The power of a model-driven approach to handle evolving Data Warehouse requirements

Saïd Taktak, Saleh Alshomrani, Jamel Feki, Gilles Zurfluh

The Power of a Model-Driven Approach to Handle Evolving Data Warehouse Requirements

Said Taktak[1], Saleh Alshomrani[2], Jamel Feki[2] and Gilles Zurfluh[3]
[1]*University of Sfax, FSEGS Faculty, Miracl Laboratory, Sfax, Tunisia*
[2]*Faculty of Computing and IT, University of Jeddah, Jeddah, Saudi Arabia*
[3]*University of Toulouse 1 Capitole, IRIT Laboratory, Toulouse, France*

Keywords: Data Warehouse, Evolution Modeling, OLAP Requirements, MDA, M2M, M2T.

Abstract: The Data Warehouse (DW) is characterized by complex architecture, specific modeling and design approaches. It integrates data issued from operational data sources in order to meet decision-makers' needs by providing answers for OLAP queries (On-Line Analytical Processing). In practice, both data source models and decision-makers' analytical requirements evolve over time and, therefore, lead to changes in the DW multidimensional model. In this evolving context, we have developed the *DWE* (Data Warehouse Evolution) framework. DWE automatically propagates the changes of the data source data-model on the DW data-model. This paper proposes a model-driven approach for extending DWE in order to consider a further related evolutionary aspect: The evolution of decision-makers' needs. It deals with the propagation of these evolutions on the DW multidimensional model. This approach relies on a classification of evolution scenarios and a set of transformation rules for the identification of evolution operations to apply on the DW.

1 INTRODUCTION

DW modeling has been considered, for more than one decade, as a real challenging research topic for which several approaches are proposed. Three major categories of approaches for designing a DW schema (i.e., data-model) are well known in the literature: Top-down (Kimball and Ross, 2002), bottom-up (Golfarelli et al., 2001; Rusu et al., 2005), and mixed (Nabli et al., 2005) approaches.

All these DW design methods rely on a rigid assumption that the *conceptual model of the DW istime-invariant*. However, in practice, this assumption restricts the evolution of the real world and, therefore, does not hold most of the time since, the DW model may evolve due to internal and/or external factors (e.g., business processes, organization environment). Furthermore, it is difficult to determine definitively the DW model at the design phase; in fact, for sustainability issues, it is often necessary to undergo changes after its implementation. These changes are due to two main reasons: (a) Evolution of analytical needs of decision-makers: changes in these needs might require extending the DW model (e.g. adding new axes or subject of analysis), and (b) Evolution of data source model (DS) dictated by the evolution of the organization business processes (e.g., adding or even deleting conceptual entities). To the best of our knowledge, we claim that the problem of changes in the DW model has not been sufficiently addressed yet neither by the research community nor by the DW software editors. As well, all evolution strategies of the literature are at a single modeling level: schemas before and after changes are conform to the same meta-model. To the best of our knowledge, in the DW domain, the evolution of schemas expressed in different models have not yet received their full part of investigation.

To alleviate this problem, we proposed in (Taktak et al., 2015) an MDA (Model Driven Architecture) approach that automates the propagation of the evolution of the DS model towards its associated DW model. In this paper, we extend our contribution by considering the impacts of changes of decision-makers' requirements on the DW data-model (DW). In this context, we suggest an approach based on a classification of evolution scenarios and a set of transformation rules for the identification of evolution operations to apply on the DW model.

This paper is organized as follows. In Section 2, we give a review of works dealing with the DW evolution problem. Section 3 describes our MDA-based approach for the propagation of decisional requirements evolution towards the multidimensional DW model; this approach extends our framework *DWE* «Data Warehouse Evolution». Section 4 introduces our classification of evolutions of plausible decisional requirements. Section 5 details our extension; it presents the process of identifying DW evolution alternatives according to our classification; in addition, it develops a set of algorithms to derive the appropriate changes that should apply on the DW model. Section 6 describes the DWE technical implementation including MDA transformations at two levels: Model-To-Model (M2M) and Model-To-Text (M2T). The conclusion section summarizes the paper and enumerates its perspectives.

2 RELATED WORKS

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

2.1 Approaches based on DS Evolution Model

Organizations' business processes evolve over time. This is due to the modification of existing processes or to the emergence of new processes with new real world objects. These evolutions affect the data-model of the information system that feeds the DW. In turn, the DW cannot be immunized against these evolutions in its data source; consequently this evolution deserves to be studied so that it becomes semi-(automatically) propagated towards the DW data-model, the DW ETL (Extract Transformed and Load) process, and stored data. This evolution problem was addressed from different viewpoints. We can 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, Rundensteiner *et* al. (1997) and Bellahsene (2002) proposed approaches for a dynamic adaptation of materialized views in response to an evolution of the DS DM. These approaches maintain not only the schema views, but also their instances (i.e., data). The main idea of this contribution is to avoid recalculating views after DS changes so that a new schema view derives from the old one. More details on views maintenance in multidimensional context are available in (Bellahsene, 2002).

