A Model for Data Warehouse Operational Processes

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 (DW) 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 DW, i.e., with a metamodel for DW operational processes. The proposed metamodel is capable of modeling complex activities, their interrelationships, and the relationship of activities with data sources and execution details. Finally, the metamodel complements proposed architecture and quality models in a coherent fashion, resulting in a full framework for DW metamodeling, capable of supporting the design, administration and evolution of a DW. We have implemented this metamodel using the language Telos and the metadata repository system ConceptBase.

1 Introduction

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

All the data warehouse components, processes and data are – or at least should be – tracked and administered from a metadata repository. The metadata repository controls the data warehouse components and is therefore the essential starting point for design and operational optimization. Moreover, the schema of the metadata repository, expressed as the *DW architecture model* should also be powerful enough to capture the semantics and structure of the data warehouse processes (design, refreshment, cleaning, etc.). Expressiveness and ease of use of the metadata schema are crucial for data warehouse quality [JQJ98].

In [JJQV99], we have presented a metadata modeling approach which enables the capturing of the *static* parts of the architecture of a data warehouse, along with information over different quality dimensions of these components. The linkage of the architecture model to quality parameters (quality model) and its implementation in ConceptBase have been formally described in [JQJ98]. [VBQ99] presents a methodology for the actual exploitation of the information found in the metadata repository and the quality-oriented evolution of a data warehouse based on the architecture and quality model. In this paper, we complement these approaches with the metamodeling for the *dynamic* part of the data warehouse environment: the *processes*.

As we will show in this paper, this kind of meta-information can be used to support the design and the evolution of the DW. In these phases of the DW lifecycle, the designers of a DW need information about processes: what are they supposed to do, why are they necessary, how are they implemented and how they affect other processes in the DW. To this end, we present a model for DW processes, influenced by ideas developed in the area of workflow management systems [CCPP95, SaOr99, GeHS95, WMC98] as well as the Actor-Dependency model [YuMy94] (and its latest version, the Strategic Dependency model [Yu99]). We have identified several requirements for a DW process model, which are close to the specific nature of such an environment. Specifically, these requirements are:

1. **Complexity of structure**: the DW processes are quite complex in their nature, in terms of tasks executed within a single process, execution coherence, contingency treatment, etc. The complexity of structure is also confusing for the zooming in and out the repository, which is a well-known requirement in the field of repository management.
2. **Relationship of processes with involved data**: the DW processes are data intensive by nature. To handle the data flow in a repository, it is important that the repository is able to capture the interrelationship between processes and relevant data.
3. **Information on process traces**: not only the structure of a process is important; the specific traces of executed processes are required to be tracked down, too. The repository, thus, gains added value since, *ex ante* the DW stakeholders can

use it for design purposes (e.g., to select the DW objects necessary for the performance of a task) and *ex post*, people can relate the DW objects to decisions, tools and the facts which have happened in the real world [JaJR90].

4. **Clear separation of viewpoints**: As discussed in [YuMy94, JJQV99], there are different viewpoints for the world of an information system, which can be tracked in a metadata repository: *what* components it consists of (logical perspective), *how* they actually perform (physical perspective) and *why* these components exist (conceptual perspective). The DW process model should reflect this separation of viewpoints.
5. **Linkage to quality**: The ability of a metadata repository to track the quality of the involved objects can be exploited especially in the case where this particular quality is not satisfactory from the viewpoint of a DW stakeholder [JJQV99, JLVV99]. A process model, should thus be able to represent the linkage of the modeled processes to concrete factors which measure the quality of the DW.

The contribution of this paper is towards the fulfillment of all the aforementioned requirements. We do not claim that our approach is suitable for any kind of process, but rather we focus our attention to the internals of DW systems. Our model has been implemented in the metadata repository ConceptBase. The usefulness of our approach is further demonstrated by the fact that the proposed model enables DW management, design and evolution, as we will show in section 4. Our model supports the following steps in the DW lifecycle:

(i) The design of the DW is supported by extended use of consistency checks, to ensure the correctness of the representation.

(ii) The model facilitates the administration of the DW, by enabling the measurement of quality of DW processes and the spotting of inefficiencies.

(iii) A specific task of DW administration, namely evolution, is supported by the exploitation of the information on the interdependencies of DW components, to forecast any possible impact by the change of any of these components.

To facilitate these tasks, we derive the description of the DW materialized views from the process definitions used for their population, instead of treating the DW as a set of layers of materialized views.

This paper is organized as follows: in section 2 we describe the process metamodel and in section 3 we present its linkage to the quality model. In section 4 we present how the metadata repository can be exploited, when enriched with this kind of information. In section 5 we present related work and finally, in section 6 we conclude our results and present issues for future research.

2 Metamodel for Data Warehouse Operational Processes

In [JJQV99] a basic metamodel for data warehouse architecture and quality has been presented. A condensed graphical overview of the architecture meta model is given in (Fig. 1). The framework describes a data warehouse in three *perspectives*: a *conceptual*, a *logical* and a *physical* perspective. Each perspective is partitioned into the three traditional data warehouse *levels*: *source, data warehouse* and *client* level.

On the *metamodel* layer, the framework gives a notation for data warehouse architectures by specifying meta classes for the usual data warehouse objects like data store, relation, view, etc. On the *metadata* layer, the metamodel is instantiated with the concrete architecture of a data warehouse, involving its schema definition, indexes, tablespaces, etc. The lowest layer in Fig. 1 represents the real world where the actual processes and data reside.

