Architecture and quality in data warehouses - An extended repository approach

Jarke, M.; Jeusfeld, M.A.; Quix, C.; Vassiliadis, P.

Published in:
Information Systems

Publication date:
1999

ARCHITECTURE AND QUALITY IN DATA WAREHOUSES: AN EXTENDED REPOSITORY APPROACH

MATTHIAS JARKE[(1)], MANFRED A. JEUSFELD[(2)]
CHRISTOPH QUIX[(1)], PANOS VASSILIADIS[(3)]

(1) Informatik V, RWTH Aachen, 52056 Aachen, Germany
(2) Infolab, KUB University, Postbus 90153, 5000 LE Tilburg, The Netherlands
(3) Computer Science Division, NTUA Athens, Zographou 15773 Athens, Greece

Abstract — Most database researchers have studied data warehouses (DW) in their role as buffers of materialized views, mediating between update-intensive OLTP systems and query-intensive decision support. This neglects the organizational role of data warehousing as a means of centralized information flow control. As a consequence, a large number of quality aspects relevant for data warehousing cannot be expressed with the current DW meta models. This paper makes two contributions towards solving these problems. Firstly, we enrich the meta data about DW architectures by explicit enterprise models. Secondly, many very different mathematical techniques for measuring or optimizing certain aspects of DW quality are being developed. We adapt the Goal-Question-Metric approach from software quality management to a meta data management environment in order to link these special techniques to a generic conceptual framework of DW quality. The approach has been implemented in full on top of the ConceptBase repository system and has undergone some validation by applying it to the support of specific quality-oriented methods, tools, and application projects in data warehousing.

Keywords: data warehouses, meta data management, data quality, conceptual models, repository

1. INTRODUCTION

Data warehouses provide large-scale caches of historic data. They sit between information sources gained externally or through online transaction processing systems (OLTP), and decision support or data mining queries following the vision of online analytic processing (OLAP). Three main arguments have been put forward in favor of this caching approach:

1. *Performance and safety considerations*: The concurrency control methods of most DBMSs do not react well to a mix of short update transactions (as in OLTP) and OLAP queries that typically search a large portion of the database. Moreover, the OLTP systems are often critical for the operation of the organization and must not be under danger of corruption of other applications.
2. *Logical interpretability problems*: Inspired by the success of spreadsheet techniques, OLAP users tend to think in terms of highly structured multi-dimensional data models, whereas information sources offer at best relational, often just semi-structured data models.
3. *Temporal and granularity mismatch*: OLTP systems focus on current operational support in great detail, whereas OLAP often considers historical developments at a somewhat less detailed granularity.

Thus, quality considerations have accompanied data warehouse research from the beginning. A large body of literature has evolved over the past few years in addressing the problems introduced by the DW approach, such as the trade-off between freshness of DW data and disturbance of OLTP work during data extraction; the minimization of data transfer through incremental view maintenance; and a theory of computation with multi-dimensional data models.

However, the heavy use of highly qualified consultants in data warehouse applications indicates that we are far from a systematic understanding and usage of the interplay between quality factors and design options in data warehousing. The goal of the European DWQ project [26] is to address these issues by developing, prototyping and evaluating comprehensive Foundations for Data Warehouse Quality, delivered through *enriched meta data management facilities* in which specific analysis and optimization techniques are embedded.

This paper develops the DWQ architecture and quality management framework, and describes its implementation in a meta database. The main contributions include an extension of the standard DW architecture used in the literature by enterprise modeling aspects, and a strategy for embedding special-purpose mathematical

reasoning tools in a repository model of the architecture. Our goal is to enable a computationally tractable yet very rich quality analysis, and a quality-driven design process.

Interaction with DW tool vendors, DW application developers and administrators has shown that the standard framework used in the DW literature is insufficient to capture in particular the business role of data warehousing. A DW is a major investment made to satisfy some business goal of the enterprise; quality model and DW design should reflect this business goal as well as its subsequent evolution over time. In section 2, we discuss this problem in detail. Our new architectural framework separates (and links) explicitly the concerns of conceptual enterprise perspectives, logical data modeling (the main emphasis of DW research to date), and physical information flow (the main concern of commercial DW products to date).

In section 3, we first build on the literature for data and software quality to come up with a suitable set of DW quality goals, as perceived by different groups of stakeholders. We then adapt a variant of the so-called Goal-Question-Metric approach used in software quality management, in order to link these conceptual goals to specific techniques developed in DW research and practice, and to enable trade-off between heterogeneous quality goals. Technically, this is accomplished through materialized quality views, i.e. using the DW approach to describe its own quality. Some experiences with a prototypical implementation of the resulting meta database using the ConceptBase repository manager have been gained in cooperation with industrial case studies. Section 4 relates our approach to other work in data warehousing, data and software quality, while section 5 provides a summary and conclusions.

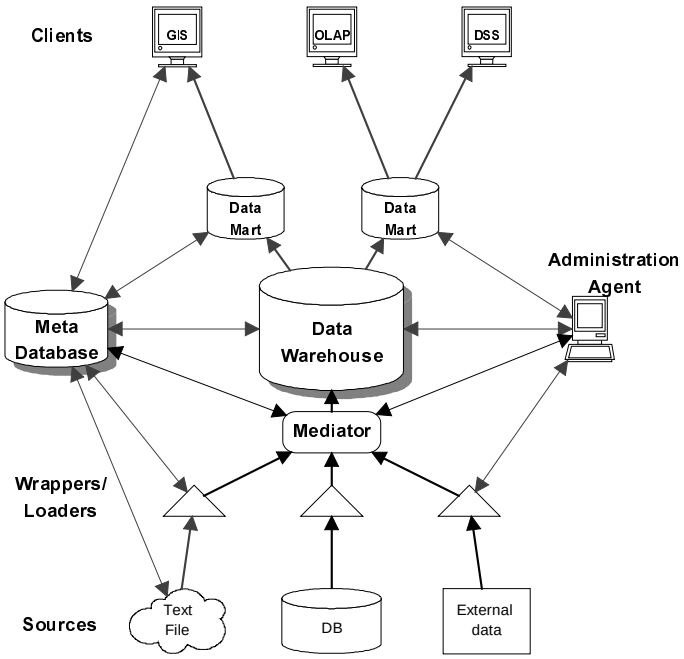

Fig. 1: Traditional Data Warehouse Architecture

2. AN EXTENDED DATA WAREHOUSE ARCHITECTURE

The traditional data warehouse architecture, advocated both in research and in the commercial trade press, is shown in figure 1. Physically, a data warehouse system consists of databases (source databases, materialized views in the data warehouse), data transport agents that ship data from one database to another, and a repository which stores meta data about the system and its evolution. In this architecture, heterogeneous information sources are first made accessible in a uniform way through extraction mechanisms called *wrappers*, then *mediators* [47] take on the task of information integration and conflict resolution. The resulting standardized and integrated data are stored as materialized views in the data warehouse. The DW base views are usually just slightly aggregated; to customize them better for different groups of analyst users, *data marts* with more aggregated data about

specific domains of interest are frequently constructed as second-level caches which are then accessed by data analysis tools ranging from query facilities through spreadsheet tools to full-fledge data mining systems based on knowledge-based or neural network techniques.

The content of the repository determines to a large extent the way how the data warehouse system can be used and evolved. The main goal of our approach is therefore to define a meta database schema which can capture and link all relevant aspects of DW architecture and quality.

We shall tackle this very difficult undertaking in several steps. First, we discuss the shortcomings of the traditional architecture and propose a conceptual enterprise perspective to solve some of these shortcomings. Then, we elaborate the extended metamodel resulting from our approach, and show how it can be implemented in a repository. Finally, the application of these repository concepts is illustrated with a more detailed description of a specific submodel developed and validated in the DWQ project.

2.1. Adding a Conceptual Perspective to Data Warehousing

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, as shown in figure 1. For example, projects such as TSIMMIS [10], Squirrel [18], or WHIPS [16] all focus on the integration of heterogeneous data via wrappers and mediators, using different logical formalisms and technical implementation techniques. The Information Manifold project at AT&T Research [33] is the only one providing a conceptual domain model as a basis for integration.

Our key observation is that the architecture in figure 1 covers only partially 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.

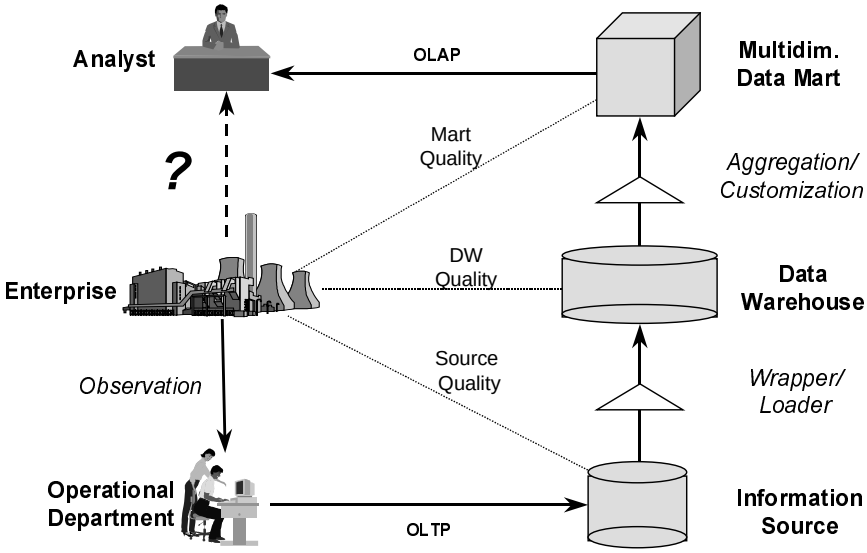

Fig. 2: Data Warehousing in the Context of an Enterprise

The main argument we wish to make is the need for a *conceptual enterprise perspective*. To explain, consider figure 2. In this figure, the flow of information in figure 1 is stylized on the right-hand side, whereas the process of creating and using the information is shown on the left. Suppose an analyst wants to know something about the business – the question mark in the figure. She does not have the time to observe the business directly but must rely on existing information gained by operational departments, and documented as a side effect of OLTP systems. This way of gathering information implies already a bias which needs to be compensated when selecting OLTP data for uploading and cleaning into a DW where it is then further pre-processed and aggregated in data marts for certain analysis tasks. Considering the long path the data has taken, it is obvious that also the last step, the formulation of conceptually adequate queries and the conceptually adequate interpretation of the answers present a major problem to the analyst.

The traditional DW literature only covers two of the five steps in figure 2. Thus, it has no answers to typical practitioner questions such as "how come my operational departments put so much money in their data quality, and still the quality of my DW is terrible" (answer: the enterprise views of the operational departments are not easily compatible with each other or with the analysts view), or "what is the effort required to analyze problem X for which the DW currently offers no information" (could simply be a problem of wrong aggregation in the materialized views, could require access to not-yet-integrated OLTP sources, or even involve setting up new OLTP sensors in the organization).

