DATA WAREHOUSE PROCESS MANAGEMENT

Panos Vassiliadis[1], Christoph Quix[2], Yannis Vassiliou[1], Matthias Jarke[2,3]

[1] National Technical University of Athens, Dept. of Electrical and Computer Eng., Computer Science Division, Iroon Polytechniou 9, 157 73, Athens, Greece
{pvassil,yv}@dbnet.ece.ntua.gr
[2] Informatik V (Information Systems), RWTH Aachen, 52056 Aachen, Germany
{quix,jarke}@informatik.rwth-aachen.de
[3] GMD-FIT, 53754 Sankt Augustin, Germany

Abstract – Previous research has provided metadata models that enable the capturing of the static components of a data warehouse architecture, along with information on different quality factors over these components. This paper complements this work with the modeling of the dynamic parts of the data warehouse. The proposed metamodel of data warehouse operational processes is capable of modeling complex activities, their interrelationships, and the relationship of activities with data sources and execution details. Moreover, the metamodel complements the existing architecture and quality models in a coherent fashion, resulting in a full framework for quality-oriented data warehouse management, capable of supporting the design, administration and especially evolution of a data warehouse. Finally, we exploit our framework to revert the widespread belief that data warehouses can be treated as collections of materialized views. We have implemented this metamodel using the language Telos and the metadata repository system ConceptBase.

Key words: data warehousing, process modeling, evolution, quality

1. Introduction

Data Warehouses (DW) integrate data from multiple heterogeneous information sources and transform them into a multidimensional representation for decision support applications. Apart from a complex architecture, involving data sources, the data staging area, operational data stores, the global data warehouse, the client data marts, etc., a data warehouse is also characterized by a complex lifecycle. In a permanent design phase, the designer has to produce and maintain a conceptual model and a usually voluminous logical schema, accompanied by a detailed physical design for efficiency reasons. The designer must also deal with data warehouse administrative processes, which are complex in structure, large in number and hard to code; deadlines must be met for the population of the data warehouse and contingency actions taken in the case of errors. Finally, the evolution phase involves a combination of design and administration tasks: as time passes, the business rules of an organization change, new data are requested by the end users, new sources of information become available, and the data warehouse architecture must evolve to efficiently support the decision-making process within the organization that owns the data warehouse.

All the data warehouse components, processes and data should be tracked and administered via a *metadata repository*. In [29], we presented a metadata modeling approach which enables the capturing of the *static* parts of the architecture of a data warehouse. The linkage of the architecture model to quality parameters (in the form of a quality model) and its implementation in the metadata repository ConceptBase have been formally described in [32]. [57] presents a methodology for the exploitation of the information found in the metadata repository and the quality-oriented evolution of a data warehouse based on the architecture and quality model. In this paper, we complement these results with metamodels and support tools for the *dynamic* part of the data warehouse environment: the *operational data warehouse processes*. The combination of all the data warehouse viewpoints is depicted in Fig. 1.

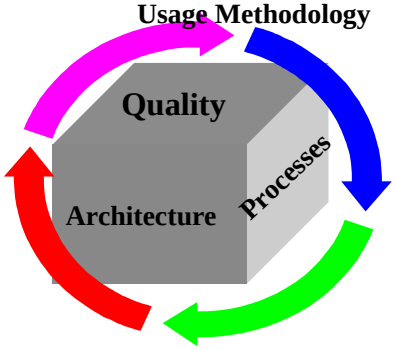

Fig. 1. The different viewpoints for the metadata repository of a data warehouse

In the three phases of the data warehouse lifecycle, the interested stakeholders need information on various aspects of the examined processes: *what* are they supposed to do, *how* are they implemented, *why* are they necessary and how they affect other processes in the data warehouse [68, 29]. Like the data warehouse architecture and quality metamodels, the process metamodel assumes the *clustering of their entities in logical, physical and conceptual perspectives*, each assigned with the task of answering one of the aforementioned stakeholder questions. In the rest of this section we briefly present the requirements faced in each phase, our solutions and their expected benefits.

The *design and implementation* of operational data warehouse process is a labor-intensive and lengthy procedure, covering thirty to eighty percent of effort and expenses of the overall data warehouse construction [55, 15]. For a metamodel to be able to efficiently support the design and implementation tasks, it is imperative to capture at least two essential aspects of data warehouse processes, *complexity of structure* and *relationship with the involved data*. In our proposal, the logical perspective is capable of modeling the structure of complex activities and capture all the entities of the widely accepted Workflow Management Coalition Standard [64]. The relationship of data warehouse activities with their underlying data stores is taken care of in terms of SQL definitions.

This simple idea reverts the classical belief that *data warehouses are simply collections of materialized views*. In previous data warehouse research, directly assigning a naïve view definition to a data warehouse table has been the most common practice. Although this abstraction is elegant and sufficient for the purpose of examining alternative strategies for view maintenance, it is incapable of capturing real world processes within a data warehouse environment. In our approach, we can deduce the definition of a table in the data warehouse table as the outcome of the combination of the processes that populate it. This new kind of definition complements existing approaches, since our approach provides the operational semantics for the content of a data warehouse table, whereas the existing ones give an abstraction of its intentional semantics.

The conceptual process perspective traces the reasons behind the structure of the data warehouse. We extend the demand-oriented concept of *dependencies* as in the Actor-Dependency model [68], with the supply-oriented notion of *suitability* that fits well with the redundancy found often in data warehouses. As an another extension to the Actor-Dependency model, we have generalized the notion of role in order to uniformly trace any person, program or data store participating in the system.

By implementing the metamodel in an object logic, we can exploit the query facilities of the repository to provide the support for consistency checking of the design. The deductive capabilities of ConceptBase [28] provide the facilities to avoid assigning manually all the interdependencies of activity roles in the conceptual perspective. It is sufficient to impose rules to deduce these interdependencies from the structure of data stores and activities.

While the design and implementation of the warehouse are performed in a rather controlled environment, the *administration* of the warehouse has to deal with problems that evolve in an ad-hoc fashion. For example, during the loading of the warehouse contingency treatment is necessary for the efficient administration of failures. In such events, not only the knowledge of the structure of a process is important; the specific *traces of executed processes* are also required to be tracked down – in an erroneous situation, not only the causes of the failure, but also the progress of the loading process by the time of the failure must be detected, in order to efficiently resume its operation. Still, failures during the warehouse loading are only the tip of the iceberg as far as problems in a data warehouse environment are concerned. This brings up the discussion on *data warehouse quality* and the ability of a metadata repository to trace it in an expressive and usable fashion. To face this problem, the proposed process metamodel is explicitly linked to our earlier quality metamodel [32]. We complement this linkage by mentioning specific quality factors for the quality dimensions of the ISO 9126 standard for software implementation and evaluation.

Identifying erroneous situations or unsatisfactory quality in the data warehouse environment is not sufficient. The data warehouse stakeholders should be supported in their efforts to react against these phenomena. The above-mentioned suitability notion in the conceptual perspective of the process metamodel allows the definition of recovery actions to potential errors or problems (e.g., alternative paths for the population of the data warehouse) in a straightforward way, during runtime.

Data warehouse evolution is unavoidable as new sources and clients are integrated, business rules change and user requests multiply. The effect of evolving the structure of the warehouse can be predicted by tracing the various interdependencies among the components of the warehouse. We have already mentioned how the conceptual perspective of the metamodel traces interdependencies between all the participants in a data warehouse environment, whether persons, programs or data stores. The prediction of potential impacts (whether of political, structural, or operational nature) is supported by this feature in several ways. To mention the simplest, the sheer existence of dependency links forecasts

a potential impact in the architecture of the warehouse in the presence of any changes. More elaborate techniques will also be provided in this paper, by taking into account the particular attributes that participate in these interdependencies and the SQL definitions of the involved processes and data stores. Naturally, the existence of suitability links suggests alternatives for the new structure of the warehouse. We do not claim that our approach is suitable for any kind of process, but focus our attention to the internals of data warehouse systems.

This paper is organized as follows: In Section 2 we present the background work and the motivation for this paper. In Section 3 we describe the process metamodel and in section 4 we present its linkage to the quality model. In section 5 we present how the metadata repository can be used for the determination of the operational semantics of the data warehouse tables and for evolution purposes. In section 6 we present related work and section 7 presents issues for future research.

2. Background and Motivation

In this section we will detail the background work and the motivation behind the proposed metamodel for data warehouse operational processes.

2.1. Background Work for the Metamodel: The Quest for Formal Models of Quality

For decision support, a data warehouse must provide high quality of data and service. Errors in databases have been reported to be up to ten percent range and even higher in a variety of applications. [65] 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 spend over 25% of their sales on wasteful practices, service companies up to 40%. In certain vertical markets (e.g., the public sector) data quality is not an option but a constraint for the proper operation of the data warehouse. Thus, data quality problems seem to introduce even more complexity and computational burden to the loading of the data warehouse. In the DWQ project (Foundations of Data Warehouse Quality [30]), we have attacked the problem of quality-oriented design and administration in a formal way, without sacrificing optimization and practical exploitation of our research results. In this subsection we summarize our results as far as needed for this paper.

In [29] a basic metamodel for data warehouse architecture and quality has been presented as in Fig. 2. The framework describes a data warehouse in *three perspectives*: a *conceptual*, a *logical* and a *physical* perspective. Each perspective is partitioned into the three traditional data *warehouse levels*: *source, data warehouse* and *client* level.

On the *metamodel* layer, the framework gives a notation for data warehouse architectures by specifying meta-classes for the usual data warehouse objects like data store, relation, view, etc. On the *metadata* layer, the metamodel is instantiated with the concrete architecture of a data warehouse, involving its schema definition, indexes, table spaces, etc. The lowest layer in Fig. 2 represents the actual processes and data.

The quality metamodel accompanying the architecture metamodel [32] involves an extension of the Goal-Question-Metric approach [47]. The metamodel introduces the basic entities around quality (including *Quality Goals, Quality Queries* and *Quality Factors*), the metadata layer is customized per warehouse with quality scenarios, and the instance layer captures concrete measurements of the quality of a given data warehouse.

The *static* description of the architecture (left part of Fig. 2) is complemented in this paper with a metamodel of the *dynamic* data warehouse operational processes. As one can notice in the middle of Fig. 2, we follow again a three-level instantiation: a *Process Metamodel* deals with generic entities involved in all data warehouse processes (operating on entities found at the data warehouse metamodel level), the *Process Model* covers the processes of a specific data warehouse by employing instances of the metamodel entities, and the *Process Traces* capture the execution of the actual data warehouse processes happening in the real world.

In [57], the Goal-Question-Metric (GQM) methodology has been extended in order (a) to capture the interrelationships between different quality factors with respect to a specific quality goal, and (b) to define an appropriate lifecycle that deals with quality goal evaluation and improvement. The methodology comprises a set of steps, involving the *design* of the quality goal, the *evaluation* of the current status, the *analysis* and *improvement* of this situation, and finally, the *re-evaluation* of the achieved plan. The metadata repository together with this quality goal definition methodology constitutes a decision support system which helps data warehouse designers and administrators to take relevant decisions, in order to achieve a reasonable quality level which fits the best user requirements.

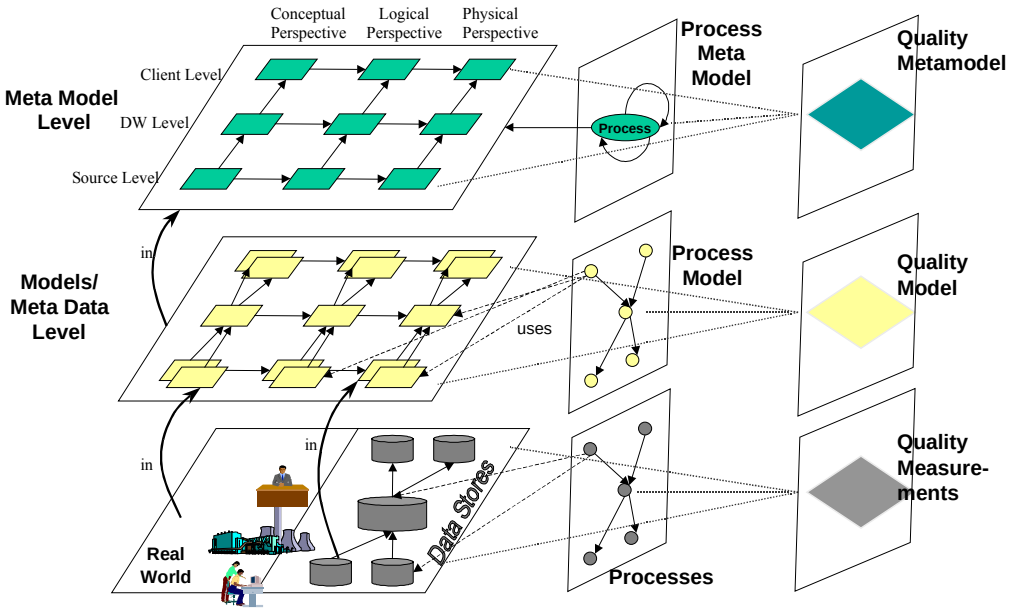

Fig. 2. Framework for Data Warehouse Architecture [29]

Throughout our models, we encourage the use of *templates* for process, quality and architecture objects. This is especially apparent in the metadata layer where an abstract specification of architecture, process or quality objects, originally coming from the designers of the data warehouse, can be properly specialized by the data warehouse administrator at runtime. For example, a high-level specification of a chain of activities involving extraction, checking for primary and foreign keys and final loading in the warehouse, can be customized to specific programs, later in the construction of the warehouse. Practical experience has shown that this kind of templates can be reoccurring in a data warehouse architecture, i.e., several tables can be populated through very similar programs (especially if the data are coming from the same source). The interested reader is referred to [49, 59] for examples of such templates.

