Toward Evolution Models for Data Warehouses

Saïd Taktak[1], Jamel Feki[1] and Gilles Zurfluh[2]
[1]*University of Sfax, FSEGS Faculty, P.O. Box 1088, Miracl Laboratory, Sfax, Tunisia*
[2]*University of Toulouse 1 Capitole, IRIT, Toulouse, France*
{said.taktak, jamel.feki}@fsegs.rnu.tn, gilles.zurfluh@ut-capitole.fr

Keywords: Data Warehouse, Evolution Model, MDA, QVT.

Abstract: A Data warehouse (DW) is characterized by a complex architecture, designed in order to integrate data derived from operational data sources (DS), hence providing advanced analytical tools of these data. The DW is highly dependent on its DS. Hence, evolutions of the DS schema need to be propagated to the DW schema and content. This paper presents a model-driven approach for the evolution of a multidimensional DW. It is based on two evolution models: a first evolution model for the DS and another for the DW. These two models concern the data structure aspects as well as the evolution operations. The transition between these two models is performed through specific transformation rules defined in QVT (Query\View\Transformation).

1 INTRODUCTION

In the data warehousing (DW) field, whatever the enterprise's philosophy falls into i) Bill Inmon's camp where the DW is one part of the overall BI system, or into ii) Ralph Kimball's camp where the DW is the conglomerate of all data marts within the enterprise, data issued from the operational systems are extracted, transformed, cleansed and finally loaded into the fact and dimension tables of the famous star schema which represents the keystone modeling diagram that has twofold objectives: first, it highlights the subject of analyses (i.e., fact representing the activity to be evaluated) and, secondly, it shows up the axes (i.e., dimensions) according to which the fact's data could be analyzed (Inmon, 2002).

This strong dependency between the DW and the data source (DS) leads to a new evolution problem that addresses the impact of the DS schema evolution on the DW. In fact, the dynamic evolution of business processes within the enterprise can lead to another evolution of the DS schema. The associated DW cannot escape from this evolution which can simultaneously affect its schema, stored data, and also the ETL process (Extract-Transform-Load) (Vassiliadis, 2009).

This paper treats this evolution problematic, it is organized as follows. In section 2, we overview researches related to the DW evolution problem. Section 3 proposes a model-driven approach for the propagation of DS schema changes towards the DW. Section 4 defines the evolution models of both the DS and the DW. Section 5 presents an example of transformation rules formalized in QVT (Query \View \Transformation); it is for the automatic passage between these two models. Finally, section 6 concludes the paper and enumerates our future perspectives.

2 RELATED RESEARCHES

The DW evolution problem has been the subject of several research studies. It was treated from several points of views: Analytical need evolution, DS schema evolution, etc.

Some researchers (Favre et al., 2007), (Benitez et al., 2004), (Blaschka et al., 1999) have limited their study to the DW evolution as a result of evolution of decision makers needs, without considering the case of the DS evolution. Other literature works (Bellahsene, 2002), (Wrembel and Bebel, 2007), (Solodovnikova, 2008) have examined the evolution of the source schema as well as its impact on the DW. This suits our concern in this paper.

Accordingly, in (Rundensteiner et al., 1999) and (Bellahsene, 2002) the authors consider the DW as a set of materialized views built directly from the data sources. In this approach, any change in the DS

schema requires views maintenance.

As a practical extension, the authors of (Rundensteiner et al., 1999) have developed a prototype to automate the rewriting of the materialized views definitions in order to reflect and to be coherent with the realized structural changes at the layer of the DS. Their EVE (Evolvable View Environment) tool consists of two basic modules: The first module is used to describe the appeared changes in the DS. The second module allows the user to evolve the views via an extended Structured Query Language (SQL) version.

In (Bellahsene, 2002), the author presents an approach for a dynamic adaptation of the materialized views in response to an evolution of the DS. This approach is applied to maintain not only the schema views, but also its instances (i.e., data). The main idea of this contribution is to avoid recalculating the views after each change done on the DS by subtracting the schema of the new view from the old one.