An adequate answer to such questions requires an explicit model of the conceptual relationships between an enterprise model, the information captured by OLTP departments, and the OLAP clients whose task is the decision analysis. We have argued that a DW is a major investment undertaken for a particular business purpose. We therefore do not just introduce the enterprise model as a minor part of the environment, but demand that *all other models are defined as views on this enterprise model.* Perhaps surprisingly, even information source schemas define views on the enterprise model – not vice versa as suggested by figure 1!

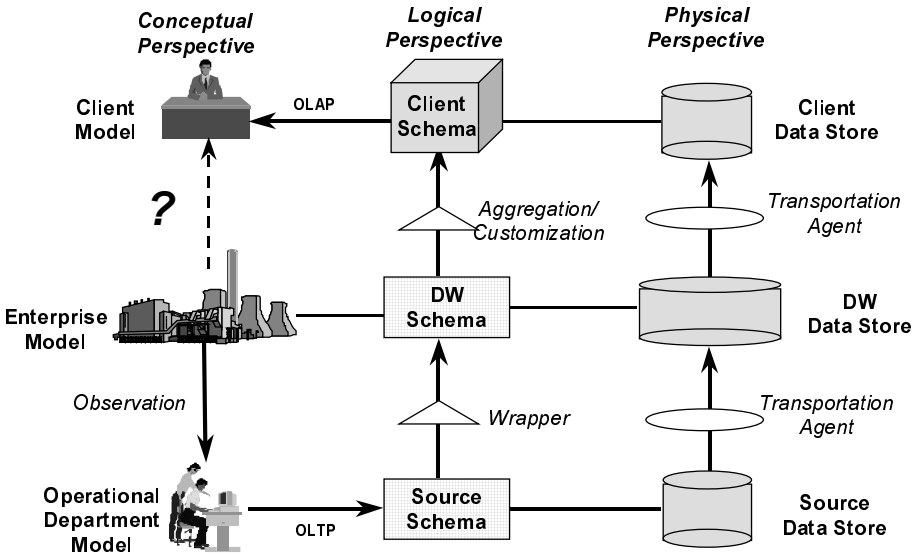

Fig. 3: The Proposed Data Warehouse Meta Data Framework

2.2. A Repository Model for the Extended Data Warehouse Architecture

By introducing an explicit business perspective as in figure 2, the wrapping and aggregation transformations performed in the traditional data warehouse literature can thus all be checked for interpretability, consistency or completeness with respect to the enterprise model – provided an adequately powerful representation and reasoning mechanism is available.

At the same time, the logical transformations need to be implemented safely and efficiently by physical data storage and transportation – the third perspective in our approach. It is clear that these physical quality aspects require completely different modeling formalisms from the conceptual ones, typical techniques stemming from queuing theory and combinatorial optimization.

As a consequence, the data warehouse meta framework we propose clearly separates three perspectives as shown in figure 3: a conceptual enterprise perspective, a logical data modeling perspective, and a physical data flow perspective.

There is no single decidable formalism that could cover the handling of all these aspects uniformly in a meta database. We have therefore decided to capture the architectural framework in a deductive object data model in a comprehensive but relatively shallow manner. Special-purpose reasoning mechanisms such as the ones mentioned above can be linked to the architectural framework as discussed in section 3, below.

We use the meta database to store an abstract representation of data warehouse applications in terms of the three-perspective scheme. The architecture and quality models are represented in Telos [35], an extensible meta modeling language which has both a graphical syntax and a frame syntax, mapped to an underlying formal semantics based on standard deductive databases. Using this formal semantics, the Telos implementation in the ConceptBase system [22] provides query facilities, and definition of constraints and deductive rules. Telos is well suited because it allows to formalize specialized modeling notations (including the adaptation of graphical representations [27]) by means of meta classes. Since ConceptBase treats all concepts including meta classes as first-class objects, it is well suited to manage abstract representations of the DW objects to be measured [28].

A condensed ConceptBase model of the architecture notation is given in figure 4, using the graph syntax of Telos. Bold arrows denote specialization links. The top level object is *MeasurableObject.* It classifies objects at any perspective (conceptual, logical, or physical) and at any level (source, data warehouse, or client). Within each perspective, we distinguish between the modules it offers (e.g. client model) and the kinds of information found within these modules (e.g. concepts and their subsumption relationships). The horizontal links *hasSchema* and *isViewOn* establish the way how the horizontal links in Figure 2 are interpreted: the types of a schema (i.e., relational or multidimensional structures) are defined as logical views on the concepts in the conceptual perspectives. On the other hand, the components of the physical perspective get a schema from the logical perspective.

Each object can have an associated set of materialized views called *QualityMeasurements.* These materialized views (which can also be specialized to the different perspectives – not in the figure) constitute the bridge to the quality model discussed in section 3.

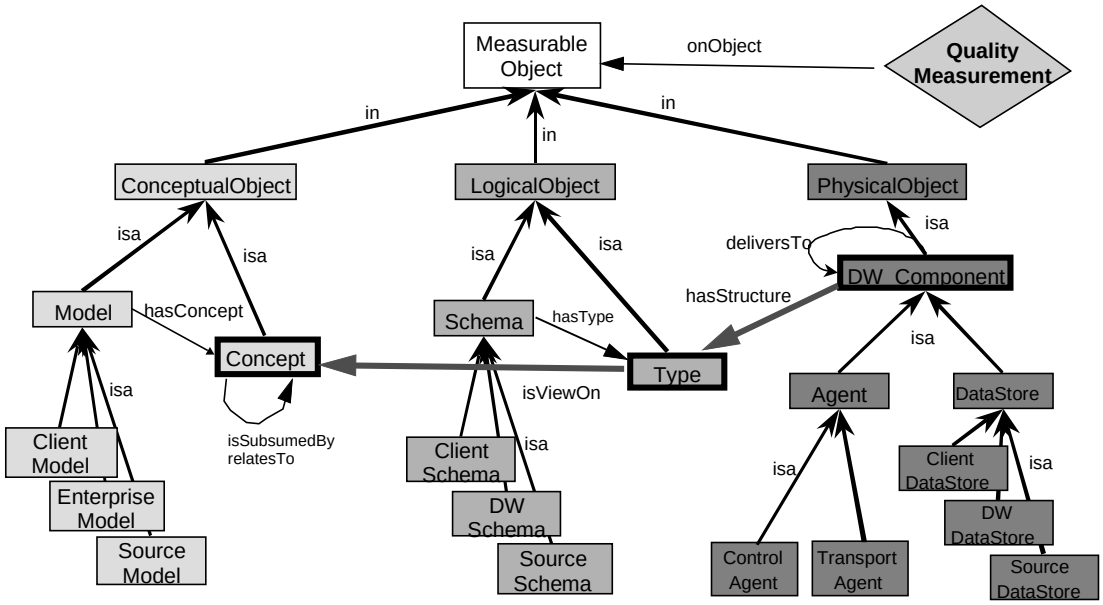

Fig. 4: Structure of the Repository Meta Model

The horizontal levels of the objects are coded by the three subclasses attached to *Model*, *Schema*, and *DataStore*. We have experimented with this notation and were able to represent physical data warehouse architectures of commercial applications, such as the SourcePoint tool marketed by Software AG [41], the DW architecture underlying a data mining project at Swiss Life [42], and a DW project in Telecom Italia (cf. section 2.6). The logical perspective currently supports relational schema definitions whereas the conceptual perspective supports the family of extended entity-relationship and similar semantic data modeling languages. Note that all objects in figure 4 are meta classes: actual conceptual models, logical schemas, and data warehouse components are represented as instances of them in the meta database.

In the following subsections, we elaborate on the purpose of representing each of the three perspectives, then demonstrate how the architecture above can be refined for particular purposes.

2.3. Conceptual Perspective

The conceptual perspective offers a business model of the information systems of an enterprise. The central role is played by 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 are 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. On the client side, the interests of user groups can also be described as views on the enterprise model.

In the implementation of the conceptual perspective in the meta database, the central class is *Model*. A model is related to a source, a client or the relevant section of the 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 distinguish the models of several sources, clients and the enterprise itself. A model consists of *Concepts*, each representing a concept of the real world, i.e. the business world. If the user provides some information about the relationship between concepts in a formal language like description logic, a reasoner can check for subsumption of concepts [7].

The results of the reasoning process are stored in the model as attribute *isSubsumedBy* of the corresponding concepts. Essentially, the repository can serve as a cache for reasoning results. Any tool can ask the repository for containment of concepts. If the result has already been computed, it can directly be answered by the repository. Otherwise, a reasoner is invoked by the repository to compute the result.

2.4. Logical Perspective

The logical perspective conceives a data warehouse from the view point of the actual data models involved, i.e. the data model of the logical schema is given by the corresponding physical component, which implements the logical schema. The central point in the logical perspective is *Schema.* As a model consists of concepts a schema consists of *Types*. We have implemented the relational model as an example for a logical data model; other data models such as the multi-dimensional or the object-oriented data model are also being integrated in this framework [14][45].

Like in the conceptual perspective, we distinguish in the logical perspective between *ClientSchema*, *DWSchema* and *SourceSchema* for the schemata of clients, the data warehouse and the sources. For each client or source model, there is one corresponding schema. This restriction is guaranteed by a constraint in the architecture model. The link to the conceptual model is implemented through the relationship between concepts and types: each type is expressed as a view on concepts.

2.5. Physical Perspective

Data warehouse industry has mostly 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 [41], the data warehouse system of RedBrick, Informix's MetaCube [19], Essbase of Arbor Software [2] or the product suite of MicroStrategy [38].We have 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.

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 a 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 rather than a complex object.

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. An *Agent* may also notify other agents about errors or termination of its process.

A *DataStore* physically stores the data which is described by models and schemata 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.

2.6 Applying the Architecture Model: The Example of Source and Data Integration

The metadata framework shown in figure 4 defines the basic metamodel of the products in the repository, and their interrelationships. As shown in figure 5, this framework can be instantiated by information models (conceptual, logical, and physical schemas) of particular data warehousing strategies which can then be used to design and administer the instances of these data warehouses – the main role of the administration system and meta database in figure 1.

However, quality cannot just be assessed on the network of nine perspectives, but is largely determined by the processes how these are constructed [24]. The process meta model defines a way how such processes can be defined, the process models define plans how data warehouse construction and administration is to be done, and the traces of such processes are captured at the lowest level; this process hierarchy accompanying the DW product model is shown on the right of figure 5.

