A Model for Data Warehouse Operational Processes

Panos Vassiliadis[1], Christoph Quix[2], Yannis Vassiliou[1], and 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 *operational data store (ODS)*, the *global data warehouse*, 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 accompanied by a detailed physical design for efficiency reasons. 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 [2]. 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,

B. Wangler, L. Bergman (Eds.): CAiSE 2000, LNCS 1789, pp. 446-461, 2000

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.

In [12], 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 [13]. [27] 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. We have identified several requirements for a DW process model. Specifically, the requirements involve: (i) the coverage of the *complexity of the structure* of DW processes, in terms of tasks executed within a single process, execution coherence, contingency treatment, etc.; (ii) the capturing of the *relationship of processes with involved data*; (iii) the *tracking of specific executed processes*, so that people can relate the DW objects to decisions, tools and the facts which have happened in the real world [10]; (iv) a *clear separation of perspectives*: *what* components a process consists of (logical perspective), *how* they perform (physical perspective) and *why* these components exist (conceptual perspective); and finally (v) the representation of the *linkage of the modeled processes to concrete quality 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 [11]. Its usefulness is demonstrated by the fact that the proposed model enables DW management, design and evolution, as we will show in section 5. 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 discuss briefly related work. Section 3 describes 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 exploited, when enriched with this kind of information. Finally, in section 6 we conclude our results and present issues for future research. For lack of space, many issues are not thoroughly presented here. We refer the interested reader to [28] for more examples and explanations.

2 Related Work

Our approach is build around the Workflow Reference Model, presented in [29] by the Workflow Management Coalition (WfMC). We have adapted this model to the specific requirements in DW systems.

We found the Workflow Reference Model too abstract for the purpose of a repository serving 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 [31] (and its latest version, the Strategic Dependency model [30]) to add a conceptual perspective to the definition of a process, capturing the reasons behind its structure. In [31], three different ways to view a process are identified: *what* steps it consists of, *how* they are to be performed and *why* these steps exist. This separation is also represented in a software process data model to support software information systems [10]. The model captures the representation of design objects ("what"), design decisions ("why") and design tools ("how").

The idea of mapping the conceptual representation of a workflow to its execution is presented in [3]. 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.

An approach to validate workflow models is presented in [24,16]. The algorithms uses a set of graph reduction rules to identify structural conflicts in a control flow specification, and to check the consistency of workflow temporal constraints.

3 Metamodel for Data Warehouse Operational Processes

In [12] a basic metamodel for data warehouse architecture and quality has been presented (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.

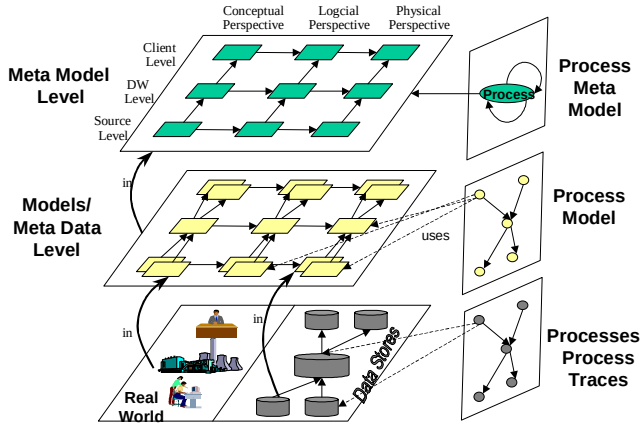

Fig. 1. Framework for Data Warehouse Architecture [12]

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.

Our process model (cf. Fig. 2.) has also different perspectives covering distinct aspects of a process: the *conceptual*, *logical* and *physical* perspective. The categorization fits naturally with the architecture model, since the perspectives of the process model operate on objects of the respective perspective of the architecture model. As mentioned in [31] 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. In the following subsections, we will elaborate on each of the three perspectives, starting from the logical one.