The authors studied the impact of the DS evolution on the DW. Nevertheless, their proposed solutions are only applicable in case the DW is composed of a set of materialized views.

In (Papastefanatos et al., 2009), the authors tackle the inconsistencies that may appear in ETL processes after the DS evolution. They proposed the "HECATAEUS" tool which offers to the designer a mechanism for adapting the ETL activities to the changes happening in the DS schema. Moreover, the tool is able to detect precociously the vulnerable components (i.e., affected) in the Information System (IS). The proposed approach is based on a technical representation which includes all the essential components of the ETL process and also produces a graph evolution model (Simitsis et al., 2005). Following a change in the graph element(s), the tool detects automatically the graph parts that have to be affected and also highlights the changes to be made according to a set of rules a priori defined.

This ensures the consistence of the ETL procedures. However, this study was limited to the ETL feeding process and did not treat the impact of the DS schema evolution on the schema of the DW.

In (Wrembel and Bebel, 2007) and (Solodovnikova, 2008), the authors were interested in studying the effect of the DS evolution on the DW schema.

They adopted an approach based on versioning in order to historize the versions of DW schemas.

In (Wrembel and Bebel, 2007), the authors presented a formal model for a multi-version DW. They identified a set of evolution operations affecting the DW schema and its instances. The authors have distinguished between two types of DW versions: *Real version* and *Alternative version*. The DW *real* version is created in order to reflect the changes in the real environment of the enterprise, whereas the aim of the DW *alternative* version is to ensure the change in the simulation process based on "*What-If*" analyses. In order to validate their approach, the authors have developed a software tool for both the maintenance of the DW and the management of its versions.

In (Solodovnikova, 2008), the author suggested a tool for the DW evolution which ensures the creation and manipulation of several versions as well as the construction and execution of associated reports. He also defined a physical representation of the schema version in the database and a logical representation of the DW, hence classifying the changes which may affect the DW into three categories: Physical, logical and semantic changes.

Solutions proposed by (Solodovnikova, 2008) and (Wrembel and Bebel, 2007) allow the automatic detection of the DS changes and assist the administrator in the propagation of these changes towards the DW. These studies are mainly based on the administrator's expertise and do not propose automatic propagation rules for the DW alterations. For example, when adding a new table/column to the DS schema, the administrator must manually define the potential role of the new table/column in the DW. For instance, (s)he indicates whether the added table could become a dimension, a fact, a measure or a parameter.

Table 1: Comparison of DW evolution approaches.

Criteria / Authors	DW Evolution			Evolution Approach	
	Mat. Views	ETL	DW schema	Classic	MDA
Rundensteiner et al., 1999	✔	-	-	✔	-
Bellahsene, 2002	✔	-	-	✔	-
Wrembel and Bebel, 2007	-	-	✔	✔	-
Solodovnikova 2008	-	-	✔	✔	-
Papastefanatos et al., 2009	-	✔	-	✔	-

Table 1 presents a recap of the approaches studied in this section. These contributions have addressed the DS evolution effect on the DW from different points of views: materialized view evolution, ETL