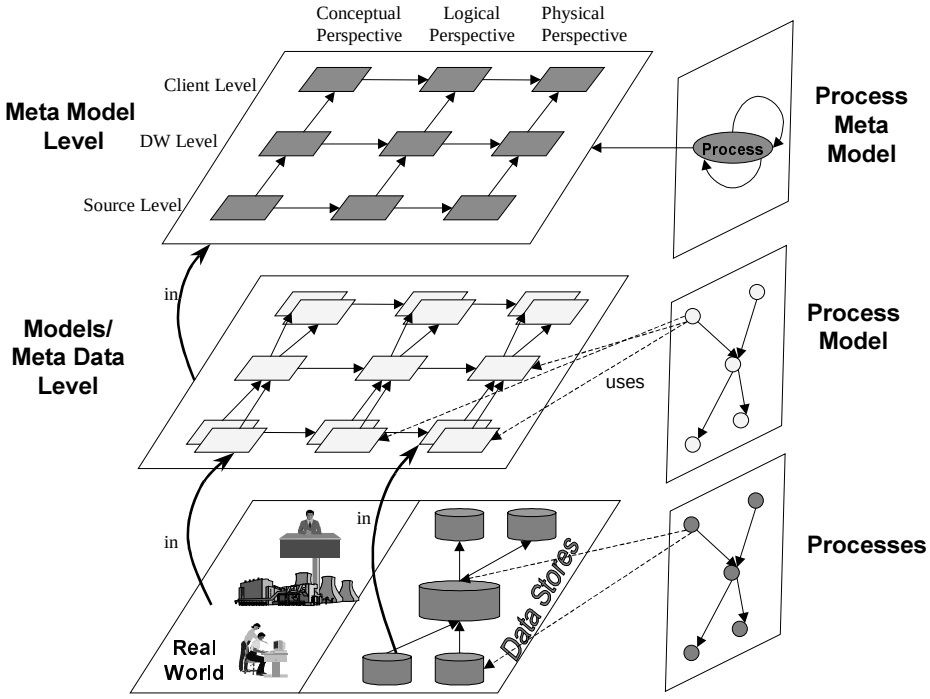

Fig. 5 : Repository Structure for Capturing Product and Process of Data Warehousing

In DWQ, we are still experimenting with suitable process modeling formalisms, based on our earlier work on software process modeling and management [24]. For the purposes of this paper, we can safely assume that the impact of such process models on the repository is some kind of query plan, a partially ordered set of queries defined over the meta database (and stored in the meta database). This is, for example, also the strategy followed in the new version of the Microsoft Repository [3].

Figure 4 only gives a rough overview of the actual model structure in the DWQ repository meta model. In reality, each perspective offers a much richer meta model structure reflecting the approach taken in addressing the tasks in this perspective.

In this subsection, we describe one of the specific DWQ methodologies, the one for source and data integration [8], in order to illustrate this refinement of models as well as the interplay between the different

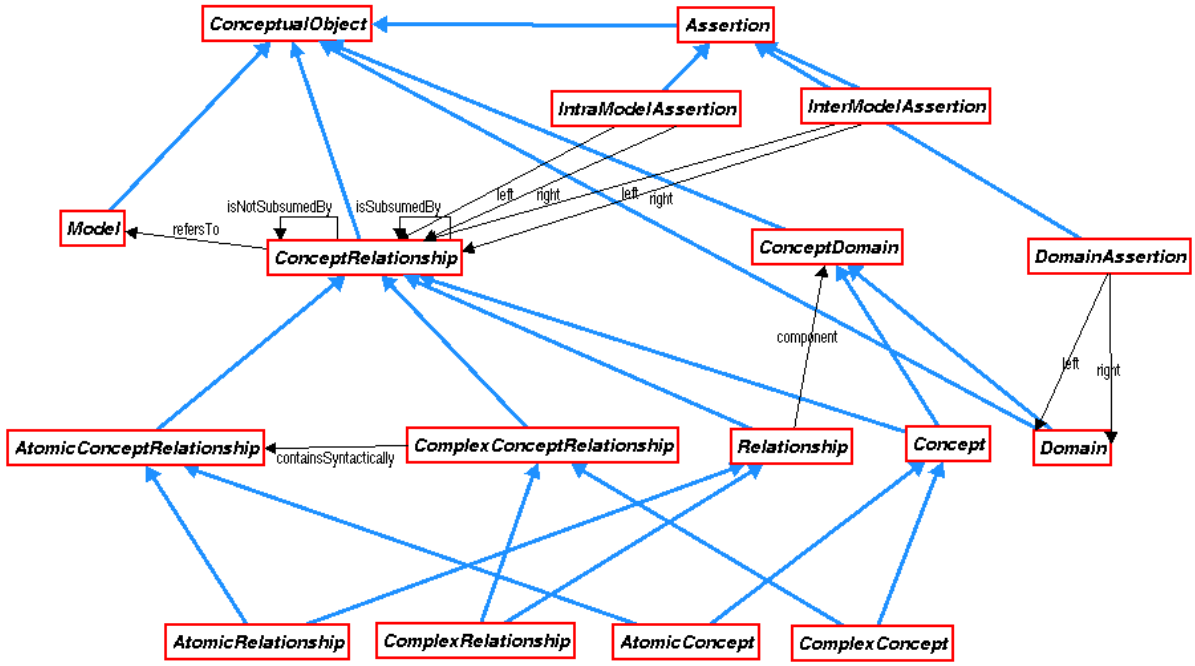

Fig. 6: Refining the Conceptual Perspective for Source Integration (ConceptBase screendump)

perspectives in our approach. While *source integration* means designing the relationships between information sources and the views in the data warehouse, data integration means the construction of acquisition plans by which these views are actually materialized.

In the context of figure 4, the example is concerned with the enterprise and source models at the conceptual perspective and with the source schemas (and possibly DW schemas) in the logical perspective.

Conceptual Perspective: According to the DWQ approach, one conceptual model is constructed for each source and one for the enterprise. These models rely on an extended entity-relationship model in which both the entities and the relationships can be interpreted as concepts formalized in a description logic, and additional logical assertions can be formulated to express generic domain knowledge (*DomainAssertions*), properties and limitations of a source (*IntraModelAssertions*), and relationships between the sources, such as containment, consistency, etc. (*InterModelAssertions*).

In the ConceptBase repository, this leads to an elaboration of the *Concept* node from figure 4, as shown in figure 6. On the one hand, this refinement structurally describes the basic structure of the extended ER model, i.e. *Concepts*, *Relationships*, and complex objects constructed from them. On the other hand, it describes the linkage of the different kinds of assertions to the objects. Despite its expressiveness, this data model allows decidable subsumption reasoning [7] between concepts. Thus, through inheritance from the central *ConceptRelationship*

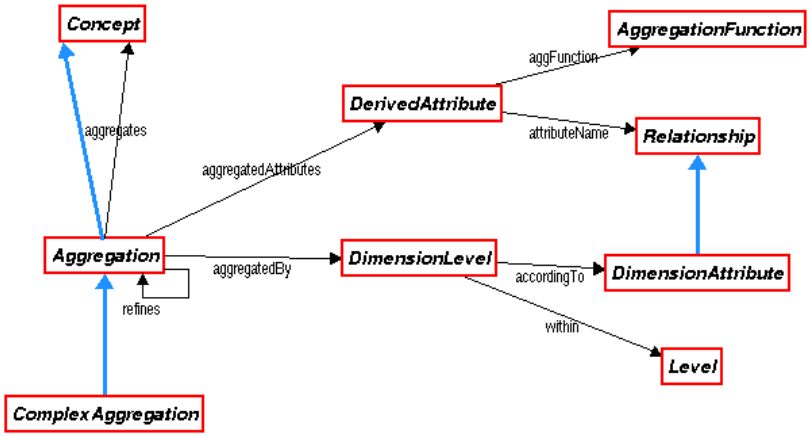

Fig. 7: Client level of the conceptual perspective (ConceptBase screendump)

object, both the assertions and the subsumption relationships computed by an external description logic reasoner on this structure can be applied to all subtypes of the meta schema.

The conceptual model is not restricted for the use in source integration. We can specialize the meta model to handle also the client side of a data warehouse, i.e. multidimensional data models. In the conceptual client model, it is important how aggregations are defined and which attributes are aggregated of a concept [13]. Figure 7 shows the client level of the meta model for the conceptual perspective.

Aggregations aggregate concepts with respect to a specific dimension level, which is defined by a dimension attribute, and a level. For example, if customers are aggregated by cities, the dimension attribute is 'address' and the level is 'city'. Furthermore, we need to know, which attributes are aggregated and which aggregation function is used for the aggregation.

Logical Perspective: As stated earlier, the present implementation of the logical perspective is limited to relational databases. In line with our basic philosophy concerning the central role of the enterprise model, the DWQ approach considers the (relational) schema of an information source to be integrated as a view on the conceptual enterprise model. As the DW schema itself consists of (possibly cleaned and merged) views over the sources, it naturally becomes also an (indirect) view over the enterprise model.

These views are, as usual, defined by conjunctive *Queries* over the enterprise model. In the merging of sources, also disjunctive queries are possible. These queries are defined at the time of source (schema) integration. For the actual data integration, i.e. to load the data warehouse schema from the sources, an *AcquistionPlan* is constructed from these queries (possibly taking into account the physical perspective).

