Model-Driven Approach to Handle Evolutions of OLAP Requirements and Data Source Model

Said Taktak[1(✉)], Jamel Feki[2], Abdulrahman Altalhi[3], and Gilles Zurfluh[4]

[1] FSEGS Faculty, Miracl Laboratory, University of Sfax, Sfax, Tunisia
said.taktak@fsegs.rnu.tn
[2] Faculty of Computing and IT, University of Jeddah, Jeddah, Saudi Arabia
jfeki@uj.edu.sa
[3] Faculty of Computing and IT, King Abdelaziz University, Jeddah, Saudi Arabia
ahaltalhi@kau.edu.sa
[4] IRIT Laboratory, University of Toulouse 1 Capitole, Toulouse, France
gilles.zurfluh@ut-capitole.fr

Abstract. Data Warehouse (DW) evolution is becoming a critical research topic for several organizations mainly because their analytical data change permanently and rapidly due to changes in the data source and decision-makers' requirements. This paper presents an MDA-compliant (Model Driven Architecture) approach and a software tool for propagating automatically the evolutions of the data source model and OLAP (On-Line Analytical Processing) requirements towards the multidimensional DW model. More accurately, we propose a DWE (Data Warehouse Evolution) framework. Being MDA compliant, we perform this DW evolution through Model-To-Model transformation rules we have defined as QVT (Query/View/Transformation) along with M2T (Model-To-Text) transformations realized using *Acceleo* templates. Thus, the evolution operations (Create table, Add column…) are firstly modeled, secondly transformed into multidimensional evolution operations (Create dimension, hierarchy…), and then are used with *Acceleo* templates for generating the DW alteration code.

Keywords: Data warehouse · Evolution modeling · Data source model
OLAP requirements · MDA · M2M · M2T

1 Introduction

Nowadays the DW is a powerful technology for strengthening the decision-making process within organizations. It gathers synthesis information from internal and/or external operational data sources.

DW modeling has been considered, for more than one decade, as a real challenging research topic for which several approaches were proposed. Three major categories of

L. F. Pires et al. (Eds.): MODELSWARD 2017, CCIS 880, pp. 401–425, 2018.
https://doi.org/10.1007/978-3-319-94764-8_17

approaches for designing a DW schema (i.e., data-model) are well known in the literature: Top-down [1], bottom-up [2, 3], and mixed [4] approaches.

All these DW design approaches rely on a rigid assumption; they consider that the conceptual model of the DW is *time-invariant*. However, in practice, this assumption is unrealistic most of the time, and therefore restricts the evolution of the real world. In fact, the DW model may evolve due to internal and/or external factors (e.g., business processes changes, organization environment evolution). Furthermore, it is difficult to fix definitively the DW model at the design phase; in fact, for sustainability issues, the DW model should often undergo changes after its implementation. These changes are due to two main reasons namely (a) Evolution of the analytical needs: changes in these needs might require extending the DW model (e.g. adding new axes or subjects of analysis), and (b) Evolution of the data source model (DS) due to the evolution of the business processes (e.g., adding/removing conceptual entities). To the best of our knowledge, we can claim that the problem of changes in the DW model needs more research investigations and appropriate software features. Indeed, all evolution strategies proposed in the DW literature are at a single level of modeling: schemas before and after changes conform to the same meta-model. In the DW domain, the evolution of schemas expressed in different models has not yet received its full share of the investigation.

To alleviate this problem, we propose in this paper an MDA (Model Driven Architecture) approach [14] that automates the propagation of the evolution of the DS model and the evolution of decision-makers' requirements (OLAP-requirements in the remaining of the paper) towards its associated DW model. In this context, we suggest an approach based on a classification of the evolution scenarios and a set of transformation rules to identifying the evolution operations to apply to the DW model.

This paper is organized as follows. Section 2 provides a review of works dealing with the DW evolution problem. Section 3 describes our MDA-based approach for the propagation of OLAP-requirements and DS evolution towards the multidimensional DW model. Section 4 discusses the effect of the evolution of the DS model on the DW. Section 5 introduces our classification of evolutions of OLAP-requirements; it develops algorithms to derive the appropriate changes that should apply on the DW model. Section 6 describes the implementation of the DW evolution through MDA transformations at two levels: Model-To-Model (M2M) and Model-To-Text (M2T). Finally, Sect. 7 concludes the paper and enumerates its perspectives.

2 Related Works

The DW evolution problem is considered from two main viewpoints: (a) *Evolution of data source model*, and (b) *Evolution of business requirements of decision-makers*. Hereafter, we review the approaches for each trends.

2.1 Approaches Based on DS Evolution Model

Organizations' business processes evolve over time due to the modification of existing processes or the emergence of new ones that may create new real world objects.

Naturally, these evolutions affect the data-model of the DS (i.e., information system) that feeds the DW with data. In turn, the DW cannot be immunized against the evolutions of its DS; consequently this evolution deserves to be studied in order to semi-(automatically) propagate towards the DW data-model and the ETL (Extract Transformed and Load) process. This evolution problem has been addressed from different perspectives; we classify the related works into three main categories: (i) *Evolution of the DW multidimensional model*, (ii) *Maintenance of materialized views*, and (iii) *Adaptation of the ETL process*.

Works addressing views maintenance consider the DW as a set of materialized views directly built on, and loaded from, the DS. In this category of approaches, any change in the DS data-model requires views maintenance efforts. As a practical extension, [5, 6] proposed approaches for a dynamic adaptation of materialized views in response to the evolution of the DS/DW. These approaches maintain not only the schema views but also their content; they mainly attempt to avoid recalculating views after DS changes by deriving a new schema view from the old one. More details on views maintenance in multidimensional context are available in [7].

Other research works adapt the ETL process when the DS data-model evolve. Among these works, the authors in [8, 9] provide a mechanism for adapting the ETL tasks to the changes occurred in the DS data-model. However, this study was restricted to the ETL process without tackling the impact of the DS evolution on the DW model components (Dimensions, facts, hierarchies…).

To lighten these shortcomings, the authors in [10] have defined a formal model for a multi-version DW. They presented a set of evolution operations that affect the DW schema and content. These authors have distinguished two types of DW versions: *real version* and *alternative version*. The DW real version reflects the changes in the real world environment of the organization whereas the DW alternative version simulates the change process; to do so, "What-If" analysis strategy was adopted. Furthermore, the authors have developed the MVDW (Multi-Version Data Warehouse) prototype for the DW maintenance and versions management. A major drawback of this contribution is the manual identification of the DW evolution operations; this identification requires high expertise of the DW administrator and, therefore, is out of reach of end-users.

2.2 Approaches Based on Business Requirement Evolution

Let us note that in mixed DW design approaches [4], the design of the DW relies, from the one hand, on the DS model and, from the other hand, on the OLAP-requirements. Obviously, OLAP-requirements could not be static in time; therefore, the DW-design driven by user requirements may become obsolete and no longer comply the new requirements. To overcome this issue, it is necessary to consider the new analytical requirements and then adapt the DW to encompass them. Among the research works of this category, the authors in [11] suggested an approach for the customization of analyses based on "If-Then" rules model; this model allows users to integrate their own knowledge in order to enlarge the panoply of analysis on the DW by changing the DW schema. The suggested evolution operations affect only two components of the DW: dimensions and hierarchies. The authors have developed a prototype called *WEDriK* (Warehouse Evolution Driven by Knowledge) based on a set of DW evolution

algorithms to create new analytical axes. The analytical requirements introduced by each user are processed and transformed into DW evolution operations. Nevertheless, the authors assume that the DW users are skilled enough to express properly their requirements. Moreover, the supported changes are simple: they do not cover all cases that decision-makers may ask for.

