QVT transformation by modeling

From UML Model to MD Model

I.Arrassen
Laboratory for Computer Science Research
Faculty of Sciences
Mohammed First University
524, Oujda, Morocco

A.Meziane
Laboratory for Computer Science Research
Faculty of Sciences
Mohammed First University
524, Oujda, Morocco

R.Sbai
Laboratory of Applied Mathematics and Computer Signal Processing
Superior School of Technology
Mohammed First University
524, Oujda, Morocco

M.Erramdani
Laboratory of Applied Mathematics and Computer Signal Processing
Superior School of Technology
Mohammed First University
524, Oujda, Morocco

***Abstract* – To provide a complete analysis of the organization, its business and its needs, it is necessary for leaders to have data that help decision making. Data warehouses are designed to meet such needs; they are an analysis and data management technology. This article describes an MDA (Model Driven Architecture) process that we have used to automatically generate the multidimensional schema of data warehouse. This process uses model transformation using several standards such as Unified Modeling Language, Meta-Object Facility, Query View Transformation, Object Constraint Language, ... From the UML model, especially the class diagram, a multidimensional model is generated as an XML file, the transformation is carried out by the QVT (Query View Transformation) language and the OCL (Object Constraint Language) Language. To validate our approach a case study is presented at the end of this work.**

Key-Words: Datawarehouse; Model Driven Architecture; Multidimensional Modeling; Meta Model; Transformation rules; Query View Transformation.

I. INTRODUCTION

To support the process of making management decisions, development of data warehouses is important for organizations. According to the definition given by Bill Inmon (1996), data warehouse is a collection of data that is subject-oriented, integrated, time-varying and non-volatile. His ultimate goal is integrating data from all corners of the enterprise in a single directory, from which users can easily find answers to queries, generate reports and perform analysis.

A data warehouse is a management and data analysis technology. On this basis, the establishment of a process of building data warehouse is very important. Through this work, we use a UML class diagram summarizing the activities: requirements expression, analysis and design of information system of the organization. From this diagram, we will generate future objects decision diagram such as facts and dimensions. The decisional diagram will be in the form of a multidimensional schema, in fact, multidimensional modeling is the one that best represents the data warehouse schema.

The approach used in this work is the MDA. A models transformation process is used to transform a UML model (Class Diagram) into a multidimensional model, the QVT and the OCL languages was chosen as implementation language processing.

In Section 2, we will discuss the works that are related to our theme. In section 3 we explain the concepts of multidimensional modeling. In Section 4 we outline the main concepts of the MDA architecture (Model Driven Architecture), which is at the heart of the approach followed in our work. Then in Section 5 we present the source and target Meta models used in the transformation program. In Section 6, we present the work of generating the multidimensional model. A case study is presented in Section 7. We conclude this work in Section 8, with suggestions and possible extensions.

II. RELATED WORKS

In recent years, several approaches to developing data warehouse have been proposed. In this section, we present a brief overview of some best known approaches.

In [1] several DWH case studies are presented. DWH design is based on using the star schema and its variants (snowflake schema and fact in constellation) using a relational approach: tables, columns, foreign keys and so on. However this work is considered as a benchmark in the field of DWH, the authors are only interested in the representation of relational DWH, and they regarded any other technology.

In [2], the authors propose a model Fact-Dimension (Dimensional-Fact Model DFM), in which they define a special notation for the conceptual model of the DWH. They also show how to derive a DWH schema from the data sources described by the entity-relationship diagram.

A goal-oriented approach has been added to DFM in [3]. This approach does not consider important aspects such as the ETL (Extract Transform Load) process. In addition, the authors consider that relational schemas of data exist, which is

not always true, in addition, the use of special notations makes it difficult to apply this approach.

In [4], a model for the design of the DWH was proposed: YAM2. It is an object oriented model that uses UML notation to represent the multidimensional structure of data. However, no method that shows how to get the conceptual model is described in the paper.