Other research contributions have offered solutions for adapting the ETL process during the DS data-model evolution. Among these works, solutions proposed in (Papastefanatos et al., 2009) and (El Akkaoui et al., 2011) 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 treating the impact of the DS evolution on the DW model (Dimensions, facts, hierarchies …).

To address these shortcomings, the authors of (Wrembel and Bebel, 2007) defined a formal model for a multi-version DW. They presented a set of evolution operations that affect the DW schema and its instances. 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; it bases on "What-If" analyses. In order to validate their approach, they developed the MVDW (Multi-Version Data Warehouse) prototype for the maintenance of the DW and the management of its versions. The major drawback of this solution is the manual identification of the DW evolution operations; it mostly requires high expertise of the DW administrator and then is out of reach of end-users.

2.2 Approaches based on Business Requirement Evolution

Let us note that in mixed approaches (Phipps and Davis, 2002; Nabli et al., 2005), the DW design is based firstly on the DS model and, secondly, on the decision-makers requirements. Obviously, we note that decision-makers needs are not static since they evolve through time. Therefore, the DW created based on initial requirements may become obsolete and not satisfy the new requirements. To overcome this issue, it is necessary to consider the new analytical requirements and adapt the DW to encompass them. Among the research works of this

category, the authors in (Favre *et al.,* 2007) suggested an approach for the customization of analyses based on "If-Then" rules model; this model allows the users to integrate their own knowledge in order to enlarge the analysis alternatives of the DW by changing its 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 main goal of this work is the analytical requirements introduced by each user are processed and transformed into DW evolution operations. However, the authors assume that users are skilled enough to intervene properly during this task. Moreover, note that the proposed changes are simple: they do not cover all cases that decision-makers may ask for.

To surmount this problem, Talwar and Gosain (2012) studied the evolution of complex hierarchies (multiple alternative hierarchies, dependent and independent parallel hierarchies). They defined a set of evolution operations equipped with constraints 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 of (Thakur and Gosain, 2011) 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 the decision-makers new needs. In fact, this task is borne entirely by the DW administrator.

In the same context, authors in (Solodovnikova et al., 2015) have also investigated the problem of business requirements evolution. They defined a formalism for modeling the new needs of decision-makers and proposed a semi-automatic approach to adjust and create a new version of the DW model. However, the evolution operations supported by this solution are simple and lack precision. For example, when adding an attribute, the proposed algorithm is able to identify the dimension to change but not the role of the new attribute in the dimension: create a new hierarchy, insert a level into an existing hierarchy... Details about this operation are really a heavy task left to a skilled user.

2.3 Discussion

The related works have focused on two complementary categories of evolutions in DW systems, namely evolution of the DS model and evolution of decision-makers' needs. We have identified three deficiencies concerning i) complementarity, ii) complexity of the evolutions, and iii) automatic propagation of changes.

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

Secondly, few works were interested in studying the DS evolution effects 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 concern in these works, and when addressed, it was carried out in a traditional way.

Finally, from the technological side, we note that all proposed solutions were realized in a conventional software engineering context; therefore, implementations are platform-dependent. Obviously, using the MDA approach (OMG, 2004) allows benefiting from its multiple advantages.

The objective of this paper is to extend our *DWE* (Data Warehouse Evolution Framework) (Taktak et al., 2015) initially designed to automate the propagation of changes from DS towards the multidimensional DW. This extension consists of enriching DWE to accommodate the evolution of decision-makers' needs. This extension should provide for a complete solution that will cover the DS evolution and the decision-makers' requirements evolution. Our solution is compliant with the Model Driven Engineering (MDE) methodology that promotes the semi-automatic propagation of business requirements towards the multidimensional DW. Relying on MDE technology in DWE is a challenging proof. In fact, MDE facilitates the realization of the proposed extension. Consequently, our proposed approach inherits benefits from this technology (i.e. independence from platforms, reduction of efforts, and improvement of the quality of result). The strength of MDE is the reuse of models; we profit from this advantage to reuse the models of evolution and transformation mechanisms already implemented in our DWE. Furthermore, we introduce a new model to define the decision-makers' needs evolutions. In the remaining of this

paper, we present our approach that addresses the DW model evolution problem, and we focus on evolutions stemming from decision-makers.

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, towards the DW multidimensional model. Figure 1 shows our approach where the evolution of the DW model is due either to an evolution of its DS model (Figure 1, panel A) already treated in (Taktak et al., 2015), or to an evolution of decision-makers needs (panel B) which will be the focus of this paper.