2.2. The 3 Perspectives for the Process Model.

Our process model (cf. Fig. 3) follows the same three perspectives as the architecture model, since the perspectives of the process model operate on objects of the respective perspective of the architecture model. As mentioned in [68], there are different ways to view a process: *what* steps it consists of (logical perspective), *how* they are to be performed (physical perspective) and *why* these steps exist (conceptual perspective). Thus, we view a data warehouse process from three perspectives: a central *logical* part of the model, which captures the basic structure and data characteristics of a process, its *physical* counterpart which provides specific details over the actual components that execute the process and a *conceptual* perspective which abstractly represents the basic interrelationships between data warehouse stakeholders and processes in a formal way.

Typically, the information about how a process is executed concerns stakeholders who are involved in the everyday use of the process. The information about the structure of the process concerns stakeholders that manage it while the information relevant to the reasons behind this structure concerns process engineers who are involved in the monitoring or evolution of the process environment. Often, all these roles are covered by the data warehouse administration team, although one could also encounter different schemes.

Another important issue shown in Fig. 3 is that we can observe a data flow in each of the three perspectives. In the logical perspective, the modeling is concerned with the functionality of an activity, describing what this particular activity is about in terms of consumption and production of information. In the physical perspective, the details of the execution of the process are the center of the modeling. The most intriguing part, though, is the conceptual perspective covering why a process exists. The answer can be either due to necessity reasons (in which case, the receiver of information depends on the process to deliver the data) and/or suitability reasons (in which case the information provider is capable of providing the requested information).

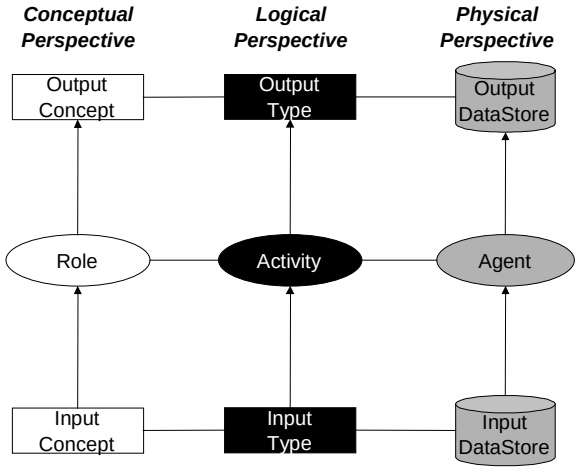

Fig. 3. The reasoning behind the 3 perspectives of the process metamodel

2.3. Complexity and Traces

Data warehouse operational processes are quite complex, in terms of tasks executed within a single process, execution coherence, contingency treatment, etc. A process metamodel should be able to capture this kind of complexity. In Fig. 4 the data warehouse refreshment process is depicted, as described in [6]. The refreshment process is composed of activities, such as *Data Extraction, History Management, Data Cleaning, Data Integration, History Management, Update Propagation* and *Customization*. Each of these activities could be executed on a different site. The activities are interlinked through rules, denoted by arrows; in a real-world case study in banking, no less than 17 kinds of knowledge sources determined this process [53]. The gray background in Fig. 4 implies that there is a composition hierarchy in the set of data warehouse operational processes. In fact, the need to isolate only a small subset of the overall processes of the warehouse is common. Any metamodel must be suitable to support zooming in and out the process structure, in order to achieve this functionality.

The most common reason for this kind of inspection is to avoid or recover erroneous execution during runtime. Not only the structure of a process is important; the specific traces of executed processes should be tracked down, too. If the repository is able to capture this kind of information, it gains added value since: *ex ante* the data warehouse stakeholders can use it for design purposes (e.g., to select the data warehouse objects necessary for the performance of a task) and *ex post*, people can relate the data warehouse objects to decisions, tools and the facts which have happened in the real world [27].

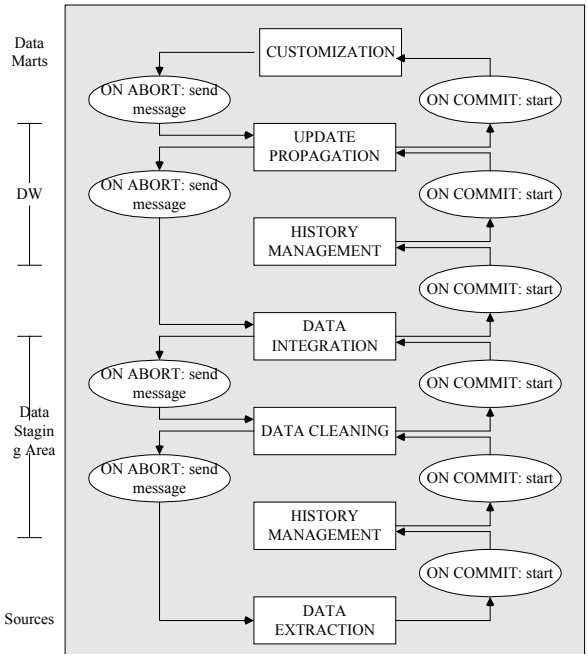

Fig. 4. The Data Warehouse Refreshment Process [6]

2.4. The Data Oriented Nature of Operational Data Warehouse Processes

Data warehouse activities are of data intensive nature in their attempt to push data from the sources to the tables of the data warehouse or the client data marts. We can justify this claim by listing the most common operational processes:

- *data extraction processes*, which are used for the extraction of information from the legacy systems;
- *data transfer* (and *loading*) *processes*, used for the instantiation of higher levels of aggregation in the data warehouse with data coming from the sources or lower levels of aggregation;
- *data transformation processes*, used for the transformation of the propagated data to the desired format;
- *data cleaning processes*, used to ensure the consistency of the data warehouse data (i.e., the fact that these data respect the database constraints and the business rules);
- *computation processes*, which are used for the derivation of new information from the stored data (e.g., further aggregation, querying , business logic, etc.).

To deal with the complexity of the data warehouse loading process, specialized *Extraction-Transformation-Loading* (*ETL*) tools are available in the market. Their most prominent tasks include:

- the identification of relevant information at the source side,
- the extraction of this information,
- the customization and integration of the information coming from multiple sources into a common format,
- the cleaning of the resulting data set, on the basis of database and business rules, and
- the propagation of the data to the data warehouse and/or data marts.

According to a study for Merrill Lynch [55], ETL and Data Cleaning tools cover a labor-intensive and complex part of the data warehouse processes, estimated to cost at least one third of effort and expenses in the budget of the data warehouse. [15] mentions that this number can rise up to 80% of the development time in a data warehouse project. Still, due to the complexity and long learning curve of these tools, many organizations turn to in-house development to perform ETL and data cleaning tasks.

2.5. Case Study Example

To motivate the discussion, we use a part of one of our real-world case studies [58]. The organization collects various data about the annual activities of all the hospitals of a particular region. The source of data, for our example, is a COBOL file, dealing with the annual information by class of beds and hospital (here we use only three classes, namely A, B and C). It yields a specific attribute for each type of class of beds. Periodically, the COBOL file is transferred from the production system to the data warehouse and stored in a "buffer" table of the data warehouse, acting as mirror of the file. Then, the tuples of the buffer table are used by computation procedures to further populate a "fact" table inside the data warehouse. Finally, several materialized views are populated with aggregate information and used by client tools for querying.

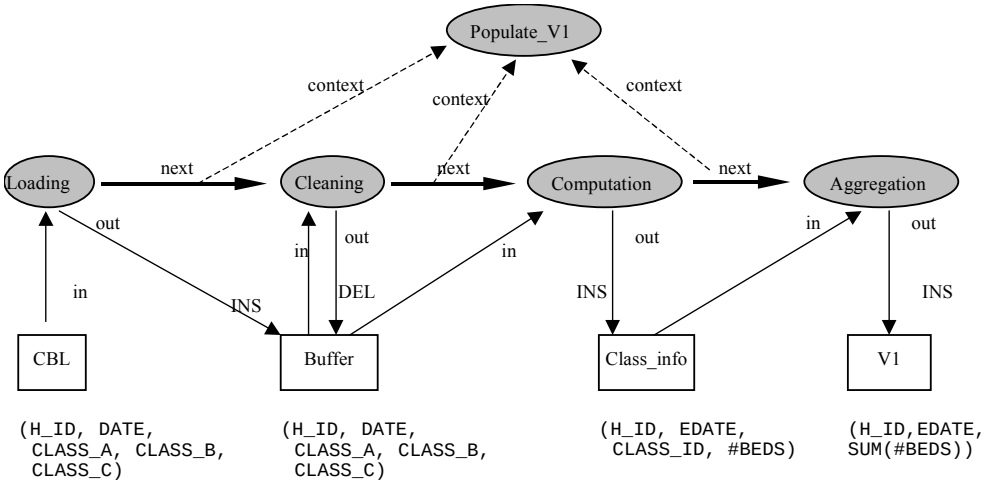

Fig. 5. Motivating Example

The entity *Type* denotes the logical schema for all kinds of data stores. Each *Type* is characterized by a name and a set of *Fields*. In our example, we assume the following four *Types*: *CBL*, *Buffer*, *Class_info* and *V1*. The schemata of these types are depicted in Fig. 5. There are four atomic *Activities* in the data warehouse: *Loading*, *Cleaning*, *Computation* and *Aggregation*. The *Loading* activity simply copies the data from the *CBL* Cobol file to the *Buffer* type. *H_ID* is an identifier for the hospital and the three last attributes hold the number of beds per class. The *Cleaning* activity deletes all entries violating the primary key constraint. The *Computation* activity transforms the imported data to a different schema; the date is converted from American to European format and the rest of the attributes are converted to a combination (*Class_id*, *#Beds*). For example, if *(03,12/31/1999,30,0,50)* is a tuple in the *Buffer* table, the respective tuples in the *Class_Info* table are *{(03,31/Dec/1999,A,30), (03,31/Dec/1999,C,50)}*. The *Aggregation* activity produces the sum of beds by hospital and year. The combination of all the aforementioned activities is captured by the composite activity *Populate V1*.

3. The Metamodel of Data Warehouse Operational Processes

We start the presentation of the metamodel for data warehouse operational processes from the *logical* perspective, to show how the metamodel deals with the requirements of structure complexity and capturing of data semantics in the next two sections. Then, in subsections 3.3 and 3.4 we present the physical and the conceptual perspectives. In the former, the requirement of trace logging will be fulfilled too. The full metamodel is presented in Fig. 6.

For the implementation, we have used a meta-database as a repository for meta-information of the data warehouse components. The architecture, quality and process models are represented in Telos [39], a conceptual modeling language for representing knowledge about information systems. A prototype was implemented in the object-oriented deductive database system ConceptBase [28], that provides query facilities, and a language for constraints and deductive rules. The implementation of the process metamodel in ConceptBase is straightforward. Thus we choose to follow an informal, bird's-eye view of the model, for reasons of presentation.