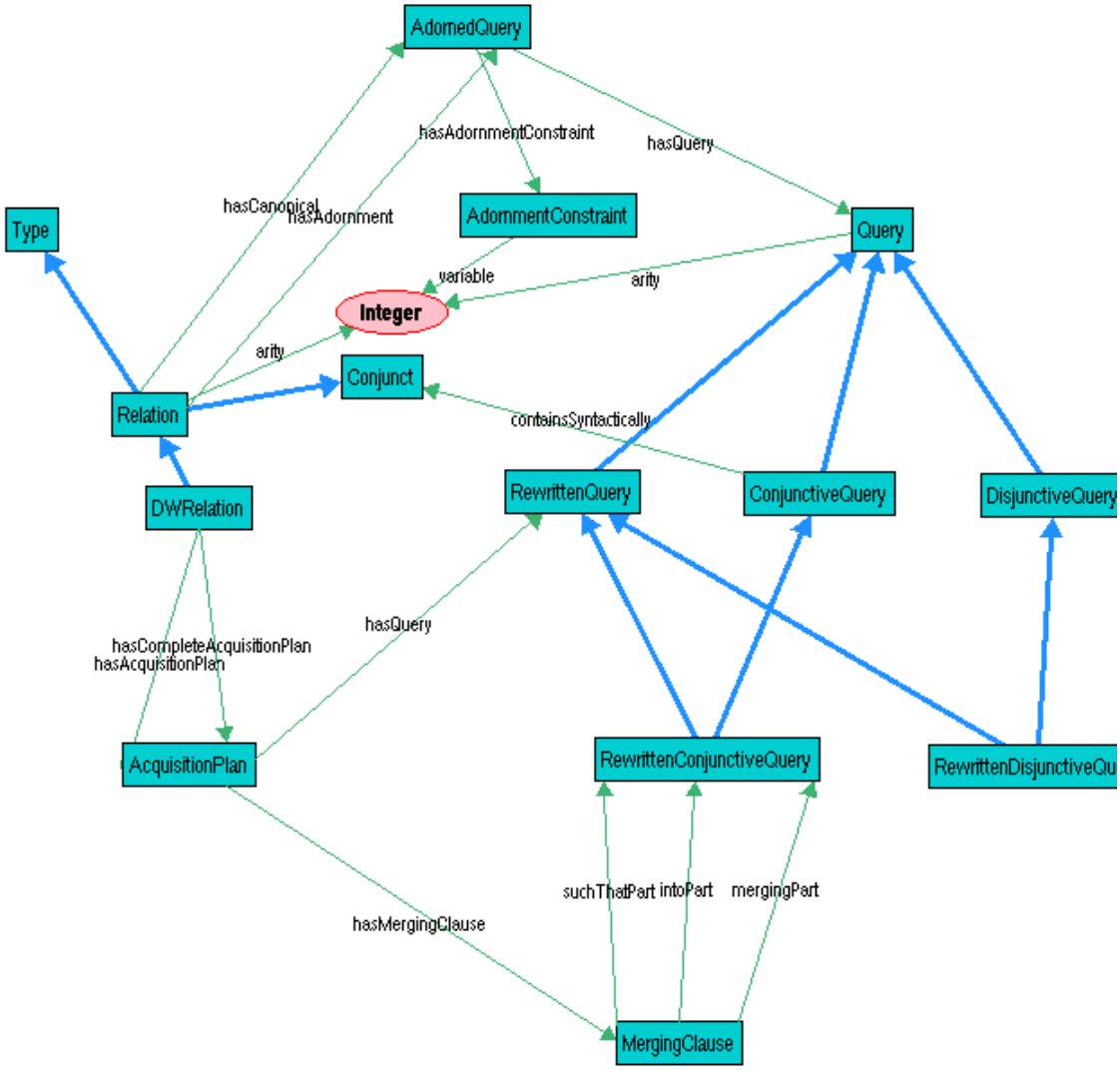

Fig. 8: Refining the Logical Perspective for (Relational) Source and Data Integration (ConceptBase screendump)

However, to capture the semantics correctly, the assertions of the conceptual model must be taken into account; this is accomplished by adding them as adorrnments to the view definition queries. From the acquisition plan and the *AdornedQueries*, a query rewriting can then be performed automatically which defines the extraction queries from the sources as well as the *MergingClauses* that need to be executed when data from more than one source need to be merged into a data warehouse relation.

Figure 8 shows how this approach is captured quite naturally in the ConceptBase repository, refining the *Type* object in figure 4. This structure also provides a suitable memory for the integration process, thus allowing reuse of specific integration techniques as well as re-loading of the DW. Of course, the latter is usually done incrementally by view maintenance techniques but their description goes beyond the scope of this paper.

The DWQ source and data integration approach is described in more detail in [8]. A validation case study involving the integration of four complex Telecom databases, reported in [9], demonstrates that this information structure is suitable for the incremental modeling of data warehouse architectures; "incremental" is meant here both in the sense of gradually refining the models of a specific information source or the enterprise as a whole and in the sense of adding a new information source, possibly overlapping in concepts with the existing enterprise model.

3. MANAGING DATA WAREHOUSE QUALITY

In this section, we discuss how to extend the DW architecture model to support explicit quality models. There are two basic issues to be resolved. On the one hand, quality is a subjective phenomenon so we must organize quality goals according to the stakeholder groups that pursue these goals. On the other hand, quality goals are highly diverse in nature. They can be neither assessed nor achieved directly but require complex measurement, prediction, and design techniques, often in the form of an interactive process. The overall problem of introducing quality models in meta data is therefore to achieve breadth of coverage without giving up the detailed knowledge available for certain criteria. Only this combination enables systematic quality management.

In the following subsections, we first categorize the relevant data warehouse quality dimensions according to the stakeholders that are typically interested in them. We also present some tables mapping these quality criteria to the DW perspectives introduced in the previous section, by giving examples of types of measurements which could help to establish the quality of a particular DW component with respect to a particular quality dimension. Then, we show how this basic structure can be formally captured in an extension to the Goal-Question-Metric approach from software engineering, and how this extension can be implemented and used in the DW meta database.

3.1. Stakeholders and Data Warehouse Quality Dimensions

There exist different roles of users in a data warehouse environment. The *Decision Maker* usually employs an OLAP query tool to get answers interesting to him. A decision maker is usually concerned with the *quality of the stored data*, their *timeliness* and the *ease of querying* them through the OLAP tools. The *Data Warehouse Administrator* needs facilities like *error reporting, meta data accessibility* and knowledge of the *timeliness* of the data, in order to detect changes and reasons for them, or problems in the stored information. The *Data Warehouse Designer* needs to measure the *quality of the schemata* of the data warehouse environment (both existing or newly produced) and the *quality of the meta data* as well. Furthermore, he needs *software evaluation standards* to test the software packages he considers for purchasing. The *Programmers of Data Warehouse Components* can make good use of *software implementation standards* in order to evaluate their work. *Meta data reporting* can also facilitate their job since they can avoid mistakes related to schema information.

Based on this analysis, we can safely argue that different roles imply a different collection of quality dimensions, which a quality model should be able to address in a consistent and meaningful way. In the following, we summarize the quality dimensions of three stakeholders, the data warehouse administrator, the programmer, and the decision maker. A more detailed presentation of quality dimensions for different stakeholder types is included in [11].

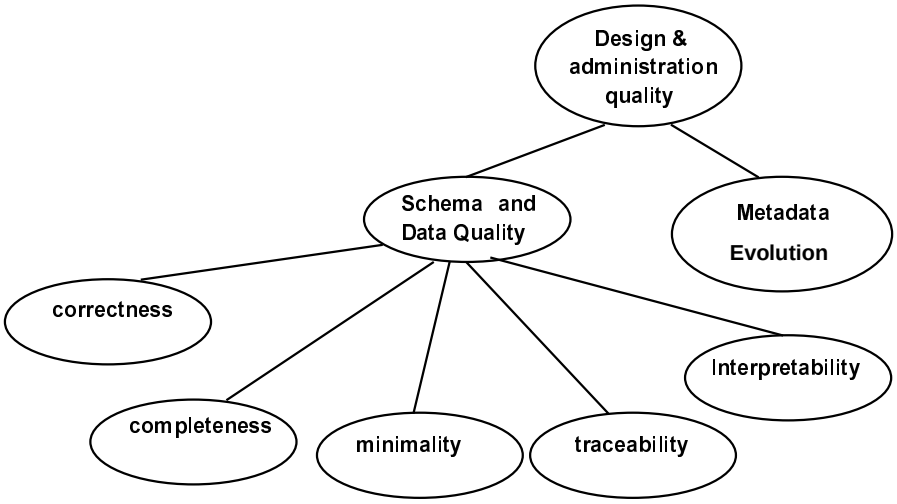

Fig. 9: Design and administration quality dimensions

Design and Administration Quality. The design and administration quality can be analyzed into more detailed dimensions, as depicted in figure 9. The *schema and data quality* refers to the ability of a schema or model to represent adequately and efficiently the information; the same criteria also apply at the data instance level. The *correctness* dimension is concerned with the proper comprehension of the entities of the real world, the schemata of the sources (models) and 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 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 *meta data evolution* dimension is concerned with the way the schema evolves during the data warehouse operation. Table 1 relates the quality dimensions to data warehouse objects and shows how the quality of these objects can be measured.

Design and Administration Quality	**Conceptual Perspective**		**Logical Perspective**	
	Model	**Concept**	**Schema**	**Type**
Correctness	Number of conflicts to other models/real world	Correctness of the description wrt. real world entity	Correctness of mapping of the conceptual model to logical schema	Correctness of the mapping of the concept to a type
Completeness	Level of covering, number of represented business rules	Number of missing attributes; Are the assertions related to the concept complete?	Number of missing entities wrt. conceptual model	Number of missing attributes wrt. conceptual model
Minimality	Number of redundant entities/relationships in a model	Equivalence of the description with that of other concepts in the same model	Number of redundant relations	Number of redundant attributes
Traceability	Are the designer's requirements and changes recorded?	Are the designer's requirements and changes recorded?	Are the designer's requirements and changes recorded?	Are the designer's requirements and changes recorded?
Interpretability	Quality of documentation	Quality of documentation	Quality of documentation	Quality of documentation
Metadata Evolution	Is the evolution of the model documented?	Is the evolution of the concept documented?	Is the evolution of the schema documented?	Is the evolution of the type documented?

Table 1: Examples for measurement types for design and administration quality dimensions

Software Implementation 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, but adopt the ISO 9126 standard [20]. The quality dimensions of ISO 9126 are *Functionality* (Suitability, Accuracy,

Interoperability, Compliance, Security), *Reliability* (Maturity, Fault tolerance, Recoverability), *Usability* (Understandability, Learnability, Operability), *Software Efficiency* (Time behavior, Resource Behavior), *Maintainability* (Analyzability, Changeability, Stability, Testability), *Portability* (Adaptability, Installability, Conformance, Replaceability).

These quality dimensions apply only to the physical perspective of the architectural, where the software (agents and data stores) are represented. Table 2 gives some examples how these quality dimensions can be measured for specific components.

Software Implementation Quality	**Physical Perspective** **DW Component**
Functionality	Number of functions not appropriate for specified tasks, number of modules unable to interact with specified systems
Reliability	Frequency of failures, Fault tolerance
Usability	Acceptance of the users
Software Efficiency	Performance, response time, processing time
Maintainability	Man-hours needed for maintaining and testing this software
Portability	Number of cases where the software failed to adopt to new environments; man-hours needed to install software in new environments

Table 2: Examples for measurement types for software implementation quality dimensions

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. In figure 10 the hierarchy of quality dimensions related to data usage is depicted.

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. *System availability* 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.