To overcome this problem, in [12] the authors studied the evolution of complex hierarchies (multiple alternative hierarchies, dependent and independent parallel hierarchies). They defined constraint-based evolution operations to ensure data integrity and schema consistency of the new DW model. Operations and constraints are defined in ULD (Uni-Level Description language) and MDD (Multilevel Dictionary Definition). This study is an extension of the work in [12] where the authors presented a conceptual requirement-oriented framework called *DWEVOLVE* for DW evolution. It analyzes the changes in the requirements specified by stakeholders as well as developers, and then incorporates them into the DW by performing appropriate additions, deletions and updates. Nevertheless, the authors do not suggest mechanism for automatic inference of evolution operations from OLAP-requirements. In fact, this task is borne entirely by the DW administrator.

In the same context, in [13] authors have also investigated the problem of business requirements evolution. They defined a formalism for modeling the new analytical needs and proposed a semi-automatic approach to adjust and create a new version for the DW model. However, the evolution operations supported by this solution are simple and lack accuracy. For instance, when adding an attribute, the proposed algorithm is able to identify just the dimension to change but cannot find the role of the new attribute in the dimension, i.e., whether it creates a hierarchy or inserts a level into an existing hierarchy… How to find the role of the new component is really a hard task left to a skilled user.

2.3 Discussion

In the related works section we have focused on two complementary categories of evolutions in DW systems, namely evolution of the DS model and evolution of OLAP needs. We have identified three deficiencies concerning (i) complementarity, (ii) complexity of the evolutions, and (iii) automatic propagation of changes from the DS toward the DW.

First, concerning the complementarity, to the best of our knowledge, no solution has combined the DS evolution with business requirements evolution so far. Indeed, contributions have addressed these two evolutions separately.

Secondly, few works were interested in studying the DS evolution effect on the multidimensional model. Moreover, most of these works provide solutions touching a few isolated aspects and treating simple evolution cases (i.e., Dimension evolution, Fact evolution, ETL evolution).

Thirdly, automatic propagation was not a main concern in these works, and where addressed, it was carried out according to traditional modeling and programming approaches.

Finally, from the technological viewpoint, we note that all proposed solutions were realized in a conventional software engineering context; therefore, implementations are

platform-dependent. Obviously, using the MDA approach allows benefiting from its multiple advantages.

The objective of this paper is to propose a Data Warehouse Evolution framework (*DWE*) as a complete solution covering the DS evolution and the OLAP-requirements evolution. Our proposal is MDA compliant, it promotes the automatic propagation of changes occurred in the DS model along with business requirements changes towards the multidimensional DW. Relying DWE on the MDA technology is really a challenging proof. In fact, MDA facilitates realizing our proposed approach, which inherits benefits from this technology (i.e. platform-independent, reduced efforts, and improved quality of results). We define one for the OLAP needs and one model for the DS evolution. In the remaining of this paper, we present our approach that addresses the DW model evolution problem.

3 Overview of the Proposed Approach

Our MDA-based approach aims to automate the propagation of the changes raised by decision-makers (as new needs) and DS model (as new evolution operations) towards the DW multidimensional model. Figure 1 depicts our approach where the evolution of the DW model is due either to an evolution of its DS model (Fig. 1, panel A), or to an evolution of OLAP needs (panel B). To do so, we define an appropriate evolution model for the new OLAP needs; this enables us reusing our on-hand DW evolution model [15]: we keep the same M2T transformation rules for code generation.

Our approach relies on three evolution models: (i) *DS Evolution Model* (DSEM), (ii) *DW Evolution Model* (DWEV), and (iii) *Requirements Evolution Model* (REM). In addition, it applies M2M and M2T transformations:

- DSEM: This model describes all evolution operations that may affect the relational DS elements (table, column…).
- DWEM: It describes all operations that may affect the multidimensional structures (dimensions, facts…). These operations should be derived from the DSEM model.

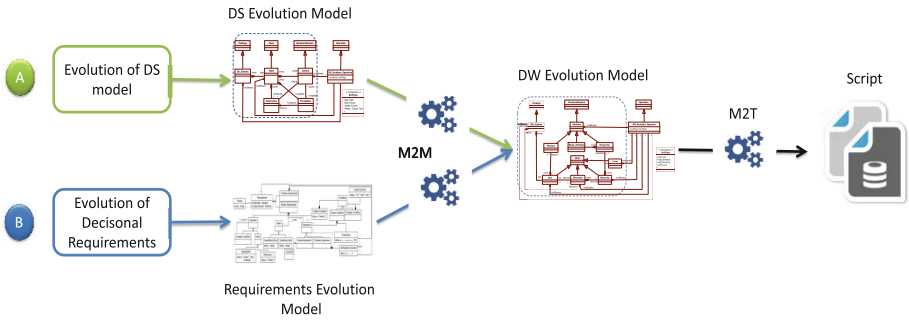

Fig. 1. Overview of our MDA-based DW evolution approach.

- REM: This model describes the new needs of decision-makers in terms of subject and axes of analysis. It also allows defining knowledge introduced by the user (e.g. rules, formulas). This model will be transformed into a DWEM model.
- M2M transformation: It generates the DWEM model from REM model. It relies on automatic mapping between these two models. M2M transformation rules are implemented in QVT (Query-View-Transformation), and use a set of meta-models stored upstream as *Ecore* files.
- M2T transformation: It generates the code that performs the DW model alteration; the generated code results from the DWEM previously generated by applying a set of transformation rules we have formalized in *MOF2Text*. M2T process takes as input the physical model (PSM) along with the DW evolution models; it produces SQL script file(s) for creating or modifying the DW model. We have defined *Acceleo* templates for transforming DWEM operations into an executable script. This transformation process is valid as well for processing the DS evolution as for processing the needs evolution. In fact, this reuse is feasible because these two transformations start from the same DWEM evolution model.

4 Evolution Inferred by the DS Model

The DW schema may evolve over time due to the evolution of its DS data-model. Naturally, the evolution frequency is domain-dependent. As an illustration, in the banking domain the DS of a DW changes every 2–4 weeks on average, also the DS of a telecommunication company is less stable since its schema changes every 7–13 days on average [16].

Two crucial questions arise when the DS evolve: (1) What are the changes to apply to the DW model (i.e., adding a dimension, fact, level of analysis…), and (2) How to perform these changes efficiently and quickly; rapidity is an imperative factor for some decisional systems as argued before. A trivial solution rebuilds the DW from the new DS data-model starting from scratch; but this is a poor approach because the DW reconstruction is a heavy and complex task requiring a lot of time and effort, and is therefore costly. Furthermore, rebuild from scratch cannot be envisaged especially in frequently changing domains. In order to address this evolution issue, we have proposed a model-driven approach for propagating changes from the relational DS towards its DW in an almost-automatic way, thus avoiding the need for the full reconstruction of the DW model (and later its full-reloading process). To do so, we have proposed an MDA-based architecture [14] for propagating the evolution operations occurred on the DS model towards the DW data-model. We have identified sets of evolution operations on the DS and their transformation rules. These operations concern tables, columns, keys…; their execution is not systematic (a precondition should be satisfied). Table 1 lists the evolution operations that could affect the relational DS, and gives for each one the corresponding set of plausible evolution operations we may apply on the DW. For example, in line 1 when we "*Add new table*" to the DW the effect of this evolution operation may create a "*New Fact*", "*New Dimension*", "*New Hierarchy*" and/or "*New Level*" within an existing hierarchy.

Table 1. DS evolution operations and their corresponding evolution operations on the DW.