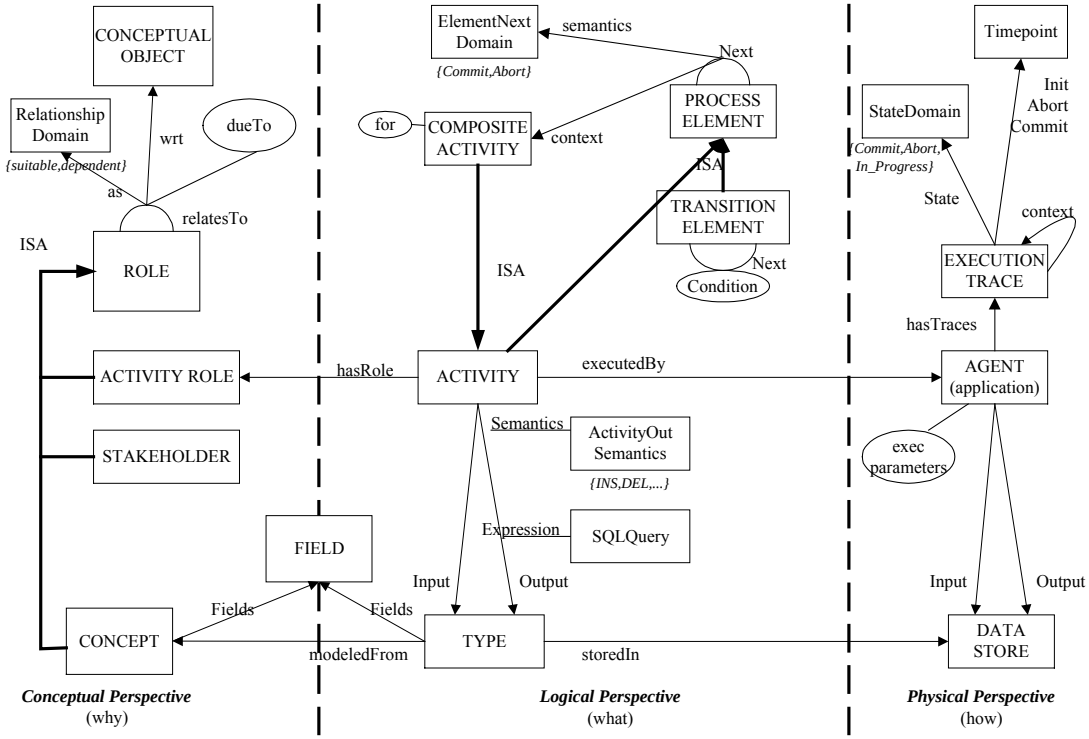

Fig. 6. The Metamodel for Data Warehouse Operational Processes

3.1. Complexity of the Process Structure

Following the Workflow Coalition [64], the main entity of the logical perspective is *Activity*. An activity represents a unit of "work which is processed by a combination of resource and computer applications". Activities can be complex, as captured by the specialization of *Activity*, namely

CompositeActivity. This gives the possibility of zooming in and out the repository. The components of composite activities are *ProcessElements*. Class *ProcessElement* is a generalization of the entities *Activity* and *TransitionElement.* A transition element is employed for the interconnection of activities participating in a complex activity. The attribute *Next* captures the sequence of events. Formally, a *Process Element* is characterized by the following attributes:

- *Name*: to uniquely identify the *Process Element* within the extension of its class.
- *Next*: a *ProcessElement* which is next in the sequence of a composite activity, characterized by:
  - *Context*: Since two activities can be interrelated in more than one complex DW process, the context of this interrelationship is captured by the relevant *CompositeActivity* instance.
  - *Semantics*: denotes whether the next activity in a schedule happens upon successful termination of the previous activity (*COMMIT)* or if a contingency action is required (*ABORT*).

A *TransitionElement* is a specialization of *ProcessElement*, used to support scheduling of the control flow within a composite activity. This extra functionality is supported by two mechanisms. First, we enrich the *Next* link with more meaning, by adding a *Condition* attribute to it. A *Condition* is a logical expression in Telos denoting that the firing of the next activity is performed when the *Condition* is met. Second, we specialize the class *TransitionElement* to four subclasses (not shown in Fig. 6), capturing the basic connectives of activities [64]: *Split_AND, Split_XOR, Join_AND, Join_XOR*. The semantics of these entities are the same with the ones of the WfMC proposal. For example, the *Next* activity of a *Join_XOR* instance is fired when (a) the *Join_XOR* has at least two activities "pointing" to it through the *Next* attribute and only one *Next* Activity (well-formedness constraint), (b) the *Condition* of the *Join_XOR* is met and (c) at least one of the "incoming" activities has *COMMIT* semantics in the *Next* attribute. This behavior can be expressed in Telos with appropriate rules. Similarly, *Split_AND* denotes a point in the process chain where more than one concurrent execution threads are initiated, *Split_XOR* denotes a point where exactly one execution thread is to be initiated (out of many different alternatives) and *JOIN_XOR* acts as a rendezvous point for several concurrent threads, where the execution of the process flow is paused until all incoming activities have completed their execution.

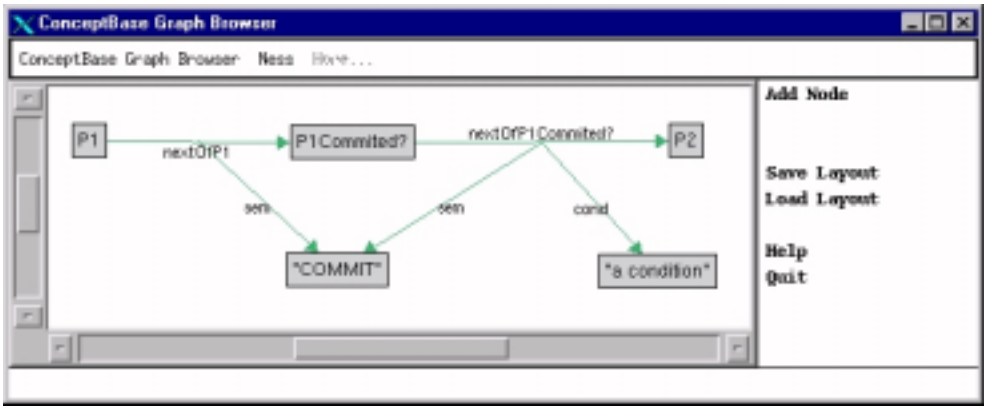

Fig. 7. Example of a composite activity: ON COMMIT(P1), IF <condition> START

The WfMC proposes two more ways of transition between *Activities*. *Dummy activities* perform routing based on condition checking, they are modeled as simple *Transition Elements*. *LOOP* activities are captured as instances of *CompositeActivity*, with an extra attribute: the *for* condition. Fig. 7 shows the modeling of a composite activity, composed of two sub-activities, where the second is fired when the first activity commits and a Boolean condition is fulfilled. *P1_Commited?* is a transition element.

3.2. Relationship with Data

A *Type* denotes the schema for all kinds of data stores. Formally, a *Type* is defined as a specialization of *LogicalObject* with the following attributes:

- *Name*: a single name denoting a unique *Type* instance.
- *Fields*: a multi-value attribute. In other words, each *Type* has a name and a set of *Fields*, exactly like a relation in the relational model.
- *Stored*: a *DataStore*, a physical object representing software used to manipulate stored data (e.g., a DBMS). This attribute will be detailed in the description of the physical perspective.
- *ModeledFrom*: a *Concept*, an object in the conceptual perspective, providing a description of the type in a more user-friendly manner. A *Concept* is the generalization of *Entities* and *Relationships* in the Entity-Relationship model (not depicted in Fig. 6).

Any kind of physical data store (multidimensional arrays, COBOL files, even reports) can be represented by a *Type* in the logical perspective. For example, the schema of multidimensional cubes is of the form $[D_1,...,D_n,M_1,...,M_m]$ where the D_i represent dimensions (forming the primary key of the cube) and the M_j measures [60]. Cobol files, as another example, are records with fields having two peculiarities: nested records and alternative representations. One can easily unfold the nested records and choose one of the alternative representations.

Each *Activity* in a data warehouse environment is linked to a set of incoming and outgoing types. We capture the relationship of activities to data by expressing the outcome of a data warehouse process as a function over its input data stores. This function is captured through SQL queries, extended with functions. An *Activity* is formally characterized by the following attributes:

- *Name, Next*: inherited from *Process Element.*
- *Input*: multi-valued *Type* attribute modeling all data stores used by the activity to acquire data.
- *Output*: single-valued *Type* attribute. This attribute models the data store or report where the activity outputs data. The *Output* attribute is further explained by two attributes:
  - *Semantics*: a single value belonging to the set {*Insert, Update, Delete, Select*} (captured as the domain of class *ActivityOutSemantics*). A process can either add, delete, or update the data in a data store. Also it can output some messages to the user (captured by using a "Message" Type and *Select* semantics).
  - *Expression*: a single SQL query (instance of class *SQLQuery*) to denote the relationship of the output and the input types, possibly with functions.
- *ExecutedBy*: a physical *Agent* (i.e., an application program) executing the Activity. More information on agents will be provided in the description of the physical perspective.
- *HasRole*: a conceptual description of the activity. This attribute will be properly explained in the description of the conceptual perspective.

Fig. 8 shows how various types of processes can be modeled in our approach.

Activity	**Semantics**	**Expression**
Extract & Load	INS/UPD	SELECT * FROM <In>
Transformation	INS/UPD	SELECT * FROM <In> WHERE $attr_i$ = f($attr_1$, ..., $attr_n$)
Cleaning	DEL/UPD	– Primary Key: SELECT <P.K.> FROM <IN> GROUP BY <P.K.> HAVING COUNT(*) > 1 – Foreign Key: SELECT <P.K.> FROM <IN> WHERE <F.K.> NOT IN (SELECT <F.K> FROM <TARGET>) – Any other kind of query
Computation	INS/UPD	Any kind of query
Messaging	SEL	Any kind of query

Fig. 8. Examples of *Output* attribute for particular kinds of activities

To gain more insight in the proposed modeling approach, consider the example of Fig. 5. The expressions and semantics for each activity are listed in Fig. 9. All activities append data to the involved types, so they have *INS* semantics, except for the cleaning process, which deletes data, and thus has *DEL* semantics. We do not imply that everything should actually be implemented using the employed queries, but rather that the relationship of the input and the output of an activity is expressed as a function, through a declarative language such as SQL.

Attribute	**Expression**	**Semantics**
Loading.Out:	SELECT H_ID,DATE,CLASS_A,CLASS_B,CLASS_C FROM CBL	INS
Cleaning.Out:	SELECT H_ID,DATE,CLASS_A,CLASS_B,CLASS_C FROM BUFFER B1 WHERE EXISTS (SELECT B2.H_ID, B2.DATE FROM BUFFER B2 WHERE B1.H_ID = B2.H_ID AND B1.DATE = B2.DATE GROUP BY H_ID,DATE HAVING COUNT(*)> 1)	DEL
Computation.Out:	SELECT H_ID, EUROPEAN(DATE) AS EDATE, 'A', CLASS_A AS #BEDS FROM BUFFER WHERE CLASS_A <> 0 UNION SELECT H_ID, EUROPEAN(DATE) AS EDATE, 'B', CLASS_B AS #BEDS FROM BUFFER WHERE CLASS_B <> 0 UNION SELECT H_ID, EUROPEAN(DATE) AS EDATE, 'C', CLASS_C AS #BEDS FROM BUFFER WHERE CLASS_C <> 0	INS
Aggregation.Out:	SELECT H_ID, EDATE, SUM(#BEDS) AS SUM_BEDS FROM CLASS_INFO GROUP BY H_ID, EDATE	INS

Fig. 9. Expressions and semantics for the activities of the example

3.3. The Physical Perspective

While the logical perspective covers the structure ("what") of a process, the *physical perspective* covers the details of its execution ("how"). Each process is executed by an *Agent* application program. Each *Type* is assigned to a *DataStore* (providing information for issues like table spaces, indexes, etc.). An *Agent* can be formally described as follows:

- *Name*: to uniquely identify the *Agent* within the extension of its class.
- *Execution_parameters*: a multi-value string attribute capturing any extra information about the execution of an agent.
- *In, Out*: physical *DataStores* communicating with the *Agent*. The types used by the respective logical activity must be stored within these data stores.
- *HasTraces*: a multi-value attribute capturing all the execution traces of the *Agent*.

An *Execution Trace* is an entity capturing details of activity execution from the proper *Agent*:

- *TraceID*: a unique identification number to uniquely identify each *Execution Trace*.
- *State*: a single value belonging to the domain of class *AgentStateDomain* = {*In_Progress*, *Commit*, *Abort*}.
- *Init_time, Abort_time, Commit_time*: timestamps denoting the *Timepoints* when the respective events have occurred.
- *Context*: another *Execution Trace*. This attribute is used in the case where a script (*Agent*) is simply the coordinator script for the execution of several other *Agents*, i.e. in *Composite Activities*. In this case, the *Execution Trace* of the *Agent* of the *Composite Activity* defines the context for the execution of the coordinated agents.

The information of the physical perspective can be used to trace and monitor the execution of data warehouse processes. Fig. 10 sketches the trace information after a successful execution of the process described in Fig. 5. We show the relationship between the logical and the physical perspective by linking each logical activity to a physical application program. Each agent has a set of execution traces. We can see that the trace of composite activity *Populate V1* has as *Init time* the time of the execution of the first sub-activity and *Commit time* the completion time of the last activity. Also, this composite activity defines the context of the execution of its sub-activities: this is captured by properly populating the attribute *Context*.