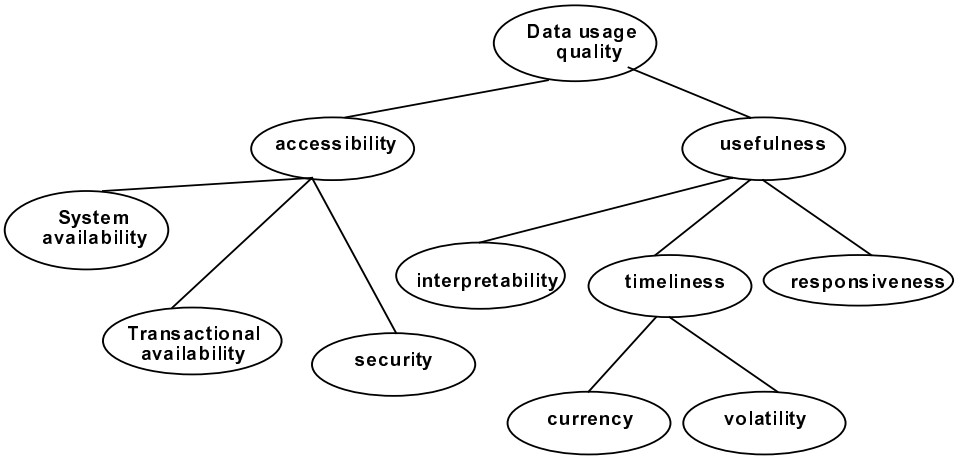

Fig. 10: Data usage quality dimensions

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. The *volatility* dimension describes the time period for which the information is valid in the real world. 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. In table 3, some examples are shown how data usage quality can be measured.

Data Usage Quality	Logical Perspective		Physical Perspective	
	Schema	**Type**	**Agent**	**Data Store**
Accessibility	Is the schema definition accessible by the users?	Is the type visible and accessible for users?	Is the network sufficient for delivered data?	Is the data store accessible?
Availability	Frequency of updates	Frequency of updates	Response time	Uptime of data store, response time
Security	Level of security (access rights)	Level of security (access rights)	Are there physical access restrictions?	Is the store able to prevent unauthorized access?
Usefulness	Is the schema used by any users?	Is the type used by any users?	Is the data delivered by the agent really used in the destination store?	Is the data in this store queried by a user?
Interpretability	Is the schema understandable?	Is the type understandable?	Is the data delivered understandable?	Is the data stored understandable?

Table 3: Examples for measurement types for data usage quality dimensions

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. We define data quality as a small subset of the dimensions proposed in other models. For example, in [48] our notion of data quality, in its greater part, is treated as a second level dimension, namely *believability*. The basic quality dimensions we introduce are shown in figure 11.

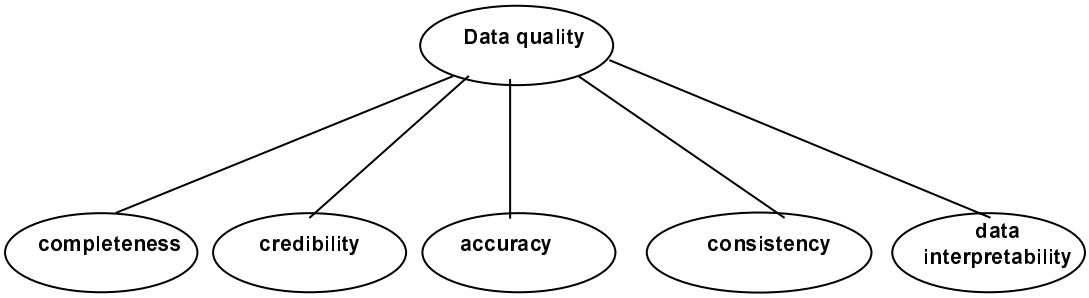

Fig.11: Data Quality Dimensions

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 the physical perspective of the the architecture representing data stores and agents 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.). Some metrics for data quality are given in table 4.

Data Quality	Physical Perspective	
	Agent	**Data Store**
Completeness	Number of tuples delivered wrt. expected number	Number of stored null values where there are not expected
Credibility	Believability in the process that delivers the values	Number of tuples with default values
Accuracy	Number of delivered accurate tuples	Level of preciseness; Number of accurate tuples
Consistency	Is the delivered data consistent with other data	Number of tuples violating constraints, number of coding differences
Data Interpretability	Number of tuples with interpretable data, documentation for key values, is the format understandable?	Number of tuples with interpretable data, documentation for key values, is the format understandable?

Table 4: Examples for measurement types for data quality dimensions

3.2. The Problem of Heterogeneous Multi-Criteria Quality Assessment

We now turn to the formal handling and repository-based management of DW quality goals such as the ones described in the previous section.

A first formalization could be based on a qualitative analysis of relationships between the quality factors themselves, e.g. positive or negative goal-subgoal relationships or goal-means relationships. The stakeholders could then enter their subjective evaluation of individual goals as well as possible weightings of goals and be supported in identifying good trade-offs. The entered as well as computed evaluations are used as quality measurements in the architecture model of figure 3, thus enabling a very simple integration of architecture and quality model.

Such an approach is widely used in industrial engineering under the label of *Quality Function Deployment*, using a special kind of matrix representation called the House of Quality [1]. Formal reasoning in such a structure has been investigated in works about the handling of non-functional requirements in software engineering, e.g. [36]. Visual tools have shown a potential for negotiation support under multiple quality criteria [14].

However, while this simple approach certainly has a useful role in cross-criteria decision making, using it alone would throw away the richness of work created by research in measuring, predicting, or optimizing individual data warehouse quality factors. In the DWQ project, such methods are systematically adopted or newly developed for all quality factors found important in the literature or our own empirical work. To give an impression of the richness of techniques to be considered, we use a single quality factor – responsiveness in the sense of good query performance – for which the DWQ project has studied three different approaches, one each from the conceptual, logical, and physical perspective.

We start with the logical perspective [43]. Here, the quality measurement associated with responsiveness is taken to be a weighted average of query and update "costs" for a given query mix and given information sources. A combinatorial optimization technique is then proposed that selects a collection of materialized views as to minimize the total costs. This can be considered a very simple case of the Quality Function Deployment approach, but with the advantage of automated design of a solution.

If we include the physical perspective, the definition of query and update "costs" becomes an issue in itself: what do we mean by costs – response time, throughput, or a combination of both (e.g. minimize query response time and maximize update throughput)? what actually produces these costs – is database access or the network traffic the bottleneck? A comprehensive queuing model [39] enables the prediction of such detailed metrics from which the designer can choose the right ones for quality measurements for his design process. In addition, completely new design options come into play: instead of materializing more views to improve query response time (at the cost of disturbing the OLTP systems longer at update time), the designer could buy a faster client PC or DBMS, or provide an ISDN link rather than using slow modems.

Yet other options come into play, when a rich logic is available for handling the conceptual perspective. For example, the description logic developed in the DWQ project for source integration [8] allows to state that information about all instances of one concept in the enterprise model is maintained in a particular information source. In other words, the source is complete with respect to the domain. This enables the DW designer to drop the materialization of all views on other sources, thus reducing the update effort semantically without any loss in completeness of the answers.

3.3. Hierarchical Quality Assessment: An Adapted GQM Approach

It is clear (and has in fact been proven in [7]) that there can be no decidable formal framework that even comes close to covering all of these aspects in a uniform language. When designing the meta database extensions for quality management, we therefore had to look for another solution that still maintains the overall picture offered by shallow quality management techniques such as QFD but is at the same time open for the embedding of specialized assessment and design techniques.

Our solution to this problem builds on the widely used Goal-Question-Metric (GQM) approach in software quality management [40]. The idea of GQM is that quality *goals* can usually not be assessed directly. Instead, their meaning is circumscribed by *questions* that need to be answered when evaluating the quality. Quality

questions again can usually not be answered directly but rely on *metrics* applied to either the product or process in question; techniques such as statistical process control charts are then applied to derive the answer of a question from the measurements.

In the above example, the *goal* of responsiveness can be refined into *questions* about the trade-off between query and update performance (logical perspective), about the present bottlenecks at the physical level, and about the completeness or even redundancy of the utilized data sources (conceptual perspective). These questions can then be answered using the above-mentioned *metrics and algorithms*.

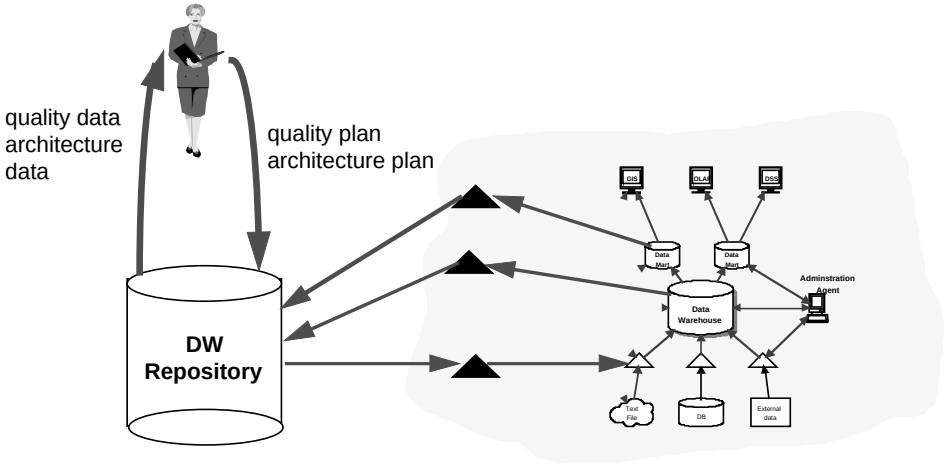

Fig. 12: Quality management via the data warehouse repository

Our repository solution uses a similar approach to bridge the gap between quality goal hierarchies on the one hand, and very detailed metrics and reasoning techniques on the other. The bridge is defined through the idea of quality queries as materialized views over the data warehouse; the views are defined through generic queries over the quality measurements. Figure 12 motivates this approach by zooming in on the repository. The stakeholder assesses the data warehouse quality by asking quality queries to the repository. The repository answers the queries by accessing quality data obtained from measurement agents (the black triangles in figure 12). The agents communicate with the components of the real data warehouse to extract measurements.

The stakeholder may re-define her quality goals at any time. This shall lead to an update of the quality model in the repository and possibly to the configuration of new measurement agents responsible to deliver the base quality data. Analogously, a stakeholder with appropriate authorization can re-define the architecture of the data warehouse via the repository. Such an evolutionary update, e.g. the specification of a new data source, leads to a re-configuration of the real data warehouse. Ultimately, the quality measurements will reflect such effect of the change and give evidence whether the evolution has led to an improvement of some quality goals.