DS evolution operation	Plausible DW evolution operation	
Add new table	New fact New dimension	New hierarchy New level
Add new column	New measure New hierarchy New level	New parameter New weak attribute*
Modify column type	Modify weak attribute type* Modify parameter type	Modify parameter type Modify measure type
Drop table	Delete fact Delete dimension	Delete level Delete parameter
Remove column	Delete measure Delete level	Delete weak attribute* Delete hierarchy
Add new constraint	New fact New dimension	New hierarchy New level
Drop constraint	Delete fact Delete dimension	Delete level Delete parameter
Split table	New fact New dimension Delete measure Delete level	New hierarchy New level Delete weak attribute* Delete hierarchy

*Less significant operation

In order to define the transformation rules we adopt the following notation:

- DS: a third normal form relational DS schema
- t: a relational table belonging to DS
- t.pk: the set of primary key columns of table t
- t.Cols: the set of non-primary key columns of t ($t.pk \cap t.Cols = \emptyset$)
- $t_i \rightarrow t_j$: table t_i references table t_j via a foreign key belonging to t_i
- DW: a multidimensional data warehouse schema loadable from tables in DS
- f: a fact table belonging to DW
- d: a dimension belonging to DW
- $d_i.h_j$: a hierarchy h_j of dimension d_i
- $d_i.h_j.l_k$: a level l_k belonging to $d_i.h_j$
- $d_i.d_j.l_k.p$: a parameter at level $d_i.h_j.l_k$
- $d_i.h_j.l_k.p.W$: a possibly empty set of weak attributes associated with parameter $d_i.h_j.l_k.p$
- f.M: the set of measures of fact f
- f.D: the set of dimensions of fact f
- Load (t, d): A Boolean function returning True if table t loads dimension d.

In this section, we limit ourselves to detail two transformation rules:

- Transforming a table into a dimension,
- Transforming a table into a fact,

Other transformation rules are available in [15].
We illustrate these transformation rules using the DS and DW of Fig. 2.

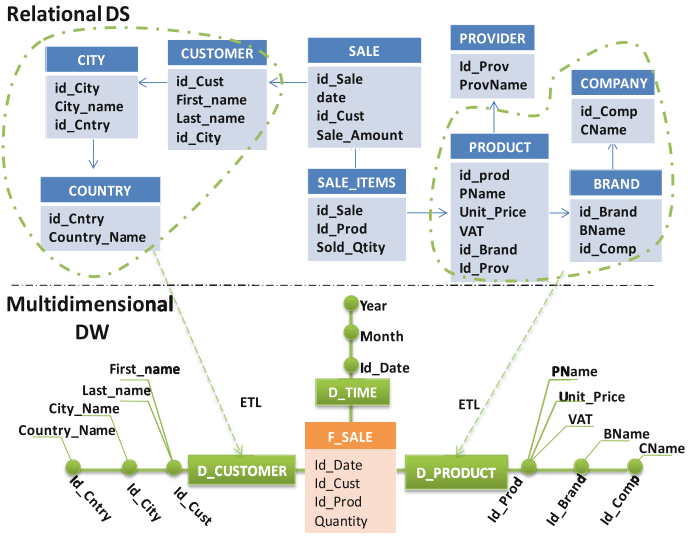

Fig. 2. A relational Data source model and its multidimensional DW model [17].

4.1 Transforming a Table into a Dimension

The creation of a table t_{new} in the DS may create a new dimension d_{new} in the DW by calling the *Add_dimension(d_{new})* operation. This performs through rule *T2D*.

Rule T2D: Table-To-Dimension
Input:
- t_{new}: table added to the DS via the *Add_Table (t: Table)* operation
- DS, DW

Condition:
- DS.ti → DS.t_{new} -- t_{new} is referenced by a table t_i in the DS
- Load (DS.t_i, DW.f_j) -- table t_i (which references t_{new}) loads a fact f_j

Processing: /*Create a new dimension d_{new}*/
- Find F' -- the set of all facts loaded from tables that reference t_{new}
- d_{new}.F := F' -- link d_{new} with all facts in F'
- d_{new}.name:= "D_" + t_{new}.name -- + denotes the string concatenation operator.
- d_{new}.H := {h_{new}} -- create a hierarchy h_{new} for dimension d_{new}
- d_{new}.h_{new}.L:= {l_{new}} -- create a level l_{new} within h_{new}
- d_{new}.h_{new}.l_{new}.p := t_{new}.pk -- parameter of l_{new} is t_{new}.pk
- d_{new}.h_{new}.l_{new}.W := t_{new}.Cols -- weak attributes of l_{new} are columns of t_{new}

Output:
- d_{new}: new dimension added to the DW using *Add_dimension (d: Dimension)*.

In our running example (Fig. 2), let us create the table *RETAIL_OUTLET (Id_Ro, Ro_name, Ro_zone…)* that is referenced by the DS table *SALE* that feeds the fact *F_SALE*. Applying rule *T2D*, the new table creates a new dimension called *D_RETAIL_OUTLET* linked to the *F_SALE fact*.

4.2 Transforming a Table into a Fact

The creation of table t_{new} in the DS using *Add_table*(t_{new}) may create a new fact f_{new} in the DW by calling the *Add_fact*(f_{new}) operation. This evolution is realized by applying rule *T2F* hereafter.

Rule T2F: Table-To-Fact
Input:
- t_{new}: table added to the DS via *Add_Table (t: Table)*
- DS, DW

Condition:
- $t_{new} \rightarrow DS.t_1 \ldots, DS.t_k$ with $k \geq 2$ -- t_{new} references *k* tables in the DS
- Numeric (t_{new}) $\neq \emptyset$ -- t_{new} has numeric attributes
- Load ($DS.t_i$, $DW.d_j$) with $1 \leq i \leq k$ -- each table t_i referenced by t_{new} loads a dim d_j

Processing: /*Create a new fact f_{new}*/
- Find D', the set of all dimensions loaded from tables referenced by t_{new}
- f_{new}.D := D' -- link f_{new} with dimensions in D'
- f_{new}.M := Numeric (T_{new}.Cols) -- numeric columns of t_{new} become Measures in f_{new}
- f_{new}.name := "F_" + t_{new}.name -- name of the new fact.

Output:
- f_{new}: new fact added to DS via the *Add_Fact (f: Fact)* operation.

Continuing with our example, we create the table *SCORE_PROD (#Id_Prod, #Id_Cust, ScoreNumeric…)* that references two tables *PRODUCT* and *CUSTOMER* in the DS. These tables feed respectively the two dimensions *D_PRODUCT* and *D_CUSTOMER*. Applying rule *T2F*, the new table creates a fact called *F_SCORE_PROD* with *Score* as a measure, related to *D_CUSTOMER* and *D_PRODUCT* dimensions.

5 Evolution Implied by the Decision Makers Needs

The evolution of the OLAP-requirements leads to several cases of evolution on the DW model. We group these evolution cases into three classes namely: *Evolution by derivation*, *Evolution by reorganization*, and *Evolution by extension*. More details about this classification are available in [17]. We clarify these classes and we textually explain the transformation rules that generate the modifications operations to apply on the DW model when the OLAP-requirements evolve.

– *Statico*: Nothing to change if the current DW model meets a new requirement.
– *Reorganization*: Applies when the necessary elements (i.e., measure or attribute) for the new requirement already exist in the DW model but their current roles are not

adequate. We change the role of such elements by creating new links between some elements of the DW model. This reorganization mainly affects temporal and spatial dimensions.

- *Derivation*: If an element is vital for a new requirement but is not in the DW model, therefore, we check if it is derivable from an existing DW element; the derivation uses knowledge introduced by the DW administrator as rules or formulae. Otherwise, if the vital element is derivable from the DS, then we extend the DW model with the derived element.
- *Extension*: This alternative is the most delicate. In fact, when the DW model cannot satisfy the new requirement, either by derivation or by re-formulation, we have to identify which element from the DS we should add to the DW and define its role, and then we expand the DW model with the new element.

In order to decide which alternative of evolution - from above -to apply to the DW model, we develop the Main algorithm (Algorithm 1).

Note that we use these alternatives independently or combined. In the following, we detail each one and specify the evolution operations to perform it. To do so, we use the notation below:

- *Req*: a new requirement
- *A*: set of attributes describing *Req*; *A* divides into two subsets $A = A_{quant} \cup A_{qual}$
- A_{qual}: all qualitative attributes of *Req*
- A_{quant}: all quantitative attributes of *Req*,
- *DW*: set of elements of the DW multidimensional model (i.e., schema)
- *DS*: set of elements of the DS model.