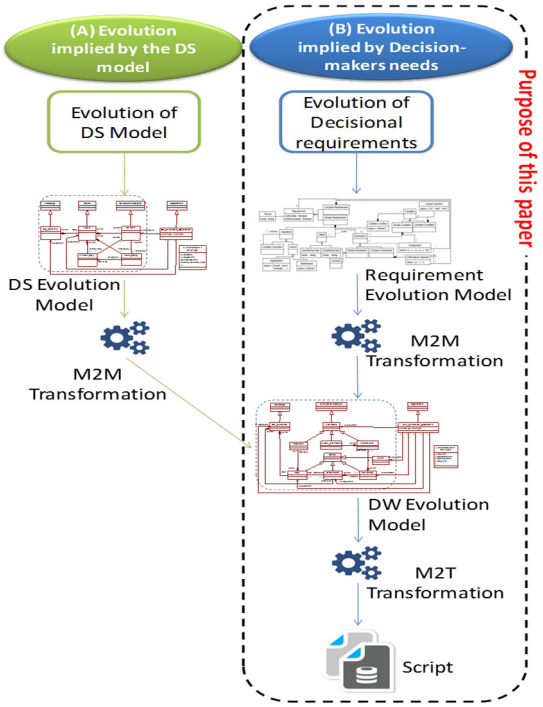

Figure 1: Overview of our DW evolution approach.

For this purpose, we define an appropriate evolution model for the new decision-makers' needs; this enables us reusing our DW evolution model (Taktak et al., 2014) so that we keep the same M2T (Model-To-Text) transformation rules for code generation.

Our approach bases on three evolution models: i) *DS Evolution Model* (DSEM), ii) *DW Evolution Model* (DWEV), and iii) *Requirements Evolution Model* (REM). Besides, it applies two types of transformations: M2M and M2T.

-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 derive from the DS evolution model.

-REM: It describes the new needs of decision-makers in terms of subject and axes of analysis. It also allows defining the knowledge introduced by the user (e.g. rules, formulas). This model will transform into DWEM.

-M2M transformation: Generates the DWEM from REM. 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: 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 the treatment of the DS evolution as for the treatment of the needs evolution. In fact, this reuse is feasible because these two transformations start from the same DWEM evolution model.

In the next section, we classify decision-makers needs into three classes; this helps us defining the impact of each class on the DW multidimensional model as a set of common operations.

4 CLASSIFICATION OF DECISIONAL REQUIREMENTS AND EVOLUTION SCENARIOS

The evolution of the decision-makers' needs leads to several cases of evolution on the DW model. We group these evolution cases in three classes namely: *Evolution by derivation*, *Evolution by reorganization*, and *Evolution by extension*.

For illustration examples, we refer to the multidimensional DW and its associated relational DS modeled described in Figure 2.

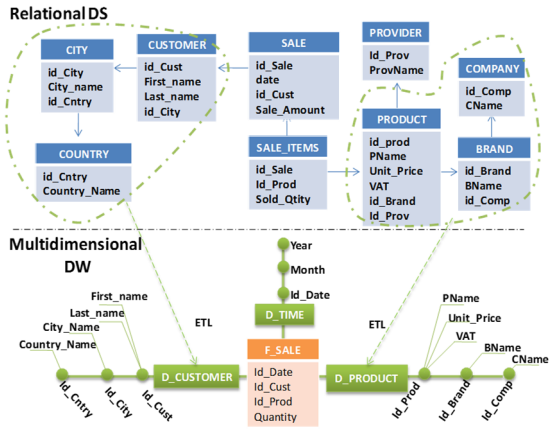

Figure 2: Data Source model and its DW model.

4.1 Evolution by Derivation

This class of evolutions consists of using elements of the DW model for which we derive new elements. The DW administrator should be skilled enough to provide knowledge to achieve this evolution. Knowledge expressed as rules or formulae.

4.1.1 Derivation by Rule

For this derivation type, simply we need to define rule(s) to apply on an element of the DW model in order to derive a new element to integrate into the same DW model. The general derivation form of this could be of the following form:

$$n = \mathcal{Y}(a)$$

where n is the derived attribute, $\mathcal{Y}$ is a user-defined extraction function, and a is a DW attribute.

The derivation by rule is convenient for complex codes of entities (i.e., identifiers composed of small pieces of information), as the French civil identifier INSEE code (2-digits department code + 3-digits city code), or a product code including the product category and sub-category or even the country of origin. We can easily derive (extract) new data from these codes. In addition, a *Date* attribute likely to generate the day, month, quarter… for a *Time* dimension.

As an illustrative example, consider the DW in Figure 2, and let us assume that a decision-maker chooses to analyze the sales by Category of products. The criterion of analysis *Category* (i.e., hierarchic parameter) is not currently present in the multidimensional model despite it exists implicitly in the data source. Nevertheless, the decision-maker knows that the last digit of the product identifier (*Id_Prod*) indicates the product category. The derivation of the category exploits this knowledge as an extraction rule.

Figure 3: Category created with a new hierarchy.

But, how to deal with the new extracted data called *Category*? What is the role of this Category in the DW model (is it a dimension, parameter within a new or an existing hierarchy?). Here the insertion of the *Category* creates a new hierarchy for the *D_PRODUCT* dimension (cf. Figure 3). Logically, the new hierarchy links the *Id_Prod* to the *Category*.