The *static* description of the architecture parts of the DW (left part of Fig. 1) is complemented in this paper with a metamodel of the *dynamic* parts of the DW, i.e. the DW operational processes. As one can notice on the right side of Fig. 1, we follow again a three level instantiation: a *Process Metamodel* deals with generic entities involved in all DW processes (operating on entities found at the DW metamodel level), the *Process Model* covers the processes of a specific DW by employing instances of the metamodel entities and the *Process Traces* capture the execution of the actual DW processes happening in the real world.

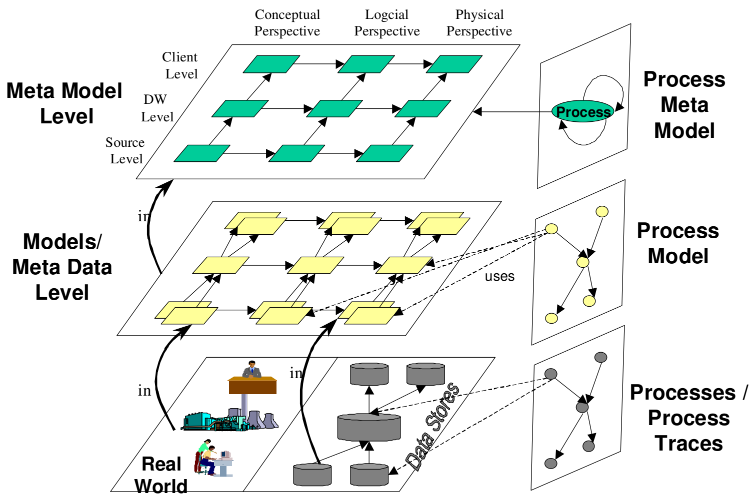

Fig. 1. Framework for Data Warehouse Architecture [JJQV99].

Our approach has been influenced by ideas on dependency and workflow modeling stemming from [CCPP95, SaOr99, JaJR90, RaDh92, YuMy94, GeHS95] and on the Workflow Reference Model, presented in [WMC98] by the Workflow Management Coalition (WfMC). We found the Workflow Reference Model too abstract for the purpose of a repository serving a specific kind of processes, the DW operational processes. First, the relationship of an activity with the data it involves is not really covered, although this would provide extensive information of the data flow in the DW. Second, the separation of perspectives is not clear, since the WfMC proposal focuses only on the structure of the workflows. To compensate this shortcoming, we employ the basic idea of the Actor-Dependency model to add a conceptual perspective to the definition of a process, capturing the reasons behind its structure.

2.1 The 3 Perspectives for the Process Model.

Our process model (cf. Fig. 3) has also different perspectives covering distinct aspects of a process: the *conceptual, logical* and *physical* perspective. The categorization fits naturally with the separation proposed in [JJQV99] (i.e., the architecture model), since the perspectives of the process model operate on objects of the respective perspective of the architecture model. As mentioned in [YuMy94] there are different ways to view a process: *what* steps it consists of (logical perspective), *how* they are to be performed (physical perspective) and *why* these steps exist (conceptual perspective). Thus, we view a DW process from three perspectives: a central *logical* part of the model, which captures the basic structure of a process, its *physical* counterpart which provides specific details over the actual components that execute the activity and the *conceptual* perspective which abstractly represents the basic interrelationships between DW stakeholders and processes in a formal way.

Typically, the information about how a process is executed concerns stakeholders who are involved in the everyday use of the process. The information about the structure of the process concern stakeholders that manage it and the information relevant to the reasons behind this structure concern process engineers who are involved in the monitoring or evolution of the process environment. In the case of DW's it is expected that all these roles are covered by the DW administration team, although one could also envision different schemes. Another important issue shown in Fig. 2 is that there is also a data flow at each of the perspectives: a type description of the incoming and outcoming data at the logical level (where the process acts as an input/output function), a physical description of the details of the physical execution for the data involved in the activity and a relationship to the conceptual entities related to these data, connected through a corresponding role.

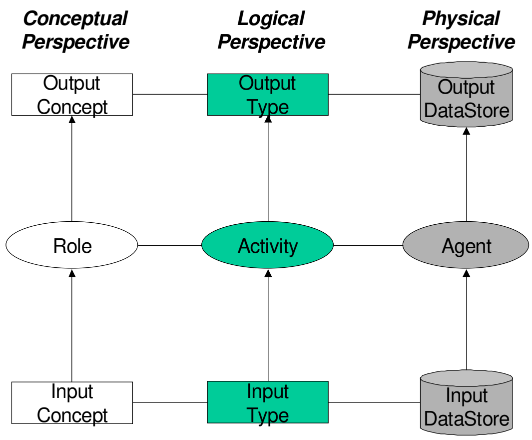

Fig. 2. The reasoning behind the 3 perspectives of the process metamodel.

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 [MBJK90], a conceptual modeling language for representing knowledge about information systems, implemented in the object-oriented deductive database system ConceptBase [JGJ+95], 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, birds'-eye view of the model, for reasons of presentation and lack of space. Wherever definitions, constraints, or queries of ConceptBase are used in the paper, they will be explained properly in natural language, too.

In the sequel, when we present the entities of the metamodel, this will be done in the context of a specific perspective. In the Telos implementation of the metamodel this is captured by specializing the generic classes C*onceptualObject, LogicalObject* and *PhysicalObject* with ISA relationships, accordingly.

We will start the presentation of the metamodel from the *logical* perspective. First, we will show how it deals with the requirements of structure complexity and capturing of data semantics in the next two sections. Then, in subsections 2.4 and 2.5 we will present the physical and the conceptual perspectives. In the former, the requirement of trace logging will be fulfilled too. The full metamodel is presented in Fig. 3.