Activity Name	Populate V1	Loading	Cleaning	Computation	Aggregation
Agent Name	Schedule.exe	sqlldr80.exe	Clean.pql	Comp.pql	Aggr.pql
Execution Parameters	--	Parfile= param.par	--	--	--
HasTraces	{…,100,…}	{…,101,…}	{…,102,…}	{…,103,…}	{…,104,…}
TraceID	100	101	102	103	104
State	COMMIT	COMMIT	COMMIT	COMMIT	COMMIT
Init time	31/12/99:00:00	31/12/99:00:00	31/12/99:01:00	31/12/99:01:15	31/12/99:03:00
Abort time	--	--	--	--	--
Commit time	31/12/99:03:30	31/12/99:01:00	31/12/99:01:15	31/12/99:03:00	31/12/99:03:30
Context	--	100	100	100	100

Fig. 10. Trace information after a successful execution of the process of the example

3.4. The Conceptual Perspective

A major purpose behind the introduction of the conceptual perspective is to help stakeholders understand the reasoning behind decisions on the architecture and physical characteristics of data warehouse processes. Our modeling approach captures dependency and suitability relationships among the basic conceptual entities to facilitate the design, administration and evolution of the data warehouse.

Each *Type* in the logical perspective is the counterpart of a *Concept* in the conceptual perspective. A concept represents a class of real-world objects, in terms of a conceptual metamodel, e.g., the Entity-Relationship or UML notation. Both *Types* and *Concepts* are constructed from *Fields* (representing their attributes), through the attribute *fields*. We consider *Field* to be a subtype both of *LogicalObject* and *ConceptualObject.* The central conceptual entity is the *Role* which generalizes the conceptual counterparts of activities, stakeholders and data stores. The class *Role* is used to express the interdependencies of these entities, through the attribute *RelatesTo. Activity Role*, *Stakeholder* and *Concept* are specializations of *Roles* for processes, persons and concepts in the conceptual perspective. Formally, a *Role* is defined as follows:

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

Each *Role* represents a person, program or data store participating in the environment of a process, charged with a specific task and/or responsibility. An instance of the *RelatesTo* relationship is a statement about the interrelationship between two roles in the real world, such as 'View V1 *relates to* table Class_Info *with respect to* the attributes H_ID, EDate and #Beds *as* dependent *due to* loading reasons'. Note that, since both data and processes can be characterized by SQL statements, their interrelationship can be traced in terms of attributes.

The conceptual perspective is influenced by the Actor Dependency model [68]. In this model, actors *depend* on each other for the accomplishment of goals and the delivery of products. The *dependency* notion is powerful enough to capture the relationships in the context of a data flow, where a data consumer (person, data store or program) depends on the proper function of its data providers, to achieve its mission. Our extension can capture *suitability* as well (e.g., in the case where more than one concepts can apply for the population of an aggregation, one concept is suitable to replace the other).

When understanding the occurring errors or the design decisions on the architecture of a data warehouse, the conceptual perspective can exploited in various ways.

1. The *design* of the data warehouse is supported, since the conceptual model serves as a documentation repository for the reasons behind the structure of the data warehouse. The model allows the tracing of the relationships between any pair of persons, programs or data stores. With minimum query facilities of the metadata repository, these interdependencies do not have to be directly stored in all the cases, but can also be computed incrementally, due to their transitivity.
2. The *administration* of the data warehouse is facilitated in several ways. The conceptual model is a good roadmap for the quality management of the warehouse and can act as an entry-point to the logical perspective, since it can enable the user to pass from the abstract relationships of roles to the structure of the system. During runtime, the suitability notion can be used to obtain solutions to potential errors or problems (e.g., alternative paths for the population of the data warehouse) in a straightforward way.
3. Data warehouse *evolution* is supported at two levels. At the *entity* level, the impact of any changes in the architecture of the warehouse can be detected through the sheer existence of dependency links. At the same time, the existence of suitability links suggests alternatives for the new structure of the warehouse. At the *attribute* level, on the other hand, internal changes in the schema of data stores or the interface of software agents can be detected, by using the details of the relationships of the data warehouse roles. For example, the previous statement for the relationship of view *V1* and table *Class_Info* could be interpreted as 'View *V1* is affected by any changes to table *Class_Info* and especially the attributes *H_ID, EDate* and *#Beds*'.

We detail all the aforementioned features and benefits through our example. As we can see in Fig. 11 our conceptual model consists of several entities, including:

- *Aggregation Process* is an *Activity Role*, corresponding to the logical activity *Aggregation*;
- *Information By Class and Date* (for brevity, *"Info By Class"* in Fig. 11) is a *Concept* corresponding to the *Type* "*Class_info*";
- *Hospital Information By Date* (for brevity, *"Hospital Info"* in Fig. 11) is also a *Concept* corresponding to the *Type* "*V1*";
- *Administrator 1, Administrator 2* (for brevity, *"Admin 1* and *2"* in Fig. 11) as well as *End User* are the involved *Stakeholders* with the obvious roles in the system.

Clearly, *Hospital Information By Date* is an aggregation over *Information By Class and Date* and actually, what the conceptual schema tells us, is that it is the *Aggregation Process* that performs the aggregation in practice. Due to the data flow, there exists a strong dependency between the aforementioned roles. *Hospital Information By Date* depends on *Aggregation* since the latter acts as a data pump for the former. *Hospital Information By Date* transitively depends on *Information By Class*. *Administrator 1* depends also on *Information By Class*. Although not depicted in Fig. 11 to avoid overloading the figure, this relationship relies on the idea of political dependency: up to now, it was the *Administrator 1* that provided the data for this kind of information and in fact, he still does within the data warehouse environment.

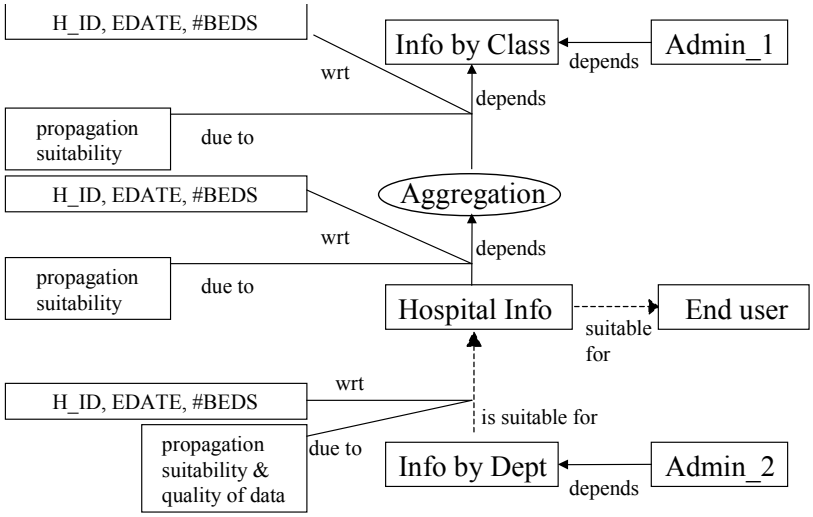

Fig. 11. The conceptual perspective for the example

Another interesting point shown in Fig. 11 is the idea of *suitability*: according to the stated needs of the users, the concept *Hospital Information By Date* represents information which is *suitable* for the *End User*. It is worth stressing the fact that although some of the entities correspond to processes, other to stakeholders and other to data stores, this is not affecting the uniformity and the simplicity of the representation. Also, notice that we do not delve into the fields of conceptual multidimensional aggregation (for example, see [2]) or requirements engineering: the specific approach one can adopt is orthogonal to our modeling.

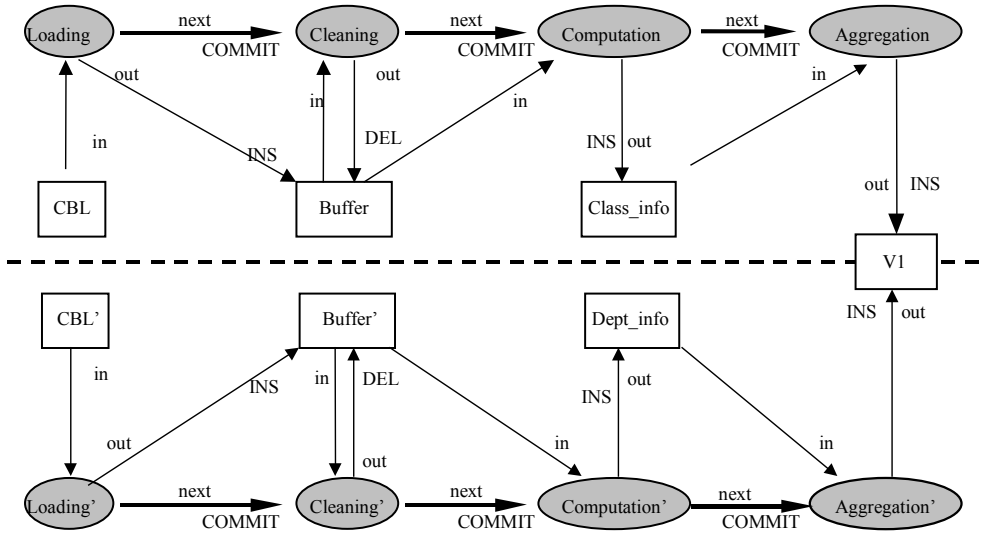

Fig. 12. Data warehouse evolution. Upper part: original problematic configuration. Lower part: the new configuration.

The notion of suitability helps to support data warehouse evolution. As already mentioned, it is the redundancy in the data warehouse that makes suitability so attractive. Consider the following real-world case, where the information in the final view *V1*, was not 100% consistent. Simple measurements of quality (cf. Section 4) indicated that the COBOL file *CBL* was responsible for this quality problem (upper part of Fig. 12). Thus, the original data provider was flawed. Out of the many different choices one could possibly have to resolve the problem, the most suitable one proved to be the exploitation of redundancy. An alternative population scheme for view *V1* used the source file *CBL'* as initial input. *CBL'* is the corresponding data store of the *Concept Information By Department and Date* (*"Info By Dept"* in Fig. 12) and captures the number of beds by department of hospital (instead of class of beds).

Sometimes, suitability in the conceptual model of a data warehouse can be automatically derived from aggregate reasoners and algorithms proposed by previous research on view containment [54, 23, 46, 12]. Again, we would like to stress that suitability in our proposal is not restricted to either persons or resources but can uniquely cover all the entities in a conceptual model of a data warehouse.

Apart from the support for evolution of the data warehouse at the entity level, the proposed model is also capable to support evolution at the attribute level. As mentioned, the relationship between two roles is normally expressed in terms of fields. In our example, *Aggregation* depends on a subset of the attributes of *Info By Class*, namely *H_ID, DATE* and *#Beds.* If the attribute *Class_ID* changes, e.g., due to change of type, removal, or renaming, the final information of the *Concept "Hospital Info"* is not affected at all. On the other hand, changes in any of the attributes depicted in Fig. 11 clearly affect the information delivered to end users.

It is interesting to note the political conflict that takes place due to the proposed change. As we can see, removing *Info By Class* from the data flow, automatically affects the Stakeholder *Administrator 1*, who depends on *Info By Class*. A simple query in the metadata repository for the dependents of the entity *Info By Class* could give a warning for the political vibrations coming from such a decision[1]. Finally, we should also mention that hidden within the short story that we have just summarized is the idea of quality measurement, which will be detailed in Section 4.

3.5. Facilitating Data Warehouse Design through Consistency Checking in the Metadata Repository

To ensure the validity of the representation in the metadata repository, consistency checks can be performed during the design of the data warehouse, by defining constraints, rules and views in the language Telos. The following view finds out whether the types used as inputs of an activity are stored in the respective data stores used as inputs of the agent, executed by the activity.

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

Other simple constraints involve the local structure of the process elements. For example, split transition elements must have at least one incoming edge and more that one outgoing edge. The timestamps of an agent should also be consistent with its state. The repository can also be used by external programs to support the execution of consistency checking algorithms as proposed in [52, 38].

3.6. Repository Support for the Automatic Derivation of Role Interdependencies

The *Roles* of the conceptual perspective can be directly assigned by the data warehouse administrator, or other interested stakeholders. Still, one could argue that this is too much of an administrative burden, based on the number of relationships that should be traced in the repository. By deductive reasoning in the metadata repository, we can impose simple rules to deduce these interdependencies from the structure of data stores and activities. For example, we can derive dependency relationships by exploiting the structure of the logical perspective of the metadata repository. Also, interdependencies do not have to be directly stored in all the cases, but can also be computed incrementally, due to the transitivity of their nature. In Fig. 13 we show three simple rules which can be used to derive the production of role interdependencies. They can also be implemented in the metadata repository.