The *Main* algorithm depicts the principle of defining the evolution strategy. It calls three algorithms *Reorganize* (Algorithm 2), *Derive* (Algorithm 3) and *Extend* (Algorithm 4).

Algorithm 1: Main.

```
        Input:
          Req, DW, DS
        Begin:
1.      if DW_answer(Req) then
2.          Null  // No changes to do on the DW model
3.      else   if A ⊆ DW  then
4.                 Reorganize( )        // see  Reorganize algorithm
5.             else
6.                for each  a  ∈ A
7.                 if a ∉ DW  and (Rule(a)  or  Formula(a)) then
8.                    Derive( )          // see Derive algorithm
9.                else if a ∉ DW  and a ∈ DS    then
10.                      Extend( )       // see Extend algorithm
11.               end if
12.               end for
13.            end if
14.     end if
        End.
```

DW_answer(Req) is a Boolean function that returns True if the DW model meets the new requirement (*Req*): *Statico* alternative, and False otherwise.

Rule(a) is a Boolean function True if attribute *a* is defined through a rule, and False otherwise.

Formula(a) is a Boolean function True if attribute *a* is defined through a formula, and False otherwise.

5.1 Reorganization

The reorganization process (see Algorithm 2) begins with the identification of the DW elements (fact, dimensions) for the new requirement. It calls two functions *Find_Fact* and *Find_Dimension*; these functions return respectively the fact containing quantitative attributes A_{quant}, and dimensions containing qualitative attributes A_{qual}. The fact *fnew* will be enriched with the set of measures A_{quant} attributes. Dimensions containing A_{qual} attributes are refined using the *Refine* function before their link to the new fact. This function prunes hierarchies by eliminating unnecessary attributes for the new requirement.

Algorithm 2: Reorganize.

```
        Input:
        A_quant , A_qual
        Begin:
1.        f = Find_Fact(A_quant)
2.        D = Find_Dimensions(A_qual)
3.        if f == ∅ Then
4.          f_new.M = A_quant
5.        else
6.          f_new = f
7.        end if
8.        for each d ∈ D
9.          d_new = Refine(d)
10.         d_new.f = f_new
11.         Add_Dimension(d_new)
12.       end for
13.       Add_Fact(f_new)
        End.
```

5.2 Derivation

The *Derive* algorithm describes the derivation process; it takes as input the attribute to derive as well as the knowledge given by the DW administrator as rules or formulae. We treat differently qualitative and quantitative attributes of this class.

If the derived attribute a_d is quantitative, and if there is, a fact *f* related to the dimension that contains the qualitative attributes of the new requirement, then we add a_d to *f* as new measure m_{new}. Otherwise, we create a new fact f_{new} for the derived attribute a_d.

If a_d is a qualitative attribute, it necessarily belongs to a dimension where its position generally depends on the a_{source} attribute in the rules. If a_{source} belongs to a terminal level l_t then a_{new} becomes a terminal level l_{new} in the same hierarchy as l_t. Otherwise, we create a new hierarchy h_{new} that contains level l_s and all its predecessor levels. l_{new} adds to the new hierarchy as a terminal level.

Algorithm 3: Derive.

Input:
a_d: a derived attribute
a_{source}: an attribute of DS model used within a rule or formula
A_{quant} , $A_{qual.}$
Begin:
1. if $a_d \in A_{quant}$ and formula(a_d) then
2. f = Find_Fact(A_{qual}) //find the fact linked to dimensions containing A_{qual}
3. if f ==∅ then
4. $f_{new}.M = f_{new}.M \cup a_d$ // define the measure of the new fact
5. $f_{new}.D$ = Find_Dimensions(A_{qual}) //find dimensions containing A_{qual}
6. ***Add_Fact (f_{new})***
7. else
8. $m_{new} = a_d$; m_{new}.fact = f
9. ***Add_Measure (m_{new})***
10. end if
11. else if $a_d \in A_{qual}$ and Rule(a_d) then
12. return level containing a_{source}
13. if Terminal_Level(l_s) then
14. $l_{new}.h = l_s.h$ //hierarchy of level l_{new} is l_s hierarchy
15. $l_{new}.p = a_d$ // parameter of l_{new} is the derived attribute a_d
16. $l_{new}.pred = l_s$ // the predecessor level of l_{new} is l_s
17. else
18. $h_{new}.d = l_s.h.d$ //dimension of h_{new} is the dimension of l_s
19. //the levels of h_{new} are all l_s predecessor levels
20. ***Add_ hierarchy (h_{new})***
21. $l_{new}.p = a_d$; $l_{new}.pred = l_s$; $l_{new}.h = h_{new}$
22. end if
23. ***Add_ Level(l_{new})***
24. end if
25. End.

5.3 Extension

The *Extend* algorithm states the principle of the extension, which enriches the DW model with elements extracted from the DS to satisfy the new OLAP-requirement. We assume that a semi-automatic association between attributes of the new requirement and the DS attributes is provided; this treatment could use a semantic resource or a dictionary of the DS attributes. The role of each element depends on the type (quantitative or qualitative) of its associated attribute and its membership table in the DS.

Algorithm 4: Extend.

```
        Input:
        DW, DS,
        a_e : attribute to retrieve from the data source
        Begin:
1.      t = Find_Table(a_e)  // returns the table that contains a_e
2.      if a_e ∈ A_qual  then
3.    returns the level which is loaded from t
4.      if l == null then
5.        t' = ref(t)        // returns the table which references table t
6.        t'' = IsRef (t)   // returns the table which is referenced by t
7.        if  t' not null  and t'' not null  then
8.         l'= Load_Level(t')
9.         l'' = Load_Level(t'')
10.          if l''.pred == l'   then
11.           l_new.h = l'.h ;   l_new.pred = l' ;   l_new.succ = l''
12.           Add_Level(l_new)                  // add level l_new
13.         end if
14.        else if  t' is not null   then
15.        l' = Load_Level (t')
16.        if Terminal_Level (l') then    // add terminal level
17.          l_new.p = a_e ;  l_new.pred = l' ;   l_new.h = l'.h
18.        else                  // add hierarchy and a new level
19.          h_new.d = l_s.h.d
20.          h_new.L = l_1.. l'   // the level in the new hierarchy
21.          Add_Hierarchy (h_new)
22.          l_new.p = a_e  ; l_new.pred = l' ;  l_new.h = h_new
23.          Add_Level (l_new)
24.         end if
25.        end if
26.       else
27.        a_new.p = l.p
28.        Add_Attribute(a_new)
29.       end if
30.     else
31.      f = Load_Fact(t)
32.      if f not null   then
33.       m_new = a_e  ;   m_new.fact = f
34.      Add_Measure (m_new)      // add measure m_new to the fact I
35.     else
36.      f_new. M = a_e ;   Add_Fact(f_new)   // add fact I
37.     end if
38.   end if
        End.
```

If the attribute to extract a_e (a_e belongs to a table t) is qualitative, four situations arise to define the role of a_e in the multidimensional model:

– If table t feeds a level l then it becomes a low attribute by applying the *Add_Attribute* evolution operation.

- If t feeds no levels, and if t is referenced by a table t' which feeds a terminal level l', then a_e becomes an attribute for a new terminal level l_{new} by applying the *Add_Level* evolution operation.
- If t does not feed any level, and if t is a table referenced by t' and refers to a table t'' - t' and t'' respectively feed the two successive levels l' and l''- a_e can then feed a hierarchical level inserted between the two levels l' and l''.
- If t does not feed any level and if t is referenced by table t' which feeds a non-terminal level l' then a_e creates a new hierarchy h_{new} by calling the *Add_Hierarchy* evolution operation. h_{new} contains the level l' and all its predecessor levels in the hierarchy of l'. Then, we create a new terminal level l_{new} for the new hierarchy h_{new}.

