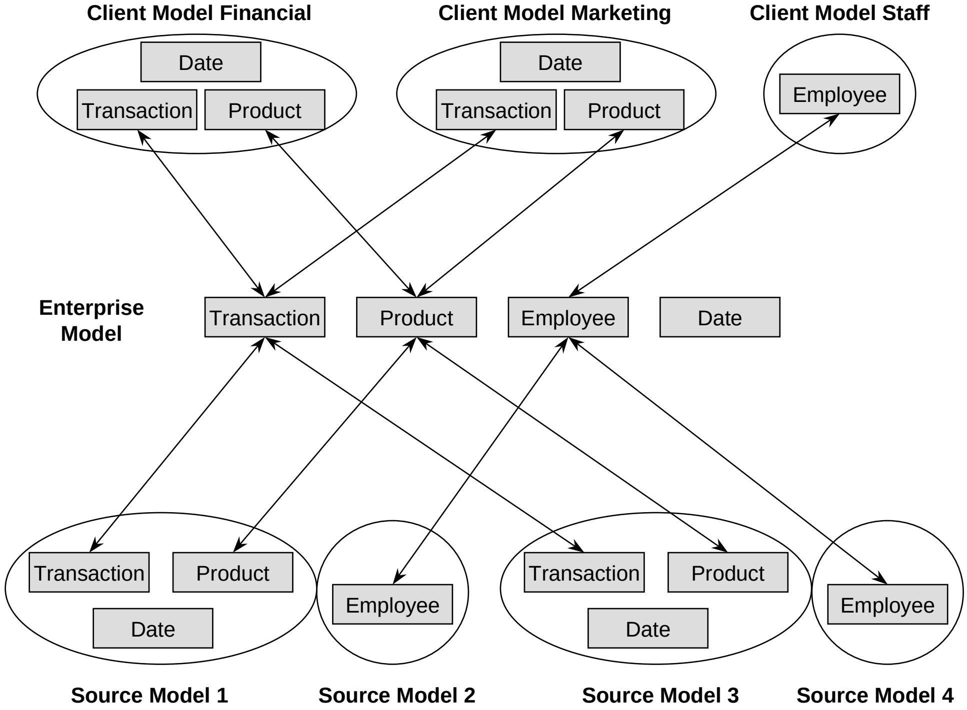

Figure 18 Conceptual Perspective

The logical perspective represents the conceptual models as relational schemas. Each concept is translated into a relation. The relationship to the conceptual perspective is shown in the architecture model by the definition of the conjunctive queries for the relations. The relation *Employee* has the fields *social_security_number* (integer), *name* (string), *salary* (integer) and *worksFor* (string). The *worksFor*-field contains a description of the department and the branch the employee works for. The relation for the concept *Product* has the fields *product_id* (integer), *name* (string), *price* (real), *brand* (string) and *category* (string).

The *Transaction* relation has three fields: *quantity*, *product_id* and date. The quantity the number of sold items in the transaction. The *product_id* is a foreign key to the relation Product, it contains a product identifier and is of the domain integer. The date field contains the date and the time of the transaction as a string. Because of this simple representation of the time concept, it is not necessary to have a relation for the concept *Date*.

The relational schemas of the sources contain the respective relations of the concepts in the source model. The schema of the source *TransactionProductDB* contains the relations *Transaction* and *Product*, and the schema of the source *EmployeeDB* contains the relation *Employee*.

The relations in the client schemas are de-normalized views on the relations of the data warehouse schema. The schema of the data mart in the financial department contains only one relation *FinancialTransaction*. This relation contains only the field *amount* and *date*, because the financial department is not interested in details of products. However, this relation is defined by a conjunctive query over the concepts *Product*, *Transaction* and *Date*. The marketing data mart contains only the relation *MarketingProduct*, which is a view on the relation *Product* and *Transaction* of the data warehouse schema. The relation *Employee*, in the staff department is an exact copy of the relation in data warehouse.

In the next step, we have to model the physical aspects of the data warehouse architecture, such as data delivery relationships and location of data stores. An overview of the physical implementation of the data warehouse architecture is shown in figure 19, which is a screenshot from the ConceptBase GraphBrowser. The bold arrows mark a data delivery relationship, the thin arrows represent a *hasPart* relationship between components. Agents are drawn as oval nodes, data stores as small buckets and all other data warehouse components as rectangular boxes.

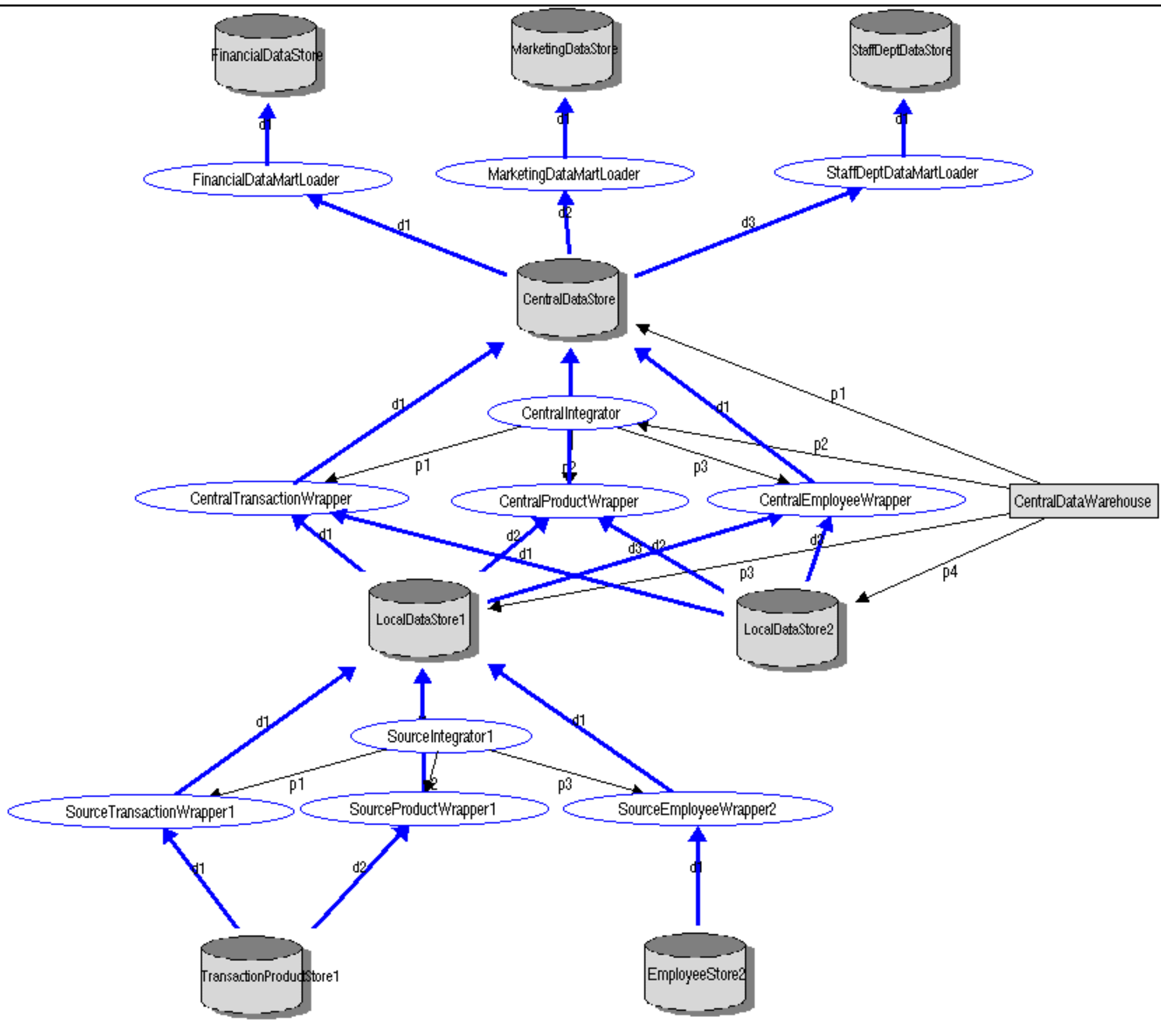

Figure 19 Physical Perspective of the data warehouse (Screenshot of the ConceptBase GraphBrowser)