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

Fig. 13. Simple rules for the production of role interdependencies

4. Process Quality

In this section we present how the process metamodel is linked to the metamodel for data warehouse quality proposed in [32]. Moreover, we complement this quality metamodel with specific dimensions for data warehouse operational processes.

[1] We take the opportunity to stress the huge importance of politics in the development lifecycle of a data warehouse. See [15, 58] for more details.

4.1. Terminology for Quality Management

The quality metamodel in [32] customizes the Goal-Question-Metric approach (GQM) of [47] for data warehouse environments. In this section, we adopt the same metamodel for the operational processes of the data warehouse.

Each object in the data warehouse is linked to a set of quality goals and a set of quality factors (Fig. 15). A *quality goal* is an abstract requirement, defined on data warehouse objects, and documented by a purpose and the stakeholder interested in it, e.g., 'improve the availability of source S1 until the end of the month in the viewpoint of the data warehouse administrator'. *Quality dimensions* (e.g. 'availability') are used to group quality goals and factors into different categories. A *Quality Factor* represents a quantitative assessment of a particular aspect of a data warehouse object, i.e., it relates quality aspects both to actual measurements and expected ranges for these quality values. Finally, the method of measurement is attached to a quality factor through a *measuring agent*.

The bridge between the abstract, subjective quality goals and the specific, objective quality factors is determined through a set of *quality queries*, to which quality factor values are provided as possible answers. Such queries are the outcome of the methodological approach described in [57] which offers template quality factors and dimensions, defined at the metadata level and instantiates them, for the specific data warehouse architecture under examination. As a result of the goal evaluation process, a set of improvements (e.g., design decisions) can be proposed, in order to achieve the expected quality.

4.2. Quality Dimensions and Factors for Data Warehouse Operational Processes

ISO 9126 standard [ISO97] on software implementation and evaluation provides a general understanding of how to measure the quality of software systems. Data warehouses do not stray from these general guidelines; thus we adopt the standard as a starting point. ISO 9126 is based on six high level quality dimensions (*Functionality, Reliability, Usability, Efficiency, Maintainability, Portability*). Time and budget constraints in the development of a data warehouse cause the addition of *Implementation Effectiveness*. The dimensions are analyzed to several sub-dimensions (Fig. 14).

Quality Dimension	***Quality Factors***
Functionality:	
Suitability	Benchmark scores, number of software requirements met.
Accuracy	Completeness, accuracy and consistency of data (which are the final product).
Interoperability	Number of modules unable to interact with specified systems.
Compliance	Number of modules not compliant with application related standards or conventions or regulations in laws and similar prescriptions.
Security	Number of modules unable to prevent unauthorized access, whether accidental or deliberate, to programs and data.
Reliability:	
Maturity	Frequency of failure by faults in the software.
Fault tolerance	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	Number of occasions where the software was unable to re-establish its level of performance and recover the data affected in the case of a failure. The time and effort needed for this reestablishment.
Usability:	
Understandability	Percentage of acceptance by the users.
Learnability	Percentage of acceptance by the users.
Operability	Existence of time prediction for operation, number of error alerts, available built in functions for ETL tasks
Software Efficiency:	
Time behavior	Response times, processing times and throughput rates.
Resource Behavior	Volume of extracted and loaded data, maximum data size the software can handle.
Maintainability:	
Analyzability	Comment percentage, readability index.
Changeability	Number of logical paths in a module, control flow intersection, GOTO statements, size, etc.
Stability	Same as Changeability
Testability	Same as Changeability
Portability:	Same as Maintainability
Implementation Effectiveness	Man-hours spent, tasks completed.

Fig. 14. Software quality dimensions [26] and proposed quality factors in data warehouse environments

ISO 9126 does not provide specific quality factors. To deal with this shortcoming, Appendix I gives a set of quality factors customized for the case of data warehouse operational processes. It does not detail the whole set of possible factors for all operational data warehouse processes, but rather, we intend to come up with a minimal representative set. This set of quality factors can be refined and enriched by the data warehouse stakeholders with customized factors. Once again, we encourage the use of "templates" in a way that fits naturally with the overall metadata framework that we propose.

4.3. Relationships Between Processes and Quality

Quality goals describe *intentions* of data warehouse users with respect to the status of the data warehouse. In contrast, our process model describes *facts* about the current and previous status of the data warehouse and what activities are performed in the data warehouse. However, the reason behind the existence of a process is a quality goal. For example, a data cleaning process is executed in the data staging area in order to improve the accuracy of the data warehouse. We have represented this interdependency between processes and quality goals by establishing a relationship between roles and data warehouse objects in the conceptual perspective of the process model (relationship *Imposed On*). This is shown in the upper part of Fig. 15.

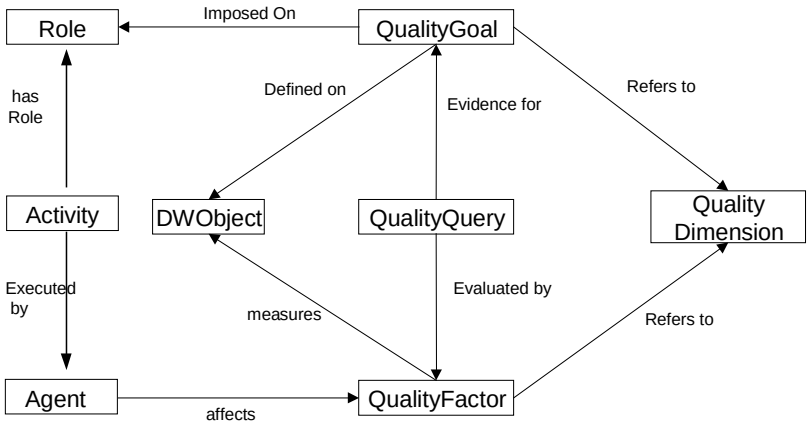

Fig. 15. Relationships between processes and quality

Our model is capable of capturing all dependency types mentioned in [68]. *Task dependencies*, where the dependum is an *activity*, are captured by assigning the appropriate role to the attribute *Wrt* of the relationship. *Resource dependencies*, where the dependum is the availability of a *resource*, are modeled when fields or concepts populate this attribute. The relationship *ExpressedFor* relates a role to a high-level quality goal; thus *Goal dependencies*, dealing with the possibility of making a condition true in the real world, are captured from the model, too. *Soft-goal dependencies* are a specialization of goal dependencies, where evaluation cannot be done in terms of concrete quality factors.

The lower part of Fig. 15 represents the relationship between processes and quality on a more operational level. The actions of an agent in the data warehouse *affect* the expected or measured quality factors of some data warehouse objects. For example, a data cleaning process affects the availability of a source: it decreases the amount of time during which it can be used for regular operations. Consequently, this process will affect the quality factors *Average Load*, *CPU state*, *Available Memory* defined on a *Source*. All these quality factors are concrete representations of the abstract notion *Availability* -- the relevant quality dimension. The effect of a data warehouse process must always be confirmed by new measurements of the quality factors. Unexpected effects of data warehouse processes can be detected by comparing the measurements with the expected behavior of the process. The measurement of the quality of the particular agents through their own quality factors is analyzed in [29].

A *Quality Query* provides the methodological bridge to link the high-level, user-oriented, subjective quality goals and the low-level, objective, component-oriented quality factors. The vocabulary (or domain) of quality queries with respect to the process model is the set of data warehouse activities, which can be mapped to reasons (roles) and conditions (of agents) of a specific situation.

Let us return to our hospital example to clarify how the process and the quality metamodels interplay, and how the different perspectives gracefully map to each other. More than one of our experiences in the public sector indicated a need of total quality of data, were no errors were allowed and no information missing. Thus, the quality goal is '100% quality of data delivered to the end users'.

For the purpose of our example, we narrow this high level goal to the subgoal, 100% consistency of the produced information. There are two objects involved in this quality goal, namely the quality dimension *consistency* and the role *end user*. Both entities, as well as the quality goal itself, belong to the conceptual perspective and can be used to explain why the chain of processes exist: to bring clean, consistent information to the involved stakeholders.

According to the GQM paradigm, a good start to examine a situation would be to find out its current status. With respect to the elements of the process model, the basic question over process status is naturally over the *correctness* dimension: are all activities performing as they should? The question, itself belonging to the logical perspective, involves an object of the logical part of the process metamodel: *activities.* If one applies the methodology of [57], this question would directly be analyzed to five consequent questions, each involving one of the activities *Loading, Cleaning, Computation, Aggregation*, and *Populate V1*.

The actual quality evaluation of the produced software is done in terms of concrete measurements. On the physical perspective (where the quality factors that provide the measurements belong) they measure the specific software agents involved, in our case, the quality factor *Correctness through White Box Software Testing* (*WBCorrectness* in the sequel) performs this kind of measurements.

The discrimination of logical, conceptual and physical perspectives is proven useful once more, in the quality management of the data warehouse: the quality goals can express "why" things have happened (or should happen) in the data warehouse, the quality questions try to discover "what" actually happens and finally, the quality factors express "how" this reality is measured (Fig. 16). On top of this, we have organized a seamless integration of the process and quality models, by mapping the objects of the same perspectives to each other.

In Fig. 17 we can also see the assignment of quality factors to the various objects at the physical perspective. We assign the quality factor *WBCorrectness* to the software agents and the quality factors *Consistency* and *Completeness* to the data stores. A simple formula derives the quality of the data stores in the latest steps of the data flow from the quality of the previous data stores and software agents. Let us take *Consistency* for example:

$$\text{Consistency}(ds) = \prod \text{correctness}(a) * \prod \text{consistency}(ds') \quad (1)$$

where `ds` is the data store under consideration, `a` denotes the agents having `ds` as their output and *COMMIT* semantics and `ds'` is any data store different than `ds`, serving as input to the agents `a`.

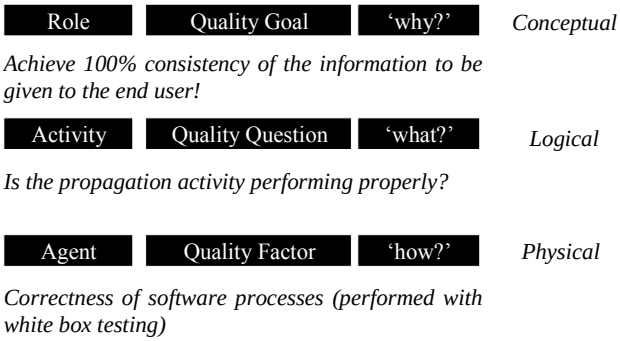

Fig. 16. Perspectives and interrelationships for the process and quality metamodels

Clearly, the consistency of the final view *V1* depends on the consistency of all the previous data stores and the correctness of all the involved software agents. Although arguably naive, the formula fitted perfectly in our real world scenario.

4.4. Exploitation of Quality Modeling in the Repository for Data Warehouse Administration

The different instantiations of the quality model can be exploited to assist both the design and the administration of the data warehouse. In [57], it has been shown how the classification of quality dimensions, factors and goals can be combined with existing algorithms to address the data warehouse design problem (i.e., the selection of the set of materialized views with the minimum overall "cost" that fulfill all the quality goals set by the involved stakeholders). As far as the administration of the warehouse is concerned, the information stored in the repository may be used to find deficiencies in a data warehouse.

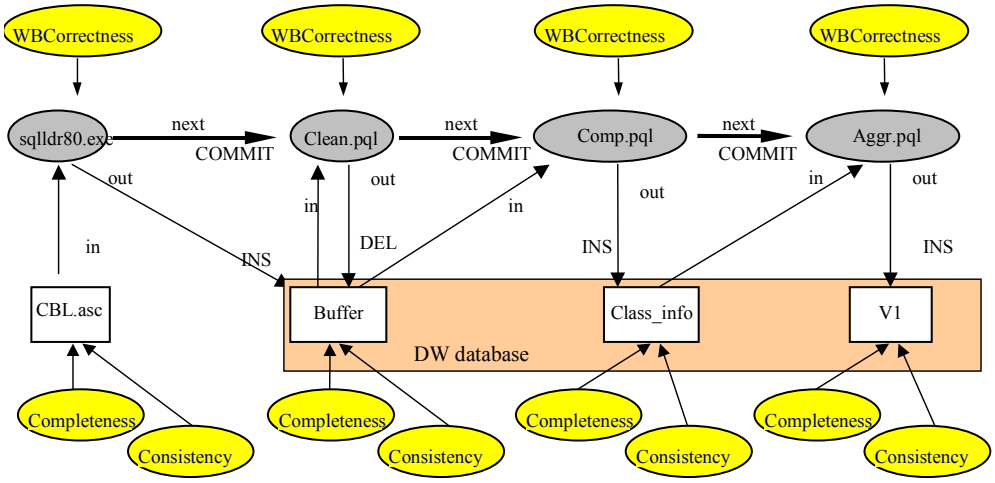

Fig. 17. Application of quality factors to the entities of the example

To show how the quality model is exploited, we take the following query. It returns all data cleaning activities which have decreased the availability of a data store according to the stored measurements. The significance of the query is that it can show that the implementation of the data cleaning process has become inefficient.