When the extracted attribute a_e is quantitative, if t (table of a_e) feeds a fact f, then a_e becomes a measure of f. Otherwise, we create a new fact with the new measure a_e.

6 Implementation

To validate our approach, we have developed a DWE (Data Warehouse Evolution) software prototype under the EMF (Eclipse Modeling Framework) platform that is a complete environment for MDA. Figure 3 shows the DWE overall architecture that offers two evolution features: (i) Evolution of the DW model as a result of changes in its DS model; (ii) Evolution of the DW model to meet new OLAP-requirements.

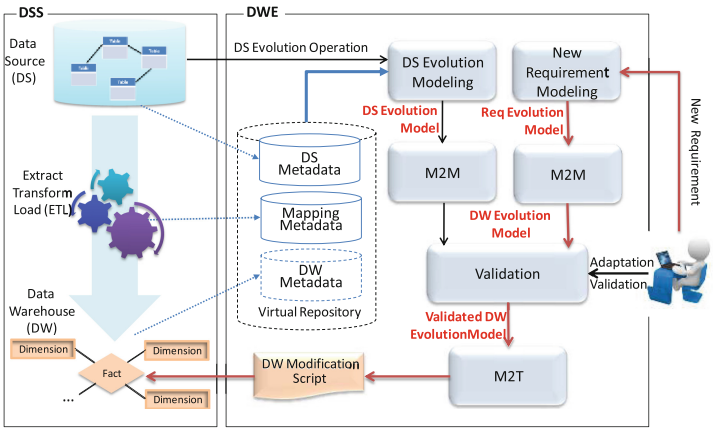

Fig. 3. Architecture of the DWE prototype [17].

The DW evolution process starts with the modelling of the new requirements or changes occurred in the DS model; it aims to generate respectively the requirement evolution model (REM) and the DS evolution model (DSEM). The next step is the M2M that transforms the REM and the DSEM into DWEM. Once the DWEM is generated, thereafter the new DW model displays graphically; this enables the DW

administrator to observe and study the effects (i.e., computed changes) of the DW-evolutions operations. At this stage, the DW administrator can accept the suggested changes or even adapt them. Finally, the M2T process transforms the DWEM into script for DW alteration. In what follows, we detail these steps.

6.1 Modeling of Evolution

We use UML (Unified Modeling Language) class diagrams to define the evolution models DSEM, REM and DWEM. The static property list in the classes define the models of the DS, Requirements and the DW whereas the operations define the changes that may affect each of these structures. Next, we detail these three evolution models.

DS Evolution Model (DSEM)
The DSEM model is the basic model for deducing the DWEM model. It defines the relational DS schema (tables, constraints…) through class properties as well as the evolution operations (add table, add column…).

The DSEM conforms to its Meta-Model in Fig. 4. The latter has two components: (i) The DS Meta-Model (enclosed within the dashed area) stores the DS schema; and (ii) The Meta-Model of the DS Schema Evolution Operations that stores the DS schema evolution operations.

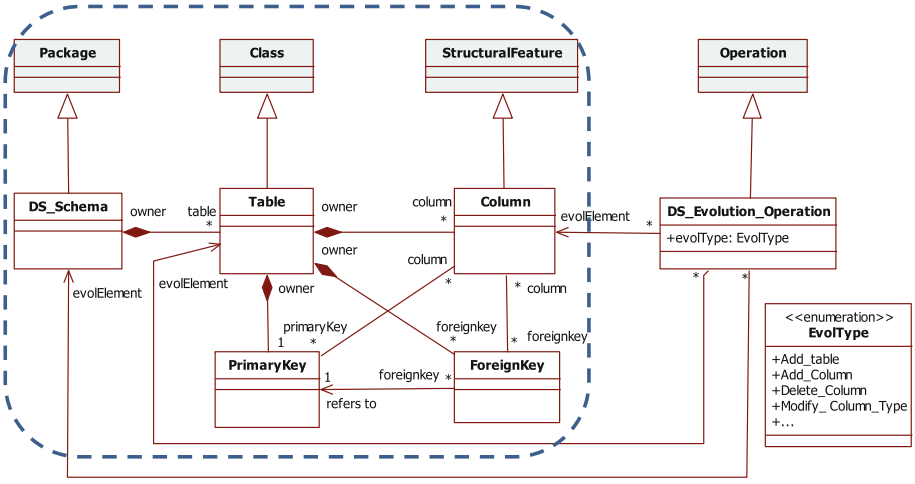

Fig. 4. DS evolution meta-model.

Modeling Decision-makers Requirements
This step takes as input the new requirements expressed as queries, rules or formulae and returns a Requirements Evolution Model (REM) compliant to the Meta-Model in [13] depicted in Fig. 5. A new requirement has *quantitative* and *qualitative* attributes, arithmetic operations (i.e., formula) and logical expressions (i.e., rules).

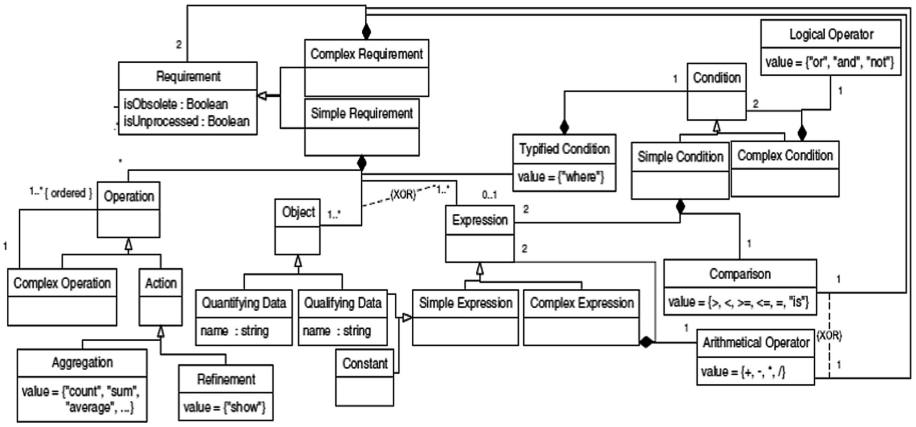

Fig. 5. Requirements evolution meta-model [13].

DW Evolution Model

The DW Evolution Meta-Model has two components (cf. Fig. 6): (i) The DW Meta-Model (dashed area) that stores the DW Schema, and (ii) The Meta-Model of the DW Schema Evolution Operations that stores the DW Schema Evolution Operations. This latter will be deduced automatically from the DSEM using transformation rules.

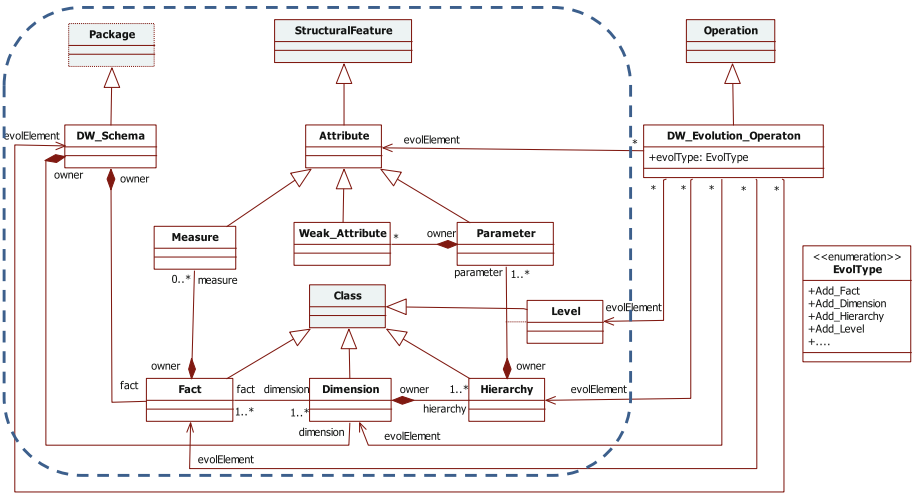

Fig. 6. DW evolution meta-model.