In [5] the authors propose a multidimensional meta-model extended by generalizing the model heart, based on medical data, this work was done under the MAP project (Personalized Medicine Anticipation). In our opinion this type model is specific to medical data; in addition, the authors do not specify the modeling approach.

In [6] the authors created a decision support system called BIRD integrated data grid Decrypthon which is designed to process data in functional genomics. This system is a specific platform that does not meet all requirements of any DWH.

III. MULTIDIMENSIONAL MODELING

It is a logical design technique that aims to present data in a standard, intuitive way, which allows high performance access.

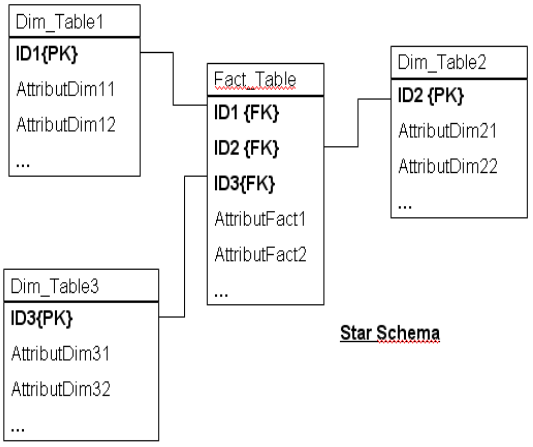

Figure 1. Dimensionnel Model Schema

IV. MDA ARCHITECTURE

In November 2000, the OMG, a consortium of over 1,000 companies, initiates the MDA approach [7]. The purpose of this standard is to separate the business logic of the enterprise, from any technical platform. It is a new way to design applications. Indeed, the technical architecture is unstable and undergoes many changes over time, unlike the business logic. It is therefore easy to separate the two to face the increasing complexity of information systems and high costs of technology migration. This separation then allows the capitalization of software knowledge and the know-how of the company.

Figure 2 below shows schematically the architecture of MDA [7]. It is divided into four layers. The OMG was based on several standards. In the center are the standard UML (Unified Modeling Language) [8], MOF (Meta-Object Facility) [9] and CWM (Common Warehouse Meta-model) [10].

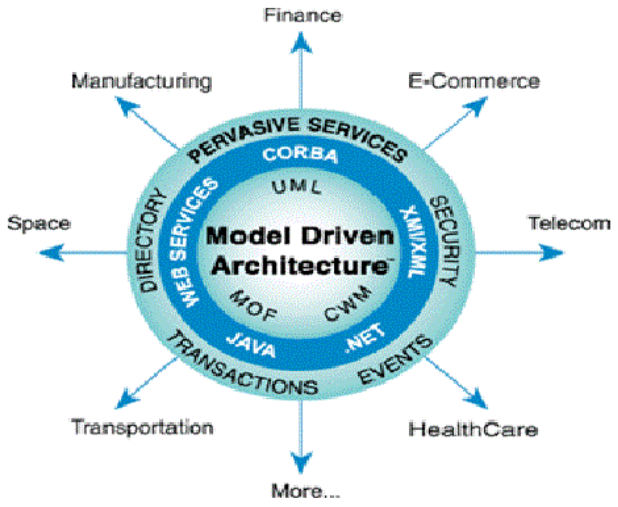

Figure 2. MDA Architecture [7]

In the next layer, it is also a standard XMI (XML Metadata Interchange), which allows communication between the middlewares (Java, CORBA, .NET and Web Services). The third layer contains the services that manage events, security, directories, and transactions. The final layer offers specific frameworks in fields (Finance, Telecommunications, Transportation, Space, Medicine, Commerce, Manufacturing ...)

A. CIM (Computation Independent Model)

CIM stands for Computation Independent Model [12]. In UML, a requirements model can be summarized as a use case diagram. Because they contain the functionality provided by the application and the various entities that interact with them (actors) without providing information on the operation of the application. The role of requirements models in an MDA approach is the first models to be perennial. The modeled requirements provide a contractual basis validated by the customer and vary little.