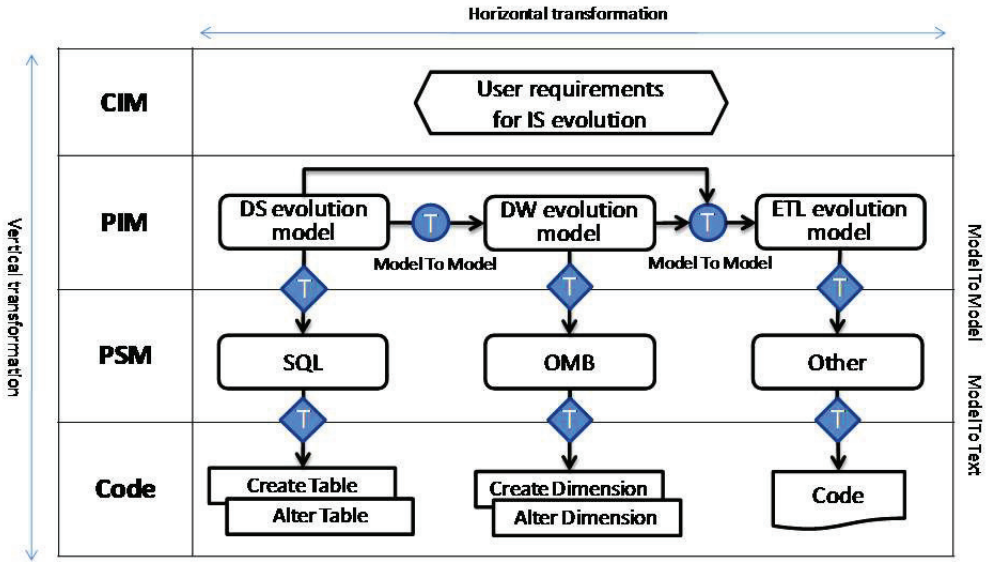

Figure 1: Model-driven approach for DW evolution.

evolution and DW schema evolution. We note that all suggested solutions are realized in a software engineering conventional context; therefore, their implementations are platform-dependent and, thus, it is so hard to be adapted on different platforms.

In our current research, we are addressing the evolution problem of the decision information system by adopting the Model-Driven Architecture (MDA). This choice is motivated by the fact that MDA provides a flexible and effective evolution of the management support. Indeed, the DW evolution process requires less effort when it is managed with a high level of abstraction (i.e. when using models and transformations), hence improving the quality. This is particularly advantageous because MDA provides mechanisms for automatic transformation between models at different levels, unlike the traditional approaches which directly affect the implementation part. Moreover, MDA can provide support for the development, integration, interoperability, scalability, portability and reusability of information systems (Mazon and Trujillo, 2008).

3 PROPOSED APPROACH

We adopt the MDA to automate the extension of the DS schema evolution towards the DW which is loaded from this DS.

The MDA provides an approach to systems development based on models and model transformations in accordance with a set of OMG (Object Management Group) standards (OMG, 2004). This approach separates functional system specifications details and their implementation. In fact, everything in MDA is considered as a model, as well as the schema and source code too.

Figure 1 shows the different steps of our proposed approach where we find three modeling levels and two types of transformations: *Vertical* and *Horizontal*.

The vertical transformation involves different levels of abstraction. It allows the passage from the requirements model (CIM: Computation Independent Model) to the analysis and design model (PIM: Platform Independent Model) and then to model concrete design (PSM: Platform Specific Model) in order to reach the end of a code of impaired DW. The passage between these different levels of models is achieved through transformation rules (Section 5).

- CIM: is a model independent from any computer system. This is the application domain model. It represents the starting point of the DW alteration process which describes the administrator needs in terms of changes to be applied on the decision information system.
- PIM: is a model independent from any technological platform. It defines the structure and behavior of the system without reference to the execution platform. This level constitutes the major part of the proposed approach. It allows the management of changes in the decision information system levels, namely Data Warehousing and ETL.

- PSM: it is a model dependent on technological platforms, and represents a projection of a PIM on a given platform for the generation of corresponding executable code.

The horizontal transformation allows the passage from one or more sources to a target model having the same level of abstraction. Figure 1 shows this type of model transformation at the PIM level.

We distinguish three models of evolution:

- Data source evolution model: This model describes all the evolution operations that may affect a relational DS (table, column...).
- Data warehouse evolution model: It describes all operations that may affect the multidimensional structures (dimensions, facts ...). It should be derived from the DS evolution model.
- ETL evolution model: It describes the ETL applicable evolution operations. It is intended to be derived from the two previous evolution models: DS and DW evolution models.