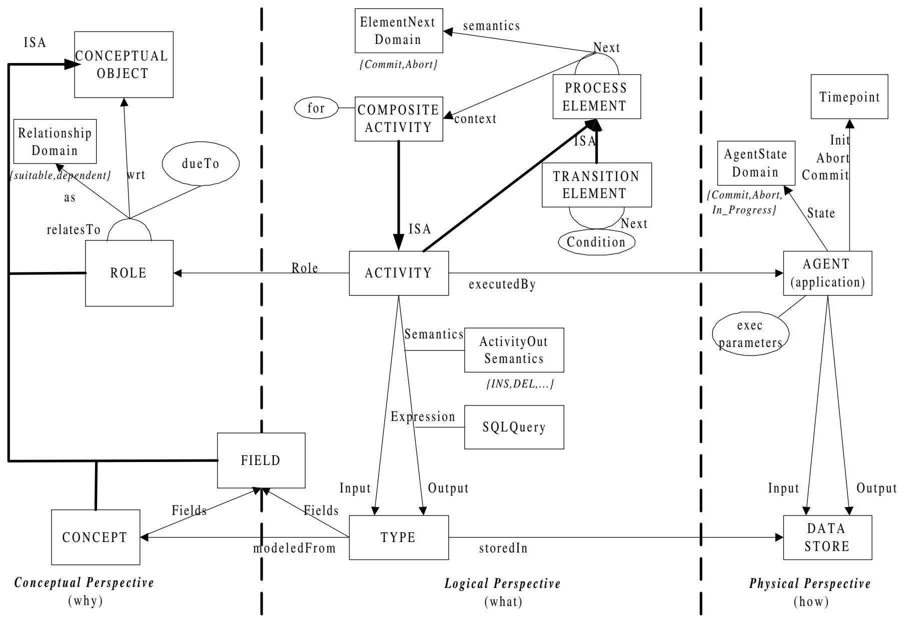

Fig. 3. The DW Operational Process Metamodel.

2.2 Complexity of the Process Structure

Following [WMC98], the main entity of the logical perspective is an *Activity*. An activity represents an amount of "work which is processed by a combination of resource and computer applications". Activities are rather complex in nature and this complexity is captured by the specialization of *Activity*, namely *CompositeActivity*. We follow here the lessons coming both from repository and workflow management: there must be the possibility of zooming in and out the repository. Composite activities are composed of *ProcessElements* which is a generalization of the entities *Activity* and *TransitionElement.* A transition element is the "bridge" between two activities: it is employed for the interconnection of activities participating in a complex activity. The attribute *Next* of the process elements captures the sequence of events.

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

- *Next*: a *ProcessElement* which is next in the sequence of a composite activity. The attribute *Next* has itself two attributes, that characterize it:
  - *Context*: A *CompositeActivity*. Since two activities can be interrelated in more than one complex DW processes, the context of this interrelationship is captured by the relevant *CompositeActivity* instance.
  - *Semantics*: This attribute denotes whether the next activity in a schedule happens on successful termination of the previous activity (*COMMIT)*

or in the case where a contingency action is required (*ABORT*). Thus, the class *ElementNextDomain* has two instances {*COMMIT, ABORT*}.

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

Second, we specialize the class Transition Element to four prominent subclasses, capturing the basic connectives of activities, as suggested by [WMC98][1]: *Split_AND, Split_XOR, Join_AND, Join_XOR*. Their semantics are obviously the same with the ones of the WfMC proposal. For example, the *Next* activity of a *Join_XOR* instance is fired when (a) the *Join_XOR* has at least two activities "pointing" to it through the *Next* attribute and only one *Next* Activity (well-formedness constraint), (b) the *Condition* of the *Join_XOR* is met and (c) only one of the "incoming" Activities has *COMMIT* semantics in the *Next* attribute. This behavior can be expressed in Telos with the appropriate rules.

The WfMC proposes two more ways of transition between Activities. The *dummy activities*, which perform routing based on condition checking are modeled in our approach, as simple *Transition Elements*. The *LOOP* activities are captured as instances of *CompositeActivity*, with an extra attribute: the *for* condition, expressed as a string.

In Fig. 4 we depict how two transitions are captured in our model. In case (a), we model a composite activity, composed of two sub-activities, where the second is fired when the first activity commits and a boolean condition is fulfilled. *P1_Commited?* is a transition element. In case (b) we give two alternatives for the capturing of two concurrent activities, fired together. *Dummy* is an activity with no real operations performed (e.g., like the *dummy* activity of WfMC).

[1] This specialization is not depicted in Fig. 3 to avoid overloading the figure with information which is not in the central focus of the paper. The same happens with several attributes of the other entities, too.

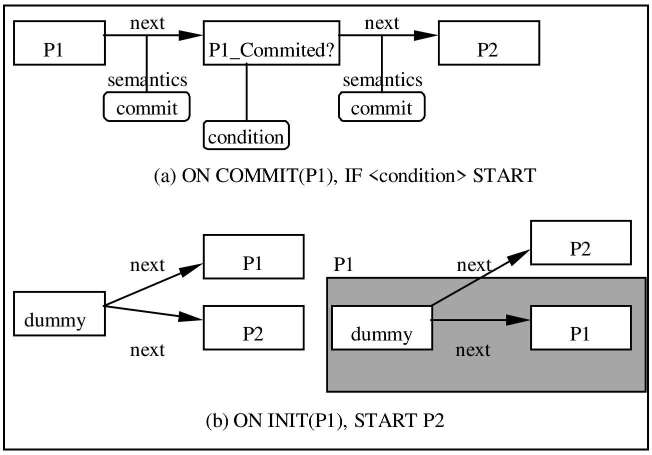

Fig. 4. Examples of complex activities.