The use of the repository for data warehouse quality management has significant advantages:

- data warehouse systems already incorporate repositories to manage meta data about the data warehouse; extending this component for quality management is a natural step
- existing meta data about the data warehouse, e.g. source schemas, can be directly used for formulating quality goals and measurement plans
- the quality model can be held consistent with the architecture model, i.e. the repository can prevent the stakeholders to formulate quality goals that cannot be validated with the given architectural data
- the stakeholder accesses the repository as a *data source* to deliver quality reports to the stakeholders who formulate quality goals; in fact, producing such reports is the same kind of activity that is used to deliver aggregated data to the client tools of a data warehouse

The last argument is not just a technical remark. Quality data, i.e. values of quality measurements, are derived from DW components. The values are materialized views of properties of these components. These values do

have quality properties like timeliness and accuracy themselves. It makes a difference whether value of a quality measurement is updated each hour or once a month. While we do not go into detail with this "second-level" quality, we note that the same methods that are used to maintain quality of the DW can also be used to maintain the quality of the DW repository (hosting the quality model).

3.4. The Quality Meta Model

Quality data is derived data and is maintained by the data warehouse system. This implementation strategy provides more technical support than GQM implementations for general software systems. Such system lack the built-in repository. The expressive query language offered by the ConceptBase repository system makes a large portion of quality management tasks a matter of query formulation. In the sequel, we elaborate how a version of GQM can be modeled by Telos meta classes in ConceptBase and then be used for quality goal formulation and quality analysis.

Telos provides a logical representation for class membership *(x in class),* specialization between classes *(c isA d),* and attributes *(x label y).* This logical representation can be mapped to a graphical layout as shown for the quality model below, as well as to a frame syntax which we sometimes use for the formulation of queries. Since all items (objects, classes, meta classes, and attributes) are uniformly treated in the logical representation, the Telos language is used – extending the approach shown in figure 5 – for formulating

1. a meta model by a collection of meta classes (here for defining the architecture and quality models),
2. a collection of classes (here the use of the architecture and quality meta models to express quality goals, queries, and measurement types on DW components), and
3. instances of the classes (here for representing results of measurements as class instances).

Data warehouse systems are unique in the sense that they rely on a run-time meta database (or repository) that stores information about the data and processes in the system. This opens the opportunity to implement the GQM approach such that it directly refers to the concepts in the meta database of the data warehouse.

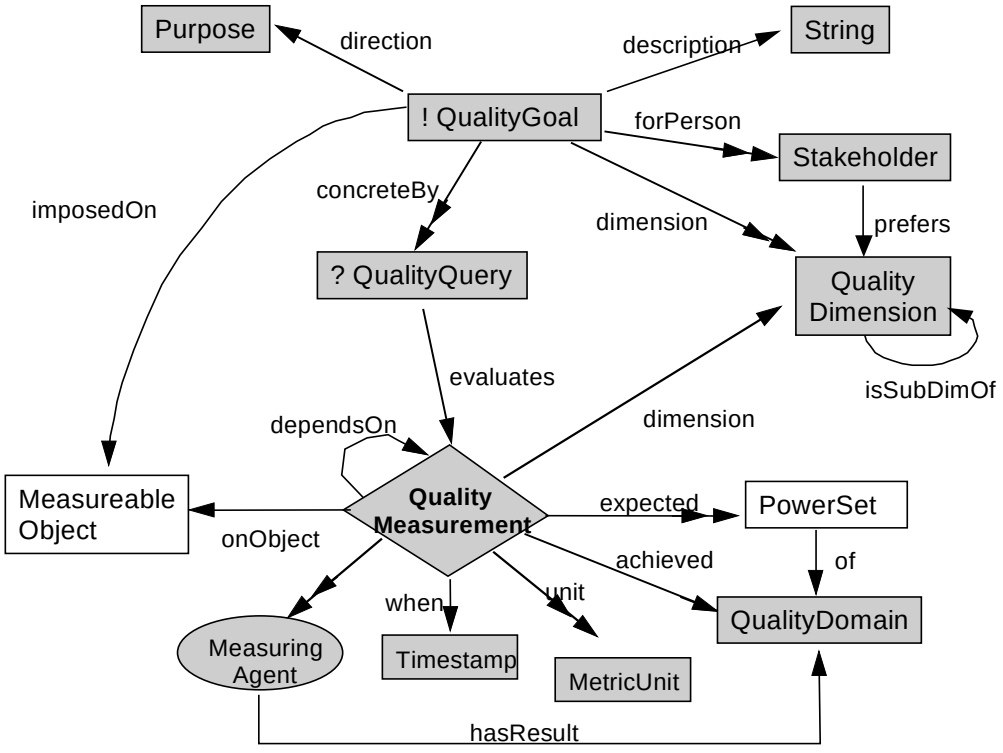

Fig. 13: A meta model for data warehouse quality

Figure 13 shows the Telos meta classes for managing data warehouse quality. Quality goals, e.g. 'improve the timeliness of data set sales-per-month', are assigned to stakeholders. The *purpose* attribute for quality goals is used to specify the intended direction of quality improvement (e.g., to increase the quality or to achieve a certain quality level at a certain time). The quality goal is imposed on measurable data warehouse objects as classified by the architecture model of figure 4. *Quality goals* are mapped to a collection of *quality queries* which are used to decide whether a goal is achieved or not. In our version of the GQM, these queries are queries to the DW repository. A quality goal is linked to one or more *quality dimensions* according to the preferences of the stakeholder who formulates the goal (see figures 9-11).

The next key concept is the quality query. While this is just a text in the original GQM approach, we encode a quality query as an executable query on the data warehouse repository using the expressive deductive query language of ConceptBase. The answer to a quality query is regarded as evidence for the fulfillment of a quality goal. The most simple kind of quality query would just evaluate whether the current *quality measurement* for a data warehouse object is within the expected interval. A quality measurement uses a metric unit, e.g. the average number of null values per tuple of a relation.

3.5. Implementation Support for the Quality Meta Model

The abstraction levels of the concepts in the quality model require a closer consideration [28]. In standard software metrics, a *quality measurement* is a function that maps a real world entity to a value of a domain, usually a number. In our case, we maintain abstract representations of all "interesting" real world entities in the DW repository itself. Thus, quality measurements can be recorded as explicit relationships between the abstract representations, i.e. measurable objects, and the quality values. By nature, such a quality measurement relates objects of different abstraction levels. For example, a quality value of 0.8 could be measured for the percentage of null values of the *Employee* relation of some data source. *Employee* is a relation (the type of instances of the *Employee* data structure) whereas 0.8 is just a number. For this reason, we require a framework like Telos which is able to relate objects at different abstraction levels.

A second remark has to be made on the use of the quality model by instantiation. Typical instances of the *MeasurableObject* are items like *Relation* (logical perspective) or entity type (conceptual perspective). These items are independent of the DW application domain. They are used to describe a DW architecture but they are not components of a concrete DW architecture[1]. A concrete architecture consists of items like data source for *Employee*, concrete wrapper agents etc. Therefore, when we instantiate the quality model we describe types of quality goals, types of queries, and types of measurements. For example, we can describe a completeness goal for relational data sources (instances of the *Relation* concept in figure 4) which is measured by counting the percentage of null values in the relation. Such types (or patterns) can be reused for any concrete DW architecture. For example, the measurement for a relational source for *Employee* would be instantiated from the measurement type by instantiating the expected and achieved quality values. The quality factors listed in the tables 1 to 4 are such measurement types and they need to be instantiated by concrete measurements. This two-step instantiation is essential in our approach since it allows to pre-load the repository with quality goal, query and measurement types independent of the application domain. In other words, the repository has knowledge about quality management methods.

Quality goals – whose dimensions are organized in hierarchies such as shown in figures 9 to 11 – are made operational as types of queries defined over quality measurements. These queries will support the evaluation of a specific quality goal when parameterized with a given (part of a) DW meta database. Such a query usually compares the analysis goal to a certain expected interval in order to assess the level of quality achieved.

As a consequence, the quality measurement must contain information about both expected and actual values. Both could be entered into the meta database manually, or computed inductively by a given metric through a specific reasoning mechanism. For example, for a given physical design and some basic measurements of

---

[1] Formally, this is expressed by means of class instantiation in Telos. The concept *Relation* is represented by a tuple *(Relation in MeasurableObject)*. The concept *Employee* is introduced in Telos by a tuple *(Employee in Relation)*. Thus, *MeasurableObject* is a meta class of *Employee*.

component and network speeds, the queuing model in [39] computes the quality measurement response time and throughput, and it could indicate if network or database access is the bottleneck in the given setting. This could then be combined with conceptual or logical quality measurements at the level of optimizing the underlying quality goal.

A number of quality queries have been developed, focusing on some that turned out to be relevant when validating the architecture against three case studies: creating a model of Software AG's SourcePoint DW loading environment, modeling the data quality problems hindering the application of data mining techniques in Swiss Life, and conceptually re-constructing some design decisions underlying the administrative data warehouses of the City of Cologne, Germany. Details about these case studies can be found in [11][41].

Generally speaking, quality queries access information recorded by quality measurements. A quality measurement stores the following information about data warehouse components:

1. an interval of expected values
2. the achieved quality measurement
3. the metric used to compute a measurement
4. causal dependencies to other quality measurements