6.2 M2M Transformations in QVT: REM to DWEM

The first aim of our approach is to determine the evolution operations to apply on the DW model after the appearance of new analytical needs. Figure 7 lists transformations potentially applicable to the DW according to the evolution strategies.

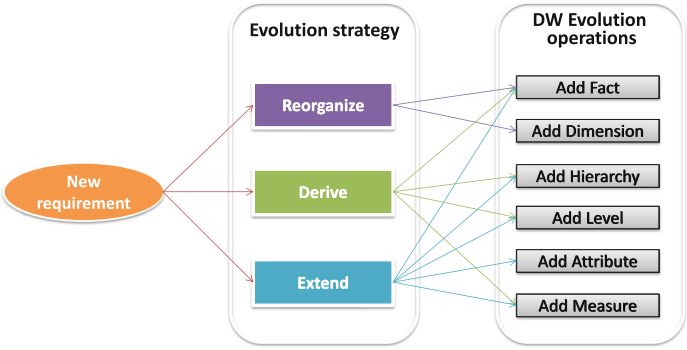

Fig. 7. DW evolution operations for new requirements [17].

Due to space limitation, we define the rules that transform a new requirement into the *Add_Fact* evolution operation that adds a fact into the DW.

Each new requirement, defined using an appropriate model, is converted automatically into a set of evolution operations on the target model (DW evolution model).

The relation *Main* is the entry point of the transformation process; it has elements of the two following models (cf. Fig. 8):

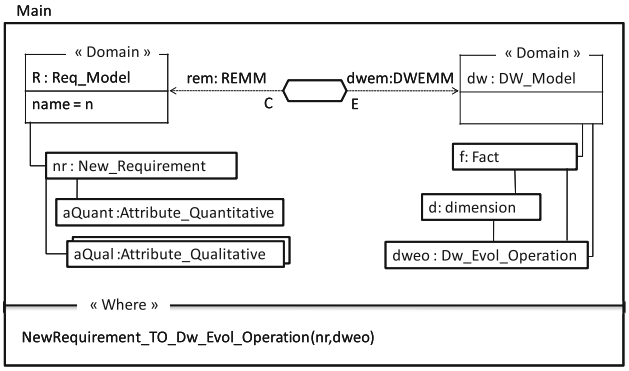

Fig. 8. Graphical representation of the QVT relation *Main* [17].

– « *rem* » model conform to *REMM* (Requirement Evolution Meta-Model),
– « *dwem* » model conform to *DWEMM* (DW Evolution Meta-Model).

The *Domain* element of the « *rem* » model is marked with « C » (*Checkonly*); this means when a transformation occurs in this direction (i.e. the direction of a Checkonly domain) it simply checks if there is a valid match in the relevant model that satisfies the relationship. The domain of the « *dwem* » model is marked with « E » (*Enforce*); this means when a transformation occurs in this direction (i.e. the direction of the model of an enforced domain) if the checking fails then the target model « *dwem* » is modified to satisfy this relation. The left side of this relation describes the elements of the source model « *rem* » , which transforms into elements of the target model « *dwem* » . More specifically, a new requirement from the left « *nr: New_Requirement* » transforms into evolution operation(s) for the DW « *dweo: Dw_Evol_Operation* » by invoking the relation *New_Requirement_TO_Dw_Evolution_Operation (nr, dweo)* specified in the *where* clause. Consequently, the following relations executes:

- *New_Requirement_TO_AddDimension,*
- *New_Requirement_TO_AddLevel,*
- *New_Requirement_TO_AddFact,*
- *New_Requirement_TO_AddMeasure,*
- *New_Requirement_TO_AddParameter, and*
- *New_Requirement_TO_AddAttribute.*

Let us focus on the *New_Requirement_TO_Add_Fact* relation. Figure 9 describes the relation that transforms a new requirement « *nr* » into the DW evolution operation *Add_Fact*.

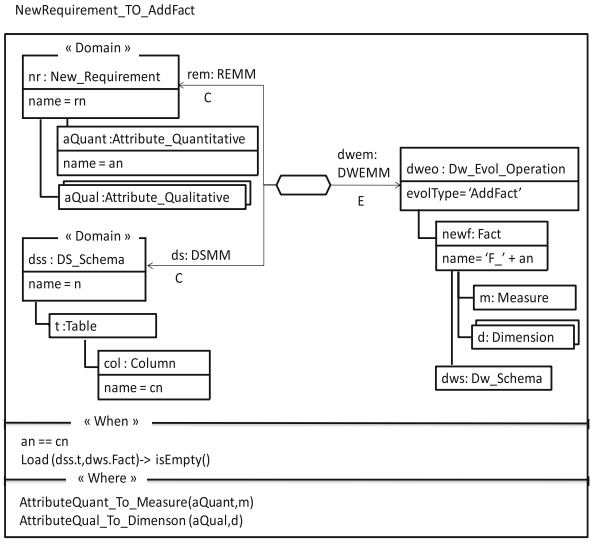

Fig. 9. QVT relation *NewRequirement_TO_AddFact* [17].

Since we are treating the DW evolution problem according to the extension strategy, we have elements from the DS model *(« Domain: Ds_Schema »)* in the *New_Requirement_ TO_Add_Fact* relation. Truthfully, a quantitative attribute a_{Quant} (in a new requirement *nr*) that belongs to a table *t* of the DS model « *dss* » may create a new fact *newf* in the DW model « *dws* » if table *t* does not load any fact of the « *dws* » . Then, the a_{Quant} attribute feeds a measure of the new fact *newf* via the relation *AttributeQuant_ to_Measure*(a_{Quant},*m)*. The dimensions of *newf* will be deducted from the qualitative attributes present in the new *nr* requirement using the relation *AttributeQual_To_Dimension* (a_{Qual},*d)*.

6.3 M2M Transformations in QVT: DSEM to DWEM

Here, we define our QVT rules to transform the DS evolution model into a DW evolution model. Figure 10 depicts how the evolution operations performed on the DS model will be transformed into evolution operations on the DW model. Among these relations, we have selected to detail *AddTable_TO_AddDimension* and *AddTable_ TO_AddFact*.

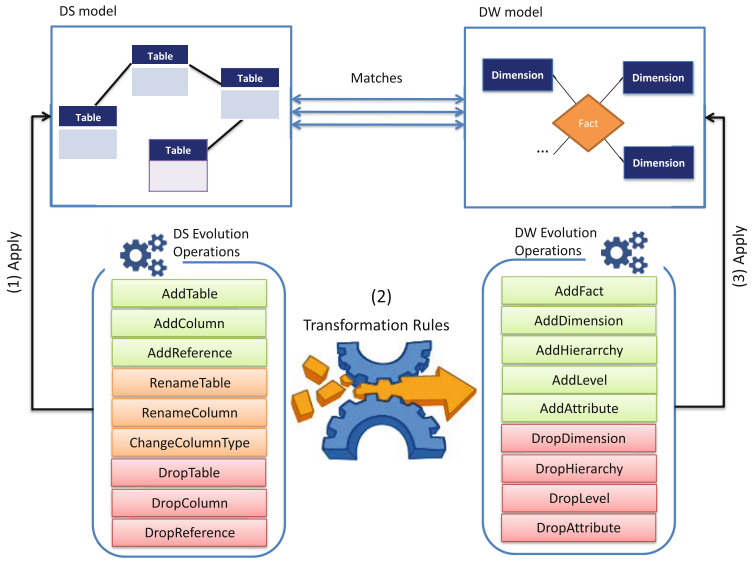

Fig. 10. Principle of transforming DS-evolution operations into DW-evolution operations.