2.3 Relationship with Data

We introduce the entity *Type* to capture the logical representation of a data store. 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*, i.e., a physical object representing any application used to manipulate stored data (e.g., a DBMS). This attribute will be detailed later, in the description of the physical perspective.

Any kind of physical data store (multidimensional arrays, COBOL files, etc.) can be represented by a Type in the logical perspective. For example, the schema of multidimensional cubes is of the form $[D_1,...,D_n,M_1,...,M_m]$ where the D_i represent dimensions (forming the primary key of the cube) and the M_j measures [Vass98]. Cobol files, as another example, are in general records with fields having two peculiarities (a) nested records and (b) alternative representations. One can easily unfold the nested records and choose only one of the alternative representations, reducing, thus, the intentional description of the Cobol file to a *Type*.

Each activity in a DW environment is linked to a set of incoming types as well as to a set of outcoming types, too. The DW activities are of data intensive nature in their attempt to push data from the data sources to the DW materialized views or client data marts. We can justify this claim by listing the most common operational processes:

- *data transfer processes*, used for the instantiation of higher levels of aggregation in the ODS and the DW 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 DW 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.);
- *extraction processes*, which are used for the extraction of information from the legacy systems;
- *loading processes*, which are used for the loading of the extracted information from the legacy systems inside the DW.

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

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

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

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

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

Fig. 5. Examples of *Output* attribute for particular kinds of activities

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

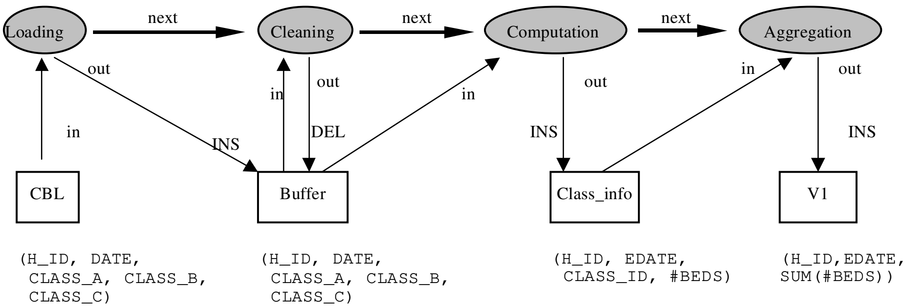

Fig. 6. Motivating Example

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

Fig. 7. Expressions and semantics for the motivating example

We assume the following four *Types*: *CBL, Buffer, Class_info* and *V1*. The schemata of these types are depicted in Fig. 6. There are four *Activities* in the DW: *Loading, Cleaning, Computation* and *Aggregation*. The *Loading* activity simply copies the data from the *CBL* Cobol file to the *Buffer* type. *H_ID* is an identifier for the hospital and the three last attributes hold the number of beds per class. The *Cleaning* activity deletes all the entries violating the primary key constraint. The *Computation* activity transforms the imported data into a different schema. The date is converted from American to European format and the rest of the attributes are converted to a combination (*Class_id, #Beds*). For example, if (03,12/31/1999,30,0,50) is a tuple in the *Buffer* table, the respective tuples in the *Class_Info* table are {(03,31/Dec/1999,A,30), (03,31/Dec/1999,50)}. The *Aggregation* activity simply produces the sum of beds by hospital and year. The expressions and semantics for each activity are listed in Fig. 7.

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

2.4 The Physical Perspective

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

- *State*: a single value belonging to the domain of class *AgentStateDomain* = {*In_Progress*, *Commit*, *Abort*}.
- *Init_time, Abort_time, Commit_time*: timestamps denoting the *Timepoints* when the respective events have occurred.
- *Execution_parameters*: a multivalue string attribute capturing any extra information about 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.

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

Fig. 8. Trace information after a successful execution of the process of the motivating example

The information of the physical perspective can be used to trace and monitor the execution of DW processes. In Fig. 8 we present the trace information after a successful execution of the process described in Fig. 6. The relationship between the logical and the physical perspectiveis done by linking each activity to a specific application program.

2.5 The Conceptual Perspective

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

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

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

A role represents any program or data store participating in the environment of a process, charged with a specific task and/or responsibility. The interrelationship between roles is modeled through the *RelatesTo* relationship.Since both data and processes can be characterized by SQL statements, their interrelationship can be traced in terms of attributes (Fig. 9). An instance of this relationship is a statement about the interrelationship of two roles in the real world, such as 'View V1 *relates to* table Class_Info *with respect to* the attributes Id, Date and number of beds *as* dependent *due to* data loading reasons'. Since both data and processes can be characterized by SQL statements, their interrelationship can be traced in terms of attributes. In Fig. 9 two examples are also used to clarify this instantiation, based on the example of Fig. 6.

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

Fig. 9. Examples of role interrelationships for the motivating example

The conceptual perspective is influenced by the Actor Dependency model [YuMy94]. In this model, the actors *depend* on each other for the accomplishment of goals and the delivery of products. The *dependency* notion is powerful enough to capture the relationship of processes where the outcome of the preceding process in the input for the following one. Still, our approach is more powerful since it can capture *suitability,* too (e.g., in the case where more than one concept can apply for the population of an aggregation, one concept is suitable to replace the other).