The dependencies between quality measurements can be used to trace quality problems, i.e. measurements that are outside the expected interval, to their causes. The following two ConceptBase queries exemplify how quality measurements classify data warehouse components and how the backtracing of quality problems can be done by queries to the meta database:

```
QualityQuery BadQuality isA QualityMeasurement
 with constraint
      c: $ not (this.expected contains this.current) $
end

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

3.6. Understanding, Controlling and Improving Quality with the Repository

Summarizing the discussion above, figure 14 gives an impression how the traditional architecture of figure 1 is extended by our repository centred meta data management approach. The quality model forms the basis of the implementation in ConceptBase. Quality data (i.e., values of measurements) are entered into the ConceptBase system by external measurement agents which are specialized analysis and optimization tools. In the DWQ project, four such tools are developed. Besides the subsumption reasoning tools already mentioned in section 2.6, they include a data freshness toolkit covering the physical modeling of source integration, and tools for reasoning about multi-dimensional aggregates and query optimization on the client side. ConceptBase can trigger these agents based on the timestamp associated to them in the repository (see figure 14).

The result of the analysis of the quality data can be displayed graphically, as shown in figure 15. Quality measurements are the long ovals in the middle. The black oval indicates that the timeliness of the staff department data store (an item of the physical perspective) is not in its expected range (12 instead of 0 to 10). The white color of the other measurements indicate measurements that are in expected range. The color code of the graphical view is computed by the repository based on the *BadQuality* query shown above.

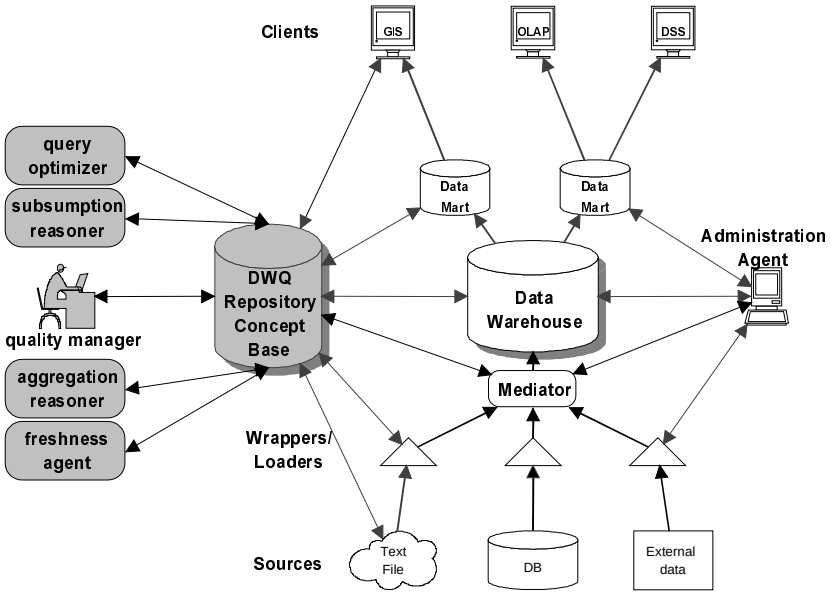

Fig. 14: Mapping the Extending Architecture and Quality Model to the Traditional DW Architecture

The graphical display is intended for controlling the quality of the data warehouse. The 'black' nodes indicate locations where some ad hoc *control* is required, or where stakeholders have to be aware of unexpected low quality. Each stakeholder has her own quality goals and hence has individualized views on the quality. The repository can also be used to maintain the knowledge about causes of quality measurements. The 'dependsOn' link in figure 13 is exactly intended to build such a symptom-to-cause model over the quality measures. Such a mathematical model shall be used to *understand* the effects of certain measures to other (dependent) measures. As soon as the mathematical models are coded into the repository, the can be used to forecast derived quality measures. If derived and measured values coincide for the same parameter, then the model is validated. This issue is still under research in the data warehouse area, however.

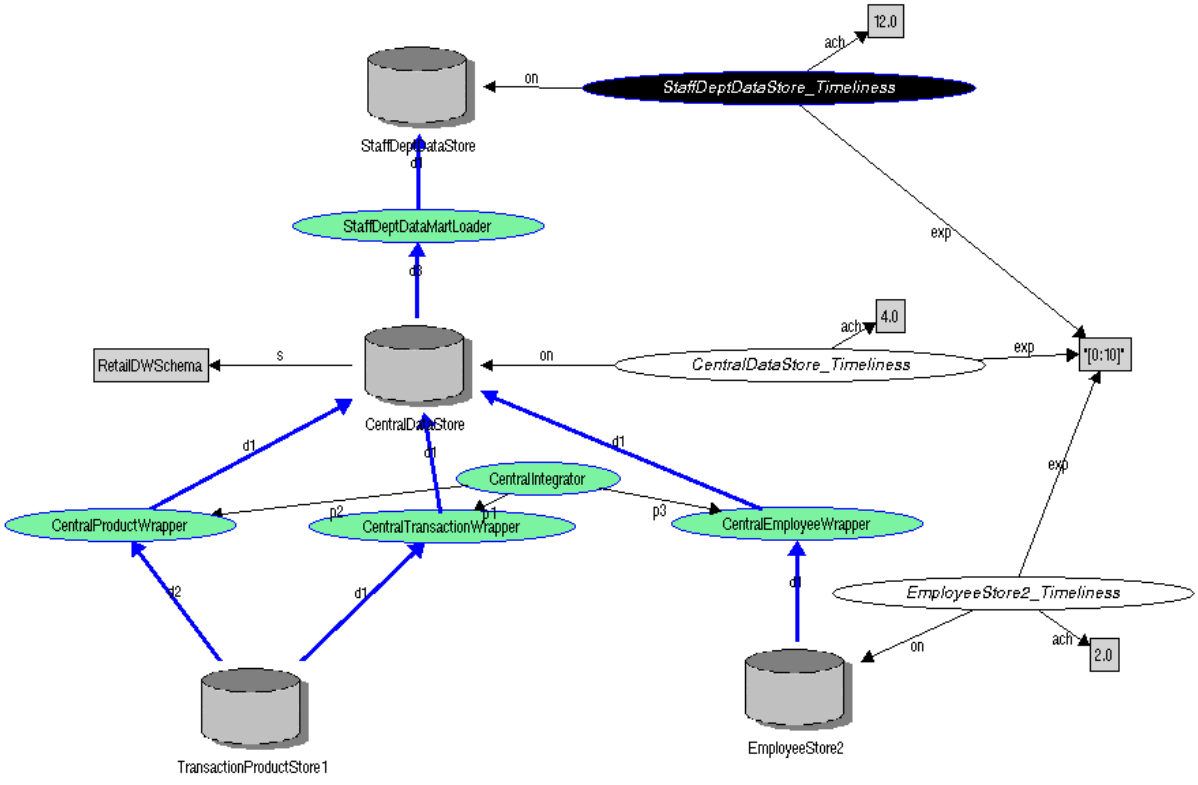

Fig. 15: ConceptBase screenshot of the graphical view on the quality data

The last and most advanced aspect of quality management is the *improvement*. Our current model does not contain constructive knowledge about how to improve the quality of a data warehouse. The first step is to incorporate the mathematical model mentioned above. Then, a data warehouse designer can make incremental changes to the data warehouse architecture, measure the local effect on quality, and then measure the effect on derived quality measures.

4. RELATED WORK

Our approach extends and merges results from data warehouse research and from data/software quality research. We mention here only some of the most relevant approaches; a comprehensive survey of research and practice in data warehousing appears in [23].

Starting with the *data warehouse literature*, the well-known projects have focused almost exclusively on what we call the logical and physical perspectives of DW architecture. While the majority of early projects have focused on source integration aspects, the recent effort has shifted towards the efficient computation and re-computation of multi-dimensional views. The business perspective is considered at best indirectly in these projects. The *Information Manifold* (IM) developed at AT&T is the only one that employs a rich domain model for information gathering from disparate sources such as databases, SGML documents, or unstructured files [29][32][33] in a manner similar to our approach (but with less powerful reasoning mechanisms for analysis).

TSIMMIS (*The Stanford-IBM Manager of Multiple Information Sources*) is a project with the goal of providing tools for the integrated access to multiple and diverse information sources and repositories [10][44]. Each information source is equipped with a *wrapper* that encapsulates the source, converting the underlying data objects to a common data model - called *Object Exchange Model* (OEM). On top of wrappers, *mediators* [47] can be conceptually seen as views of data found in one or more sources which are suitably integrated and processed.

Similarly, but with slightly different implementation strategies, the *Squirrel* Project [18][50] provides a framework for data integration based on the notion of *integration mediator*. Integration mediators are active modules that support incrementally maintained integrated views over multiple databases. Moreover, data quality is considered by defining formal properties of *consistency* and *freshness* for integrated views.

The WHIPS (*WareHouse Information Prototype at Stanford*) system [16][46] has the goal of developing algorithms for the collection, integration and maintenance of information from heterogeneous and autonomous sources. The WHIPS architecture consists of a set of independent modules implemented as CORBA objects. The central component of the system is the *integrator*, to which all other modules report.

On the client side of data warehousing, numerous tools for multi-dimensional data modeling and querying exist. In terms of our architecture model, most of them have addressed a logical perspectives, e.g. relational algebras [45], SQL extensions by data cubes [15] or visualization techniques generalizing the spreadsheet approach [14]. However, there is also some work on logical foundations of a conceptual level, as a basis for DW design [31] as well as DW operation [6]. In the DWQ project, a unified approach capturing the essence of these extensions is under construction, as an extension to the repository meta model similar to the one described in section 2.6.

Interestingly, metadata support for multi-dimensional extensions as well as for the representation of what we call acquisition plans is offered by the new version of the Microsoft Repository [3]. However, as the MS Repository is based on binary-standard object-oriented program interfaces on top of relational storage technologies, it does not offer deductive querying mechanisms or subsumption analysis techniques that support quality management in our approach. Still, this recent commercial effort accentuates the importance allocated by vendors to the question of repository support for data warehousing.

Turning to *data quality research*, Wang et al. [49] present a framework of data quality analysis, based on the ISO 9000 standard. 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. In [48], an attribute-based model is presented that can be used to incorporate quality aspects of data products. As in our approach, the basis is the assumption that the quality design of an information system should

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 measurements. Further work on data quality can be found, among others, in [5][21][30][34].

Variants of the Goal-Question-Metric (GQM) approach are widely used in software quality management [37][12]. A structured overview of the issues and strategies for information systems quality, embedded in a repository framework, can be found in [24]. Several hierarchies of quality dimensions have been proposed. For example, the GE Model [37] suggests 11 criteria of software quality, while B. Boehm's [4] suggests 19 quality factors. ISO 9126 [20] suggests six basic quality factors which are further analyzed to an overall of 21 quality factors. In [17] a comparative presentation of these three models is done and the SATC software quality model is presented, along with the metrics for the software quality dimensions.

5. DISCUSSION AND CONCLUSIONS

The goal of our work is to enrich meta data management in data warehouses such that it can serve as a meaningful basis for systematic quality analysis and quality-driven design. To reach this goal, we had to overcome two limitations of current data warehouse research.

Firstly, the basic architecture in which data warehouses are typically described turned out to be too weak to allow a meaningful quality assessment. As quality is usually detected only by its absence, quality-oriented meta data management requires that we address the full sequence of steps from the capture of enterprise reality in operational departments to the interpretation of DW information by the client analyst. This in turn implied the introduction of an explicit enterprise perspective as a central feature in the architecture. To forestall possible criticism that full enterprise modeling has proven a risky and expensive effort, we recall from section 2.6 that our approach to enterprise model formation is fully incremental such that it is perfectly feasible to construct the enterprise model step by step, e.g. as a side effect of source integration or of other business process analysis efforts.

The second major problem is the enormous richness in quality factors, each associated with its own wealth of measurement and design techniques. Our quest for an open quality management environment that can accommodate existing or new such techniques led us to an adaptation and repository integration of the GQM approach where parameterized queries and materialized quality views serve as the missing link between specialized techniques and the general quality framework.

The power of the repository modeling language determines the boundary between precise but narrow metrics and comprehensive but shallow global repository. The deductive object base formalism of the Telos language provides a fairly sophisticated level of global quality analysis in our prototype implementation but is still fully adaptable and general. Once the quality framework has sufficiently stabilized, a procedurally object-oriented approach could do even more, by encoding some metrics directly as methods, of course at the expense of flexibility. Conversely, a simple relational meta database could take up some of the present models with less semantics than offered in the ConceptBase system, but with the same flexibility.