The *CentralDataWarehouse* builds the main component on the data warehouse level and is composed out of the components *CentralDataStore*, *CentralIntegrator*, *LocalDataStore1* and *LocalDataStore2*. The *CentralIntegrator* is again composed out of the transportation agents *CentralTransactionWrapper*, *CentralProductWrapper* and *CentralEmployeeWrapper*. These agents read the corresponding relations from the local data stores (1 and 2) and store the result in the central data store. The local data stores get their data from the source databases with the help of *SourceIntegrators*. We show in the figure only the SourceIntegrator for the first local data store, the architecture for the second data store is equivalent.

The source integrator is composed out of the parts *SourceTransactionWrapper1*, *SourceProductWrapper1* and *SourceEmployeeWrapper2*. The transaction and product wrappers read the relations from the *TransactionProductDataStore1*, the employee wrapper reads its relation from the *EmployeeDataStore2*. All wrappers deliver their data to the local data store.

The data marts of the departments have also data stores. The data marts are loaded and updated by *DataMartLoaders*, one for each department. Because the data stores are located at the client level, they are instances of the class client data store. The connection to the logical perspective is not shown in this figure. Each data store has its relational schema and stores relations. Most of the components deliver data from some relations to another component. Furthermore, for simplicity we skipped the modeling of some administration tools, like a control agent or the meta database itself. The control agent invokes the wrappers for loading and updating at a certain time, whereas the meta database retrieves and delivers meta information to all other components.

Finally, we have to measure the quality of the components. Most quality indicators in the quality model deal with the physical implementation of the data warehouse. We concentrate on one quality dimension, namely *Currency*. We attach a currency quality indicator to all physical components dealing with an employee relation, i.e. *EmployeeStore*, *SourceEmployeeWrapper*, *LocalDataStore*, *CentralDataStore*, *StaffDeptDataMartLoader* and *StaffDeptDataStore*. We take as a metric for this dimension the average age of the tuples in this relation in days. At source level, we assume that the quality of the information provided by the data entry clerks is within acceptable boundaries, e.g. the data is entered at the latest two days after the change occurred. The quality measures on the following levels can be computed out of the quality of the previous levels and by comparing the timestamps of the target databases and source databases. Because of delays in the update process the data becomes older when it is transferred from one level to the next. Figure 20 shows the physical components of one data chain with the attached quality indicators.

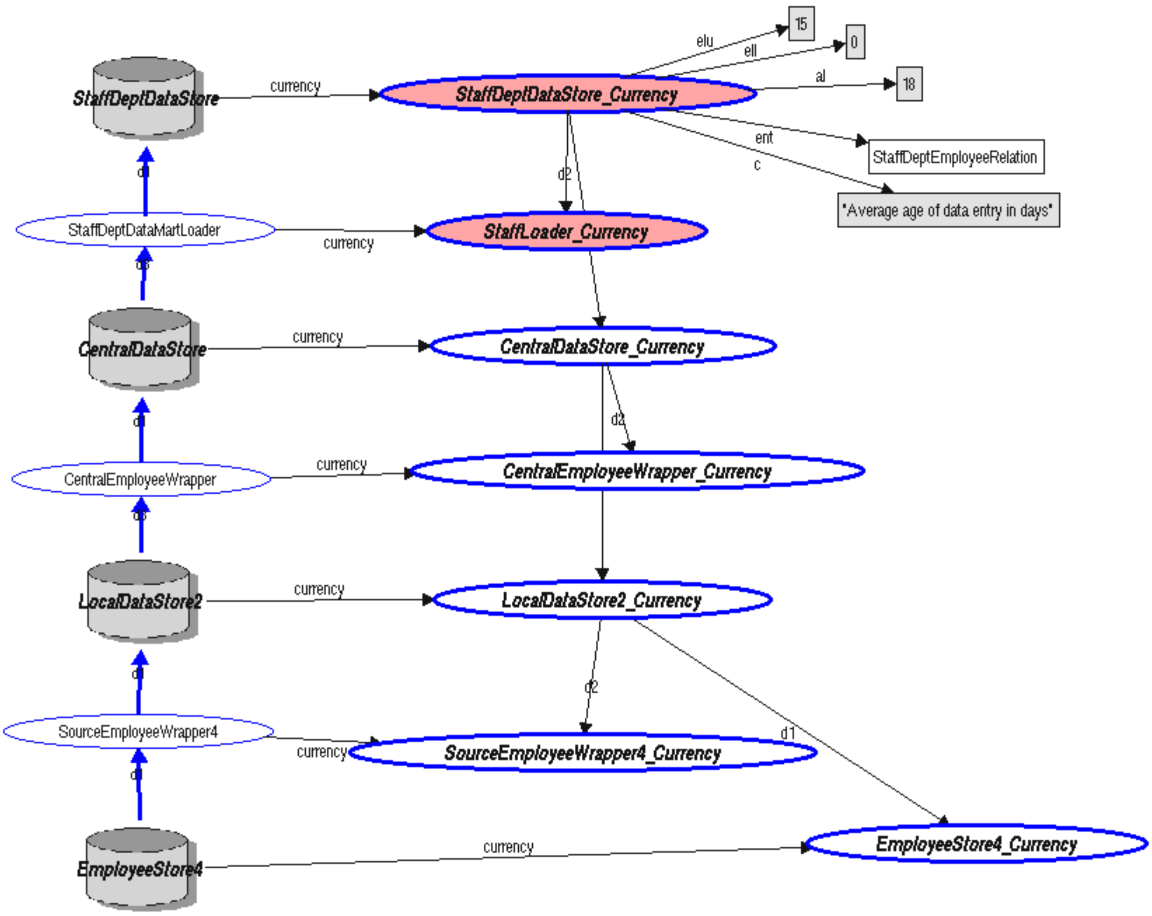

Figure 20 Physical components with attached quality indicators

Let us suppose, that due to a bad quality in the loader of the data mart of the staff department, the data in this data mart is not current. The graphical browser of ConceptBase shows bad quality indicator as red oval nodes, since it is able to display objects in different graphical types (different shapes and colors), depending on their properties. The rules for the graphical types are specified as logical formulas in the same way as other rules are specified in ConceptBase.

The data warehouse designer is now able to trace back to the reason of the bad quality. If a decision maker claims about not current information in the data mart of the staff department, the data warehouse administrator can use either the graphical browser or one of the queries proposed in section 7.3 to find the cause of the bad quality. Furthermore, the administrator can see, if any other components are affected by this component and the quality of the components change, if the quality of the one component is improved.

7 Telos Models

This chapter contains the definitions of the classes in the architecture and quality model in the Telos language. In chapter 4 and 5, we presented the definitions of the models as graphs, now we will present it more formally as text. Furthermore, we show the constraints of the models. In section 7.3, we will explain in detail the queries for the architecture and quality model. This shows also the expressiveness of the Telos language.