4.1.2 Derivation by Formula

This type of derivation consists of using a formula to derive a new element to integrate into the DW model. This derivation expresses as below:

$$n = F(a)$$

where n is the derived attribute, F is an arithmetic function, and a is a DW attribute.

Let us proceed with our running sample DW and assume that the decision-maker wants to analyze the *amount of Value Added Tax (VAT)of Sales* according to the three dimensions *D_PRODUCT*,*D_TIME* and *D_CUSTOMER*. The *VAT amount* is not a measure in the *F_SALE* fact; nevertheless, it is derivable by the following formula:

*VAT_Amount = Quantity * Unit price * VAT_Rate*

Weak attributes *VAT* and *Unit_Price* in the dimension *D_PRODUCT*, along with the measure *Quantity* in the *F_SALE* fact are useful for computing the required measure *Amount_VAT* for the *F_SALE* fact.

4.2 Evolution by Reorganization

The second class of evolution consists of creating new links between the DW model elements. This is suitable since it avoids duplicating or recreating existing elements. The reorganization process affects mainly the temporal or spatial dimensions; however, other scenarios could be possible. For example, assume that the DW contains two facts: *F_SALE* for *Sales* analysis associated with dimensions *D_CUSTOMER* and *D_PRODUCT* (cf., Figure 4, (1)), and the *F_STOCK* fact associated with dimensions *D_RETAIL_OUTLET* and *D_PRODUCT* (cf., Figure 4, (2)). If the decision-

maker chooses to analyze the *Sales* by *RETAIL_OUTLET* and *PRODUCT*, the *F_SALE* fact will then have to be connected to the dimension *D_RETAIL_OUTLET*(cf., (Figure 4, (3)).

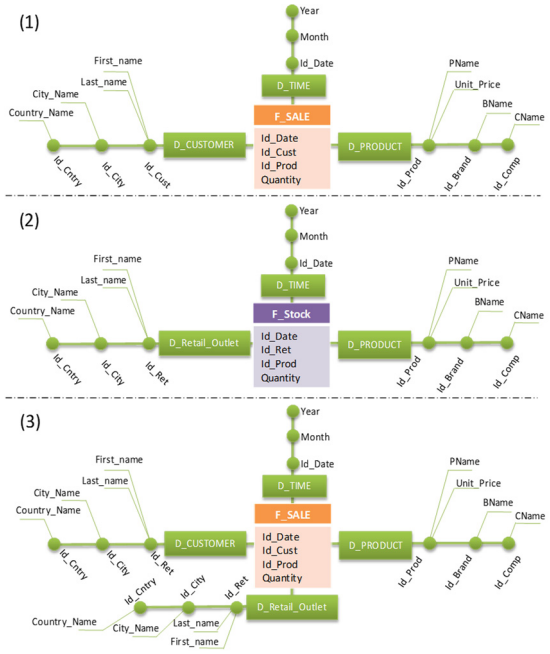

Figure 4: Reuse of the *D_Retail_Outlet* dimension.

4.3 Evolution by Extension

The third class of Evolution requires extending the DW model by new elements extracted from its associated DS. These evolutions apply when the current DW model is inadequate to meet the new requirement either by derivation or by reformulation. In this case, it is necessary to identify which element from the DS we should add to the DW.

For example, the decision-maker needs to analyzing sales according to the products' providers. This data is currently absent in the DW but the *Provider* table exists in the DS. Besides, we have a *Foreign Key* constraint from *Product* to *Provider* indicating the existence of a functional dependency associating each product to its provider. *PRODUCT* feeds the dimension *D_PRODUCT* of the *F_SALE* fact then *Provider* becomes a hierarchic level (parameter) in the *D_PRODUCT* dimension.

These sample evolution examples highlights the importance of propagating the evolutions of the decision-makers' requirements towards the DW. Two issues arise concerning the DW evolution: 1) *Which modifications* to bring to the DW model to meet the new needs (i.e. creating dimension, fact, or level), 2) *How to accomplish these modifications* efficiently and especially fast; speed is a vital factor for some decision-making systems.

The trivial solution completely rebuild the DW starting from the new collection of decision-makers' needs. This solution is inadequate since the DW reconstruction is a heavy and complex task, which requires a lot of time, efforts and high costs. More accurately, it is not feasible especially in domains where the evolution frequency is high (Bellatreche and Wrembel, 2013).

To deal with this problem, we proposed a model driven approach for propagating changes of the decision-makers' requirements towards the DW model in an almost automatic way, thus avoiding the need for full reconstruction of the DW. To do so, we have defined a set of transformation rules to transform new decision-makers' requirements into DW evolution operations. Subsequently, we present these transformation rules textually (as algorithms) then we formalize them using QVT.