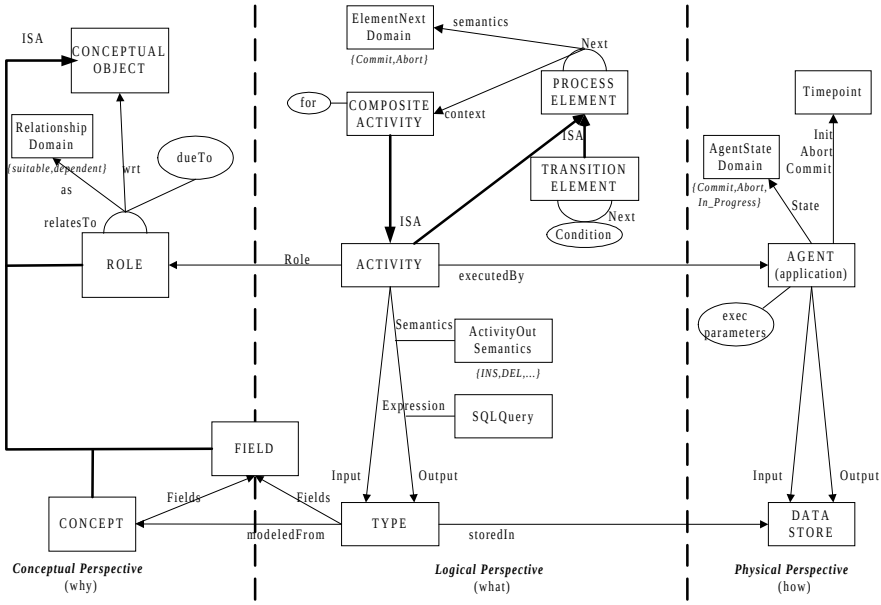

Fig. 2. The DW Operational Process Metamodel

3.1 Complexity of the Process Structure

Following [29], 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*: Since two activities can be interrelated in more than one complex DW processes, the context is captured by a *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*).

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 [29]: *Split_AND, Split_XOR, Join_AND, Join_XOR*. Their semantics are obviously the same with the ones of the WfMC proposal. The proposed model supports two other constructs of the WfMC proposal, namely the *dummy activities* (modeled as simple Transition Elements) and the *LOOP activities*, captured as instances of *CompositeActivity*, with the extra attribute *for*, expressed as a string.

3.2 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:

- *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, cf. section 3.3).

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 [26].

Each activity in a DW environment is linked to a set of incoming types as well as to a set of outcoming types. 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 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*: represents all data stores used by the activity to acquire data.
- *Output*: represents all data stores or reports, 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*}. 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.
- *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.

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.

We assume the following four *Types*: *CBL*, *Buffer*, *Class_info* and *V1*. The schemata of these types are depicted in Fig. 3. 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*). The *Aggregation* activity simply produces the sum of beds by hospital and year. The expressions and semantics for each activity are listed in Fig. 4. 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.

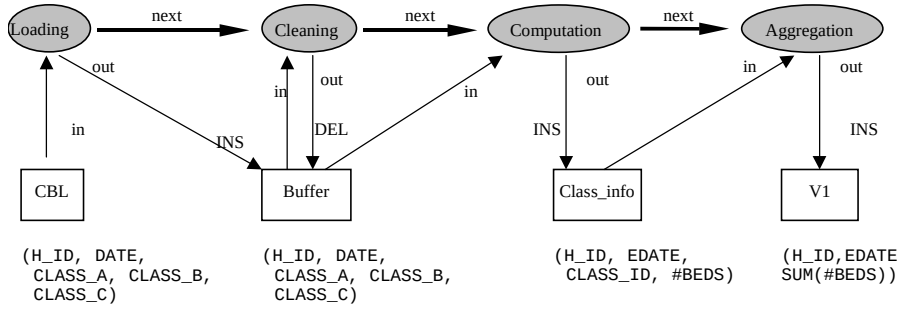

Fig. 3. Motivating Example

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. 4. Expressions and semantics for the motivating example

3.3 The Physical Perspective

Whereas the logical perspective covers the structure of a process ("what" in [31] 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 value of *AgentStateDomain* = {*In_Progress*, *Commit*, *Abort*}.
- *Init_time*, *Abort_time*, *Commit_time*: timestamps.
- *Execution_parameters*: represent any 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.

The information of the physical perspective can be used to trace and monitor the execution of the processes. The relationship between the logical and the physical perspective is done by linking each activity to a specific application program.

3.4 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*.

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 value of the *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. 5). 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'.

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

Fig. 5. Examples of role interrelationships for the motivating example

The conceptual perspective is influenced by the Actor-Dependency model [31]. 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. 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. Finally, 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 [28].

4 Issues on Process Quality

We adopt the metamodel for the relationship between architecture objects and quality factors presented in [13], which is based on the Goal-Question-Metric approach (GQM) described in [21]. Each object in a DW architecture is linked to a set of quality goals and a set of quality factors (Fig. 6). A *quality goal* is an abstract requirement, defined on DW objects, and documented by a purpose and the stakeholder interested in it. *Quality dimensions* 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.