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

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

5. Repository Support for Data Warehouse Description and Evolution

Summarizing the discussion so far, during the *design* phase, the user can check the consistency of his/her design, to determine any violations of the business logic of the data warehouse, or the respect of simple rules over the structure of the data warehouse schema. During the *administration* phase, we can use the repository to discover quality problems.

In this section, we continue to show how the metadata repository can be exploited in different ways. First, we complement the perception of data warehouses as collections of materialized views with a precise operational description of the content of data warehouse tables. Second, a particular task in the data warehouse lifecycle, *data warehouse evolution*, is examined separately, in order to determine possible impacts, when the schema of a particular table in the data warehouse changes.

5.1. Why Data Warehouses Are Not (Just) Collections of Materialized Views

Many database researchers have considered data warehouses to be collections of materialized views, organized in strata where the views of a particular stratum are populated from the views of a lower stratum. For example, in [58] where the papers of three major database conferences related to data warehousing, between the years 1995 and 1999, are classified into different categories, almost half of

the papers (around 46%) deal with view maintenance and integration issues. The papers on view maintenance have focused on algorithms for updating the contents of a view in the presence of changes in the sources. Papers related to integration have targeted the production of a single interface for the processing of distributed heterogeneous data, along with query processing techniques for that cause and resolution of conflicts at the schema level. One can observe, thus, that most of the performed research has been dedicated to *what* should be extracted and loaded, instead of *how* this process is actually performed. Practical aspects of extraction, loading and conversion processes, such as scheduling, declarative process definition, or data peculiarities (source format, errors, conversions) are clearly neglected (see [58] for a broader discussion).

In summary, although the abstraction of treating data warehouses as strata of materialized views, has efficiently served the purpose of investigating the issues of (incremental) view maintenance, it lacks the ability to accurately describe the real content of data warehouse tables, due to the fact that all the intermediate processing, transformation and reorganization of the information is systematically absent from most research.

Our modeling approach follows a different path, by treating data warehouse processes as first class citizens. The semantic definitions for data warehouse views are not assigned directly by the designer but result from combining the respective definitions of the data warehouse processes. Thus, we can argue that there are two ways to define the semantics of a table in the data warehouse:

- A *Type* can be defined as a materialized view over its previous *Types* in the data flow. This is the definition of what the *Type* should contain ideally, i.e., in a situation where no physical transformations or cleaning exists.
- A *Type* can be defined as a view again, resulting from the adoption of our modeling approach. In this case, the resulting definition explains how the contents of the *Type* are *actually* produced, due to schema heterogeneity and bad quality of data.

Of course, both kinds of definition are useful but only the former has been taken into consideration in previous research. To complement this shortcoming, the rest of this subsection is dedicated to showing how we can derive view definitions from the definitions of their populating processes.

To give an intuition of the difference between the two approaches, consider the example of Fig. 5. Ideally, we would like to express the view *V1* in terms of the input file *CBL* (or its relational counterpart, table *Buffer*). A simple formula suffices to give this *intentional* semantics:

```
SELECT H_ID, EUROPEAN(DATE) AS EDATE, CLASS_A+CLASS_B+Class_C AS SUM_BEDS
FROM CBL
```

On the other hand, reality is clearly different. It involves the identification of multiple rows for the same hospital at the same time period, and the restructuring of the information to a normalized format before the loading of data in view *V1*. The full expression capturing this operational semantics is definitely more complex than the simple SQL query denoting the intentional semantics.

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

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

Where *Reduce(expr)*:

1. Use the technique of [41] to represent SQL queries; if self-references exist (e.g. in the case of DEL statements) discriminate between multiple occurrences of the same table.
2. Use the reduction techniques of [48, 33+, 36] wherever applicable to reduce the query definition to a compact form.

Fig. 18. Algorithm for extracting the definition of a type in the repository

To construct expressions for data warehouse tables with respect to their operational semantics, we constrain ourselves to the case where we are able to construct an acyclic, partially ordered graph of activities (produced by proper queries in ConceptBase). Thus, we can treat the whole set of data warehouse activities as an ordered list. Mutually exclusive, concurrent paths in the partially ordered graph are treated as different lists (the execution trace determines which list is considered each time).

Furthermore, a set of types belonging to the set *SourceSchema*, denoting all the types found in the data sources, are treated as source nodes of a graph. For the rest of the types, we can derive an SQL expression by using existing view reduction algorithms, such as [33] corrected with the results of [19, 44, 45], [14, 11, 41, 48, 36]. Our algorithm is applicable to graphs of activities that do not involve updates. In most cases, an update operation can be considered as the combination of insertions and deletions or as the application of the appropriate function to the relevant attributes.

The results of the application of this algorithm to our example are shown in Fig. 18. For convenience, we break composite definitions of table expressions into the different lines of Fig. 19. For example, when the third iteration (`i=3`) refers to the definition of table *Buffer*, it does so with respect to the definition of line 2 (`i=2`). The expression of a single type can also be computed locally.

i	**Expression**
1	`Buffer.expr = Buffer.i_expr:=` `SELECT * FROM CBL`

2	`Buffer.d_expr:=`
	`(SELECT * FROM CBL C1 WHERE EXISTS (SELECT C2.H_ID, C2.DATE FROM CBL C2 WHERE C1.H_ID=C2.H_ID AND C1.DATE=C2.DATE GROUP BY H_ID,DATE HAVING COUNT(*)>1)`
	`Buffer.expr:=`
	`(SELECT * FROM CBL) MINUS`
	`(SELECT * FROM CBL C1 WHERE EXISTS (SELECT C2.H_ID, C2.DATE FROM CBL C2 WHERE C1.H_ID=C2.H_ID AND C1.DATE=C2.DATE GROUP BY H_ID,DATE HAVING COUNT(*)>1)`

3	`Class_info.expr = Class_info.i_expr:=`
	`SELECT H_ID, EUROPEAN(DATE) AS EDATE, 'A', CLASS_A AS #BEDS`
	`FROM BUFFER WHERE CLASS_A <> 0`
	`UNION`
	`SELECT H_ID, EUROPEAN(DATE) AS EDATE, 'B', CLASS_B AS #BEDS`
	`FROM BUFFER WHERE CLASS_B <> 0`
	`UNION`
	`SELECT H_ID, EUROPEAN(DATE) AS EDATE, 'C', CLASS_C AS #BEDS`
	`FROM BUFFER WHERE CLASS_C <> 0;`

4	`V1.expr = V1.i_expr:=`
	`SELECT H_ID, EDATE, SUM(#BEDS) AS SUM_BEDS FROM CLASS_INFO`
	`GROUP BY H_ID, EDATE`

Fig. 19. SQL expressions of the types of the example

5.2. Repository Support for Data Warehouse Evolution

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

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

In our approach, we distinguish two kinds of impact of a hypothetical change:

- *Direct impact*: the change in the data warehouse object imposes that some action must be taken against an affected object. For example, if an attribute is deleted from a materialized view, then the activity which populates it must also be changed accordingly.
- *Implicit impact:* the change in the data warehouse object might change the semantics of another object, without obligatorily changing the structure of the latter.

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

compute the two sets from queries on the metadata repository of process definitions. Queries for the successor and after relationships can be defined in a similar way.

Suppose that the final SQL expression of a type *t*, say *e*, changes into *e'*. In the spirit of [24], we can use the following rules for schema evolution in a data warehouse environment (we consider that the changes abide by the SQL syntax and the new expression is valid):

- If the *select clause* of *e'* has an extra attribute from *e*, then propagate the extra attribute to the base relations: there must be at least one path from one type belonging to a *SourceSchema* to an activity whose out expression involves the extra attribute. If we delete an attribute from the select clause of a Type, it must not appear in the select clause of the processes that directly populate the respective type, as well as in the following Types and the processes that use this Type. In the case of attribute addition, the impact is direct for the previous objects **B**(*t*) and implicit for the successor objects **A**(*t*). For deletion the impact is direct for both categories.
- If the *where clause* of *e*' is more strict than the one of *e*, the *where clause* of at least one process belonging to **B**(*t*) must change identically. If this is not possible, a new process can be added before *t* simply deleting the respective tuples through the expression *e'-e*. If the *where clause* of *e*' is less strict than the one of *e*, subsumption techniques [54, 37, 23, 46] determine which types can be used to calculate the new expression *e'* of *t*. The *having clause* is treated in the same fashion. The impact is direct for the previous and implicit for the successor objects.
- If an attribute is deleted from the *group by clause* of *e*, at least the last activity performing a *group-by* query should be adjusted accordingly. All consequent activities in the population chain of *t* must change too (as if an attribute has been deleted). If this is not feasible we can add an aggregating process performing this task exactly before *t*. If an extra attribute is added to the *group by clause* of *e*, then at least the last activity performing a *group -by* query should be adjusted accordingly. The check is performed recursively for the types populating this particular type, too. If this fails, the subsumption techniques mentioned for the *where-clause* can be used for the same purpose again. The impact is direct both for previous and successor objects. Only in the case of attribute addition it is implicit for the successor objects.

Returning to our example, suppose that we decide to remove attribute *CLASS_C* from the table *BUFFER*. This change has the following impacts:

- The activities belonging to *Previous(BUFFER)*, namely *Loading* and *Cleaning*, must change accordingly, to remove *CLASS_C* from their *select clause*.
- The activities belonging to *Next(BUFFER)*, namely *Cleaning* and *Computation* must also change to remove any appearance of *CLASS_C* in their query expression.

Previous	Loading	`SELECT H_ID, DATE, CLASS_A, CLASS_B, ~~CLASS_C~~ FROM CBL`

	Cleaning	`SELECT H_ID, DATE, CLASS_A, CLASS_B, ~~CLASS_C~~ FROM BUFFER B1`
		`WHERE EXISTS (SELECT B2.H_ID, B2.DATE FROM BUFFER B2 WHERE`
		`B1.H_ID = B2.H_ID AND B1.DATE = B2.DATE GROUP BY`
		`H_ID,DATE HAVING COUNT(*)> 1)`

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

	Cleaning	As is, there is no existence of CLASS_C in its definition any more.

Fig. 20. Impact of the removal of attribute CLASS_C from table BUFFER to the definitions of the affected activities

Note that *Cleaning* participates both in the *Previous* and in the *Next* list. Cycles in the computations are avoided, though, due to the special care we have taken in the definition of the query class *Before*. Fig. 20 shows the impact of these changes. Attribute *CLASS_C* is removed from the select list of the *Previous* activities and from the body of the queries of the *Next* list.

Due to the existence of implicit impacts, we do not provide a fully automated algorithmic solution to the problem, but rather, we sketch a methodological set of steps, in the form of suggested actions to perform this kind of evolution. Similar algorithms for the evolution of views in data warehouses can be found in [24, 4]. A tool could easily visualize this evolution plan and allow the user to react to it.

6. Related Work

In this section we discuss the state of art and practice for research efforts, commercial tools and standards in the fields of process and workflow modeling, with particular focus to data warehousing.

6.1 Standards

The standard [64] proposed by the *Workflow Management Coalition (WfMC)* includes a metamodel for the description of a workflow process specification and a textual grammar for the interchange of process definitions. A *workflow process* comprises a network of *activities*, their interrelationships, criteria for staring/ending a process and other information about *participants*, invoked *applications* and relevant *data*. Also, several other entities external to the workflow, such as system and environmental data or the organizational model are roughly described.

The *MetaData Coalition (MDC)*, is an industrial, non-profit consortium which aims to provide a standard definition for enterprise metadata shared between databases, CASE tools and similar applications. The *Open Information Model (OIM)* [40] is a proposal (led by Microsoft) for the core metadata types found in the operational and data warehousing environment of enterprises. The OIM uses UML both as a modeling language and as the basis for its core model. The OIM is divided in *packages* extend UML in order to address different areas of information management. The *Database and Warehousing Model* is composed from the *Database Schema Elements* package, the *Data Transformations Elements* package, the *OLAP Schema Elements* package and the *Record Oriented Legacy Databases* package. The *Database Schema Elements* package contains three other packages: a *Schema Elements* package (covering the classes modeling tables, views, queries, indexes, etc.), a *Catalog and Connections* package (covering physical properties of a database and the administration of database connections) and a *Data Types* package, standardizing a core set of database data types. The *Data Transformations Elements* package covers basic transformations for relational-to-relational translations. It does not deal with data warehouse process modeling (i.e., it does not cover data propagation, cleaning rules, or querying), but covers in detail the sequence of steps, the functions and mappings employed and the execution traces of data transformations in a data warehouse environment.