7.1 Telos Models of the Architecture Model

The class *DW_Object* is the base class for all objects in the data warehouse. The attribute *classifiedBy* establishes the link to the quality model.

```
Class DW_Object with
attribute
       classifiedBy : QualityIndicator
end
```

The classes *ConceptualObject*, *LogicalObject* and *PhysicalObject* represent the three levels of figure 3. The classes of the three perspectives will be a subclass of one of these classes.

```
Class ConceptualObject isA DW_Object end
Class LogicalObject isA DW_Object end
Class PhysicalObject isA DW_Object end
```

The classes *Source* and *Client* relate the components in the three perspectives to one object in the data warehouse architecture as described in section 4.4.

```
Class Source isA DW_Object end
Class Client isA DW_Object end
```

7.1.1 Conceptual Perspective

A *Model* represents the concepts, that are used in the conceptual perspective of one data warehouse object. It has three subclasses corresponding to the three levels of a data warehouse: *ClientModel*, *EnterpriseModel* and *SourceModel*. Client- and SourceModel are related the corresponding Client respectively Source instance of the data warehouse architecture.

```
Class Model isA ConceptualObject with
attribute
       hasConcept : Concept
end

Class EnterpriseModel isA Model end
Class ClientModel isA Model with
attribute
       isModelOf : Client
end
Class SourceModel isA Model with
attribute
       isModelOf : Source
end
```

A *Concept* is intentionally or extensionally included in other Concepts (most times ComplexConcepts). This is expressed by the attributes *intIncludedIn* and *extIncludedIn*. Furthermore, it may be subsumed or not subsumed by other concepts (attributes *isSubsumedBy* and *isNotSubsumedBy*). Subsumption is tested by an external reasoner, which reads the axioms from ConceptBase and stores the result of the subsumption test in ConceptBase. When the axioms change, the affected subsumption attributes must be deleted by the client tool. Intentionally axioms are only useful as intermodel axioms, which is guaranteed by the constraints *interModelAxioms* and *interModelAxiomsWithComplexConcept*. The first constraint says that, if a concept *c1* is intentionally included in a concept *c2*, and *m1* is the model of *c1* and *m2* is the model of *c2*, then the models *m1* and *m2* are not allowed to be identical. The second constraint states the indirect relationship between concepts via a complex concept definition: the concept *c1* is included in a complex concept *cc* and *cc* contains syntactically the concept *c2*.

```
Class Concept isA ConceptualObject with
attribute
       intIncludedIn : Concept;
       extIncludedIn : Concept;
       isSubsumedBy : Concept;
       isNotSubsumedBy : Concept
constraint
       interModelAxioms :
       $ forall c1,c2/Concept m1,m2/Model ((c1 intIncludedIn c2)
         and (m1 hasConcept c1) and (m2 hasConcept c2))
         ==> (not IDENTICAL(m1,m2)) $;
       interModelAxiomsWithComplexConcept :
       $ forall c1,c2/Concept cc/ComplexConcept m1,m2/Model
         ((c1 intIncludedIn cc) and (cc containsSyntactically c2)
         and (m1 hasConcept c1) and (m2 hasConcept c2))
```

```
        ==> (not IDENTICAL(m1,m2)) $
end
```

The class Concept has several subclasses to express the axioms of the conceptual models in the architecture model. *AtomicConcepts* are just names, a *ConceptDomain* is something like integer or string and a *ConceptClass* is a simple concept.

```
Class AtomicConcept isA Concept end
Class ConceptDomain isA Concept end
Class ConceptClass isA Concept end
```

Furthermore, a *ComplexConcept* is a expression in description logics describing the relationship between concepts. This object has a link to the concepts which appear syntactically in this expression. The expression is simply stored as a string, the parsing is done by the client tool.

```
Class ComplexConcept isA Concept with
attribute
      containsSyntactically : Concept
end
```

In addition, a *Relationship* is a n-ary Relation between Concepts, which has the *RelationshipType* as an attribute. A RelationshipType may be consist out of several concepts, but only instances of ConceptClass or ConceptDomain are allowed as a field of a RelationshipType. This restriction is expressed by the constraint *allowedFieldsOfRelationshipTypes*: if a concept *c* is a field of the relationship type *r*, then it must be an instance of *ConceptClass* or *ConceptDomain*.

```
Class Relationship isA Concept with
attribute
      type : RelationshipType
constraint
      allowedFieldsOfRelationshipTypes :
      $ forall r/RelationshipType c/Concept (r field c)
        ==> ((c in ConceptClass) or (c in ConceptDomain)) $
end

Class RelationshipType isA ConceptualObject with
attribute
      field : Concept
end
```

7.1.2 Logical Perspective

This section contains the Telos definition for the classes of the logical perspective in the data warehouse architecture. The base class is *Schema*, which consists of a set of *Types*. The relationship to the physical perspective is established through the attribute *storedIn*, which relates a schema with a DataStore.

```
Class Schema isA LogicalObject with
attribute
      hasType : Type;
      storedIn : DataStore
end
```

A *Type* is complex object, which is composed out of other Types.

```
Class Type isA LogicalObject with
  attribute
    field : Type
end
```

The class Schema has three different subclasses for the three levels of the architecture model. The subclasses are *SourceSchema*, *DWSchema* and *ClientSchema*. A Source- or a ClientSchema is related with the corresponding instance of *Source* or *Client* in the data warehouse architecture.

```
Class SourceSchema isA Schema with
attribute
      isSchemaOf : Source
end
```

```
Class DWSchema isA Schema end
Class ClientSchema isA Schema with
attribute
       isSchemaOf : Client
end
```

The following classes represent a relational schema. The classes are implemented as subclasses of Schema or Type. Therefore, they specialize also the attributes of these classes, e.g. a *RelationalSchema* has as types only *Relations*, and a *Relation* may only have a *Domain* as field.

```
Class RelationalSchema isA Schema with
attribute
       hasType : Relation;
       storedIn : RelationalDatabase
end

Class Relation isA Type with
attribute
       field : Domain
end

Class Domain isA Type end
```

CQRelation is a Relation defined by a conjunctive query. The attribute *containsSyntactically* refers to the concepts appearing syntactically in the expression of the conjunctive query. For the subsumption attribute applies the same rule as for the subsumption attribute of concepts in the conceptual perspective. The conjunctive query may only contain concepts which are defined in the corresponding model of the source or client. This restriction is implemented by the constraints *onlyConceptOfCorrespondingModel(1-4)*. If a conjunctive query *cqr* contains the concept *c*, and the model of *c* is *m*, and the schema of *cqr* is *s*, and *m* is the model of the source *o*, then *s* must be the schema of *o*, and vice versa. The same constraints are needed for the client level of the architecture. We need no constraints for the data warehouse level, because the error will be detected by one of the constraints.

```
Class CQRelation isA Relation with
attribute
       containsSyntactically : Concept;
       isSubsumedBy : CQRelation;
       isNotSubsumedBy : CQRelation
constraint
  onlyConceptOfCorrespondingModel1:
       $ forall cqr/CQRelation c/Concept s/SourceSchema m/SourceModel o/Source
       ((cqr containsSyntactically c) and (m hasConcept c) and (s hasType cqr)
       and (m isModelOf o)) ==> (s isSchemaOf o) $;
  onlyConceptOfCorrespondingModel2:
       $ forall cqr/CQRelation c/Concept s/SourceSchema m/SourceModel o/Source
       ((cqr containsSyntactically c) and (m hasConcept c) and (s hasType cqr)
       and (s isSchemaOf o)) ==> (m isModelOf o) $;
  onlyConceptOfCorrespondingModel3:
       $ forall cqr/CQRelation c/Concept s/ClientSchema m/ClientModel o/Client
       ((cqr containsSyntactically c) and (m hasConcept c) and (s hasType cqr)
       and (m isModelOf o)) ==> (s isSchemaOf o) $;
  onlyConceptOfCorrespondingModel4:
       $ forall cqr/CQRelation c/Concept s/ClientSchema m/ClientModel o/Client
       ((cqr containsSyntactically c) and (m hasConcept c) and (s hasType cqr)
       and (s isSchemaOf o)) ==> (m isModelOf o) $
end
```