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 [27]. 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. An extensive list of such "templates" can be found in [12].

Quality goals describe *intentions* or *plans* of the DW users with respect to the status of the DW. In contrast, our process model presented in section 3 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. 6.

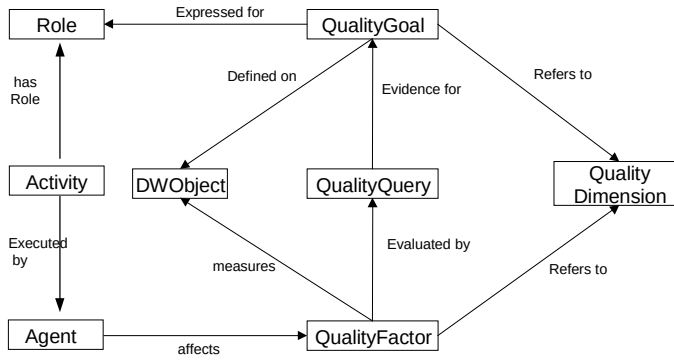

Fig. 6. Relationships between processes and quality

The lower part of Fig. 6 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.

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.

We define a set of generic quality dimensions to classify quality goals and factors of DW processes. It is influenced mainly from the quality criteria for workflows defined in [7] and the quality dimensions for software evaluation presented in [9]: (i) *Correctness*: a specification exists, describing the conditions under which the process has achieved its aim; (ii) *Functionality*: the process satisfies specific needs of data warehouse stakeholders; (iii) *Efficiency*: the process has a good balance between level of performance and amount of used resources; (iv) *Maintainability*: the degree of easiness with which the process can be modified.

5 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 consistency of the metadata repository should be checked to ensure the validity of the representation of the real world in the repository. For example, the repository can check if the type definitions of activities, agents and data stores are consistent with each other. The repository can also be used by external programs to support the execution of consistency checking algorithms like the ones proposed in [24, 16].

In addition, the quality information stored in the repository may be used to find deficiencies in data warehouse. For example, we can search the repository for all the data cleaning activities which have decreased the availability of a data store according to the stored measurements. The significance of such a query is that it can show that the implementation of the data cleaning process has become inefficient.

We do not elaborate these two issues and proceed to deal with the problem of DW evolution. We refer the interested reader to [28] for a complete discussion.

Algorithm `Extract_Type_Definitions`
Input: a list of processes **P**=[$P_1, P_2, \ldots, P_n$], a set of types **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 [17] 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 [22,14,15] wherever applicable to reduce the query definition to a compact form.

Fig. 7. Algorithm for extracting the definition of a type in the repository

5.1 Interdependencies of Types, Processes and Roles

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 [14], corrected with the results of [5, 18, 19] (to which we will collectively refer to as [14] in the sequel) as well as [15, 17, 22]. 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.

5.2 Repository Support for Data Warehouse Evolution

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*).

In Fig. 7, 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,…,*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 [8], 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 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 [20, 25] 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 [1,8]. A tool could easily visualize this evolution plan and allow the user to react to it.

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 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 [4].

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.