Each of these models must be conform to a meta-model which must be, in its turn, based on a meta-meta-model (MOF: Meta-model Object Facility) (OMG, 2001).

The transformation rules are defined with the QVT transformation language which is also based on a MOF meta-model (OMG, 2009). These rules allow defining mappings between source and target meta-models and their execution results in the generation of the target model from the source model. Figure 2 shows the source and target evolution models. They will be presented in the next section.

4 EVOLUTION MODELS

In order to represent the three evolution models, we use the UML (Unified Modeling Language) class diagram (OMG, 2001) which is a graphical object-oriented language (Prat et al, 2006).

We use UML in order to model the structural and behavioral aspects of a system through a set of models. Specifically, the class models are used to represent simultaneously these two aspects. Indeed, a UML class diagram is composed of a static property list and a set of operations to describe the evolution models (Figure 2).

On the one hand, we use the property list in order to define schemas of the DS, ETL process and the DW and, on the other hand, the operations to identify the changes which may affect each of these structures.

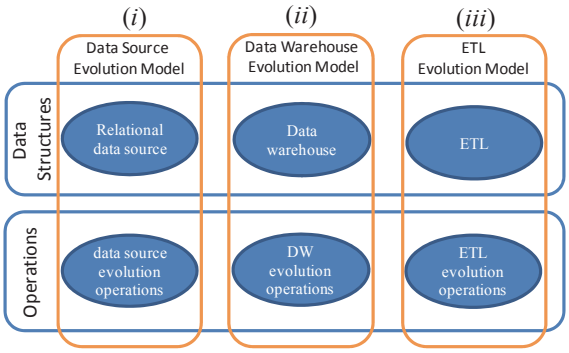

Figure 2: The structure of the evolution models.

Remember that in this paper, we are only interested in studying the impact of the DS evolution model on the DW model. So, we limit ourselves to two evolution models: (*i*) Data Source Evolution Model and (*ii*) Data Warehouse Evolution Model (Figure 2).

The following sub-sections detail these two evolution models.

4.1 Data Source Evolution Model

The DS evolution model is the basic model for the deduction of the ETL and DW evolution models. It defines the relational DS schema (such as tables, constraints...) using the class properties as well as the evolution operations that apply to it (add table, add column...).

The DS evolution model conforms to the meta-model presented in Figure 3. This latter is composed of a *DS_Schema* class which is an extension of the *Package* meta-class. The schema of the DS, in its turn, is composed of several tables. Each one of them is defined by a *Table* class which is an extension of the *Class* meta-class.

Each table contains one or more columns; a column may be a primary key (or part of it) and/or even a foreign key.

The *Column* class is an extension of the *StructuralFeature* meta-class.

The dashed region of Figure 3 models the DS schema. We add operations describing the changes affecting the DS model. These operations (addition, deletion, modification...) mainly concern tables and columns. We modeled them with the *DS_Evolution_Operation* class which represents an extension of the *Operation* meta-class.

A transformation enables the automatic generation of a target model starting from a source model by applying a set of rules. It requires generation of a target model starting from a source