In the process of understanding errors or design decisions over the architecture of a DW, the conceptual perspective can be used as a reasoning aid, to discover the interdependencies of the actors (possibly in a transitive fashion) and the possible alternatives for different solutions, through a set of suitable candidates. For example the previous statement for the relationship of view V1 and table Class_Info could be interpreted as 'View V1 is affected from any changes to table Class_Info and especially the attributes Id, Date and no. of beds'. Moreover, the provided links to the logical perspective can enable the user to pass from the abstract relationships of roles to the structure of the system. On top of these, as we shall show in the last section, DW evolution can be designed and influenced by the interdependencies tracked by the *Role* entities. It can be shown that these interdependencies do not have to be directly stored,in all the cases, but can also be computed due to the transitivity of their nature.

3 Issues on Process Quality

In this section we present how the process model is linked to the quality model which has been proposed previously in [JQJ98] for DW quality, along with specific quality dimensions and factors of DW operational processes.

3.1 Terminology for Quality Management

We adopt the metamodel for the relationship between architecture objects and quality factors presented in [JQJ98], which is based on the Goal-Question-Metric approach (GQM) described in [OB92]. Each object in a DW architecture is linked to a set of quality goals and a set of quality factors (Fig. 10). A *quality goal* is an abstract requirement, defined on DW objects, and documented by a purpose and the stakeholder interested in it. A quality goal roughly expresses natural language requirements like 'improve the availability of source s1 until the end of the month in the viewpoint of the DW administrator'. *Quality dimensions* (e.g. 'availability') are used to classify quality goals and factors into different categories. A *Quality Factor* represents a quantitative assessment of particular aspect of a DW 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 the 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* (or questions), to which quality factor values are provided as possible answers. Quality questions are the outcome of the methodological approach described in [VBQ99]. The methodology offers "template" quality factors and dimensions, defined at the metadata level and instantiates them, for the specific DW 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.

3.2 Quality Dimensions of Data Warehouse Processes

For the case of processes, we will first define a set of generic quality dimensions. We do not wish to detail the whole set of possible dimensions for all the data warehouse processes, rather, we intend to come up with a minimal characteristic set, which is generic enough to cover the most basic characteristics of data warehouse processes. Then, this set of quality dimensions can be refined and enriched with customized dimensions by the DW stakeholders.

The set of generic quality dimensions is influenced mainly from the quality criteria for workflows defined in [GeRu97] and the quality dimensions for software evaluation presented in [ISO91].

1. *Correctness*: a specification exists, describing the conditions under which, the process has achieved its aim (e.g. provides the correct results).
2. *Functionality*: the process satisfies specific needs of data warehouse stakeholders.
3. *Efficiency*: the process has a good balance between level of performance and amount of used resources. Efficiency can be decomposed to sub-dimensions like *timeliness* and *resource consumption*.
4. *Maintainability*: the degree of easiness with which the process can be modified. It can also be decomposed to sub-dimensions like *analyzability* (effort required to detect deficiencies and propose changes) and *changeability* (effort required to implement these changes).

An extensive list of quality dimensions for data warehouses, and in particular DW relations and data can be found in [JJQV99].

3.3 Relationships between Processes and Quality

Quality goals are high-level abstractions of the data warehouse quality: they describe *intentions* or *plans* of the DW users with respect to the status of the DW. Moreover, we use the repository to track how quality goals are evaluated by specific questions and measurements. In contrast, our process model describes *facts* about the current status of the DW and what activities are performed in the DW. However, the reason behind the execution of a process is a quality goal which should be achieved or improved by this process. For example, a data cleaning process is executed on the ODS in order to improve the accuracy of the DW. We have represented this dependency between processes and quality goals by extending the relationship between roles and DW objects in the conceptual perspective of the process model (relationship *Expressed For*). This is shown in the upper part of Fig. 10.

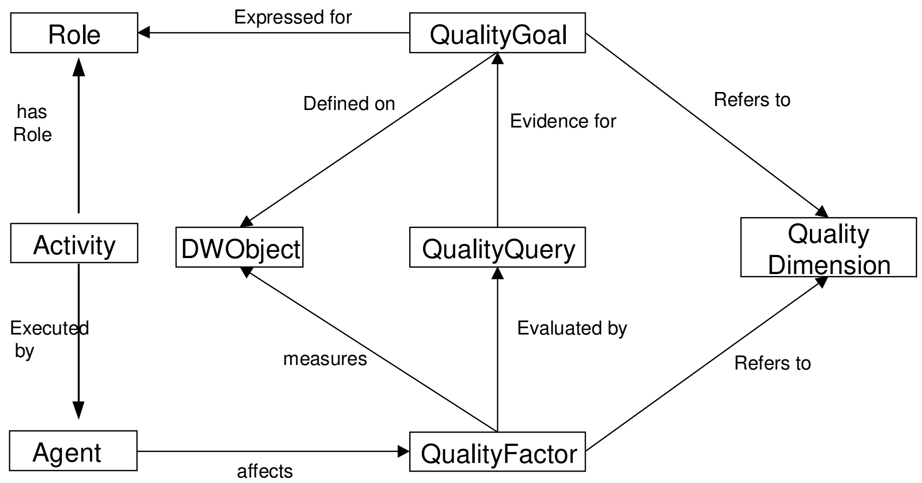

Fig. 10. Relationships between processes and quality

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

The lower part of Fig. 10 represents the relationship between processes and quality on a more operational level. The operation of an agent in the DW will have an impact on quality factors of DW objects. The relationship a*ffects* represents both the measured and expected effect of a DW process on DW quality. The effect of a DW process must always be confirmed by new measurements of the quality factors. Unexpected effects of DW processes can be detected by comparing the measurements with the expected behavior of the process. For example, a data cleaning process affects the availability of a source in the sense that it decreases the amount of time during which it can be used for its regular operations. Consequently, this process will affect the quality factor "Availability" defined on a "Source". The measurement of the quality of the particular agents through their own quality factors is analyzed in [JJQV99].