B. PIM (Platform Independent Model)

The PIM represents the business logic specific to the system or the design model. It represents the operating entities and services. It must be perennial and last over time. It describes the system, but shows no details of its use on the platform. At this level, the formalism used to express a PIM is a class diagram in UML, which can be coupled with a constraint language like OCL (Object Constraint Language). Analysis models and design are independent of platforms where they are implemented J2EE, .NET, PHP, etc.. [12].

C. PSM (Platform Specific Model)

The PSM is the work of code generation after performing the analysis models and design. This phase, the most delicate of MDA, must also use templates. It includes the application of design patterns techniques [12]. PSM code models facilitate the generation of code from a model analysis and design. They contain all information necessary to operate an execution platform, such as information systems to manipulate file systems or authentification systems.

D. OCL (Object Constraint Language)

OCL [11] was developed in 1997 by Jos Warmer (IBM), on the basis of language IBEL (Integrated Business Engineering Language). It was formally incorporated into UML 1.1 in 1999. It is a formal language that is based on the notion of constraint. A constraint is a boolean expression that can be attached to any UML element. It usually indicates a restriction or gives information on a model. The constraints are used in particular to describe the semantics of UML and its various extensions, participating in the definition of profiles.

E. MOF2.0 (MétaObject Facility) QVT

MOF [9] defines the structure that should have any meta-model. A meta-model is a set of meta-classes with meta-associations. MOF2.0 meta-meta-model is unique, and UML 2.0 meta-model is dedicated to the modeling of object-oriented applications. Among the objectives of MOF2.0: capitalize on existing commonalities between UML and MOF-level class diagrams and to explain the differences. The advantage of the MOF is to structure all meta-models in the same way.

QVT (Query / View / Transformation) in the MDA architecture is a standard for model transformations defined by the OMG (Object Management Group) in 2007. It is central to any proposed MDA. It defines a way to transform source models to target models. These source and target models must conform to the MOF meta-model. Specifically, this means that the abstract syntax of QVT must conform to the MOF 2.0 meta-model. QVT defines three specific languages named: Relations, Core Operational/Mapping. These languages are organized in a layered architecture. Relations and Core are declarative languages to two different levels of abstraction. QVT Core is to relational, what is the Java byte code to Java source code. The QVT Operational / Mapping is an imperative language, it provides common constructions in imperative languages (loops, conditions ...).

F. Kinds of MDA transformation.

The MDA identifies several transformations during the development cycle [12]. It is possible to make four different types of transformations:

1) CIM to CIM
2) CIM to PIM
3) PIM to PIM
4) PIM to PSM

There are three approaches in MDA to perform these transformations:

- Approach by programming: using the object-oriented programming languages such as Java, to write computer programs that are unique to manipulate models. This approach was used in [13] and [14].The authors automatically generate a web application from a simplified class diagram.
- Approach by template: Consists of taking a "template model", canvas of configured target models, these settings will be replaced by the information contained in the source model. This approach requires a special language for defining model template.
- Approach by Modeling: The objective is to model the transformations of models, and make transformation models sustainable and productive and to express their independence of execution platforms. The standard MOF2.0 QVT [9] is used to define the meta model for the development of models for the models transformations.

In this paper we chose to transform PIM to PSM, with an approach by modeling. This type of transformation will allow us to automatically generate the multidimensional data warehouse schema from UML schema. Indeed, as shown in Figure 2, the definition of model transformation is a model structured according to the meta-model MOF2.0 QVT. Models instances of meta-model MOF2.0 QVT express structural mapping rules between meta-model source and meta-model target of transformation. This model is a sustainable and productive, it must be transformed to allow the execution of processing on an execution platform. The following figure illustrates the approach by modeling MOF2.0 QVT [12].