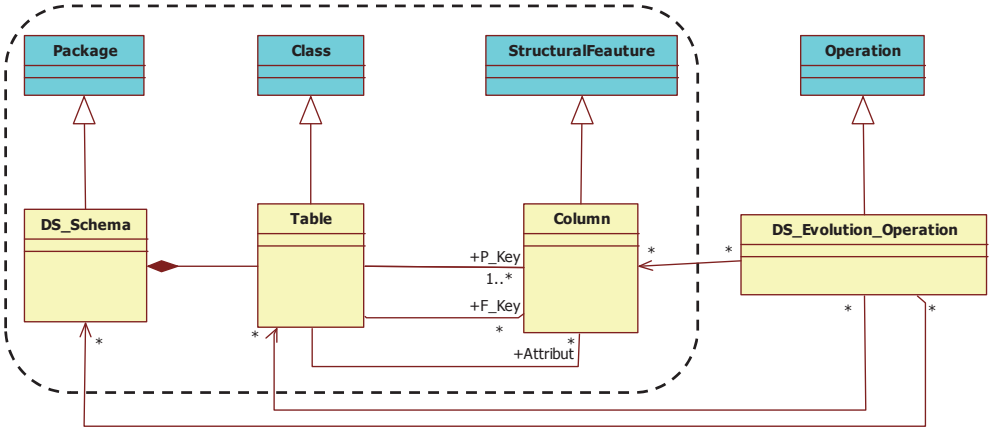

Figure 3: Data Source Evolution Meta-model.

model by applying a set of rules. It requires specifying the meta-models describing these models.

In this section we have defined the source meta-model (DS Evolution Meta-Model). Then we describe the target meta-model (i.e. DW Evolution Meta-Model).

4.2 Data Warehouse Evolution Model

DW Evolution Model conforms to the DW Evolution Meta-model depicted in Figure 4.

The DW Evolution Meta-model has two parts (Figure 4): The first one, in dashed line, describes the DW schema and the second part deals with the DW Evolution Operations.

For the first part, the DW structure is loaded with the DW schema meta-data. As to the second part, the operations of evolution will be deducted automatically from the DS Evolution Model based on the transformation rules.

Figure 4 shows the DW Evolution Meta-model which is composed of the warehouse schema *DW_Schema* (meta-class *Package* extension).
This schema is composed of fact tables *Fact* and dimensions tables *Dimension*. A *Dimension* is composed of one or more hierarchies *Hierarchy* which contains one or more attribute levels *Level*.
Fact, *Dimension*, *Hierarchy* and *Level* are extensions of the meta-class *Class*. The fact table contains one or more measures *Measure*. Each level of hierarchy is composed of parameters *Parameter* which may be associated with one or more weak attributes *Weak_Attribute*.
Measure, *Parameter* and *Weak_Attribute* inherit from the class *Attribute* which is an extension of the meta-class *StructuralFeature*.

Evolution operations of the DW level can be applied to the various DW model components (fact table, dimensions hierarchies, parameters and weak attributes).

These operations can be constructive (add dimension, add measure, etc.) or destructive (delete dimension, delete hierarchy, etc.). We model these operations through the class *DW_Evolution _Operation* which is an extension of the meta-class *Operation*.

In this section, we have defined the DS and target evolution meta-models. In the next section, we present the transformation rules.

5 MODEL TRANSFORMATION

This section presents the QVT formalization of the transformation rules allowing the automatic passage between the source and warehouse evolution models. These transformations mainly relate to evolution operations. Each operation in the DS evolution model can be transformed automatically into one or more evolution operations in the target model (i.e. DW Evolution Model).To propagate a DS evolution operation, first we need to determine the modified element (table, column, etc), its corresponding element in the DW schema (dimension, fact, parameter, etc) as well as the evolution operation (add table/column, alter column, etc). All these details are traceable from the two evolution models described in Figures 3 and 4.

Table 2 lists the possible transformations applicable to the DW after adding a new table or column to the DS schema. For example, the evolution operation *AddTable* could be transformed into *AddDimension* operation in the DW evolution model when the rule *TableToDim* is applied.