5 TRANSFORMATION RULES BY EVOLUTION STRATIGIES

We textually explain the transformation rules for generating the modifications to apply on the DW multidimensional model due to the evolution of decision-makers' requirements. To do so, we rely on the evolutions classification introduced in the previous section for determining the different evolution cases. These cases are:

- *Statico*: Nothing to change if the current DW model allows meeting the new requirement. Otherwise, we verify if a reorganization of the DW model is possible.
- *Reorganization*: Applies when the necessary elements for the new requirement (i.e., measure or attribute) already exist in the DW model but their current role is not adequate. In this case, we redefine the role of these elements to support the new requirement.
- *Derivation*: If a necessary element for the new requirement does not exist in the DW model, we verify whether it is derivable from another DW element. In such a case, the derivation applies according to knowledge introduced by the DW administrator (i.e., rule or expression). Otherwise, if the missing element is derivable from the DS, we extend the DW model with the derived element.
- *Extension*: This alternative is the most delicate evolution. It consists of expanding the

current DW model with new elements to extract from the DS.

Choosing the evolution alternative to apply to the DW model (i.e., Reordering, Derivation, and Extension) relates to the *Main algorithm*.

Note that these alternatives are usable independently or in combination.

In the following, we detail these alternatives and we specify for each case the evolution operations to apply on the DW multidimensional model. To do so, we consider the following notation:

- *Req*: a new requirement
- *A*: the 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*: the set of elements of the DW multidimensional model (i.e., schema)
- *DS*: the set of elements of the DS model.

The *Main* algorithm depicts the principle of defining the evolution strategy. It calls three algorithms *Reorganize* (cf., Algorithm 2), *Derive* (cf., Algorithm 3) and *Extend* (cf., 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 algo
9.              else if a ∉ DW  and a ∈ DS    then
10.                   Extend( )       // see Extend algo
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 can already meet the new requirement (*Req*),and False otherwise.

Rule(a) is a Boolean function True if the 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 starting point of the reorganization process (see algorithm 2) is the identification of the DW model elements (fact, dimensions) that meet the new requirement. This is through two functions *Find_Fact* and *Find_Dimension;* they return respectively the fact containing quantitative attributes A_{quant} and dimensions containing qualitative attributes A_{qual}. The fact f_{new} will be enriched with the set of measures A_{quant} attributes. Dimensions containing A_{qual} attributes are refined by the function *Refine* 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 admin 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 a_d adds to *f* as new measure m_{new}. Otherwise, we create a new fact f_{new} for the derived attribute a_d.

If the derived attribute a_d is a qualitative attribute, it necessarily belongs to a dimension; 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 l_s level and all its predecessors

levels. l_{new} adds to the new hierarchy as a new 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. l_s = Find_Level(a_{source}) // 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. $h_{new}.L = l_1..l_s$ //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
End.

5.3 Extension

The *Extend* algorithm defines the principle of this extension, which enriches the DW model with elements extracted from the DS according to the new decision-maker 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.

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 predecessors levels of 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.

Algorithm 4: Extend.

Input:
DW, DS,
a_e : attribute to retrieve from the data source
Begin:
1. *t = Find_Table(a_e)*// return the table which contains a_e
2. if $a_e \in A_{qual}$ then
3. *l = Load_Level(t)* //return the level which is loaded from *t*
4. if *l* == null then
5. *t' = ref(t)* // return the table which references table t
6. *t'' = IsRef (t)* // return 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

```
        anew.p = l.p
        Add_Attribute(anew)
      end if
    else
      f = Load_Fact(t)
      if f not null  then
        mnew = ae ;  mnew.fact = f
        Add_Measure (mnew)          // add measure mnew to the fact f
      else
        fnew. M = ae ;   Add_Fact(fnew)   // add fact fnew
      end if
    end if
    End.
```

6 IMPLEMENTATION

To validate our approach, we are currently improving our software prototype DWE (Data Warehouse Evolution). DWE© is implemented on the platform Eclipse Modeling Framework (EMF) which is a complete environment for MDA paradigm. Figure 5 shows the overall architecture of DWE© which offers two features for evolution: i) Evolution of DW model according to changes occurred in its DS model, which is developed in (Taktak et al., 2015); ii) Evolution of the DW model to meet the new needs of decision-makers. This paper focuses on this latter feature.

The DW evolution process starts with the modeling of the new requirements and aims to generate the requirement evolution model (REM). Next is the M2M step that transforms the REM 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., suggested changes) of the DW-evolutions operations. At this stage, the DW administrator can validate these changes or adapt them according to the evolution requirements. Finally, the M2T process executes; it transforms the DWEM into script for DW alteration. Next, we detail these steps.

6.1 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 (Solodovnikova et al., 2015) depicted in Figure 6. A new requirement has *quantitative* and *qualitative* attributes, arithmetic operations (i.e., formulations) and logical expressions (i.e., rules).