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

4 Exploitation of the Metadata Repository

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

4.1 Consistency Checking in the Metadata Repository

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

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

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

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

4.2 Interdependencies of Types and Processes

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

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

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

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

Fig. 11. Algorithm for extracting the definition of a type in the repository

We have not implemented this algorithm in ConceptBase yet, although external programs could easily do this.

Suppose that we apply the algorithm to our motivating example. Then the results are as shown in the following table. For reasons of convenience, we break composite definitions of table expressions into the different lines of Fig. 12. For example, when the 3rd iteration refers to the definition of table *Buffer*, it does so with respect to the definition of line 2.

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. 12. SQL expressions of the types of the motivating example

To give a better intuition of the problem, we present an extended version of our motivating example. In Fig. 13. we can see that we added a second COBOL file (of identical schema) as an input to the table *Buffer*. The rest of the definitions for the processes remain the same.

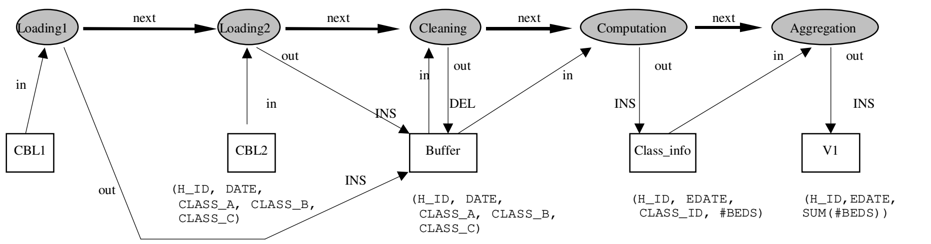

Fig. 13. Extended version of the motivating example

Attribute name	Expression	Semantics
Loading1.Out	`SELECT * FROM CBL1`	INS
Loading2.Out	`SELECT * FROM CBL2`	INS
Cleaning.Out	`SELECT * FROM BUFFER B1 WHERE EXISTS (SELECT B2.H_ID, B2.DATE FROM BUFFER B2 WHERE B1.H_ID = B2.H_ID AND B1.DATE = B2.DATE GROUP BY H_ID,DATE HAVING COUNT(*)> 1)`	DEL
Computation. Out	`SELECT H_ID, EUROPEAN(DATE) AS EDATE, 'A', CLASS_A AS #BEDS FROM BUFFER WHERE CLASS_A <> 0 UNION SELECT H_ID, EUROPEAN(DATE) AS EDATE, 'B', CLASS_B AS #BEDS FROM BUFFER WHERE CLASS_B <> 0 UNION SELECT H_ID, EUROPEAN(DATE) AS EDATE, 'C', CLASS_C AS #BEDS FROM BUFFER WHERE CLASS_C <> 0`	INS
Aggregation. Out	`SELECT H_ID, EDATE, SUM(#BEDS) AS SUM_BEDS FROM CLASS_INFO GROUP BY H_ID, EDATE`	INS

Fig. 14. Expressions and semantics for the extended example

The execution of the algorithm for the is as follows:

```
i=1
Buffer.i_expr:= SELECT * FROM CBL1
Buffer.expr:= SELECT * FROM CBL1

i=2
Buffer.i_expr:= (SELECT * FROM CBL1) UNION (SELECT * FROM
CBL2)
Buffer.expr:= (SELECT * FROM CBL1) UNION (SELECT * FROM CBL2)

i=3
Buffer.d_expr:=
(SELECT * FROM BUFFER_OLD B1 WHERE EXISTS (SELECT B2.H_ID,
B2.DATE FROM BUFFER_OLD B2 WHERE B1.H_ID=B2.H_ID AND
B1.DATE=B2.DATE GROUP BY H_ID,DATE HAVING COUNT(*)>1);
BUFFER_OLD (H_ID, DATE, CLASS_A, CLASS_B, CLASS_B) AS
((SELECT * FROM CBL1) UNION (SELECT * FROM CBL2))

Buffer.expr:=
((SELECT * FROM CBL1) UNION (SELECT * FROM CBL2)) MINUS
(SELECT * FROM BUFFER B1 WHERE EXISTS (SELECT B2.H_ID, B2.DATE
FROM BUFFER B2 WHERE B1.H_ID=B2.H_ID AND B1.DATE=B2.DATE GROUP
BY H_ID,DATE HAVING COUNT(*)>1))

i=4
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);

BUFFER (H_ID, DATE, CLASS_A, CLASS_B, CLASS_B) AS
(SELECT * FROM BUFFER_OLD B1 WHERE EXISTS (SELECT B2.H_ID,
B2.DATE FROM BUFFER_OLD B2 WHERE B1.H_ID=B2.H_ID AND
B1.DATE=B2.DATE GROUP BY H_ID,DATE HAVING COUNT(*)>1);

BUFFER_OLD (H_ID, DATE, CLASS_A, CLASS_B, CLASS_B) AS
((SELECT * FROM CBL1) UNION (SELECT * FROM CBL2))
```

i=5

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

CLASS_INFO (H_ID,DATE, CLASS_ID,#BEDS) AS
(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);

BUFFER (H_ID, DATE, CLASS_A, CLASS_B, CLASS_B) AS
(SELECT * FROM BUFFER_OLD B1 WHERE EXISTS (SELECT B2.H_ID,
B2.DATE FROM BUFFER_OLD B2 WHERE B1.H_ID=B2.H_ID AND
B1.DATE=B2.DATE GROUP BY H_ID,DATE HAVING COUNT(*)>1);