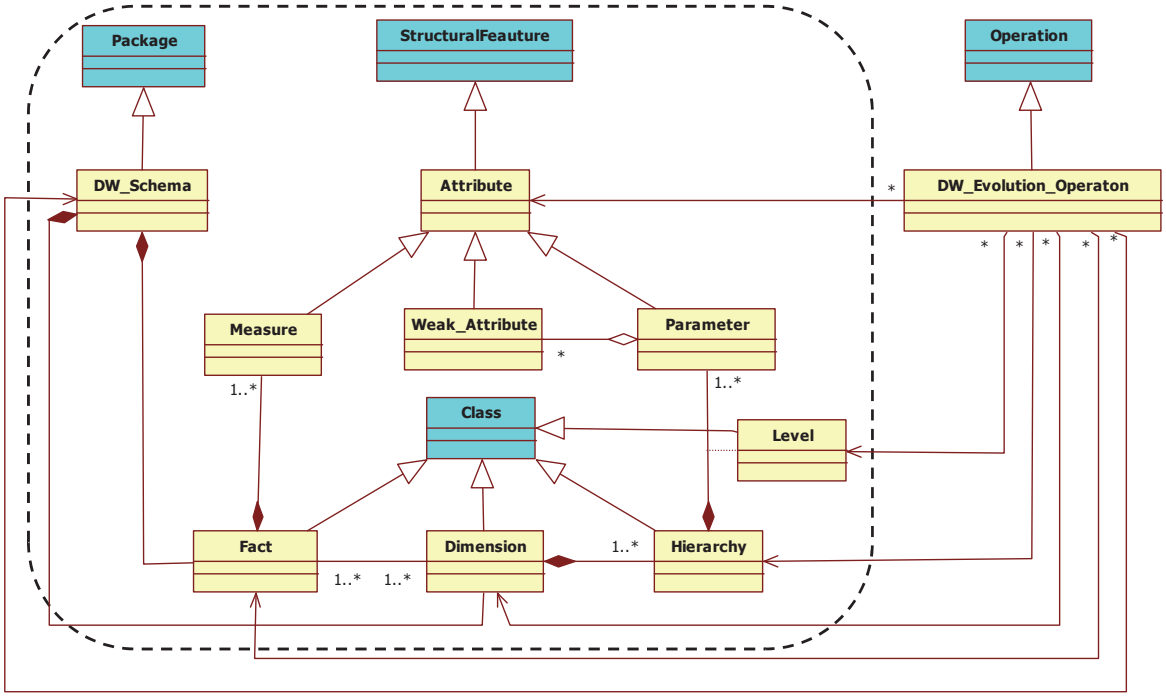

Figure 4: Data Warehouse Evolution Meta-Model.

```
Transformation semTOwem
(sem : SourceEvolutionMetaModel,
 wem : WarehouseEvolutionMetaModel)
        {
                Relation TableToDim   {…}
                Relation TableToFact  {…}
                Relation TableToHier  {…}
                Relation ColToMeas    {…}
                Relation ColToLev     {…}
                Relation ColToPara    {…}
                …
        }
```

Figure 5: Description of the transformation *semTOwem*.

Figure 5 shows a textual description of the transformation *semTOwem* which takes as input the two models *sem* and *wem*.

sem is an instance conform to the Source Evolution Meta-model (Figure 3). *wem* is an instance conform to the Warehouse Evolution Meta-model (Figure 4).

The transformation *semTOwem* contains a set of relations (*TableToDim*, *TableToFact*, *ColToMeas* ...) which must be verified on the candidate models (*sem*, *wem*) for the achievement of the transformation.

As an example, the relation *ColToMeas* transforms the evolution operation *AddColumn* into *AddMeasure* (Figure 6). This transformation is defined by:

- Two domains such as *DS_Evolution_Operation* (*dseo*) and *DW_Evolution_Operation* (*dweo*) which must be matched through the elements belonging to them.
- A *when* clause specifies the relation condition. For this example, the relation *ColToMeas* is only applicable when: (1) the DS evolution operation is *AddColumn*, and (2) the added column is numeric and (3) the modified table loads a fact.

We illustrate the *ColToMeas* relation on the relational data source of Figure 7 and its DW of Figure 8.