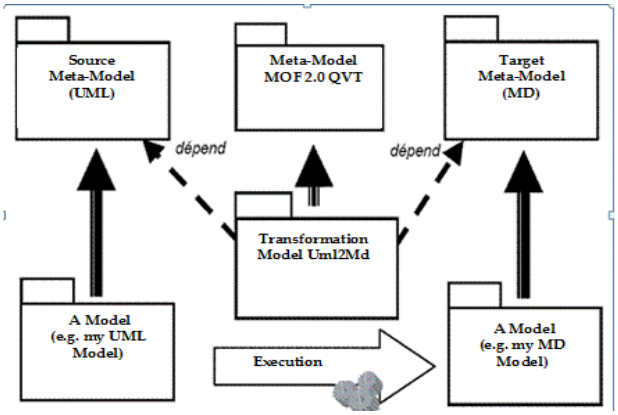

Figure 3. Modeling approach (MOF2.0 QVT)

V. Source Meta Model - Target Meta Model

In our MDA approach, we opted for the modeling approach to generate the data warehouse multidimensional schema. As mentioned above, this approach requires a source meta-model and a target meta-model.

A. Source Meta-Model

We present in this section the different meta-classes which form the source UML meta model used in [13], and our target meta-model MD to achieve the transformation between the source and target model. This transformation is based on that proposed by [13]. The source meta-model is a simplified UML model based on packages containing Class and Datatype. These classes contain typed properties and are characterized by multiplicities (upper and lower). The classes contain operations with typed parameters. The following figure illustrates the source meta-model:

- UmlPackage: expresses the notion of UML package. This meta-class is related to the meta-class Classifier.

- Classifier: it is a meta abstract class that represents both the concept of UML class and the concept of data type.
- Class: represents the concept of UML class.
- DataType: represents the data type of UML.
- Operation: expresses the concept of operations of a UML class
- Parameter: parameter expresses the concept of an operation. They can be of two types or Class Datatype. This explains the link between meta-Parameter class and meta-class Classifier.
- Property: expresses the concept of Property of a UML class. These properties are represented by the multiplicities and meta-attributes upper and lower. A UML class is composed of properties, which explains the link between the meta-class Property and meta-class Class. These properties can be primitive type or class type. This explains the link between the meta-class and the Classifier meta-class Property.

B. *Target Meta-Model*

The target is a simplified dimensional meta-model based on packages containing Facts and Dimensions. A Fact contains attributes related to dimensions. Figure 5 shows the Target Meta Model:

- FactPackage: expresses the concept of package and we consider that each class contains a FactPackage.
- Fact: expresses the concept of Fact in a multidimensional schema. Each fact is contained in FactPackage, which explains the link between FactPackage and Fact.
- FactAttribute: expresses the concept of attributes for the element Fact and Fact contains FactAttribute
- Dimension: expresses the concept of dimension in a multidimensional schema.
- DimensionAttribute: expresses the concept of dimensions attributes. A dimension contains attributes, which explains the link between a DimensionAttribute and Dimension.
- Datatype: represents the type of data in a multidimensional schema.

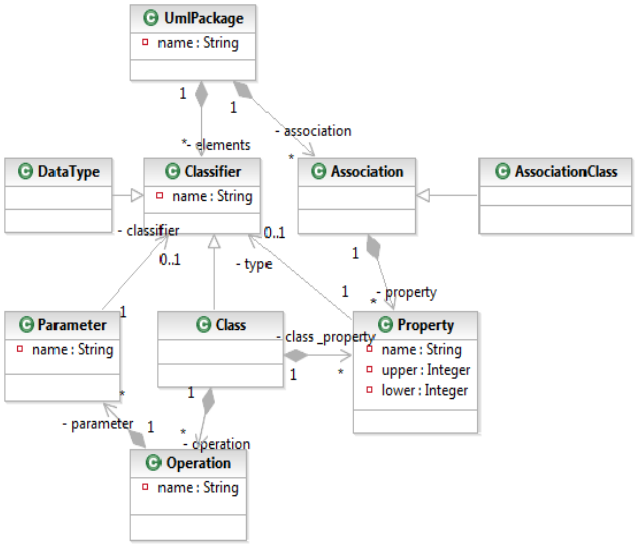

Figure 4. Simplified Meta-Model class diagram of UML.