As shown throughout the paper, the approach has been fully implemented and some validation has taken place to fine-tune the models. In part, this validation was by testing earlier versions of the model in real-world DW projects, such as [42], or by reconstructing features of existing systems, such as [41]; another important strain of validation efforts is through the definition and validation of specific methodologies within our framework, such as the source integration methodology discussed in section 2.6 [8].

Obviously, much remains to be done. One direction of current work therefore continues the validation against several major case studies, in order to set priorities among the quality criteria to be explicated in specific metrics and analysis techniques. A second overlapping strain concerns the development of these techniques themselves, and their linkage into the overall framework through suitable quality measurements and extensions to global design and optimization techniques. Especially when progressing from the definition of metrics and prediction techniques to actual design methods, it is expected that these will not be representable as closed algorithms but must take the form of interactive work processes defined over the DW architecture.

As an example, feedback from at least two case studies suggests that, in practice, the widely studied strategy of incremental view maintenance in the logical sense is far less often problematic than the time management at the physical and conceptual level, associated with the question when to refresh DW views such that data are sufficiently fresh for analysis, but neither analysts nor OLTP applications are unduly disturbed in their work due to locks on their data. Our research therefore now focuses on extending the conceptual level by suitable (simple) temporal representation and reasoning mechanisms for representing freshness requirements, complemented by an array of design and implementation methods to accomplish these requirements and the definition of processes at the global level to use these methods in a goal-oriented manner to fulfill the requirements.

As another example, one of our industrial cooperation partners – a small data warehouse application vendor – has recognized that data quality for data analysis is not enough, because data analysis is only meaningful if it also results in operational action. Jointly with this company, we are therefore devising a process and repository implementation which allows to propagate the application of the data quality techniques in the data warehouse "backwards" into the information sources [25].

While such extensions will certainly refine the approach reported here, the experiences gained so far indicate that it is a promising way towards more systematic and computer-supported quality management in data warehouse design and operation.

Acknowledgements — This research was partially supported by the European Commission in ESPRIT Long Term Research Project 22469 DWQ (Foundations of Data Warehouse Quality, http://www.dbnet.ece.ntua.gr/~dwq/), by the General Secretariat of Research and Technology (Greece) under the PENED program; and by the Deutsche Forschungsgemeinschaft through Graduiertenkolleg "Informatik und Technik". The authors would like to thanks their project partners in DWQ, especially Maurizio Lenzerini, Mokrane Bouzeghoub and Enrico Franconi, for fruitful discussions of the architecture and quality model.

REFERENCES

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

[2] Arbor Software Corporation. *Arbor Essbase*. http://www.arborsoft.com/essbase.html (1996).

[3] P.A. Bernstein, Th. Bergstraesser, J. Carlson, S. Pal, P. Sanders, D. Shutt. Microsoft Repository Version 2 and the Open Information Model. *Information Systems* **24**(2) (1999).

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

[5] D.P. Ballou, R.Y. Wang, H.L. Pazer, and 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).

[6] L. Cabibbo, and R. Torlone. A logical approach to multidimensional databases. In *Proc. 6th Intl. Conf. Extending Database Technology (EDBT 98)*, Avignon, France, Springer Verlag, pp. 183-197 (1998).

[7] D. Calvanese, G. De Giacomo, and M. Lenzerini. Conjunctive query containment in Description Logics with n-ary relations. In *Proc. International Workshop on Description Logics*, Université Paris-Sud, Centre d'Orsay, Laboratoire de Recherche en Informatique LRI, URA-CNRS 410, Paris (1997).

[8] D. Calvanese, G. De Giacomo, and M. Lenzerini, D. Nardi, R. Rosati. Schema and data integration methodology for DWQ. DWQ Deliverable 4.1, ESPRIT Project 22469 Foundations of Data Warehouse Quality, University of Rome (1998).

[9] D. Calvanese, G. De Giacomo, and M. Lenzerini, D. Nardi, R. Rosati. Experimentation with the incremental view integration. DWQ Deliverable 3.3, ESPRIT Project 22469 Foundations of Data Warehouse Quality, University of Rome (1998).

[10] 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 *10th Meeting of the Information Processing Society of Japan (IPSJ)*, Tokyo, Japan, pp. 7-18, (1994).

[11] DWQ Consortium. *Deliverable D2.1, Data Warehouse Architecture and Quality Model.* Technical Report DWQ–RWTH-002 (1997).

[12] N.E. Fenton, and S. L. Pfleeger. *Software Metrics - A Rigorous & Practical Approach*. Second Edition, PWS Publ., Boston, MA. (1998).

[13] E. Franconi et al. Algorithms for Reasoning over Multidimensional Aggregation. DWQ Deliverable D 6.1, ESPRIT Project 22469 Foundations of Data Warehouse Quality, DWQ Consortium, 1998.

[14] M. Gebhardt, M. Jarke, and S. Jacobs. CoDecide – a toolkit for negotiation support interfaces to multi-dimensional data. In *Proc. ACM-SIGMOD Conf. Management of Data*, Tucson, Arizona, ACM Press, pp. 348-356 (1997).

[15] J. Gray, A. Bosworth, A. Layman, H. Pirahesh. Data cube: a relational aggregation operator generaizing group-by, cross-tab, and sub-totals. *20th Intl. Conf. Data Engineering*, New Orleans, IEEE Computer Society, pp.152-159 (1996).

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

[17] L. Hyatt, and L. Rosenberg. A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality. *ESA 1996 Product Assurance Symposium and Software Product Assurance Workshop*, European Space Agency, ESTEC, Noordwijk, The Netherlands, pp. 209-212 (1996).

[18] R. Hull, and G. Zhou. A Framework for supporting data integration using the materialized and virtual approaches. In *Proc. ACM SIGMOD Intl. Conf. Management of Data*, ACM Press, pp. 481-492, Montreal (1996).

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

[20] ISO/IEC 9126. *Information technology -Software product evaluation- Quality characteristics and guidelines for their use*. International Organization for Standardization, http://www.iso.ch/ (1991).

[21] M. Janson. Data quality: The Achilles heel of end-user computing. *Omega J. Management Science*, **16**(5), (1988).

[22] M. Jarke, R. Gallersdörfer, M.A. Jeusfeld, M. Staudt, and S. Eherer. ConceptBase - a deductive objectbase for meta data management. *Journal of Intelligent Information Systems*, **4**(2):167-192 (1995).

[23] M. Jarke, M. Lenzerini, Y. Vassiliou, P. Vassiliadis (eds.). *Fundamentals of Data Warehouse Quality*. Springer-Verlag, to appear 1999.

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

[25] M. Jarke, C. Quix, G. Blees, D. Lehmann G. Michalk, and S. Stierl. Improving OLTP data quality using data warehouse mechanisms. To appear in *Proc. ACM-SIGMOD Conf. Management of Data*, ACM Press. Philadelphia, PA. (1999).

[26] M. Jarke, Y. Vassiliou. Foundations of data warehouse quality – a review of the DWQ project. In *Proc. 2nd Intl. Conf. Information Quality (IQ-97)*, Cambridge, Mass. (1997).

[27] M.A. Jeusfeld, M. Jarke, H.W. Nissen, and M. Staudt. ConceptBase - Managing conceptual models about information systems. In P. Bernus, K. Mertins, and G. Schmidt: *Handbook on Architectures of Information Systems*, pp. 265-285, Springer-Verlag (1998).

[28] M.A. Jeusfeld, C. Quix, and M. Jarke. Design and Analysis of Quality Information for Data Warehouses. In *Proc. 17th International Conference on Conceptual Modeling (ER'98)*, Springer Verlag, Singapore, pp. 349-362 (1998).

[29] T. Kirk, A.Y. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. In *Working Notes of AAAI 1995 Spring Symp. on Information Gathering from Heterogeneous, Distributed Environments*, pp. 85-91 (1995).

[30] C. Kriebel, Evaluating the quality of information system, *Design and Implementation of Computer Based Information Systems*, N. Szyperski/ E.Grochla ,eds. Sijthoff and Noordhoff (1979).

[31] W. Lehner. Modeling large scale OLAP scenarios. *Proc. 6th Intl. Conf. Extending Database Technology (EDBT'98)*, Valencia, Spain, Springer-Verlag, pp. 153-167 (1998).

[32] A.Y. Levy, A. Rajaraman, and J. J. Ordille. Query answering algorithms for information agents. In *Proc. 13th Nat. Conf. on Artificial Intelligence (AAAI-96)*, Portland, Oregon, AAAI Press/MIT Press, pp. 40-47 (1996).

[33] A.Y. Levy, D. Srivastava, and T. Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems*, **5**(2):121-143 (1995).

[34] G.E. Liepins, and V.R.R. Uppuluri, Accuracy and Relevance and the Quality of Data. In *A.S. Loebl*, ed., vol. 112, Marcel Dekker (1990).

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

[36] J. Mylopoulos, L. Chung, and B. Nixon. Representing and using non-functional requirements – a process-oriented approach. *IEEE Trans. Software Engineering*, **18**(6):483-497 (1992).

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

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

[39] M. Nicola, and M. Jarke. Increasing the Expressiveness of Analytical Performance Models for Replicated Databases. To appear in *Proc. International Conference on Database Theory (ICDT '99)*, Jerusalem, February 1999.

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

[41] Software AG. *SourcePoint White Paper*. Software AG, Uhlandstr 12, 64297 Darmstadt, Germany (1996).

[42] M. Staudt, J.U. Kietz, and U. Reimer. ADLER: An Environment for Mining Insurance Data. In *Proc. 4th Workshop KRDB-97*, Athens, August 1997, Online http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-10/.

[43] D. Theodoratos, and T. Sellis. Data Warehouse Configuration. In *Proc. 23th VLDB Conference*, Athens, Greece, pp. 126-135, Morgan Kaufmann (1997).

[44] J.D. Ullman. Information integration using logical views. In *Proc. 6th Int. Conf. on Database Theory (ICDT-97)*, pp. 19-40, Springer-Verlag, Delphi, Greece (1997).

[45] P. Vassiliadis. Modeling multidimensional databases, cubes, and cube operations. In *Proc. 10th Intl. Conf. Scientific and Statistical Database Management*, Capri, Italy, IEEE Computer Society, pp. 53-62 (1998).

[46] J. L. Wiener, H. Gupta, W. J. Labio, Y. Zhuge, H. Garcia-Molina, and J. Widom. A System Prototype for Warehouse View Maintenance. In *Proceedings ACM Workshop on Materialised Views: Techniques and Applications*, Montreal, Canada, pp. 26-33 (1996).

[47] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, **25**(3):38-49 (1992).

[48] R.Y. Wang, M.P. Reddy, and H.B. Kon. Towards quality data: an attribute-based approach, *Decision Support Systems*, **13**(3/4), pp. 349-372, (1995).

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

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