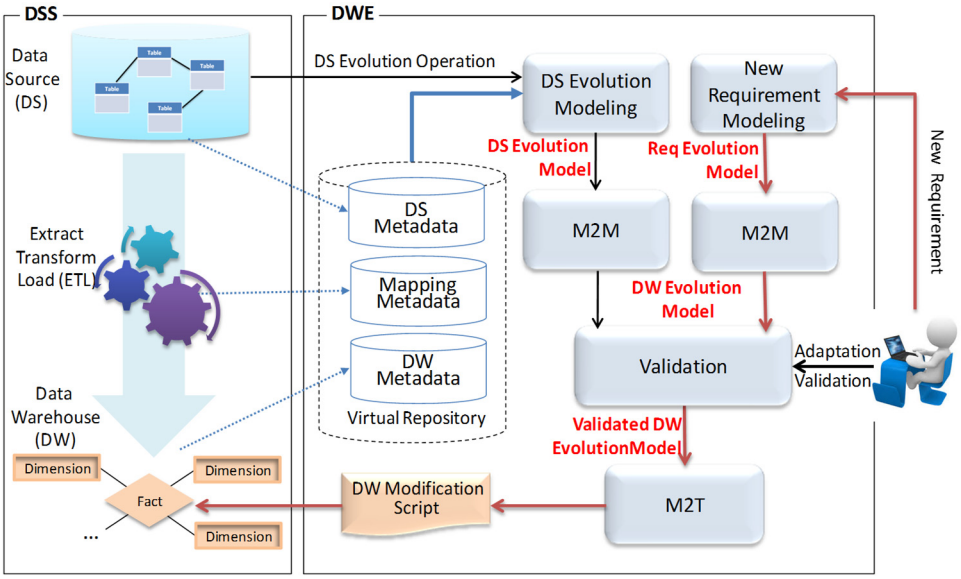

Figure 5: Architecture of the DWE prototype.

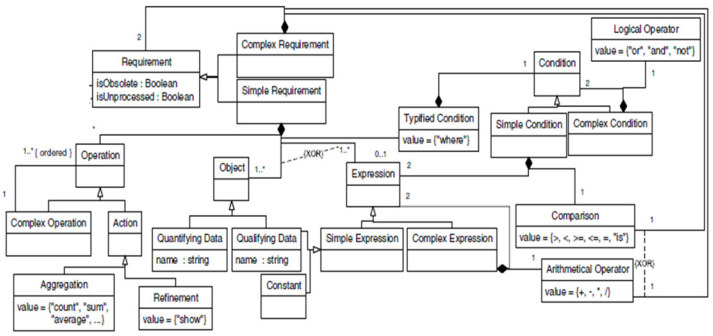

Figure 6: Requirements Evolution Meta-Model (Solodovnikova et al., 2015).

6.2 Implementation of M2M Transformations in QVT

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

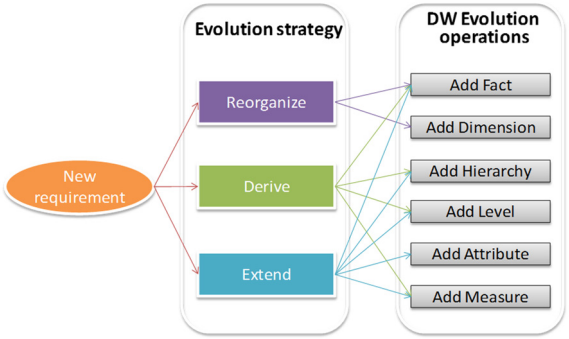

Figure 7: DW evolution operations for new requirements.

In this section, due to space constraint, we illustrate only the definition of transformation rules, which transform a new requirement into the *Add_Fact* evolution operation on the DW.

Each new requirement, defined through a dedicated model for analytical needs, automatically converts into evolution operations in the target model (DW evolution model).

In QVT, a set of *relations* specify a transformation between two models. A relation has:

- Two or more domains: Each domain has a set of elements related to the candidate model;
- Checkonly / Enforce: A domain can be marked Checkonly or Enforce.

Checkonly (C) checks if there is a match in the model that satisfies the relation. If the match is not satisfied then the domain must not be modified. When a domain is marked Enforce (E) and the match is not satisfied, then the model must be modified in order to satisfy the relation:

- Pattern matching: A pattern appears in a domain and allows the selection of a portion of the candidate model.
- *When*: Defines the necessary conditions in order to verify the relation (i.e. Precondition).
- *Where*: Defines the conditions that all elements of the involved model in the relation must satisfy (i.e. Post-condition).

In a transformation, we find two types of relations: *top-level* and *non-top-level*. The execution of a transformation causes the execution of many top-level relations. Non-top-level relations are invoked from the Where clause of top-level relations or from other non-top-level relations OMG (2009).

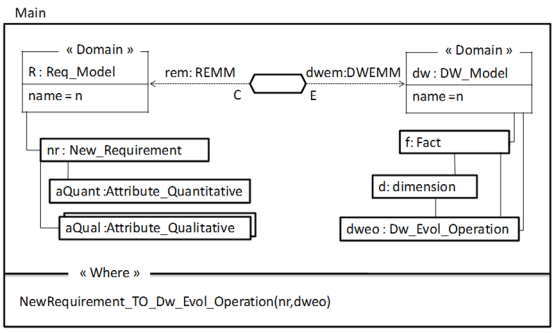

Figure 8: Graphical representation of the QVT relation Main.