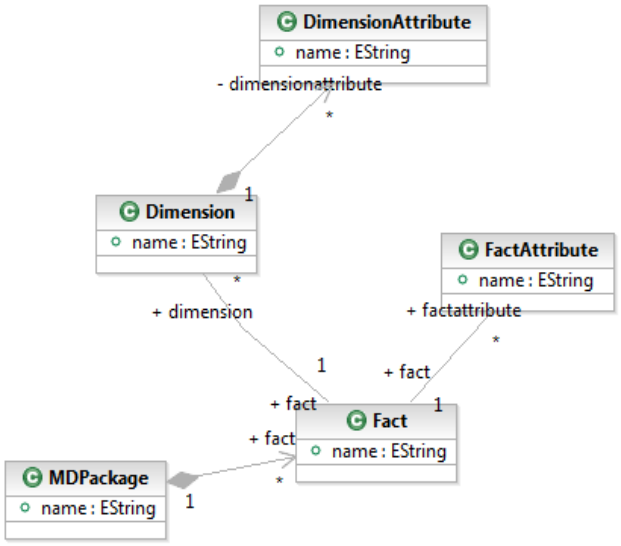

Figure 5. Simplified Meta-Model of multidimensional star schema.

VI. QVT TRANSFORMATION OF THE UML META MODEL TO THE MULTIDIMENSIONAL META MODEL.

A. *SmartQvt*

To achieve the transformation cited above, we opted for the tool SmartQvt. This tool appeared in 2007 [17] which is also an implementation of the QVT-Operational language standard that allows the transformation of models. This tool compiles model transformations specified in QVT to produce Java code used to performs these transformations. This tool is provided as Eclipse plug-ins based on the meta-modeling framework, EMF, and is composed of three elements:

- QVT Editor: helps end users to write QVT specifications.
- QVT Parser: converts the textual concrete syntax on its corresponding representation in terms of meta-model QVT.
- QVT Compiler: produces, from the QVT model, a Java API on top of EMF generated for the implementation of transformations. The input format is a QVT specification provides in XMI 2.0 format in accordance with the QVT meta-model.

B. The QVT transformation

The QVT transformation is a function of two arguments: the first is the source model, the second is the target model as shown in the figure below:

```
transformation uml2md(in srcModel:UML,out dest:MD);
main(){
srcModel.objects()[UmlPackage] ->
map UmlPackage2MDPackage(); }
```

Figure 6. Transformation function uml2md()

In this figure, we see the transformation function called uml2md() that takes as input source model, the simplified UML model shown in Figure 4, and as output the target model, the simplified multidimensional model shown in Figure 5.

As explained in the previous section, we consider a simplified UML model that consists of a package called UmlPackage. This package will be transformed into a multidimensional package called MDPackage, using the transformation rule sets in Figure 7. The MDPackage generated will be named MD followed by with the name of UmlPackage. The Association class will be transformed into a Fact by using the function associationclass2fait ().

```
mapping UmlPackage::UmlPackage2MDPackage () : MDPackage { name := 'MD' + self.name;
fact:=srcModel.objects()[AssociationClass]->map associationclass2fact();}
```

Figure 7. Transformation rule of UMLPackage to MDPackage

```
mapping AssociationClass::associationclass2fact () : Fact {
name := 'Fait'+self.name;
factattribute:=self.assoproperty->map property2factattribute();
dimension= self.assoproperty->select (upper=-1)->type->map type2dimension(); }
```

Figure 8. Transformation rule of Associationclass to a Fact

Figure 8 shows the transformation rule of an association class in a UML class diagram to a Fact.

```
mapping Classifier::type2dimension () : Dimension {
srcModel.objects()[Class]->forEach(c){
if(self=c){ name:='Dimension'+self.name ;
base:= c._property->select (upper=-1)->type-> map type2base(); } } }
```