BUFFER_OLD (H_ID, DATE, CLASS_A, CLASS_B, CLASS_B) AS
((SELECT * FROM CBL1) UNION (SELECT * FROM CBL2))
```

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

4.3 Deriving Role Interdependencies Automatically

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

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

Fig. 15. Simple rules for the production of role interdependencies.

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

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

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

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

Fig. 16. Role interdependencies for the motivating example.

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

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

The information stored in the repository may be used to find deficiencies in data warehouse. To show how the quality model is exploited, we take the following query. It returns all data cleaning activities 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. the query will return only cleaning processes that are related to the specified data store. The query returns the agents which have worked on the specified data store and which were executed between the measurements of quality factors *qf1* and *qf2*, and the measured value of the newer quality factor is lower than the value of the older quality factor.

4.5 Repository Support for Data Warehouse Evolution

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

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

In our approach, we distinguish two kinds of impact:

- *Direct impact*: the change in the DW object imposes that some action must be taken against an affected object, e.g., 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 DW 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 (which can be produced by proper queries in Telos): for each *Type* instance, say *t*, there is a set of types and activities, used for the population of *t* ("before" *t*), denoted as **B**(*t*). Also, there is another set of objects using *t* for their population ("after" *t*), denoted as **A**(*t*). We can recursively compute the two sets from the following queries in ConceptBase:

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

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

Queries for the successor and after relationships can be defined in a similar way.

In a previous subsection we showed how we could derive an SQL definition for each type in the repository. Suppose that a type *t* is characterized by an expression *e* which is supported by a set of auxiliary SQL expressions producing, thus the set **e**=$\{e_1,e_2,\ldots,e\}$. Obviously some of the expressions belonging to **e** belong also to **B**(*t*). Thus, we extend **B**(*t*) as **B**(*t*)∪**e** (with set semantics). Suppose, then, that the final SQL expression of a type *t*, say *e*, changes into *e'*. Following the spirit of [GuMR95], we can use the following rules for schema evolution in a DW environment (we consider that the changes abide by the SQL syntax and the new expression is valid):

- If the *select clause* of *e'* has an extra attribute from *e*, then propagate the extra attribute down the line to the base relations: there must be at least one path from one type belonging to a *SourceSchema* to an activity whose out expression involves the extra attribute. If we delete an attribute from the select clause of a Type, it must not appear in the select clause at least of the processes that directly populate the respective type, as well as in the following Types and the processes that use this particular Type. In the case of addition of an attribute, the impact is direct for the previous objects **B**(*t*) and implicit for the successor objects **A**(*t*). In the case of deletion the impact is direct for both categories.
- If the *where clause* of *e*' is more strict than the one of *e*, then the *where clause* of at least one process belonging to **B**(*t*) must change identically. If this is not possible, a new process can be added just before *t* simply deleting the respective tuples through the expression *e'-e*. If the *where clause* of *e*' is less strict than the one of *e*, then we can use well known subsumption techniques [SDJL96, LMSS95, GuHQ95, NuSS98] to determine which types can be (re)used to calculate the new expression *e'* of *t*. The *having clause* is treated in the same fashion. The impact is direct for the previous and implicit for the successor objects.

– If an attribute is deleted from the *group by clause* of *e*, then at least the last activity performing a *group-by* query should be adjusted accordingly. All the consequent activities in the population chain of *t* must change too (as if an attribute has been deleted). If this is not feasible we can add an aggregating process performing this task exactly before *t*. If an extra attribute is added to the *group by clause* of *e*, then at least the last activity performing a *group -by* query should be adjusted accordingly. The check is performed recursively for the types populating this particular type, too. If this fails, the subsumption techniques mentioned for the *where-clause* can be used for the same purpose again. The impact is direct both for previous and successor objects. Only in the case of attribute addition it is implicit for the successor objects.

We do not claim that we provide a concrete algorithmic solution to the problem. 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 DW's can be found in [GuMR95, Bell98]. A tool could easily visualize this evolution plan and allow the userto react to it.

5 Related Work

In the field of workflow management, [WMC98] is a standard proposed by the Workflow Management Coalition (WfMC). The standard includes a metamodel for the description of a workflow process specification and a textual grammar for the interchange of process definitions. A *workflow process* comprises of a network of *activities*, their interrelationships, criteria for staring/ending a process and other information about *participants*, invoked *applications* and relevant *data*. Also, several other kinds of entities which are external to the workflow, such as system and environmental data or the organizational model are roughly described. A widely used web server for workflow literature is maintained by [Klam99].

[SaOr99] use a simplified workflow model, based on [WMC98], using *tasks* and *control flows* as its building elements. The authors present an algorithm for identifying structural conflicts in a control flow specification. The algorithm uses a set of graph reduction rules to test the correctness criteria of *deadlock freedom* and *lack-of-synchronization freedom*. In [MaOr99] the model is enriched with modeling constructs and algorithms for checking the consistency of workflow temporal constraints.