The addition of a column (*Reduction_Rate*, *Number*) to the table *SALE* of the DS to express the reduction rate of each transaction is translated into an *AddColumn ('Reduction_Rate', Number)* evolution operation affecting the class *SALE* of the DS evolution model. In this evolution operation:

(1) The column is added using the *AddColumn* operation,

(2) The column is added using the *AddColumn* operation,

(3) the type of column *Reduction_Rate* is numeric, and

Table 2: Example for the evolution operation (Add) and transformation rules.

Evolution Operation	Data Source (DS)	Relations (R)	Data Warehouse (DW)						
			Fact	Dimension	Hierarchy	Level	Measure	Weak_Att	Parameter
Add	Table	TableToFact	✔						
		TableToDim		✔					
		TableToHier			✔				
		TableToLev				✔			
	Column	ColToHier			✔				
		ColToLev				✔			
		ColToMeas					✔		
		ColToWatt						✔	
		ColToPara							✔

```
Relation ColToMeas
{
   Domain sem dseo : DS_Evolution_Operation
   {
      DS_Schema = ds : DS_Schema{ },
      table = t : table
                  {name = tn,
                  column= cl:Column{ }},
      type ='AddColumn',
      Parameter = Ps : Parameter
                       {c_name = cn,
                        c_type = ct}
   }
   Domain wem dweo : DW_Evolution_Operation
   {
      DW_Schema = ds : DW_Schema{ },
      fact = f : fact { },
      type ='AddMeasure',
      Parameter = Pw : Parameter
                       {m_name = 'M_'+cn,
                        m_type = ct }
   }
   When
   {
      dseo.type = 'AddColumn'
      and ct='Number'
      and Fact ( t ) = f      /*returns a fact
                     loadable from a table t */
      }
}
```

Figure 6: Description of the relation *ColToMeas*.

(4) the table *SALE* of the DS loads the *SALE* fact table of the DW.

According to the *when* clause (Figure 6), the condition of the relation *ColToMeas* is satisfied. Therefore, the operation *AddColumn (' Reduction_Rate', Number)* is transformed into an evolution operation applied to the fact table *AddMeasure (M_Reduction_Rate, number)*.

```
CUSTOMER (Id_Cust, First_Name, Last_Name, #Id_City…)
CITY (Id_City, City_Name…,#Id_Cntry)
COUNTRY (Id_Cntry, Country_Name…)
SALE (Id_Sale, Date, #Id_Cust, Sale_Amount)
SALE_ITEM (#Id_Sale, #Id_Prod, Sold_Qtity)
PRODUCT (Id_Prod, PName, Unit_Price, #Id_Categ)
CATEGORIE (Id_Categ, CName)
```

Figure 7: Relational data source schema.

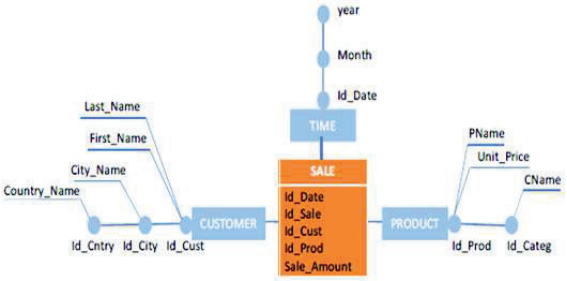

Figure 8: DW Star schema SALE built on the DS schema of Figure 7.

6 CONCLUSIONS

In this paper we have discussed the problem of the data sources evolution and then studied its impact on a multidimensional data warehouse.

To address this problem, we proposed a model-driven approach in order to automate the propagation of the data source schema evolution toward the multidimensional data warehouse. This approach is based on two evolution models presenting simultaneously the structural and

behavioral aspects of data in terms of the evolution operations. The passage between these two models is achieved through a set of transformation rules. In this paper we have illustrated one rule and defined it for the case of adding a new column to the DS; this column transforms into a measure in the multidimensional DW. The remaining rules exist in (Taktak and Feki, 2012) as textual description but they are not yet coded as QVT (Query\View\Transformation) transformations.