Figure 9. Transformation rule of a Classifier to a Dimension

Figure 9 shows the transformation rule of Classifier to an element Dimension of Dimensional Model.

```
mapping Property::property2factattribute(): FactAttribute
{ name:=self.name; }
```

Figure 10. Transformation rule of Property to FactAttribut

Figure 10 shows the transformation rule of UML class property to a Fact entity attribute, this rule uses the function property2factattribute ().

```
mapping Parameter::property2factattribute () : FactAttribute {
name:=self.name; }
```

Figure 11. Transformation rule of Parameter to FactAttribut

```
mapping Property::property2dimensionAttribute () : DimensionAttribute
{ name:=self.name;}
```

Figure 12. Transformation rule of Property to DimensionAttribut

VII. CASE STUDY

In our case study, we use the UML model representing a business, among its activities, the sale of products to its customers, these products are classified under categories, and several sub categories form a category. The UML model representing part of the Information System of the company is represented in a format file XMI2.0. This file is composed of the element UmlPackage, which contains the following elements:

```
<?xml version="1.0" encoding="ASCII"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:UmlMdMM="http:///UmlMdMM.ecore">
<UmlMdMM:UmlPackage name="Factory">
<elements xsi:type="UmlMdMM:DataType" name="String"/>
<elements xsi:type="UmlMdMM:Class" name="Custommer">
<_property upper="1" lower="1" name="CustomerID"
type="//@elements.0"/>
<_property upper="1" lower="1" name="AccountNumber"
type="//@elements.0"/>
<_property upper="1" lower="1" name="CustomerType"
type="//@elements.0"/>
</elements>
<elements xsi:type="UmlMdMM:Class" name="Product">
<_property upper="1" lower="1" name="ProductID"
type="//@elements.0"/>
<_property upper="1" lower="1" name="Name"
type="//@elements.0"/>
<_property upper="1" lower="1" name="StandardCost"
type="//@elements.0"/>
<_property upper="1" lower="1" name="ListPrice"
type="//@elements.0"/>
<_property upper="1" lower="1" name="Weight"
type="//@elements.0"/>
<_property upper="1" lower="1" name="Style"
type="//@elements.0"/>
<_property upper="1" lower="1" name="Color"
type="//@elements.0"/>
<_property upper="1" lower="1" name="DayToManufacture"
type="//@elements.0"/>
</elements>
 <association xsi:type="UmlMdMM:AssociationClass"
name="SalesOrder">
 <assoproperty upper="1" lower="1" name="SalesOrderID"
type="//@elements.0"/>
 <assoproperty upper="1" lower="1" name="OrderQty"
type="//@elements.0"/>
<assoproperty upper="1" lower="1" name="UnitPrice"
type="//@elements.0"/>
<assoproperty upper="1" lower="1" name="OrderDate"
type="//@elements.0"/>
 <assoproperty upper="-1" lower="1" name="CustomerID"
type="//@elements.1"/>
<assoproperty upper="-1" lower="1" name="ProductID"
type="//@elements.3"/>
  </association>
  </UmlMdMM:UmlPackage>
</xmi:XMI>
```

Figure 13. Overview of the instance of the UML model of our case study

An element named Customer, type of class, which has the following properties:

- CustomerID : Custormer Identifier.
- CustomerName : Customer Name.
- AccountNumber:Numbre of the Customer Account.
- Customer Type: type of Customer, I=Individual, S=Store.
- AddressID: the Adress Identifier.

An element named Address, type of class, which has the following properties:

- AddressID: Identifier of customer address.
- AdressLine1: Line 1 of the address.
- AdressLine2: Line 2 of the address.
- City: defines the city of customer.
- State: defines the state of customer.
- PostalCode: defines the postal code.

An element named Product, type of class, which has the following properties:

- ProductID: Product Identifier.
- ProductName: Product Name.
- StandardCost: Product Standard Cost.
- ListPrice : sale price
- Weight : Product Weight
- Style : W=Women, M=Male, U=Both
- Color: Product Color.
- DayToManufacture: Number of days to manufacture the product.
- ProductSubcategoryID : he product belongs to this sub category.