7.1.3 Physical Perspective

DW_Component is the super-class of the physical objects in the data warehouse architecture. Components are programs or (database) systems, which store, deliver or read data from other components. They run on a host (a *Computer* system) and may be composed out of other components (attribute *hasPart*). The attribute *hasSchema* implements the link to a logical schema.

```
Class DW_Component isA PhysicalObject with
  attribute
       hasPart : DW_Component;
       deliversTo : DW_Component;
       hasSchema: Schema;
```

```
       host : Computer
end
```

An *Agent* is a program, which does not store data, it just delivers the data to another component. It may have a small buffer for caching of data, but the main difference from a data store is that it does not store this data persistently on disk for a long time. An Agent may *notify* another agent about certain events which occurred. It has two subclasses, namely *ControlAgent* and *TransportationAgent*. A ControlAgent is a program for the administration of a data warehouse, whereas a TransportationAgent transports data physically from one DW_Component to another.

```
Class Agent isA DW_Component with
  attribute
       notifies : Agent
end

Class ControlAgent isA Agent end
Class TransportationAgent isA Agent end
```

A *DataStore* is a system which stores data persistently for a long time on a disk or another kind of hardware, e.g. a database system. It has three subclasses corresponding to three levels in the data warehouse architecture: *ClientDataStore*, *DW_DataStore* and *SourceDataStore*. Source- and ClientDataStore refer to the corresponding Client or Source object.

```
Class DataStore isA DW_Component end
Class ClientDataStore isA DataStore with
attribute
       isStoreOf : Client
end
Class SourceDataStore isA DataStore with
attribute
       isStoreOf : Source
end
Class DW_DataStore isA DataStore end
```

The implementation of the relational schema in the logical perspective was done straightforward by specializing the corresponding classes. We do the implementation in the physical perspective in the same way. A *RelationalDBMS* is declared as a subclass of DW_Component. It is not a subclass of *DataStore*, because it is composed out of *RelationalDatabases*, which are subclasses of DataStore. The link to a relational schema is implemented on the one hand by the *hasRelation* attribute of RelationalDatabase, and on the other hand, by the attribute *supportedDomain* of RelationalDBMS. A *RelationalWrapper* is implemented as specialization of TransportationAgent. The special property of a RelationalWrapper is that it just delivers a relation. The constraint *supportedDomainConstraint* of RelationalDatabase ensures that the domains of fields of a relation in a database are only domains, which are supported by the corresponding relational database management system.

```
Class RelationalDBMS isA DW_Component with
attribute
       supportedDomain : Domain;
       hasPart : RelationalDatabase
end

Class RelationalDatabase isA DataStore with
attribute
       hasRelation : Relation
constraint
       supportedDomainConstraint :
       $ forall d/Domain rdbms/RelationalDBMS
         (exists rdb/RelationalDatabase r/Relation
         (rdb hasRelation r) and (r field d) and (rdbms hasPart rdb))
         ==> (rdbms supportedDomain d) $
end

Class RelationalWrapper isA TransportationAgent with
attribute
       delivers : Relation
end
```

The attribute *deliversTo* of DW_Component refers to a type which is a delivered by the corresponding component and a *DeliveryMode*. The DeliveryMode describes the mode of delivery, for example, *StreamWise* if there is a continuous stream of data, or *RecordWise* if the data is delivered tuple by tuple.

```
DW_Component!deliversTo with
  attribute
       type : Type;
       mode : DeliveryMode
end

Class DeliveryMode end
StreamWise in DeliveryMode end
RecordWise in DeliveryMode end
```

The notification by one agent to another agent includes sending of data of a specific type. For example, if an agent reports an error, the notification will probably have the type *String*.

```
Agent!notifies with
  attribute
       type : Type
end
```

The class *Computer* is a physical object. Currently, it has only two attributes. The IP number says something about the location of the computer. The operating system of computer reveals about the quality of the system, e.g. if it supports multi-tasking, multi-user, etc.

```
Class Computer isA PhysicalObject with
attribute
       ipNumber : String;
       operatingSystem : String
end
```

7.2 Telos Models of the Quality Model

We presented the base class QualityIndicator for the quality model already in chapter 5. In this section, we will show the formal representation of the quality model in Telos. *QualityMeasure* is the domain of quality indicators. QualityMeasure has to be in SimpleClass because the builtin classes Integer, String, Real and Boolean are defined as specializations of QualityMeasure, and the builtin classes are simple classes, too. The expressiveness of Telos allows that one can define super-classes even for very basic classes, such as Integer, String, etc.

```
SimpleClass QualityMeasure in Class end
Integer isA QualityMeasure end
Real isA QualityMeasure end
String isA QualityMeasure end
Boolean isA QualityMeasure end
```

Furthermore, we define *constraints* which ensure that if an expected_level_lower attribute exists, there must be an expected_level_upper attribute and vice versa and that the priority of a quality indicator is normalized between 0 and 1, for homogeneity reasons.

```
QualityIndicator with
constraint
             lower_upper_constraint:
                    $ forall q/QualityIndicator ((exists qm1/QualityMeasure
                           (q expected_level_lower qm1))
                           ==> (exists qm2/QualityMeasure
                                  (q expected_level_upper qm2))) $;
             upper_lower_constraint:
                    $ forall q/QualityIndicator ((exists qm1/QualityMeasure
                           (q expected_level_upper qm1))
                           ==> (exists qm2/QualityMeasure
                                  (q expected_level_lower qm2))) $;
             priority_constraint:
                    $ forall q/QualityIndicator r/Real ((q priority r)
                           ==> ((r >= 0.0) and (r <= 1.0) )) $
end
```

7.2.1 Design and Administration Quality

All quality indicators relevant to the design and administration tasks, are referring to logical and conceptual objects. Each one of the quality dimensions, described in the previous sections is modeled in Telos. The reader might notice that we define all expected and achieved levels as boolean for the design and administration quality dimensions. In section 5.2.1 the metrics for these dimensions were supposed to be numerical. This difference is due to the fact that in section 5.2.1 we try to measure the overall quality of the data warehouse environment, whereas in the model each quality indicator is related to a specific object. Consequently, in the model we represent the existence or not of the required quality dimension in a specific entity and it is up to the measurement policy to calculate the numerical value for all the data warehouse (e.g. calculate an average value as percentage). The full specifications are following:

The class *Correctness* checks whether real-world entities, source schema entities and user needs have been properly mapped in the relevant schema.

```
Class Correctness isA Dgn_AdmQIndicator with
      attribute, single
            expected_level_upper : Boolean;
            expected_level_lower : Boolean;
            achieved_level : Boolean
       attribute
            relevant_user_req : String;
            reason_for_non_correspodence : String;
            other_related_entities: DW_Object
end
```