Relation AddTable_TO_AddDimension.
Note that in multidimensional modeling, each fact *f* is associated with a finite set of *n (n > 1)* dimensions; each dimension is an analysis axes of the measures in *f*. Dimensions are loaded from the DS tables directly or indirectly related to the table that feeds *f* [18, 19]. Hence, if a new table *newt* is added to the DS and is referenced by a table that feeds *f*, then *newt* transforms into a dimension for *f*.

This *AddTable* evolution operation is achieved through the QVT relation *AddTable_TO_AddDimension* in Fig. 11 that transforms the operation « *AddTable* » on the DS data-model into the operation « *AddDimension* » on the DW data-model. The *When* clause specifies the condition to check for executing this relation. It means if the new table noted « *newt* » is referenced by a table noted « *reft* » that feeds a fact « *f* » through the relation « *Load(reft, f)* » then « *newt* » will be transformed into a new dimension « *newd* » via the relation « *Table_TO_Dimension(newt, newd)* » specified in the *Where* clause.

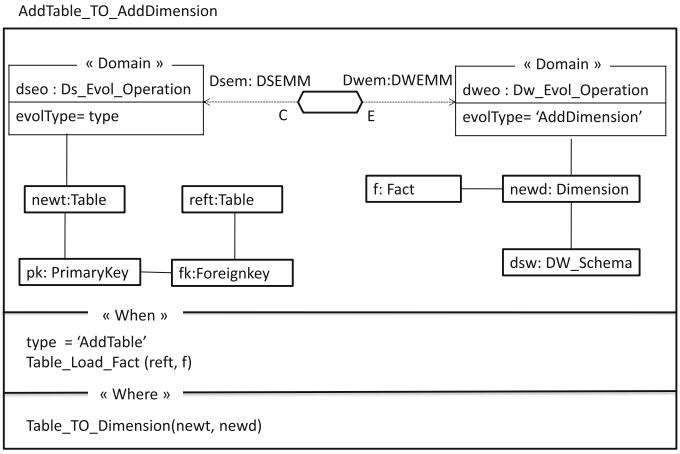

Fig. 11. QVT relation *AddTable_TO_AddDimension*.

Relation AddTable_TO_AddFact

In DW design approaches, an n-ary relationship having non-(prime and foreign key) numeric columns transforms into a fact [20, 21].

This heuristic helps us to decide whether a new table added to the DS will transform into a fact or not. Thus, the relation *AddTable_TO_AddFact* transforms the *AddTable* operation into *AddFact* operation on the DW. Figure 12 gives its formalization in QVT explained hereafter. If the new table « *newt* » refers to two tables « *ta* » and « *tb* » that feed two dimensions « *da* » and « *db* » respectively, and if « *newt* » has numeric columns then « *newt* » is likely to transform into fact via the relation « *Table_To_Fact (newt, newf)* » . Numeric columns in *newt* transform into measures through a relation called « *Column_To_Measure (c, m)* » not defined in this paper.

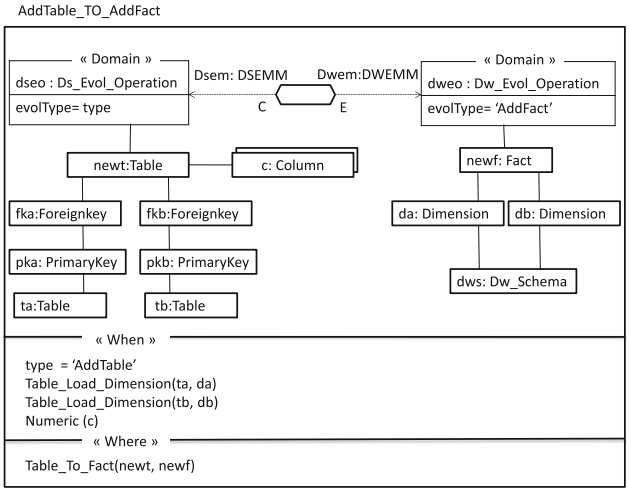

Fig. 12. Relation *AddTable_TO_AddFact* in QVT.

6.4 Validation and Adaptation Module

Once the DWEM is generated, thereafter the new DW model could be visualized graphically; this enables the DW Administrator (DWA) to follow/study the effects (i.e., suggested changes) of the DS-evolutions operations on the original DW model. Figure 13 shows the DWE graphical interface after adding the *Retail_Outlet* table to the DS model. At this stage, the DWA can validate these changes or adapt them according to the evolution requirements. Consequently, the DWEM is automatically modified and then the M2T process generates the code.

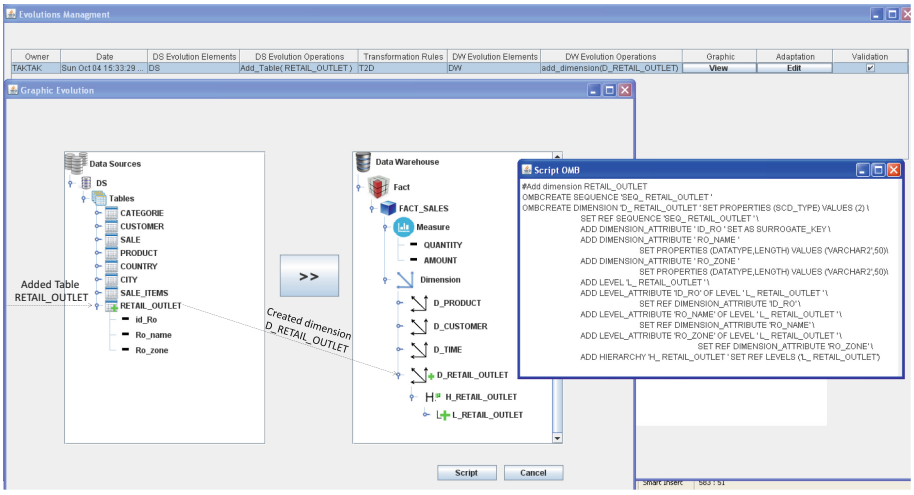

Fig. 13. Sample DWE interfaces (graphical and code).

6.5 Implementing M2t Transformations

We use *Acceleo* plugin that implements the *MOFM2T* standard of the OMG [22]. *Acceleo* provides tools for generating codes from models. This generation of code conforms to a template-based approach.

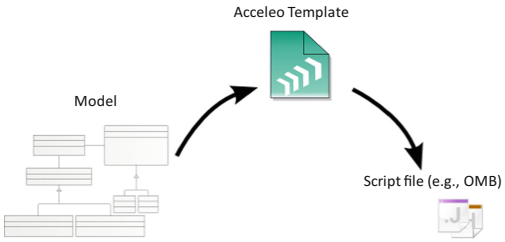

Fig. 14. *Acceleo* schema for the generation of OMB script.

A template is a text containing placeholders to fill with information extracted from the input model (Fig. 14). For our running example, the input model is the DW evolution model issued from the Requirement Evolution Model (REM) or DS Evolution Model (DSEM). For M2T transformations, we developed a PSM (Platform Specific Model) as an *Acceleo* template for generating the code [17] for the target platform *Oracle Warehouse Builder* (OWB). Our Template generates *OMB* (Oracle MetaBase) script that runs under *OMB-Plus* with *Oracle JDeveloper* or *OMB-Plus console*. The execution of this template generates the code to connect to OWB and propagates the changes to the DW data-model (Fig. 13).

7 Preliminary Results and Evaluation

Using the case study of Fig. 2, we have conducted a preliminary assessment by considering a significant set of DS and OLAP-requirements evolution scenarios leading to changes on the DW model, as the creation of new facts and dimensions. The achieved results are very promising. Hereafter, we present four evolution scenarios:

7.1 Evolution Scenarios of the DS

The creation of the DS-table RETAIL_OUTLET (Id_Ro, Ro_name, Ro_zone…) with a reference from the SALES DS-table to the RETAIL_OUTLET DS-table causes applying rule T2D that creates a dimension D_RETAIL_OUTLET linked to the F_SALE fact.