An element named ProductSubcategory, type of class, which has the following properties:

- ProductSubcategoryID: Product Sub category Identifier.
- ProductSubcategoryName: Product Sub category Name.
- ProductCategoryID: the sub category belongs to this category.

An element named ProductCategory, type of class, which has the following properties:

- ProductCategoryID: Product Category Identifier.
- ProductCategoryName : Product Category Name.

An element named SalesOrderDetail, type of class, which has the following properties:

- SalesOrderDetailID: Identifier of sales order.
- OrderQty: Quantity of Sales.
- UnitPrice: the selling price of a single product.
- OrderDate: Creation date of the sales order.
- CustomerID: Customer Identifier.
- ProductID: Product Identifier.

This file in XMI format is provided as input of QVT transformation, achieved under Eclipse Europa. As output of the QVT transformation, file in XMI format is generated

automatically. It represents the multidimensional schema of company Datawarehouse in our case study. This file is composed of a Fact named FactSalesOrderDetail surrounded by Dimensions: Customer, Adress, Product, Category, SubCategory.

The generated XMI file contains the element MDPackage composed of the following:

```
<?xml version="1.0" encoding="ASCII"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:BdMdMM="http:///BdMdMM.ecore">
 <BdMdMM:MDPackage name="MDFactory">
  <fact name="FactSalesOrder" factattribute="/1 /2 /3 /4 /5 /6"/>
 </BdMdMM:MDPackage>
 <BdMdMM:FactAttributename="SalesOrderID" fact="/0/@fact.0"/>
 <BdMdMM:FactAttribute name="OrderQty" fact="/0/@fact.0"/>
 <BdMdMM:FactAttribute name="UnitPrice" fact="/0/@fact.0"/>
 <BdMdMM:FactAttribute name="OrderDate" fact="/0/@fact.0"/>
 <BdMdMM:FactAttribute name="CustomerID" fact="/0/@fact.0"/>
 <BdMdMM:FactAttribute name="ProductID" fact="/0/@fact.0"/>
 <BdMdMM:Dimension name="DimensionCustommer">
  <dimensionattribute name="CustomerID"/>
  <dimensionattribute name="AccountNumber"/>
  <dimensionattribute name="CustomerType"/>
  <dimensionattribute name="AddressID"/>
 </BdMdMM:Dimension>
 <BdMdMM:Dimension name="DimensionProduct">
  <dimensionattribute name="ProductID"/>
  <dimensionattribute name="Name"/>
  <dimensionattribute name="StandardCost"/>
  <dimensionattribute name="ListPrice"/>
  <dimensionattribute name="Weight"/>
  <dimensionattribute name="Style"/>
  <dimensionattribute name="Color"/>
  <dimensionattribute name="DayToManufacture"/>
  <dimensionattribute name="ProductSubcategoryID"/>
 </BdMdMM:Dimension>
</xmi:XMI>
```

Figure 14. Overview of the instance of Multidimensional Model generated after execution of the transformation

VIII. CONCLUSION AND FUTURE WORKS

We applied the MDA approach for the engineering of SID. The objective is to define a model transformation. It is based on a UML source model and a Multidimensional target model. This transformation takes as input the source model and as output the target model. Once done, we can generate from a simplified UML model instance, a simplified multidimensional model.

Looking ahead, we plan to develop a graphical plug-in integrated to Eclipse, which automatically generates from a UML class diagram, a multidimensional diagram. In addition, we can extend this work to more generalized meta-models.

REFERENCES

[1] R. Kimball The Data Warehouse Toolkit, Second Edition, Wiley Computer Publishing 2002.

[2] M. Golfarelli, D. Maio, S. Rizzi, The dimensional fact model: a conceptual model for data warehouses, International Journal of Cooperative Information Systems 7 (2-3) (1998) 215–247.