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

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

In [DaRe99] several interesting research results on workflow management are presented in the field of electronic commerce, distributed execution and adaptive workflows.

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

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

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

As far as quality management is concerned, there has be much research on the definition and measurement of data quality dimensions [WW96,WKM93,WSG94,TB98]. A very good review of research literature is found in [WSF95]. [JJQV99, JLVV99] provide extensive reviews of methodologies employed for quality management, too (e.g., the GQM approach, introduced in [OB92]).

6 Conclusions

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

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

Acknowledgments. This research is sponsored in part by the European Esprit Projects "DWQ: Foundations of Data Warehouse Quality", No. 22469, and "MEMO: Mediating and Monitoring Electronic Commerce", No. 26895, and by the Deutsche Forschungsgemeinsschaft (DFG) under the Collaborative Research Center IMPROVE (SFB 476). We would like to thank all our DWQ partners who contributed to the progress of this work, and especially Professors Timos Sellis, Mokrane Bouzeghoub, Manfred Jeusfeld and Maurizio Lenzerini. Many thanks are also due to the anonymous reviewers of the short version of this paper (accepted in CAiSE'00), for useful comments.

References

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

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

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

[CCPP95] F. Casati, S. Ceri, B. Pernici, G. Pozzi. Conceptual Modeling of Workflows.

In Proc. Of OO-ER Conference, Australia, 1995.

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

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

[ChSh96] S. Chaudhuri, K. Shim. Optimizing Queries with Aggregate Views. In EDBT Conference, 1996.

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

[Dayal87] U. Dayal. Of Nests and Trees: A unified approach to processing queries that contain nested subqueries, aggregates and quantifiers. In 13th VLDB Conference, 1987.

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

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

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

[GuHQ95] Ashish Gupta, Venky Harinarayan, Dallan Quass: Aggregate-Query Processing in Data Warehousing Environments. In Proc. of 21st VLDB Conference 1995.

[GuMR95] A. Gupta, I. Mumick, K. Ross. Adapting Materialized Views after Redefinitions. In ACM SIGMOD Conference, 1995.

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

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

[JGJ+95] M. Jarke, R. Gallersdörfer, M.A. Jeusfeld, M. Staudt, S. Eherer: ConceptBase - a deductive objectbase for meta data management. In *Journal of Intelligent Information Systems, Special Issue on Advances in Deductive Object-Oriented Databases*, 4, No. 2, pp. 167-192, 1995.

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

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

[JQB+99] M. Jarke, C. Quix, G. Blees, D. Lehmann, G. Michalk, S. Stierl. Improving OLTP Data Quality Using Data Warehouse Mechanisms. To appear in *Proc.*

ACM-SIGMOD Conf. Management of Data, Philadelphia, PA., June 1999.

[JQJ98] M.A. Jeusfeld, C. Quix, M. Jarke: Design and Analysis of Quality Information for Data Warehouses. In *Proc. of the 17th International Conference on the Entity Relationship Approach (ER'98),* Singapore, 1998.

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

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

[KLCO96] D. Kuo, M. Lawley, C. Liu, M. Orlowska. A General Model for Nested Transactional Workflows. In Proc. of Intern. Workshop on Advanced Transaction Models and Architecture, India, 1996.

[LeMS94] A. Levy, I. Mumick, Y. Sagiv. Query Optimization by Predicate Move-Around. In 20th VLDB Conference, Chile, 1994.

[LMSS95] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, Divesh Srivastava. Answering Queries Using Views. In Proc. of 14th PODS, 1995.

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

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

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

[MFPR90] I. Mumick, S. Finkelstein, H. Pirahesh, R. Ramakrishnan. Magic is Relevant. In ACM SIGMOD Conference, 1990.

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

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

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

[OB92] M. Oivo, V. Basili: Representing software engineering models: the TAME goal-oriented approach. IEEE Transactions on Software Engineering, 18, 10, 1992.

[PiHH92] H. Pirahesh, J. Hellerstein, W. Hasan. Extensible/Rule Based Query Rewrite Optimization in Starburst. In ACM SIGMOD Conference, 1992.

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

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

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

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

[TB98] G. K. Tayi, D. P. Ballou: Examining Data Quality. In Communications of the ACM, 41, 2, Feb. 1998.

[Vass98] P. Vassiliadis. Modeling Multidimensional Databases, Cubes and Cube Operations. In Proc. of the 10th SSDBM Conference, Capri, Italy, July 1998.

[VBQ99] P. Vassiliadis, M. Bouzeghoub, C. Quix. Towards Quality-Oriented Data Warehouse Usage and Evolution. In *Proc. 11th Conference of Advanced Information Systems Engineering (CAiSE '99)*, Heidelberg, Germany, 1999.

[Wang98] R. Y. Wang. A product perspective on total data quality management. In Communications of the ACM, 41, 2, Feb. 1998.

[WKM93] R.Y. Wang, H.B. Kon, S.E. Madnick. Data Quality Requirements Analysis and Modeling. In Proc. of 9th International Conference on Data Engineering, pp. 670-677, IEEE Computer Society, Vienna, Austria, 1993.

[WMC98] Workflow Management Coalition. Interface 1: Process Definition Interchange Process Model. Document number WfMC TC-1016-P, 1998. Available at www.wfmc.org

[WSF95] R.Y. Wang, V.C. Storey, C.P. Firth. A Framework for Analysis of Data Quality Research. IEEE Transactions on Knowledge and Data Engineering, Vol. 7, No. 4, August 1995.

[WW96] Y. Wand, R.Y. Wang. Anchoring Data Quality Dimensions in Ontological Foundations. Communications of the ACM, Vol. 39, No. 11, November 1996.

[Yu99] E. Yu. Strategic Modelling for Enterprise Integration. In *Proc. 14th World Congress of Int. Federation of Automatic Control*, China 1999.

[YuMy94] E. Yu, J. Mylopoulos. Understanding 'Why' in Software Process Modelling, Analysis and Design. In *Proc. 16th Int. Conference Software Engineering*, May 1994.