The class *Schema_Completeness* checks whether any crucial information is missing.

```
Class Schema_Completeness isA Dgn_AdmQIndicator with
      attribute, single
            expected_level_upper : Boolean;
            expected_level_lower : Boolean;
            achieved_level : Boolean
      attribute
            missing_info : String
end
```

Instances of the class *Minimality* deal with undesired redundancy.

```
Class Minimality isA Dgn_AdmQIndicator with
      attribute, single
            expected_level_upper : Boolean;
            expected_level_lower : Boolean;
            achieved_level : Boolean
      attribute
            relevant_dw_components : DW_Object
end
```

The class *Traceability* checks whether all user/designer/administrator requirements are traceable.

```
Class Traceability isA Dgn_AdmQIndicator with
      attribute, single
            expected_level_upper : Boolean;
            expected_level_lower : Boolean;
            achieved_level : Boolean
      attribute
            relevant_user_req: String
end
```

Class *Interpretability* checks to ensure whether a component of a schema/model is commented properly.

```
Class Interpretability isA Dgn_AdmQIndicator with
      attribute, single
            expected_level_upper : Boolean;
            expected_level_lower : Boolean;
            achieved_level : Boolean
      attribute
            comment : String
end
```

Schema quality is related to the Source Integration Process. It characterizes the quality of a schema, as it is produced from the source interation point of view. Obviously, it is computed from its attributes, yet we leave the computation method for the DW designer.

```
Class Schema_Quality isA Dgn_AdmQIndicator with
	attribute, single
		expected_level_upper : Integer;
		expected_level_lower : Integer;
		achieved_level : Integer
	attribute
		correctness : Correctness;
		schema_completeness : Schema_Completeness;
		minimality: Minimality;
		traceability: Traceability;
		interpretability: Interpretability
end
```

Class *Metadata_Evolution* describes the changes which have happened at the DW metadata in the ConceptBase information directory itself.

```
Class Metadata_Evolution isA Dgn_AdmQIndicator with
	attribute, single
		expected_level_upper : Boolean;
		expected_level_lower : Boolean;
		achieved_level : Boolean
	attribute
		old_component : DW_Object;
		new_component  : DW_Object;
		reason_of_change : String
end
```

Class *Design_Administration_Quality* is the top level class, describing the way the overall design and administration process can be related to quality issues.

```
Class Design_Administration_Quality isA Dgn_AdmQIndicator with
	attribute, single
		expected_level_upper : Integer;
		expected_level_lower : Integer;
		achieved_level : Integer
	attribute
		schema_quality : Schema_Quality;
		metadata_evolution : Metadata_Evolution
end
```

7.2.2 Software Evaluation/Implementation Quality

The Software evaluation/implementation quality dimensions are characterized by the fact that they relate to pieces of software , which is modeled through the class *DW_Component* in the architecture. So the attribute *related_objects* must be refined to the respective class. All the other software quality dimensions inherit from the parent class *Software_Quality_Indicator*.

```
Class Software_Quality_Indicator isA QualityIndicator with
	attribute, single
		expected_level_upper : Integer;
		expected_level_lower : Integer;
		achieved_level : Integer
	attribute
		related_objects : DW_Component
end

Class Software_Suitability isA Software_Quality_Indicator end
Class Software_Accuracy isA Software_Quality_Indicator end
Class Software_Interoperability isA Software_Quality_Indicator end
Class Software_Compliance isA Software_Quality_Indicator end
Class Software_Security isA Software_Quality_Indicator end
Class Software_Functionality isA Software_Quality_Indicator with
	attribute
		suitability : Software_Suitability;
		accuracy : Software_Accuracy;
```

```
                interoperability: Software_Interoperability;
                compliance : Software_Compliance;
                security : Software_Security
end

Class Software_Maturity isA Software_Quality_Indicator end
Class Software_Fault_Tolerance isA Software_Quality_Indicator end
Class Software_Recoverability isA Software_Quality_Indicator end
Class Software_Reliability isA Software_Quality_Indicator with
        attribute
                maturity : Software_Maturity;
                fault_tolerance : Software_Fault_Tolerance;
                recoverability : Software_Recoverability
end

Class Software_Understandability isA Software_Quality_Indicator end
Class Software_Learnability isA Software_Quality_Indicator end
Class Software_Operability isA Software_Quality_Indicator end
Class Software_Usability isA Software_Quality_Indicator with
        attribute
                understandability : Software_Understandability;
                learnability : Software_Learnability;
                operability : Software_Operability
end

Class Software_Time_Behavior isA Software_Quality_Indicator end
Class Software_Resource_Behavior isA Software_Quality_Indicator end
Class Software_Efficiency isA Software_Quality_Indicator with
        attribute
                time_behavior : Software_Time_Behavior;
                resource_behavior : Software_Resource_Behavior
end

Class Software_Analyzability isA Software_Quality_Indicator end
Class Software_Changeability isA Software_Quality_Indicator end
Class Software_Stability isA Software_Quality_Indicator end
Class Software_Testability isA Software_Quality_Indicator end
Class Software_Maintainability isA Software_Quality_Indicator with
        attribute
                analyzability : Software_Analyzability;
                changeability : Software_Changeability;
                stability : Software_Stability;
                testability : Software_Testability
end

Class Software_Adaptability isA Software_Quality_Indicator end
Class Software_Installability isA Software_Quality_Indicator end
Class Software_Conformance isA Software_Quality_Indicator end
Class Software_Replaceability isA Software_Quality_Indicator end
Class Software_Portability isA Software_Quality_Indicator with
        attribute
                adaptability : Software_Adaptability;
                installability : Software_Installability;
                conformance : Software_Conformance;
                replaceability : Software_Replaceability
end

Class Software_Implementation_Efficiency isA Software_Quality_Indicator end
Class Software_Quality isA Software_Quality_Indicator with
        attribute
                functionality : Software_Functionality;
                reliability : Software_Reliability;
                usability : Software_Usability;
                software_efficiency: Software_Efficiency;
                maintenability: Software_Maintainability;
                portability: Software_Portability;
                implementation_efficiency: Software_Implementation_Efficiency
end
```

7.2.3 Data Loading Quality

The data loading quality dimensions are characterized by the fact that they relate to physical objects, such as data stores and agents, which are generally modeled through the class *PhysicalObject* in the architecture. So the attribute *related_objects* must be refined to the respective class. All the other data loading quality dimensions inherit from the parent class *Data_Loading_QIndicator*.

```
Class Data_Loading_QIndicator isA QualityIndicator with
       attribute
              related_objects: PhysicalObject
end
```

The class *Analyzability* deals with the validation of each process and the interaction with the user. It can refer to both agents and data stores. We define the measure of analyzability as boolean, whereas in section 5.2.3 the metrics for these dimension was supposed to be numerical. The difference lies in the fact that the metrics try to measure the overall quality of the data warehouse environment, whereas in the model an analyzability quality indicator is related to a specific physical object.

```
Class Analyzability isA Data_Loading_QIndicator with
       attribute, single
              expected_level_upper : Boolean;
              expected_level_lower : Boolean;
              achieved_level : Boolean
end
```

Transactional_Availability is the percentage of time data is not write locked. It refers to *data stores*, so the *related_objects* attribute is subsequently refined.

```
Class Transactional_Availability isA Data_Loading_QIndicator with
       attribute, single
              expected_level_upper : Integer;
              expected_level_lower : Integer;
              achieved_level : Integer
       attribute
              related_objects : DataStore
end
```