Adding the DS-table SCORE_PROD (#Id_Prod, #Id_Cust, $Score_{Numeric}$…) referencing tables PRODUCT and CUSTOMER has caused applying rule T2F that creates the fact F_SCORE_PROD referring dimensions D_CUSTOMER and D_PRODUCT and having Score as a measure.

7.2 Evolution Scenarios Due to OLAP Requirements

Assume the decision-maker wants to analyze the Sales by *Category* (analysis parameter) of products. The Category is in the DS but not in the multidimensional model. To do so, he gives a rule indicating that the last digit of the product identifier (Id_Prod) codifies the *Category* of the product. Because of this evolution in requirement, a new parameter "*Category*" is created within a new hierarchy *Id_Prod* → *Category* for the *D_PRODUCT* dimension.

Suppose the decision-maker needs to analyze The Sales by product provider. The DW does not exist in the DW but the *Provider* table exists in the DS. The prototype creates a new parameter *Id_Prov* within a new hierarchy *Id_Prod* → *Id_Prov* for the D_PRODUCT dimension.

Actually, DWE offers the DW administrator the ability to graphically view the changes suggested on the DW model, adjust these changes, and automatically generate the DW alteration script. This allows a considerable grain in terms of quality and time. Further experiments are in progress to improve the quality of the propagations obtained, for example, the systematic addition of any weak attributes to be associated with a new inserted parameter.

8 Conclusion

In this paper, we have proposed a model-driven based approach in order to automate the propagation of the evolution of OLAP-requirements and the data source model towards its associated data warehouse. To do so we have defined three evolution models: DSEM (DS Evolution Model), REM (Requirement Evolution Model) and DWEM (DW Evolution Model). Furthermore, we have defined a set of transformation rules and formalized them in QVT (Query/View/Transformation) language; these rules implement the transformation process for the passage between these models; they support the propagation of changes due to changes occurred in the data source or to new OLAP-requirements.

In order to validate our approach, we have developed a software prototype called *DWE* (Data Warehouse Evolution). DWE is compliant to the Model Driven Approach. Moreover, we have presented the functional architecture of DWE based on two levels of transformations. The first is Model-to-Model (M2M) which transforms the DS and the requirements evolution data-models into a DW evolution data-model. The second transformation is Model-To-Text (M2T), which generates the script for the DW alteration using *Acceleo* templates that we have defined for generating OMB (Oracle MetaBase) code. The execution of this template allows log in to Oracle Warehouse Builder and executing the OMB scripts that alter the DW data-model.

Our *DWE* prototype differs from the literature solutions mainly because it provides (semi-)automatic propagation of evolutions applied to the OLAP-requirements and DS data-model towards the DW data-model. Indeed, *DWE* covers the whole cycle of the DW evolution starting from the identification of the DW evolutions and extends to code generation. Additionally, being MDA-based, *DWE* allows benefits offered by this

technology (i.e. independence of platforms, reduction of efforts, reuse of models, and improvement of the quality of result).

This work is currently opening up many perspectives. As a further step, we intend to study the effect of such evolutions on the ETL (Extract-Transform-Load) process. Obviously, the ETL process must evolve to consider the effects of the DS-DW changes on the existing loading procedures. We are also planning a case study for efficiency measurement and performance evaluation of the transformation rules.

References

1. Kimball, R., Ross, M.: The Data Warehouse Toolkit, 2nd edn. Wiley, New York (2002)
2. Golfarelli, M., Rizzi, S., Vrdoljak, B.: Data warehouse design from XML sources. In: Proceedings of ACM International Workshop on Data Warehousing and OLAP (DOLAP 2001), Atlanta, GA, USA, pp. 40–47 (2001)
3. Rusu, L.I., Rahayu, W., Taniar, D.: A methodology for building XML DW. Int. J. Data Warehous. Min. **1**(2), 67–92 (2005)
4. Nabli, A., Soussi, A., Feki, J., Ben Abdallah, H., Gargouri, F.: Towards an automatic data warehouse and data mart design. In: 7th International Conference on Enterprise Information Systems (ICEIS 2005), Miami, USA, pp. 226–231 (2005)
5. Rundensteiner, E.A., Nica, A., Lee, A.J.: On preserving views in evolving environments. In: The 4th International Workshop Knowledge Representation Meets Databases, pp. 131–141 (1997)
6. Bellahsene, Z.: Schema evolution in data warehouses. Knowl. Inf. Syst. **4**(3), 283–304 (2002)
7. Thakur, G., Gosain, A.: A comprehensive analysis of materialized views in a data warehouse environment. Int. J. Adv. Comput. Sci. Appl. (IJACSA) **2**(5), 76–82 (2011)
8. Papastefanatos, G., Vassiliadis, P., Simitsis, A., Sellis, T., Vassiliou, Y.: Rule-based management of schema changes at ETL sources. In: Grundspenkis, J., Kirikova, M., Manolopoulos, Y., Novickis, L. (eds.) ADBIS 2009. LNCS, vol. 5968, pp. 55–62. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12082-4_8
9. El Akkaoui, Z., Zimànyi, E., Mazón, J.N., Trujillo. J.: A model-driven framework for ETL process development. In: Proceedings of the ACM 14th International Workshop on Data Warehousing and OLAP (DOLAP 2011), New York, USA, pp. 45–52 (2011)
10. Wrembel, R., Bębel, B.: Metadata management in a multiversion data warehouse. In: Spaccapietra, S., et al. (eds.) Journal on Data Semantics VIII. LNCS, vol. 4380, pp. 118–157. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-70664-9_5
11. Favre, C., Bentayeb, F., Boussaid, O.: Dimension hierarchies updates in data warehouses: a user-driven approach. In: 9th International Conference on Enterprise Information Systems (ICEIS 2007), Madeira, Portugal, pp. 206–211 (2007)
12. Thakur, G., Gosain, A.: DWEVOLVE: a requirement based framework for DW evolution. SIGSOFT Softw. Eng. Notes **36**(6), 1–8 (2011)
13. Solodovnikova, D., Niedrite, L., Kozmina, N.: Handling evolving data warehouse requirements. In: Morzy, T., Valduriez, P., Bellatreche, L. (eds.) ADBIS 2015. CCIS, vol. 539, pp. 334–345. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23201-0_35
14. Object Management Group (OMG): Model Driven Architecture (MDA) (2004)
15. Taktak, S., Feki, J., Zurfluh, G.: Toward evolution models for data warehouses. In: 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2014), Lisbon, Portugal, pp. 472–479 (2014)

16. Bellatreche, L., Wrembel, R.: Evolution and versioning in semantic data integration systems. J. Data Semant. **2**, 57–59 (2013)
17. Taktak S., Alshomrani S., Feki J., Zurfluh G.: The power of a model-driven approach to handle evolving data warehouse requirements. In: Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017), pp. 169–181 (2017). ISBN 978-989-758-210-3
18. Hachaichi, Y., Feki, J., Ben-Abdallah, H.: Designing data marts from XML and relational data sources. In: Design and Advanced Engineering Applications: Methods for Complex Construction. Advances in Data Warehousing and Mining Series, pp. 55–80. IGI Global (2009). Bellatreche Edition
19. Taktak, S., Alshomrani, S., Feki, J., Zurfluh, G.: An MDA approach for the evolution of data warehouses. Int. J. Decis. Support Syst. Technol. (IJDSST) **7**(3), 65–89 (2015)
20. Golfarelli, M., Maio, D., Rizzi, S.: The dimensional fact model: a conceptual model for data warehouses. Int. J. Coop. Inf. Syst. **7**(2–3), 215–247 (1998)
21. Hachaichi, Y., Feki, J.: An automatic method for the design of multidimensional schemas from object oriented databases. Int. J. Inf. Technol. Decis. Mak. **12**(06), 1223–1259 (2013)
22. Object Management Group (OMG): MOF Model to Text Transformation Language, v1.0 (2008). http://www.omg.org/spec/MOFM2T/1.0/