6.2 Commercial tools

Basically, commercial ETL tools are responsible for the implementation of the data flow in a data warehouse environment which is only one (albeit important) of the data warehouse processes. Most ETL tools are of two flavors: *engine-based*, or *code-generation based*. The former assumes that all data have to go through an engine for transformation and processing. In code-generating tools all processing takes place only at the target or source systems. There is a variety of such tools in the market; we mention three engine-based tools, from Ardent [1], DataMirror [16] and Microsoft [42, 3], and one code-generation based from ETI [18].

6.3 Research Efforts

Workflow Modeling. There is a growing research interest in the field of workflow management. [52] use a simplified workflow model, based on [64], using *tasks* and *control flows* as its building elements. The authors present an algorithm for identifying structural conflicts in a control flow specification. The algorithm uses a set of graph reduction rules to test the correctness criteria of *deadlock freedom* and *lack-of-synchronization freedom*. In [38] the model is enriched with modeling constructs and algorithms for checking the consistency of workflow temporal constraints. In [8], the authors propose a conceptual model and language for workflows. The model gives the basic entities of a workflow engine and semantics about the execution of a workflow. The proposed model captures the mapping from workflow specification to workflow execution (in particular concerning exception handling). Importance is paid to task interaction, the relationship of workflows to external agents and the access to databases. Other aspects of workflow management are explored in [7, 9]. In [35] a general model for transactional workflows is presented. A transactional workflow is defined to consist of several tasks, composed by constructs like ordering, contingency, alternative, conditional and iteration. Nested workflows are also introduced. Furthermore, correctness and acceptable termination schedules are defined over the proposed model. In [13] several interesting research results on workflow management are presented in the field of electronic commerce, distributed execution and adaptive workflows. A widely used web server for workflow literature is maintained by [34].

Process modeling. Process and workflow modeling have been applied in numerous disciplines. In [27] the authors propose a software process data model to support software information systems with emphasis on the control, documentation and support of decision making for software design and tool integration. Among other features, the model captures the representation of design objects ("what"), design decisions ("why") and design tools ("how"). A recent overview on process modeling is given in [51], where a categorization of the different issues involved in the process engineering field is provided. The proposed framework consists of four different but complementary viewpoints (expressed as "worlds"): the *subject* world, concerning the definition of the process with respect to the real world objects, the *usage* world, concerning the rationale for the process with respect to the way the system is used, the *system* world, concerning the representation of the processes and the capturing of the specifications of the system functionality and finally, the *development* world, capturing the engineering meta-process of constructing process models. Each world is characterized by a set of *facets*, i.e., attributes describing the properties of a process belonging to it.

Data Quality and Quality Management. There has been a lot of research on the definition and measurement of data quality dimensions [66,63,62,56]. A very good review of research literature is found in [65]. [29] give an extensive list of quality dimensions for data warehouses, and in particular data warehouse relations and data. Several goal hierarchies of quality factors have been proposed for software quality. For example, the GE Model [43] suggests 11 criteria of software quality, while B. Boehm's [5] suggests 19 quality factors. ISO 9126 [26] suggests six basic factors which are further refined to an overall 21 quality factors. In [25] a comparative presentation of these three models is offered and the SATC software quality model is proposed, along with metrics for all their software quality dimensions. In [21] a set of four basic quality dimensions for workflows is suggested also. Variants of the Goal-Question-Metric (GQM) approach are widely adopted in software quality management [47]. A structured overview of the issues and strategies, embedded in a repository framework, can be found in [31]. [29, 30] provide extensive reviews of methodologies employed for quality management, too.

Research focused specifically on ETL. The *AJAX* data cleaning tool developed at INRIA [22] deals with typical data quality problems, such as the object identity problem, errors due to mistyping and data inconsistencies between matching records. AJAX provides a framework wherein the logic of a data cleaning program is modeled as a directed graph of data transformations (mapping, matching, clustering and merging transformations) that start from some input source data. AJAX also provides a declarative language for specifying data cleaning programs, which consists of SQL statements enriched with a set of specific primitives to express mapping, matching, clustering and merging transformations.

6.4 Relationship of our Proposal to State-of-the-Art Research and Practice.

Our approach has been influenced by ideas on dependency and workflow modeling stemming from [8, 52, 27, 50, 68, 20, 64]. As far as the standards are concerned, we found both the Workflow Reference Model and the Open Information Model too abstract for the purpose of a repository serving well focused processes like the ones in a data warehouse environment. First, the relationship of an activity with the data it involves is not really covered, although this would provide extensive information of the data flow in the data warehouse. Second, the separation of perspectives is not clear, since the standards focus only on the structure of the workflows. To compensate this shortcoming, we employ the basic idea of the Actor-Dependency model [68] to add a conceptual perspective to the definition of a process, capturing the reasons behind its structure. Moreover, we extend [68] with the notion of suitability. As far as data quality and quality engineering are concerned, we have taken into account most of the previous research for our proposed quality dimensions and factors.

7. Conclusions

In this paper we have described a metamodel for data warehouse operational processes and techniques to design, administrate and facilitate the evolution of the data warehouse through the exploitation of the entities of this metamodel. This metamodel takes advantage of the clustering of its entities in logical, physical and conceptual perspectives, involving a high level conceptual description, which can be linked to the actual structural and physical aspects of the data warehouse architecture. This approach is integrated with the results of previous research, where data warehouse architecture and quality metamodels have been proposed assuming the same categorization.

The physical perspective of the proposed metamodel covers the execution details of data warehouse processes. At the same time, the logical perspective is capable of modeling the structure of complex

activities and capture all the entities of the Workflow Management Coalition Standard. Due to the data oriented nature of the data warehouse activities, their relationship with data stores is particularly taken care of, through clear and expressive semantics in terms of SQL definitions. This simple idea reverts the classical belief that data warehouses are simply collections of materialized views. Instead of directly assigning a naïve view definition to a data warehouse table, we can deduce its definition as the outcome of the combination of the processes that populate it. This new kind of definition does not necessarily refute the existing approaches, but rather complements them, since the former provides the operational semantics for the content of a data warehouse table, whereas the latter give an abstraction of its intentional semantics. The conceptual perspective is a key part of our approach as in the Actor Dependency model [68]. Also, we generalize the notion of role to uniformly capture any person, program or data store participating in the system. Furthermore, the process metamodel is linked to a quality metamodel, thereby facilitating the monitoring of the quality of data warehouse processes and a quality-oriented evolution of the data warehouse.

In the process of taking design decisions or understanding the occurring errors over the architecture of a data warehouse, the proposed metamodel can exploited in various ways. As far as the design of the data warehouse is concerned, simple query facilities of the metadata repository are sufficient to provide the support for consistency checking of the design. Moreover, the entities of the conceptual perspective serve as a documentation repository for the reasons behind the structure of the data warehouse. Second, the administration of the data warehouse is also facilitated in several ways. The measurement of data warehouse quality, through the linkage to a quality model, is crucial in terms of enabling the desired functionality during the everyday use of the warehouse.

Evolution is supported by the role interdependencies with two ways. At the entity level, the impact of any changes in the architecture of the warehouse can be detected through the existence of dependency links. At the same time, the existence of suitability links suggests alternatives for the new structure of the warehouse. At the attribute level, on the other hand, internal changes in the schema of data stores or the interface of software agents can also be detected by using the details of the relationships of the data warehouse roles.

We have used our experiences from real world cases as a guide for the proposed metamodel. As far as the practical application of our ideas in the real world is concerned, we find that the field of ETL and data warehouse design tools is the most relevant to our research. As a partial evaluation of our ideas and to demonstrate the efficiency of our approach, we have developed a prototype ETL tool.

Research can follow our results in various ways. First, it would be interesting to explore automated ways to assist the involved stakeholders (data warehouse designers and administrators) to populate the metadata repository with the relevant information. An example for how this can be achieved is explained in Section 3.4 for the suitability interrelationships. Specialized tools and algorithms could assist in extending this kind of support for more aspects of the proposed metamodel. Also, in this paper we have dealt only with the operational processes of a data warehouse environment. Design processes in such an environment may not fit this model so smoothly. It would be worth trying to investigate the modeling of the design processes and to capture the trace of their evolution in a data warehouse. Finally, we have used the *global-as-view* approach for the definitions of the data warehouse processes, i.e., we reduce these definitions in terms of the sources in the warehouse architecture. We plan to investigate the possibility of using the *local-as-view* approach, which means reducing both the process definitions and the data sources to a global enterprise model. The local-as-view approach appears to be more suitable to environments where a global, corporate view of data is present and thus, provides several benefits that the global-as-view approach lacks [10].

Acknowledgments. This research is sponsored in part (a) by the European Esprit Projects "DWQ: Foundations of Data Warehouse Quality", No. 22469, and "MEMO: Mediating and Monitoring Electronic Commerce", No. 26895, (b) by the Deutsche Forschungsgemeinsschaft (DFG) under the Collaborative Research Center IMPROVE (SFB 476) and (c) by the Greek General Secretariat of Research and Technology in the context of the project "Data Quality in Decision Support Systems" of the French-Greek Cooperation Programme "Plato", 2000. We would like to thank our DWQ partners who contributed to the progress of this work, especially Mokrane Bouzeghoub, Manfred Jeusfeld, Maurizio Lenzerini, and Timos Sellis. Many thanks are also due to the anonymous reviewers for their useful comments and the interesting issues they have raised.

References

[1] Ardent Software. DataStage Suite. Available at http://www.ardentsoftware.com/

[2] F. Baader, U. Sattler. Description Logics with Concrete Domains and Aggregation. In *Proc. of the 13th European Conf. on Artificial Intelligence (ECAI-98)*, pp. 336-340, Brighton, UK (1998).

[3] P.A. Bernstein, T. Bergstraesser. Meta-Data Support for Data Transformations Using Microsoft Repository. *Bulletin of the Technical Committee on Data Engineering*, **22**(1): 9-14, (1999).

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

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

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

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

[8] F. Casati, S. Ceri, B. Pernici, G. Pozzi. Conceptual Modeling of Workflows. In *Proc. 14th Intl. Conf. on Object-Oriented and Entity-Relationship Modelling (ER'95)*, pp. 341-354, Gold Coast, Australia, (1995).

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

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

[11] S. Chaudhuri, K. Shim. Optimizing Queries with Aggregate Views. In *Proc. 5th Intl. Conf. on Extending Database Technology (EDBT)*, pp. 167-182, Avignon, France 1996.

[12] S. Chaudhuri, S. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *Proc. of the 11th Intl. Conf. on Data Engineering (ICDE)*, pp. 190-200, Taipei (1995).

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

[14] U. Dayal. Of Nests and Trees: A unified approach to processing queries that contain nested subqueries, aggregates and quantifiers. In *Proc. 13th Intl. Conf. on Very Large Data Bases (VLDB)*, pp. 197-208, Brighton, UK, (1987).

[15] M. Demarest. The politics of data warehousing. http://www.hevanet.com/demarest/marc/dwpol.html (1997).

[16] DataMirror Corporation. Transformation Server. Available at http://www.datamirror.com

[17] The Data Warehouse Information Center. An (Informal) Taxonomy of Data Warehouse Data Errors. Available at http://www.dwinfocenter.org/errors.html

[18] Evolutionary Technologies Intl.. ETI*EXTRACT. Available at http://www.eti.com/

[19] Richard A. Ganski, Harry K. T. Wong: Optimization of Nested SQL Queries Revisited. In *Proc. ACM SIGMOD Intl. Conf. on the Management of Data*, pp. 23-33, San Francisco, California (1987).

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

[21] D. Georgakopoulos, M. Rusinkiewicz. Workflow management: From business process automation to inter-organizational collaboration. *Tutorials 23rd Intl. Conf. Very Large Data Bases (VLDB)*, Athens, Greece (1997).

[22] H. Galhardas, D. Florescu, D. Shasha and E. Simon. Ajax: An Extensible Data Cleaning Tool. In *Proc. ACM SIGMOD Intl. Conf. on the Management of Data*, pp. 590, Dallas, Texas, (2000).

[23] Ashish Gupta, Venky Harinarayan, Dallan Quass: Aggregate-Query Processing in Data Warehousing Environments. In *Proc. 21st Conf. on Very Large Data Bases (VLDB)* pp. 358-369, Zurich, Switzerland (1995).