The class *Data_Loading_Quality* can model the quality of a whole process. i.e. having a data store and a loader as *related_objects* and recording the overall performance.

```
Class Data_Loading_Quality isA Data_Loading_QIndicator with
       attribute, single
              expected_level_upper : Integer;
              expected_level_lower : Integer;
              achieved_level : Integer
       attribute
              analyzability : Analyzability;
              transactional_availability : Transactional_Availability
end
```

7.2.4 Data Usage Quality

The data usage quality dimensions are characterized by the fact that they relate to physical objects, such as data stores and agents, which are generally modeled through the class *PhysicalObject* in the architecture. So the attribute *related_objects* must be refined to the respective class. All the other data loading quality dimensions inherit from the parent class *Data_Usage_QIndicator*.

```
Class Data_Usage_QIndicator isA QualityIndicator with
       attribute
              related_objects: PhysicalObject
end
```

System_availability is the percentage of time the system is up.

```
Class System_Availability isA Data_Usage_QIndicator with
       attribute, single
              expected_level_upper : Integer;
              expected_level_lower : Integer;
              achieved_level : Integer
end
```

The class *Security* describes the privileges each user has, for each *DataStore*. We define the metric of the security quality dimension as boolean. In section 5.2.4 we proposed the metric to be numerical. The difference is due to the fact, that the proposed measurement deals with the overall quality of the data warehouse, whereas the quality indicator in this model is related to a physical object.

```
Class Security isA Data_Usage_QIndicator with
      attribute, single
            expected_level : Boolean;
            achieved_level : Boolean
      attribute
            relevant_user : String;
            mode : String
end
```

The *Accessibility* of a system (computer, data store, etc.) is defined from the *system* and *transactional availability* and the *security* characteristics.

```
Class Accessibility isA Data_Usage_QIndicator with
      attribute, single
            expected_level_upper : Integer;
            expected_level_lower : Integer;
            achieved_level : Integer
      attribute
            system_availability : System_Availability;
            transactional_availability : Transactional_Availability;
            security : Security
end
```

Currency describes when data were entered in the *DataStore*. The currency can be defined in terms of data stores, modeled through the inherited *related_objects* attribute, or in terms of entities (e.g. Relations) modeled through the *relevant_entities* attribute.

```
Class Currency isA Data_Usage_QIndicator with
      attribute, single
           expected_level_upper : Integer;
           expected_level_lower : Integer;
           achieved_level : Integer
      attribute
           relevant_entities : Type
end
```

Volatility describes when data are valid at the real world. Volatility can be defined in terms of data stores, modeled through the inherited *related_objects* attribute, or in terms of entities (e.g. Relations) modeled through the *relevant_entities* attribute.

```
Class Volatility isA Data_Usage_QIndicator with
      attribute, single
            expected_level_upper : Integer;
            expected_level_lower : Integer;
            achieved_level : Integer
      attribute
            relevant_entities : Type
end
```

Class *Timeliness* combines currency and volatility of the information. The achieved level can be produced through the user defined *method of computation*, modeled from the respective inherited attribute.

```
Class Timeliness isA Data_Usage_QIndicator with
        attribute, single
           expected_level_upper : Integer;
           expected_level_lower : Integer;
           achieved_level : Integer
        attribute
            currency : Currency;
            volatility : Volatility
end
```

Responsiveness describes the interaction of employed software with the user, e.g. self reporting etc. It has to do only with software components, so the *related_objects* attribute is refined accordingly. We define the metric of the responsiveness quality dimension as boolean. In section 5.2.4 we proposed the metric to be numerical. The difference is due to the fact, that the proposed measurement deals with the overall quality of the data warehouse, whereas the quality indicator in this model is related to a physical object.

```
Class Responsiveness isA Data_Usage_QIndicator with
      attribute, single
            expected_level_upper : Boolean;
            expected_level_lower : Boolean;
            achieved_level : Boolean
      attribute
            related_objects: DW_Component
end
```

Usefulness can be used to model whole quering processes. e.g. having for related objects a client datastore and a query tool. *Interpretability* is used to model the existence of description of the stored data, so that the user can query them easily.

```
Class Usefulness isA Data_Usage_QIndicator with
      attribute, single
            expected_level_upper : Integer;
            expected_level_lower : Integer;
            achieved_level : Integer
        attribute
            timeliness : Timeliness;
            responsiveness : Responsiveness;
            interpretability : Interpretability
end
```

Class *Data_Usage_Quality* is the top level class, describing the way the overall data usage process can be related to quality issues.

```
Class Data_Usage_Quality isA Data_Usage_QIndicator with
        attribute, single
            expected_level_upper : Integer;
            expected_level_lower : Integer;
            achieved_level : Integer
        attribute
            accessibility : Accessibility;
            usefulness : Usefulness
end
```

7.2.5 Data Quality

Data Quality is dealing with the quality of the stored data in terms of completeness, correctness, etc. Obviously it is related to *physical* data, namely *data stores*, modeled through the *related_objects* attribute, or in terms of entities (e.g. Relations) modeled through the *relevant_entities* attribute. All the other data quality dimensions inherit from the parent class *Data_Quality_Indicator*.

```
Class Data_Quality_Indicator isA QualityIndicator with
        attribute, single
            expected_level_upper : Integer;
            expected_level_lower : Integer;
            achieved_level : Integer
      attribute
            related_objects : DataStore;
            related_entity: Type
end
```

Data_Completeness estimates the percentage of real world information which is stored. The credibility of the source that provided the information is modeled through class *Data_Credibility*. The consistency of stored information is modeled through the class *Data_Consistency*. The accuracy of the data entry process is modeled through the class *Data_Accuracy*.

```
Class Data_Completeness isA Data_Quality_Indicator end
Class Data_Credibility isA Data_Quality_Indicator end
Class Data_Consistency isA Data_Quality_Indicator end
Class Data_Accuracy isA Data_Quality_Indicator with
      attribute
            responsible : String;
```

```
                tool : Agent
end
```

The class *Data_Interpretability* describes the layout of data.

```
Class Data_Interpretability isA Data_Quality_Indicator with
        attribute
                data_layout : String;
                aliases : String;
                defaults : String;
                coded_values : String;
                primary_keys : String;
                foreign_keys : String
end
```

Finally, the class *Data_Quality_Indicator* is the top level class, describing the way the overall data quality of a data store can be modeled and calculated.

```
Class Data_Quality isA Data_Quality_Indicator with
         attribute
                completeness : Data_Completeness;
                credibility  : Data_Credibility;
                accuracy     : Data_Accuracy;
                consistency  : Data_Consistency;
                interpretability : Data_Interpretability
end
```

7.3 Queries for the Architecture and Quality Model

The expressiveness of the Telos language for the description and modeling of data warehouses, can be further proclaimed through the presentation of user queries. In the sequel, we will present queries a user can pose to the Architecture and Quality Model.