Now, we illustrate the QVT formalization of the *NewRequirement_To_AddFact*rule.

We present the relation *Main* that is the entry point of the transformation process. It contains elements of the two following models (cf. Figure 8):

- « *rem* » model conforms to REMM (Requirement Evolution Meta-model),

- « *dwem* » model conforms 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 perform:

- *New_Requirement_To_AddDimension,*
- *New_Requirement_AddLevel,*
- *New_Requirement_To_AddFact,*
- *New_Requirement_To_AddMeasure,*
- *New_Requirement_To_AddParameter, and*
- *New_Requirement_To_AddAttribute.*

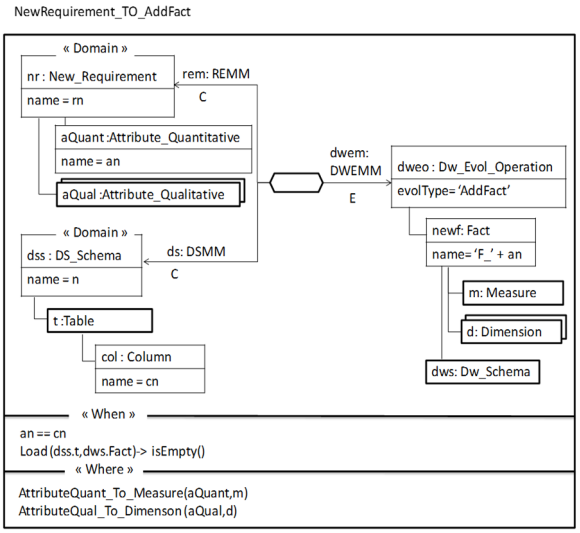

Figure 9: QVT relation *NewRequirement_To_AddFact*.

We focus on the definition of *New_Requirement_TO_Add_Fact* relation. Figure 9 describes the relation that transforms a new requirement « *nr* » into a DW evolution operation *Add_Fact*. Since we are treating the DW evolution according to the extension strategy, we have elements from the DS model *(« Domain: Ds_Schema»)* in the *New_Requirement_TO_Add_Fact* relation. In fact, a quantitative attribute a_{Quant} (in a new requirement *nr*) which belongs to a table *t* of the DS model « *dss*» may create a new fact *newf* in the DW model « *dws*», if the 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 requirement *nr* using the relation *AttributeQual_To_Dimension* (a_{Qual},*d)*.

6.3 Implementing M2T Transformations

We use *Acceleo* plugin that implements the *MOFM2T* standard of the OMG (OMG, 2009). *Acceleo* provides tools for generating codes from models. This generation of code conforms to a template-based approach. A template is a text containing placeholders to fill with information extracted from the input model. For our running example, the input model is the DW evolution model issued from the requirement evolution model (REM).

```
[comment encoding = UTF-8 /]
[module generate('DWEV_MODELS/META-
MODELS/DW_EV_MM.ecore')]
   [template public generate(y : dwevm)]
   [comment @main /]
   [file ('script OMB.txt',false,'UTF-8')]
   OMBCONNECT
orcl/orcl@localhost:1522:ORCL
   OMBLIST PROJECTS
   OMBCC 'MY_PROJECT'
   OMBLIST ORACLE_MODULES
   OMBCC 'SOURCE'
   [for (op :
DW_Evolution_Operation|y.DW_Evolution_Oper
ation)]
<%if (op.evolutionType == 'AddFact') {%>
OMBCREATE CUBE
'[op.fnew.name.trim().toUpper()/]'\

SET PROPERTIES (BUSINESS_NAME,
DESCRIPTION, DEPLOYMENT_OPTIONS)\
```

Figure 10: Extract from the *Acceleo* template for the generation of OMB script.

```
VALUES              ('[op.fnew.name/]_Cube',
'[op.fnew.name/]_Cube', 'Deploy All')
[for (m : measure|op.fnew.measure)]
OMBALTER                                CUBE
'[op.fnew.name.trim().toUpper()/]'      ADD
MEASURE '[m.name/]'\
SET                                 PROPERTIES
DATATYPE,PRECISION,SCALE,BUSINESS_NAME,DE
SCRIPTION)\
VALUES
('NUMBER',10,2,'[op.fnew.name/][m.name/]'
,'[op.fnew.name/][m.name/]')
[/for]<%}%>
#.............................
[/template]
```

Figure 10: Extract from the *Acceleo* template for the generation of OMB script (Cont.).

For M2T transformations, we developed a PSM (Platform Specific Model) as an *Acceleo* Template for generating the code (cf., Figure 10) for the target platform *Oracle Warehouse Builder* (OWB). Our Template generates *OMB* (Oracle MetaBase) script that runs under *OMB-Plus* with an Oracle *JDeveloper* or in an OMB-Plus console. The execution of this template generates the code to connect to OWB; it modifies the DW data-model.