[24] A. Gupta, I. Mumick, K. Ross. Adapting Materialized Views after Redefinitions. In *Proc. ACM SIGMOD Intl. Conf. on the Management of Data*, pp. 211-222, San Jose, CA (1995).

[25] L. Hyatt, L. Rosenberg, A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality, In *Proc. 8th Annual Software Technology Conf.*, Utah, (1996).

[26] ISO, Intl. Organization for Standardization. *ISO/IEC 9126:1991 Information Technology – Software product evaluation – Quality characteristics and guidelines for their use* (1991).

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

[28] 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**(2): 167-192 (1995).

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

[30] M. Jarke, M. Lenzerini, Y. Vassiliou, P. Vassiliadis (eds.). *Fundamentals of Data Warehouses*. Springer, (2000).

[31] M. Jarke, 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*, pp. 345-375, Minneapolis (1992).

[32] M.A. Jeusfeld, C. Quix, M. Jarke: Design and Analysis of Quality Information for Data Warehouses. In *Proc. of the 17th Intl. Conf. on Conceptual Modeling (ER'98)*, pp. 349-362, Singapore (1998).

[33] Won Kim. On Optimizing an SQL-like Nested Query. *ACM Trans. On Database Systems* **7**(3): 443-469 (1982)

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

[35] D. Kuo, M. Lawley, C. Liu, M. Orlowska. A General Model for Nested Transactional Workflows. In *Proc. of Intern. Workshop on Advanced Transaction Models and Architecture*, India (1996).

[36] A. Levy, I. Mumick, Y. Sagiv. Query Optimization by Predicate Move-Around. In *Proc. 20th Intl. Conf. on Very Large Data Bases (VLDB)*, pp. 96-107, Chile (1994).

[37] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, Divesh Srivastava. Answering Queries Using Views. In *Proc. of 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pp. 95-104, San Jose, California (1995).

[38] O. Marjanovic, M. Orlowska. On Modeling and Verification of Temporal Constraints in Production Workflows. *Knowledge and Information Systems* **1**(2): 157-192 (1999).

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

[40] MetaData Coalition. *Open Information Model, version 1.0* (1999). Available at www.MDCinfo.com

[41] I. Mumick, S. Finkelstein, H. Pirahesh, R. Ramakrishnan. Magic is Relevant. In *Proc. ACM SIGMOD Intl. Conf. on the Management of Data*, pp. 247-258, Atlantic City, NJ (1990).

[42] Microsoft Corp. MS Data Transformation Services. Available at ww.microsoft.com/sq

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

[44] M. Muralikrishna: Optimization and Dataflow Algorithms for Nested Tree Queries. In *Proc. 15th Intl. Conf. on Very Large Data Bases (VLDB)*, pp. 77-85, Amsterdam, The Netherlands (1989).

[45] M. Muralikrishna. Improved Unnesting Algorithms for Join Aggregate SQL Queries. *Proc. 18th Intl. Conf. on Very Large Data Bases (VLDB)*, pp. 91-102, Vancouver, Canada (1992).

[46] W. Nutt, Y. Sagiv, S. Shurin. Deciding Equivalences among Aggregate Queries. *Proceedings 17th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems (PODS'98),* pp. 214-223, Seattle, (1998).

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

[48] H. Pirahesh, J. Hellerstein, W. Hasan. Extensible/Rule Based Query Rewrite Optimization in Starburst. In *Proc. ACM SIGMOD Intl. Conf. on the Management of Data*, pp. 39-48, San Diego, California (1992).

[49] C. Quix, M. Jarke, M. Jeusfeld, P. Vassiliadis, M. Lenzerini, D. Calvanese, M. Bouzeghoub. *Data warehouse architecture and quality model. DWQ Technical Report DWQ–RWTH–002*, (1997). Available at http://www.dbnet.ece.ntua.gr/~dwq

[50] B. Ramesh, V. Dhar: Supporting systems development by capturing deliberations during requirements engineering. *IEEE Transactions on Software Engineering* **18**(6) :498-510 (1992).

[51] C. Rolland: A comprehensive view of process engineering. In *Proc. 10th Intl. Conf. Advanced Information Systems Engineering,* (CAiSE'98), pp. 1-25, Pisa, Italy (1998).

[52] W. Sadiq, M. Orlowska. Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models. In *Proc. 11th Conf. Advanced Information Systems Engineering,* (CAiSE'99), pp. 195-209, Heidelberg, Germany (1999).

[53] E. Schäfer, J.-D. Becker, M. Jarke: DB-PRISM – integrated data warehouses and knowledge networks for bank controlling. *Proc. 26th Intl. Conf. Very Large Data Bases (VLDB),* Cairo, Egypt, pp. 715-718, (2000).

[54] D. Srivastava, S. Dar, H. V. Jagadish, A.Y. Levy. Answering Queries with Aggregation Using Views. In *Proc. of 22nd Intl. Conf. on Very Large Data Bases (VLDB)*, pp. 318-329, Bombay, India, (1996).

[55] C. Shilakes, J. Tylman. Enterprise Information Portals. Enterprise Software Team. Available at http://www.sagemaker.com/company/downloads/eip/indepth.pdf (1998).

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

[57] P. Vassiliadis, M. Bouzeghoub, C. Quix. Towards Quality-Oriented Data Warehouse Usage and Evolution. *Information Systems*, **25**(2): 89-115 (2000). A previous version appeared in *Proc. 11th Conf. of Advanced Information Systems Engineering (CAiSE '99)*, pp. 164-179, Heidelberg, Germany (1999).

[58] P. Vassiliadis. Gulliver in the land of data warehousing: practical experiences and observations of a researcher. In *Proc. 2nd Intl. Workshop* on *Design and Management of Data Warehouses (DMDW)*, pp. 12.1 –12.16, Stockholm, Sweden (2000).

[59] P. Vassiliadis. Data Warehouse Modeling and Quality Issues. *Ph.D. Thesis* (2000).

[60] P. Vassiliadis. Modeling Multidimensional Databases, Cubes and Cube Operations. In *Proc. 10th Intl. Conf. on Statistical and Scientific Database Management (SSDBM)*, pp. 53-62, Capri, Italy (1998).

[61] ViaSoft. Visual Recap Quality Measurements. ViaSoft White Papers, 1996. Available at http://www.viasoft.com/download/white_papers/

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

[63] R.Y. Wang, H.B. Kon, S.E. Madnick. Data Quality Requirements Analysis and Modeling. In *Proc. of 9th Intl. Conf. on Data Engineering*, pp. 670-677, IEEE Computer Society, Vienna, Austria (1993).

[64] Workflow Management Coalition. *Interface 1: Process Definition Interchange Process Model. Document number WfMC TC-1016-P* (1998). Available at www.wfmc.org

[65] R.Y. Wang, V.C. Storey, C.P. Firth. A Framework for Analysis of Data Quality Research. *IEEE Transactions on Knowledge and Data Engineering*, **7**(4): 623-640 (1995).

[66] Y. Wand, R.Y. Wang. Anchoring Data Quality Dimensions in Ontological Foundations. *Communications of the ACM*, 39(11): 86-95 (1996).

[67] E. Yu. Strategic Modeling for Enterprise Integration. In *Proc. 14th World Congress of Intl. Federation of Automatic Control (IFAC'99)*, pp. 127-132, Beijing, China (1999).

[68] E. Yu, J. Mylopoulos. Understanding 'Why' in Software Process Modelling, Analysis and Design. In *Proc. 16th Intl. Conf. on Software Engineering (ICSE)*, pp. 159-168, Sorrento, Italy (1994).

Appendix I

In this Appendix, we list quality factors for the software involved a data warehouse environment. We organize the presentation of these quality factors around the lifecycle stage during which they are introduced. The examined lifecycle stages are *requirements analysis, system design, implementation and testing* (including the quality of the software module itself), *management* and *administration*. For each of these stages, we list the involved stakeholder and its products (depicted in the header of each table). Each of the following tables comprises three columns: the leftmost describes a high level quality dimension, the middle column shows the related quality factors of the dimension and the rightmost column suggests measurement methods for each quality factor. We make a full presentation, including both generic quality factors, applicable to any kind of software module and data warehouse specific quality factors. The former are presented in gray background. To provide the list of these quality factors we have relied on existing research and practitioner publications, market tools and personal experiences [25, 61, 17, 57, 29].

Who: **Designer**	**When:** **Requirements Analysis**	**Output:** **Requirements Document** **Role Conceptual model**
Generic Requirements Quality	Ambiguity	Number of weak phrases
		Number of optional phrases
	Completeness	Number of to-be-announced, to-be-added
	Understandability	Document structure (depth, breadth)
		Readability index
	Volatility	Count of changes / Count of requirements
		Lifecycle stage when change is made
	Traceability	Number of software requirements not traced to system requirements
		Number of software requirements not traced to code and tests
DW Process Consistency	Consistency	Number of contradictions to the conceptual schema of data stores
		Number of contradictions to enterprise model (concepts + constraints)

Who: **Designer**	**When:** **System Design**	**Output:** **Activity model**
DW Design Quality	Completeness	Number of modules not specified in terms of data
		Number of modules without care for trace management
		Number of modules without care for error reporting
	Consistency	Number of modules outside a flow sequence
		Number of contradictions to the logical schema of data stores
		Number of contradictions to the physical schema of data stores

Who: **Programmer**	**When:** **Testing**	**Output:** **Software cases**
Testing Effectiveness	Correctness	Reason for error
		Time of finding errors
		Time of error fixes & cumulative error curve
		Code location of fault
		Criticality of error
Traceability	Traceability	Number of empty cells in traceability matrix

Who: **Programmer**	**When:** **Implementation**	**Output:** **Software module (agent)**
Functionality		
Suitability	Reputation (for purchased SW)	Score in benchmark
		Market share
		Stock price
		Comments from other users
	Traceability	Number of empty cells in traceability matrix of requirements, roles, activities, tests, SW modules
		Number of enterprise rules & constraints tracked in cleaning software
Accuracy		Data Quality (see below)
	Data Completeness	Numbers of loaded records / number of expected records
		Number of null values in a column / number of rows
		Selectivity of extracted rows

		Percentage of involved columns
	Data Accuracy	Performance of statistic check: number of rows contradictory to real world values
	Data Consistency	Number of rows with reference violations / number of rows
		Number of duplicate rows / number of rows
		Number of rows violating business rules / number of rows
		Number of groups of tables with inconsistent granularity for the same information
		Number of rows with different codes for the same object / number of rows
		Number of rows with same codes for different objects / number of rows
		Number of rows having unknown codes / number of rows
	Interoperability	Number of modules unable to interact with specified systems.
	Compliance	Number of modules not compliant with application related standards or conventions or regulations in laws and similar prescriptions.
	Security	Number of modules unable to prevent unauthorized access, whether accidental or deliberate, to programs and data.
Reliability	Maturity	Number of error alerts over time period
	Fault tolerance	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	Number of occasions where the software was unable to re-establish its level of performance and recover the data affected in the case of a failure. The time and effort needed for this reestablishment.
Usability	Understandability	Percentage of acceptance by the users.
	Learnability	Percentage of acceptance by the users.
	Operability	Existence of time prediction for operation
		Number of error alerts over time period
		Number of built-in transformations in extraction and transformation SW
		Number of constructed transformations in extraction and transformation SW
Software Efficiency	Time behavior	Throughput: maximum (average) propagation time / available time window
		Availability window of each source
		Estimated response time of extraction for each source
		Extraction frequency of each source
	Resource Behavior	History duration for each DW store
		Volume of data extracted and integrated
		Max data size the software can handle / required data size
Maintainability	Analyzability	Comment Percentage
		Readability Index
	Changeability, Stability, Testability	Number of logical paths in a module
		Number of groups of code having more than one entry or exit point
		Number of paths + number of control variables
		Total number of operators and operands / discrete number of operators and operands
		Number of control flow intersections
		Number of lines of dead code
		Number of data items that are not referenced
		Number of GOTO commands
		Number of entry points for a module (having more than one entry points)
		Number of exit points for a module (having more than one exit points)
		Number of recursions
		Size
		Correlation of complexity, size
Portability		Same as Maintainability

Who: **Manager**	**When:** **Management**	**Output:** **Observation of the whole process**
Implementation Effectiveness	Resource Usage	Staff hours spent on life cycle activities
	Completion Rates	Tasks completions
		Planned task completions

Who: **Administrator**	**When:** **DW and Source Administration**	**Output:** **Administration activities**
Administration Operability	DW Availability	Same as SW efficiency
	Source Availability	Same as SW efficiency
	Error Reporting	Same as SW Reliability