We have started the step of testing the set of rules on samples DS/DW schemas (Azaiez et al, 2013). In order to experimentally validate the proposed models and all transformations, we are looking to apply them on a real case study.

In the future research, we have the intention to extend the DW alteration process to take into account expected evolutions of the ETL (Extract Transform and Load procedures.

REFERENCES

Azaiez, N., Taktak, S., Feki, J., 20113. DWEv : Un prototype pour l'évolution partielle du schéma multidimensionnel. *7éme édition de la Conférence Maghrébine sur les Avancées des Systèmes Décisionnels (ASD 2013)*, Marrakech, Maroc, pages 457-462.

Bebel, B., Eder,J., Koncilia, C., Morzy, T., Wrembel, R., 2004. Creation and Management ofVersions in Multiversion Data Warehouse.In *XIXth ACM Symposium on Applied Computing (SAC)*, Nicosia, Cyprus, pages 717–723.

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

Benitez-Guerrero, E.I., Collet, C., Adiba, M., 2004. The Whes Approach To Data Warehouse Evolution, *e-Gnosis (online)*, 2, Art. 11.

Blaschka, M., Sapia, C., and Hofling, G., 1999. On Schema Evolution in Multi-dimensional Databases.In *Ist International Conference on Data Ware-housing and Knowledge Discovery(DaWaK)*, Florence, Italy, volume1676 of LNCS, pages 153–164.

Favre, C., Bentayeb, F., Boussaid, O., 2007. A Survey of Data Warehouse Model Evolution, *Encyclopedia of Database Technologies and Applications*, Second Edition, Idea Group Publishing.

Inmon, W., 2002. *Building the Data Warehouse* (3rd Edition). New York. Wiley & Sons.

Mazon, J.N., Trujillo, J. 2008. An MDA approach for the development of data warehouses. *Decision Support Systems* (DSS'08), Vol. 45 (1), pages 41-58.

OMG, 2001. Object Management Group: Unified Modeling Language Specification 1.4. *http://www.omg.org/cgi-bin/doc?formal/01-09-67.*

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

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

Papastefanatos, G., Vassiliadis, P., Simitsis, A., Sellis, T., Vassiliou, Y., 2009. Rulebased Managementof Schema Changes at ETL Sources. In *The International Workshop on ManagingEvolution of Data Warehouses (MEDWa)*, Riga, Latvia.

Prat, N., Akoka, J., Comyn-Wattiau. I., 2006. A UML-based data warehouse design method, *Decision Support Systems (DSS"06)*, Vol. 42(3), pages 1449-1473.

Rundensteiner, E. A., Koeller, A., Zhang, X., Lee, A.J., Nica, A., 1999. Evolvable View Environment EVE: A Data Warehouse System Handling Schema and Data Changes of Distributed Sources. *The International Database Engineering and ApplicationSympo-sium (IDEAS'99)*, Montreal, Canada.

Solodovnikova, D., 2008. The Formal Model for Multiversion Data Warehouse Evolution. Postconference proceedings of *the 8th International Baltic Conference on Databases and Information Systems*, Tallinn, Estonia, Frontiers in Artificial Intelligence and Applicationsby IOS Press, pages 91-102.

Taktak, S., Feki, J., 2012. Toward Propagating the Evolution of Data Warehouse on Data Marts, *2nd International Conference on Model & Data Engineering (MEDI'2012)*, Poitiers, France, pages 178-185.

Vassiliadis, P., 2009. A survey of Extract-transform-Load technology. *International Journal of Data Warehousing & Mining (IJDWM'09)*, Vol. 5(3), pages. 1-27.

Wrembel, R., Bebel, B., 2007. Metadata management in a multiversion data warehouse.In *Journal on data semantics VIII*.Lecture Notes In Computer Science, Vol. 4380. Springer-Verlag, Berlin, Heidelberg pages 118-157.