The first query returns the bad quality identifiers with the bad quality measure as an attribute. We will reuse this query in the definitions of the following queries, so that their definition becomes simpler.

```
QueryClass BadQualityIndicator isA QualityIndicator with
computed_attribute
        quality_level : QualityMeasure
constraint
        c : $ (this achieved_level quality_level) and
              (exists qm3,qm4,qm5/QualityMeasure (this expected_level_lower qm3)
              and (this expected_level_upper qm4) and (this achieved_level qm5)
              and (not ((qm3 <= qm5) and (qm4 >= qm5)))) $
end
```

The next query returns all data warehouse objects which are related to a specific bad quality indicator, either the data warehouse object is classified by it or the object is affected by it. The bad quality indicator is returned as an attribute.

```
QueryClass Bad_DW_Objects isA DW_Object with
computed_attribute
        qualityInd : QualityIndicator
constraint
        c : $ ((this classifiedBy qualityInd) or (qualityInd related_objects this))
               and (qualityInd in BadQualityIndicator) $
end
```

We use the following query, to retrieve bad quality indicators of a special quality dimension. The parameter can be substituted with a subclass of QualityIndicator, e.g. Schema_Quality.

```
GenericQueryClass BadSpecialQualityIndicator isA QualityIndicator with
parameter
        quality_class : Class
retrieved_attribute
        achieved_level : QualityMeasure
constraint
        c: $ (this in quality_class) and (this in BadQualityIndicator) $
end
```

The following queries are used for tracing quality in data warehouse architecture. As mentioned in chapter 5, we want to find the reason of a problem with bad quality in a data warehouse component. The query *CauseOfBadQuality* finds the data warehouse objects which influences the quality of the object, which is given as a parameter to the query. The query uses recursion and must therefore be an instance of the system class Magic.

```
GenericQueryClass CauseOfBadQuality in Magic isA DW_Object with
parameter
      badObject : DW_Object
constraint
      c: $ exists q1,q2/QualityIndicator (badObject classifiedBy q1)
           and (q1 dependsOn q2) and (q1 in BadQualityIndicator) and
           (q2 in BadQualityIndicator) and ((this classifiedBy q2)
            or (exists o/DW_Object (o classifiedBy q2) and
            (this in CauseOfBadQuality[o/badObject]))) $
end
```

Quality indicators are classified with a priority value. This query returns all the quality indicators with "high" priority. It can be used in conjunction with the query BadQualityIndicator to find the quality indicators with bad quality and high priority.

```
QueryClass HighPriorityIndicator isA QualityIndicator with
computed_attribute
      priority_rank: Real
constraint
      c: $ (this priority priority_rank) and (priority_rank >= 0.5 ) $
end
```

This query returns all the objects with quality indicators of high priority.

```
QueryClass HighPriority_Objects isA DW_Object with
computed_attribute
      qualityInd : QualityIndicator
constraint
      c : $ (qualityInd related_objects this) and
            (qualityInd in HighPriorityIndicator) $
end
```

8 Conclusions and Future Work

We presented a reference architecture for data warehouses and linked it to a quality model. The data warehouse architecture model represents physical, logical and conceptual objects of a data warehouse and their relationships. The quality model for data warehouses represents user requirements (expected values) and actually measured quality (achieved values). The architecture and quality model can be queried for analyzing the architecture and quality of any data warehouse, e.g. to find problems or conflicts in the architecture and to find undesired quality values.

Quality plays an important role in information systems and it applies to all levels and perspectives of a data warehouse. It can be measured in many different ways, however, the actual method of measurement for a quality indicator must be given by the data warehouse designer or administrator. Furthermore, we show different groups of persons related to a data warehouse and their interests into a data warehouse and the quality of the warehouse.

The architecture model enables the data warehouse designer to *design* a data warehouse in a conceptual, logical and physical perspective. Used as reference model, it allows us to compare different data warehouse solutions. Furthermore, certain weaknesses of the data warehouse can be detected during the design phase.

We used our metadata repository ConceptBase as tool for experiments and the demonstration of the architecture and quality model. It offers a graphical user interface, so that a data warehouse architecture can be shown in a graphical browser. In addition, it offers application programming interfaces to languages like C/C++ or Java, so that other DWQ tools or external tools can be easily integrated. It has virtually no limitation in the number of analysis queries, so a lot of development policies for data warehouse design can be encoded. We have shown in an example, how we can represent a data warehouse architecture in our model, and how we can use the meta database for analyzing the design of the warehouse.

Currently, the quality of components is used for the analysis of the data warehouse design. We plan to use the methodology of Quality Function Deployment (QFD) [Akao90] for the quality-oriented design of a data warehouse. A data warehouse design tool will use this methodology for modeling a data warehouse architecture and its quality aspects.

In addition, we will connect external tools for source integration and for reasoning on multi-dimensional data structures to the meta database. The model must be extended to store the information provided and retrieved by these tools.

9 References

[AA87] N. Agmon, N.Ahituv, Assessing data reliability in an information system, *J. Management Information Systems*, vol. 4, no. 2, 1987

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

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

[BBM+95] D. H. Besterfield, C. Besterfield-Michna, G. Besterfield, M. Besterfield-Sacre, Total Quality Management, *Prentice Hall*, 1995

[BHS+97] P. A. Bernstein, B. Harry, P. Sanders, D. Shutt, J. Zander. The Microsoft Repository. In *Proc. 23nd International Conference on Very Large Data Bases*, Athens, August 1997. Morgan Kaufmann Publishers.

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

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

[BP82] D.P. Ballou, H.L. Pazer, The impact of inspector failability on the inspection policy serial production system, *Management Science*, vol. 28, no.4, 1982

[BP85] D.P. Ballou, H.L. Pazer, Modeling data and process quality multi-input, multi-output information systems, *Management Science*, vol. 31, no.2 1985

[BP87] D.P. Ballou, H.L. Pazer, Cost/quality tradeoffs for control procedures information systems, *OMEGA: Int'l J. Management Science*, vol. 15, no. 6, 1987

[BT89] D.P. Ballou, K.G. Tayi, Methodology for allocating resources for data quality enhancement, *Comm. ACM*, vol. 32, no. 3, 1989

[BWPT93] D.P. Ballou, R.Y. Wang, H.L. Pazer, K.G. Tayi, Modeling 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

[CD96] S. Chaudhuri, U.Dayal, Data warehousing and OLAP for Decision Support, *Tutorials of 22nd International Conference on Very large Data Bases*, Mumbai, India, September 1996

[CGMH+94] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *Proc. of IPSI Conference*, Tokyo (Japan), 1994.

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

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

[DJLS96] S. Dar, H.V. Jagadish, A.Y. Levy, and D. Srivastava. Answering queries with aggregation using views. In *Proc. 22nd International Conference on Very Large Data Bases*, Bombay (India), September 1996. Morgan Kaufmann Publishers.

[DM92] W.H. Delone, E.R. McLean, Information Systems Success: The quest for the dependent variable, *Information Systems Research*, vol. 3, no. 1, 1992

[DWQ97] DWQ, *Deliverable D1.1*, *Data Warehouse Quality Requirements and Framework*, NTUA, RWTH, INRIA, DFKI, Uniroma, IRST, DWQ TR DWQ – NTUA - 1001, 1997

[Hall78] Halloran et al., Systems development quality control, *MIS Quarterly*, vol. 2, no.4, 1978

[HC88] Hauser, J. R., D. Clausing, The House of Quality, *The Harvard Business Review*, May-June, No. 3, 1988

[HGMW+95] J. Hammer, H. Garcia-Molina, J. Widom, W. Labio, Y. Zhuge. The Stanford Data Warehousing Project. *Data Engineering*, *Special Issue on Materialized Views on Data Warehousing*, 18(2), pp. 41-48. June 1995.

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

[HR96] 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.

[HRU96] V. Harinarayan, A. Rajaraman, and J. Ullman. Implementing data cubes efficiently. In *Proc. 1996 ACM SIGMOD International Conference on Management of Data*, pages 205-227, Montreal (Canada), June 1996.