7 CONCLUSION

In this paper, we have addressed the problem of evolution in the decision support systems. In particular, we have studied the effect of the evolution of decision-makers' requirements on the multidimensional DW model. To do so, we have proposed an extension of our DWE (DW Evolution) prototype that addresses evolutions coming from data sources as well as evolutions due to the new requirements of decision-makers. We have elaborated a classification of scenarios of possible evolutions namely reorganization, extension or diversion. Furthermore, for each of these evolution classes, we have proposed a process for identifying DW evolution operations (add fact, add level...). Our approach takes advantage of the Model-To-Text transformations implanted in our DWE, which we reuse to transform these new evolution operations into OMB executable scripts.

This work is currently opening up many perspectives. First, we plan for a case study to evaluate the efficiency of transformation rules. As a further step, we intend to study the effect of such evolutions on the ETL (Extract-Transform-Load) process, which in turn, must evolve to take into account the effects of the DW changes on the existing loading procedures.

REFERENCES

Bellahsene, Z., 2002. Schema Evolution in Data Warehouses. *Knowledge and Information Systems*, *4*(3), (pp. 283-304).

Bellatreche, L., Wrembel, R. (2013). Evolution and Versioning in Semantic Data Integration Systems. Journal on Data Semantics, (2), (pp. 57-59).

El Akkaoui, Z., Zimànyi, E., Mazón, J. N., Trujillo. J., 2011. A model-driven framework for ETL process development. In *Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP (DOLAP '11),* (pp. 45-52). New York, USA.

Favre, C., Bentayeb, F., Boussaid, O., (2007). Dimension Hierarchies Updates in Data Warehouses: a User-driven Approach. In *9th International Conference on Enterprise Information Systems (ICEIS'07)*, (pp. 206-211), Madeira, Portugal.

Golfarelli, M., Rizzi, S., Vrdoljak, B., 2001. Data Warehouse Design from XML Sources. In *proceedings of ACM International Workshop on Data Warehousing and OLAP (DOLAP'01)*, (pp. 40-47), Atlanta, GA, USA.

Kimball, R., Ross, M., 2002. *The Data Warehouse Toolkit.* Wiley & Sons. New York, 2nd edition.

Nabli, A., Soussi, A., Feki, J., Ben Abdallah, H., Gargouri, F., 2005. Towards an Automatic Data Warehouse and Data Mart Design, *7th International Conference on Enterprise Information Systems (ICEIS'05)*, (pp. 226-231), Miami, USA.

OMG, 2004. Object Management Group: Model Driven Architecture (MDA). *http://www.omg.org/cgibin/doc?formal/03-06-01.*

OMG, 2009. Object Management Group: Meta Object Facility (MOF) 2.0 Query/View/Transformation, version 1.1. *http://www.omg.org/spec/QVT/1.1/Beta2/*

Papastefanatos, G., Vassiliadis, P., Simitsis, A., Sellis, T. Vassiliou, Y., 2009. Rule based Management of Schema Changes at ETL Sources. *International Workshop on Managing Evolution of Data Warehouses (MEDWa'09),* (pp. 55-62), Riga, Latvia.

Phipps, C., Davis, K., 2002. Automating data warehouse conceptual schema design and evaluation. In *the 4th Intern. workshop Design and Management of Data Warehouses.* Canada, 2002, (pp. 23-32).

Rundensteiner, E. A., Nica, A., Lee, A. J., 1997. On Preserving Views in Evolving Environments. In *the 4th International Workshop Knowledge Representation Meets Databases* (pp.131–141).

Rusu, L. I., Rahayu, W., Taniar, D., 2005. A methodology for Building XML DW. In *International Journal of Data warehousing & Mining, 1(2),* (pp.67-92).

Solodovnikova, D., Niedrite, L., Kozmina, N., 2015. Handling Evolving Data Warehouse Requirements. In *the East European Conference on Advances in Databases and Information Systems* (pp. 334-345). Springer International Publishing.

Taktak, S., Feki, J., Zurfluh, G., 2014. Toward Evolution Models for Data Warehouses. *2nd Intern. Conference on Model-Driven Engineering and Software Development*. Lisbon, Portugal, (pp. 472-479).

Taktak S., Alshomrani S., Feki J., Zurfluh, G., 2015. An MDA Approach for the Evolution of Data Warehouses. *Intern. Journal of Decision Support System Technology (IJDSST), 7(3)* (pp. 65-89).

Talwar, K., Gosain. A., 2012. Implementing schema evolution in data warehouse through complex hierarchy semantics. In *the International Journal of Scientific and Engineering Research 3* (pp.917–922).

Thakur, G., Gosain, A., 2011. DWEVOLVE: A Requirement Based Framework for DW Evolution. In *SIGSOFT Softw. Eng. Notes 36, 6* (pp. 1-8).

Wrembel, R., Bębel, B., 2007. Metadata Management in a Multiversion DW. *In Journal on data semantics VIII* (pp. 118-157). Springer Berlin Heidelberg.