References

1. Z. Bellahsène. Structural View Maintenance in Data Warehousing Systems. Journées Bases de Données Avancées (BDA '98), Tunis, October 1998.
2. M. Bouzeghoub, F. Fabret, M. Matulovic. Modeling Data Warehouse Refreshment Process as a Workflow Application. Workshop on Design and Management of Data Warehouses (DMDW'99), Heidelberg, Germany, 1999.
3. F. Casati, S. Ceri, B. Pernici, G. Pozzi. Conceptual Modeling of Workflows. Conf. On Object-Oriented and Entity-Relationship Modelling (OOER'95), Australia, 1995.
4. D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, R. Rosati. A principled approach to data integration and reconciliation in data warehousing. Workshop on Design and Management of Data Warehouses (DMDW'99), Heidelberg, 1999.
5. Richard A. Ganski, Harry K. T. Wong. Optimization of Nested SQL Queries Revisited. In Proc. of ACM SIGMOD Conference, San Francisco, pp. 23-33, 1987
6. D. Georgakopoulos, M. Hornick, A. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. Distributed and parallel databases, 3(2):119-153, 1995.
7. 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.
8. A. Gupta, I. Mumick, K. Ross. Adapting Materialized Views after Redefinitions. In Proc. of ACM SIGMOD Conference, San Jose, CA, pp. 211-222, 1995.
9. ISO/IEC 9126 International Standard. Intl. Organization for Standardization. 1991
10. 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.
11. M. Jarke, R. Gallersdörfer, M.A. Jeusfeld, M. Staudt, S. Eherer. ConceptBase - a deductive objectbase for meta data management. J. Intelligent Information Systems, 4(2):167-192, 1995.
12. 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 CAiSE '98, Pisa, Italy, 1998)
13. M.A. Jeusfeld, C. Quix, M. Jarke. Design and Analysis of Quality Information for Data Warehouses. 17th Conf. Entity Relationship Approach (ER'98), Singapore, pp. 349-262, 1998.
14. W. Kim. On Optimizing an SQL-like Nested Query. ACM TODS 7(3):443-469, 1982.
15. A. Levy, I. Mumick, Y. Sagiv. Query Optimization by Predicate Move-Around. In Proc. of 20th VLDB Conference, Chile, pp. 96-107, 1994.
16. O. Marjanovic, M. Orlowska. On Modeling and Verification of Temporal Constraints in Production Workflows. Knowledge and Information Systems, 1(2):157-192, 1999.
17. I. Mumick, S. Finkelstein, H. Pirahesh, R. Ramakrishnan. Magic is Relevant. In Proc. of ACM SIGMOD Conference, Atlantic City, NJ, pp. 247-258, 1990.
18. M. Muralikrishna. Optimization and Dataflow Algorithms for Nested Tree Queries. In Proc. of 15th VLDB Conference, Amsterdam, The Netherlands, pp. 77-85, 1989.
19. M. Muralikrishna. Improved Unnesting Algorithms for Join Aggregate SQL Queries. In Proc. of 18th VLDB Conference, Vancouver, Canada, pp. 91-102, 1992.
20. W. Nutt, Y. Sagiv, S. Shurin. Deciding Equivalences among Aggregate Queries. In Proc. of 17th Symposium on the Principles of Databases (PODS'98), Seattle, pp. 214-223, 1998.
21. M. Oivo, V. Basili. Representing software engineering models: the TAME goal-oriented approach. IEEE Trans .Software Engineering, 18(10):886-898, 1992.

22. H. Pirahesh, J. Hellerstein, W. Hasan. Extensible/Rule Based Query Rewrite Optimization in Starburst. In Proc. of ACM SIGMOD Conference, San Diego, CA, pp. 39-48, 1992.
23. B. Ramesh, V. Dhar. Supporting systems development by capturing deliberations during requirements engineering. IEEE Trans. Software Eng., 18(6):498-510, 1992.
24. W. Sadiq, M. Orlowska. Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models. In Proc. of 11th Intl. Conf. Advanced Information Systems Engineering, (CAiSE'99), Heidelberg, Germany, pp. 195-209, 1999.
25. D. Srivastava, S. Dar, H.V. Jagadish, A.Y. Levy. Answering Queries with Aggregation Using Views. In Proc. of 22nd VLDB Conference, Bombay, India, pp. 318-329, 1996.
26. P. Vassiliadis. Modeling Multidimensional Databases, Cubes and Cube Operations. In Proc. of 10th SSDBM Conference, Capri, Italy, pp. 53-62, 1998.
27. P. Vassiliadis, M. Bouzeghoub, C. Quix. Towards Quality-Oriented Data Warehouse Usage and Evolution. In Proc. of 11th Conference of Advanced Information Systems Engineering (CAiSE '99), Heidelberg, Germany, pp. 164-179, 1999.
28. P. Vassiliadis, C. Quix, Y. Vassiliou, M. Jarke. A Model for Data Warehouse Operational Processes (long version). Available at http://www.dblab.ece.ntua.gr/~pvassil/publications/process-model00.gz
29. Workflow Management Coalition. Interface 1: Process Definition Interchange Process Model. Doc. No. WfMC TC-1016-P, 1998. (http://www.wfmc.org)
30. E. Yu. Strategic Modelling for Enterprise Integration. In Proc. 14th World Congress of Int. Federation of Automatic Control, China 1999.
31. E. Yu, J. Mylopoulos. Understanding 'Why' in Software Process Modelling, Analysis and Design. In Proc. of 16th Intl. Conference Software Engineering, Sorrento, Italy, pp. 159-168, 1994.