[3] P. Giorgini, S. Rizzi, M. Garzetti, Goal-oriented requirement analysis for data warehouse design, DOLAP, 2005, pp. 47–56

[4] A. Abelló, J. Samos, F. Saltor, YAM2: a multidimensional conceptual model extending UML, Information Systems 31 (6) (September 2006) 541–567

[5] Dj. Midouni, J.Darmont, F.Bentayeb, Approach to modeling complex multidimensional data: Application to medical data, « 5ème journée francophones sur les entrepôts de données et l'analyse en ligne » Montpellier, France (EDA 2009).

[6] A. Nguyen, A. Friedrich, G. Berthommier, L. Poidevin, L. Moulinier, R. Ripp,O. Poch, Introduction du nouveau Centre de Données Biomédicales Décrypthon, CORIA - Conférence en Recherche d'Information et Applications-, 2008

[7] Object Management Group (OMG), MDA Guide 1.0.1. http://www.omg.org/cgi-bin/doc?omg/03-06-01.

[8] Object Management Group (OMG) Unified Modeling Language Specification 2.0. http://www.omg.org/cgi-bin/doc?formal/05-07-04.

[9] Object Management Group (OMG), MOF 2.0 Query/View/Transformation.http://www.omg.org/spec/MOF/2.0/PDF/

[10] Object Management Group (OMG), Common WarehouseMetamodel (CWM) Specification 1.1. http://www.omg.org/spec/cwm/1.1/PDF/

[11] Object Management Group (OMG), Object Constraint Language (OCL) Specification 2.0. http://www.omg.org/spec/OCL/2.2/PDF/

[12] MDA en action Ingénierie logicielle guidée par les modèles, Xavier Blanc, édition Eyrolles 2005.

[13] S. Mbarki, M. Erramdani, Model-Driven Transformations: From Analysis to MVC 2 Web Model. (I.RE.CO.S.), Vol 4 N 5 Septembre 2009.

[14] S. Mbarki, M. Erramdani, Toward automatic generation of mvc2 web applications InfoComp, Journal of Computer Science, Vol.7 n.4, pp. 84-91, December 2008, ISSN: 1807-4545

[15] J. Trujillo, S. Lujan-Mora, , I.Song , a UML profile for multidimensional modeling in data warehouses, Data & knowledge Engineering, (2006) 725-769

[16] JN. Mazón , J. Trujillo, An MDA approach for the development of data warehouses, Decision Support Systems 45 (2008) 41–58

[17] SmartQVT, http://smartqvt.elibel.tm.fr/

[18] Ganapathy, G., & Sagayaraj, S. (2011). Extracting Code Resource from OWL by Matching Method Signatures using UML Design Document. International Journal of Advanced Computer Science and Applications - IJACSA, 2(2), 90-96.

[19] Dhindsa, K. S. (2011). Modelling & Designing Land Record Information System Using Unified Modelling Language. International Journal of Advanced Computer Science and Applications - IJACSA, 2(2), 26-30.

[20] Acharya, A. A. (2010). Model Based Test Case Prioritization for Testing Component Dependency in CBSD Using UML Sequence Diagram. International Journal of Advanced Computer Science and Applications - IJACSA, 1(6).

AUTHORS PROFILE

I.Arrassen Graduate as Computer Science Enginner from the INPT(National Institut of Poste and Telecommunication) and Ph-D-Student at Faculty of Sciences, Laboratory for Computer Science Research, Mohammed First University, Oujda, Morocco.

A.Meziane is a Professor of Computer Sciences, Laboratory for Computer Science Research , Mohammed First University, Oujda, Morocco.

R.Esbai Ph-D-Student at Faculty of Sciences, Laboratory AMCSP: Applied Mathematics and Computer Signal Processing, Mohammed First University, Oujda, Morocco.

M.Erramdani is a Professor of Computer Sciences at Superior School of TechnologyLaboratory AMCSP: Applied Mathematics and Computer Signal Processing Mohammed First University, Oujda, Morocco.