[HS95] E. Horowitz, S. Sahni, Fundamentals of Computer Algorithms, *Galgotia Publications, Ltd*. N. Delhi, 1995

[HZ96] R. Hull, G. Zhou. A Framework for Supporting Data Integration Using the Materialized and Virtual Approaches. In *Proceedings of the ACM SIGMOD International Conference of the Management of Data*, pp. 481 - 492, June 1996.

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

[ISO92] ISO, ISO9000 International Standards for Quality Management, *Geneva: International Organization for Standardization*, 1992

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

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

[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, 167-192, 1995.

[JP92] M.Jarke, K.Pohl. Information systems quality and quality information systems. In Kendall/Lyytinen/DeGross (eds.): *Proc. IFIP 8.2 Working Conf. The Impact if Computer-Supported Technologies on Information Systems Development* (Minneapolis 1992), North-Holland 1992, 345-375

[JP93] M.Jarke, K.Pohl: Vision-driven systems engineering. In Prakash/Rolland/Pernici (eds.): *Proc. IFIP 8.1 Working Conf. on Information Systems Development Process* (Como/Italy), North-Holland 1993, 3-20.

[JPD+94] M.Jarke, K.Pohl, R.Dömges, S.Jacobs, H.W.Nissen: Requirements information management – the NATURE approach. *Engineering of Information Systems*, 2, 4 (1994), 609-637.

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

[KLSS95] Thomas Kirk, Alon Y. Levy, Yehoshua Sagiv, and Divesh Srivastava. The Information Manifold. In *Proceedings of the AAAI 1995 Spring Symp. on Information Gathering from Heterogeneous, Distributed Enviroments*, pages 85-91, 1995.

[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

[LM96] A. Y. Levy and I. S. Mumick. Reasoning with aggregation constraints. In *Proceedings of the International Conference on Extending Database Technology (EDBT-96)*, Avignon, France, 1996.

[LMSS95] A.Y. Levy, A.O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. *In Proc. 14th Symposium on Principles of Database Systems*, 95-104, San Jose, Ca, 1995.

[LRO96] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Query answering algorithms for information agents. In *Proc. of the 13th Nat. Conf. on Artificial Intelligence (AAAI-96)*, pages 40-47, 1996.

[LSK95] A. Levy, D. Srivastava, and T. Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems*, 5:121-143, 1995.

[LU90] 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

[MA94] Mizuno, S. 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

[MBJK90] J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis: Telos – a language for representing knowledge about information systems.. In *ACM Trans. Information Systems*, 8, 4, 1990, pp. 325-362.

[Meta96] Metadata Coalition: *Proposal for Version 1.0 Metadata Interchange Specification*. Http://www.metadata.org/standards/toc.html, July 1996.

[More82] R.C. Morey, Estimating and improving the quality of information the MIS, *Comm. ACM*, vol. 25, no. 5, 1982

[MRW78] J.A. McCall, P.K. Richards, G.F. Walters, Factors in software quality, *Technical Report, Rome Air Development Center*, 1978

[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. *MicroStrategy's 4.0 Product Line*. http://www.strategy.com/launch/4_0_arc1.htm, 1997.

[NJ97] H.W. Nissen, M. Jarke, Goal-Oriented Inconsistency Management in Customizable Modeling Environments. Submitted to *Special Issue of IEEE Transactions on Software Engineering on*

Managing Inconsistency in Software Development, 1997.

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

[PR90] R.W. Pautke, T.C. Redman, Techniques to control and improve quality of data large databases, *Proc. of Statistics Canada Symp. 90*, Canada, 1990

[QGMW96] D.Quass, A. Gupta, I. S. Mumick, and J. Widom. Making Views Self-Maintainable for Data Warehousing. *Proceedings of the Conference on Parallel and Distributed Information Systems*. Florida, December 1996.

[RedB97] 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

[SAG96] Software AG: *SourcePoint White Paper*. Software AG, Uhlandstr 12, 64297 Darmstadt, Germany, 1996.

[Sahi96] Dr. Kenan Sahin. *Multidimensional Database Technology and Data Warehousing*. Http://www.kenan.com/acumate/byln_mdw.htm, 1996.

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

[SDJL96] D. Srivastava, S. Dar, H. V. Jagadish, and A. Y. Levy. Answering queries with aggregation using views. In *Proceedings of the 22. International Conference on Very Large Data Bases (VLDB-96)*, Bombay, India, 1996.

[SLW94] D. M. Strong, Y.W. Lee, 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

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

[Ull97] Jeffrey D. Ullman. Information integration using logical views. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT-97)*, Lecture Notes in Computer Science, pages 19-40. Springer-Verlag, 1997

[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

[Weld97] Jay-Louise Weldon. *Warehouse Cornerstones*. Byte, pp. 85-88, January 1997.

[WGL+96] J. L. Wiener, H. Gupta, W. J. Labio, Y. Zhuge, H. Garcia-Molina, J. Widom. *A System Prototype for Warehouse View Maintenance*. Proceedings of the ACM Workshop on Materialised Views: Techniques and Applications , Montreal, Canada, June 7, 1996, 26-33.

[Wid95] J. Widom, Research Problems in Data Warehousing, *Proceedings of the 4th Int'l Conference on Information and Knowledge Management (CIKM)*, November 1995.

[Wie92] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, pages 38-49, March 1992.

[WKM93] R.Y. Wang, H.B. Kon, S.E. Madnick, Data quality requirements analysis and modeling, *Proc. Ninth Int'l conference on Data Engineering*, Vienna, 1993

[WM90] R.Y. Wang, S.E. Madnick, A Polygen model for heterogeneous database systems: The source tagging perspective, *16th VLDB Conference*, Brisbaine, Australia, 1990

[WRG93] R.Y. Wang, M.P. Reddy, A. Gupta, An object-oriented implementation of quality data products, *Proc. Third Ann. Workshop Information Technologies and Systems*, 1993

[WRK95] R.Y. Wang, M.P. Reddy, H.B. Kon, Towards Quality Data: An attribute-based Approach, *Decision Support Systems*, 13(1995)

[WSF95] R.Y. Wang, V.C. Storey, C.P. Firth, A framework for Analysis of Data Quality Research, *IEEE TKDE*, Vol. 7, No. 4, Aug. 1995

[WSG94] R.Y. Wang, D. M. Strong, L.M. Guarascio, Beyond Accuracy: What Data Quality Means to Data Consumers, (No. TDQM-94-10). *Cambridge Mass: Total Data Quality Management Research Program*, MIT Sloan School of Management, 1993

[WW90] Y. Wand, R. Weber, An ontological model for an information system, *IEEE Trans. Software Engineering*, vol. 16, no. 11, 1990

[WW96] Y. Wand, R.Y. Wang, Anchoring data quality dimensions ontological foundations, *Comm. ACM*, November 1996, Vol. 39, No. 11

[ZHK96] G. Zhou, R. Hull, R. King. Generating Data Integration Mediators that Use Materialization. In *Journal of Intelligent Information Systems*, Volume 6(2), pp. 199-221, 1996.

[ZHKF95a] Gang Zhou, Richard Hull, Roger King, and Jean-Claude Franchitti. Data integration and warehousing using H20. *IEEE Bulletin of the Technical Committee on Data Engineering*, 18(2):29-40, 1995.

[ZHKF95b] Gang Zhou, Richard Hull, Roger King, and Jean-Claude Franchitti. Using object matching and materialization to integrate heterogeneous databases. In *Proc. of the 3rd Int. Conf. on Cooperative Information Systems (CoopIS-95)*, pages 4-18, 1995.