Journal of Theoretical and Applied Information Technology
15th May 2023. Vol.101. No 9
© 2023 Little Lion Scientific

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

INTEGRATION OF THE MDA APPROACH IN DOCUMENT-ORIENTED NOSQL DATABASES, GENERATION OF A PSM MODEL FROM A PIM MODEL

AZIZ SRAI[1], FATIMA GUEROUATE[2]
[1]ERCI2A, FSTH, Abdelmalek Essaadi University, Tetouan, Morocco
[2]LASTIMI Laboratory, Superior School of Technologies of Sale, Mohammadia School of engineering, Mohamed V University city of Rabat, Morocco
E-mail: [1]a.srai@uae.ac.ma, [2]guerouate@gmail.com

ABSTRACT

The volume of data and its diversity are so important today that is the reason why many relational databases are unable to handle this type and large volume of data. To respond to this problem, many NoSQL databases have emerged, such as document-oriented, graph-oriented, key-value-oriented and column-oriented databases. These databases, which revolve around Big Data, have shown an important power in the management of big data. The scientific contribution of the work presented in this article is the application of an MDA approach on a document-oriented approach. We demonstrate the capacity and adaptability of the model-oriented approach on NoSQL approaches, in particular the document-oriented approach. For the presentation of this work we started by introducing the context of our studies which is the MDA approach and the document-oriented approach, then we defined the different metamodels of the sources and targets. We then introduced the different possible model-to-model and model-to-Text transformations using the QVTo transformation language. Finally, we presented as result the document-oriented PSM model and the XMI model generated from the Model to Text transformation with Acceleo. Our motivation for this contribution comes down to the fact that a minority of authors who integrate the concept of programming by model.

Keywords: *Big Data, MDA approach, NoSQL, QVT, PIM model.*

1. INTRODUCTION

The data management landscape has become extremely rich and complex. The wide variety of needs of current information systems has led to the emergence of many heterogeneous data management solutions. Relational DBMSs are still widely used but exist alongside a large number of other so-called NoSQL systems. These NoSQL systems meet various needs such as reliable storage systems for masses of data, efficient mechanisms for their analysis, structures allowing complex data to be represented flexibly. The data model is one of the important characteristics of a DBMS and any type of data management system. NoSQL systems cover a range of solutions whose data model is not relational. Their data models are generally classified into four main families: key-value, column-oriented, graphs and document-oriented. This classification is common, but there is no unique data model definition for each family. There are few and recent formalization efforts. The concepts handled in these models are not however new and take up aspects of other models such as the relational and its extensions which are not in first normal form, complex values and object oriented models. NoSQL systems allow a lot of freedom to represent data. In general, these systems do not handle the notion of database schema. We speak of Schema less system because it is not necessary to define a data schema before creating the database. In addition, the structure of the data may be different even if they are grouped together within the same collection or table. Type checking is very basic. For example, in key-value systems, the type of the value is unknown and may differ completely from entry to entry. In this work we present an application of the MDA approach to document-oriented NoSQL databases more particularly MongoDB. We approve that this approach is compatible with this type of database in order to support its use in this sense. In this article we have applied model programming on NoSQL databases, in particular document oriented. The application of model programming on this type of databases can make them independent during the implementation on a database management system and this is the main objective of the MDA approach.

2. LITERATURE REVIEW

The work of Bézivin et al. [4] is dedicated to the application of the MDA approach for the web services platform. in this work the authors presented a development of an illustrative example of e-business based on two different applications of a Model-Driven Architecture (MDA) approach. In the first application, the Platform Independent Model (PIM) is created using the Unified Modeling Language (UML). This PIM is transformed using Atlas Transformation Language (ATL) to generate the Platform Specific Model (PSM) based on three target platforms: Java, Web service and Java Web service developer pack (JWSDP). In the second application, the PIM is created using Enterprise Distributed Object Computing (EDOC) and transformed into another PSM based on the same target platforms.

The objective of work [1] is to generate a model respecting the n-tier architecture using the MDA approach; this model represents an E-learning application. The authors proposed two metamodels, a source metamodel based on UML, and a target metamodel based on an n-tier architecture. The authors have chosen the QVTo transformation language to perform all possible transformations between the two metamodels. The authors in [2], introduced an MDA approach to generate a PSM model for EJB platforms. The authors in [3], introduced a model-based approach for modeling and generating AJAX applications. The author in [5] proposes a framework based on OMG's Model Driven Architecture. These platform-independent frameworks specify and classify existing and future Learning Management Systems (LMS).

According to the analysis of the works cited, we have found that a majority of authors do not invoke the interest of applying the MDA approach on NoSQL platforms via a transformation from a PIM model to a model PSM or through code generation through a PSM-to-code transformation. To our knowledge, we are the first authors to have proposed a total generation of code for NoSQL platforms in order to use models independent of all implementation platforms. No work has presented a global study to transform a source model (uml diagram) into a target model (document-oriented NoSQL databases), i.e. a generation of a document-oriented NoSQL database through an MDA approach.

3. METHODOLOGY

The MDA (Model Driven Architecture) was designed and promoted by the OMG (Object Management Group). It is the result of the confrontation of different model-oriented design approaches and in particular thanks to the advent of UML. MDA comes in the form of a set of similar standards (but not co-dependent), used to create a model and refine it until a finished product is obtained. In practice, we practice a MDA method when we define an application using UML formalism and generating the target code (Java, C #, or others) then the corresponding executable. Today, many tools provide such functionality. Globally, MDA provides the development world with methods (models and practices) for the efficient creation of non-platform models and their reuse. In the perspective of our research on the construction of mobile collaborative applications, we propose to study these models and methods.

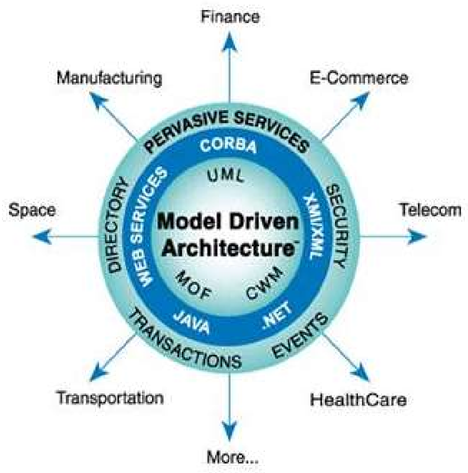


Figure 1: Diagram summarizing the languages and methods provided by MDA architecture

The OMG is based on several standards. In the center is the UML (Unified Modeling Language) standard, MOF (Meta-Object Facility) and CWM (Common Warehouse Metamodel). In the next layer, there is also an XMI (XML Metadata Interchange) standard, which allows dialogue between middleware (Java, CORBA, .NET and web services). The third layer contains the services, which make it possible to manage events, security, directories and transactions. Finally, the last layer offers frameworks specific to the field of application (Finance, Telecommunications, Transport, Space, medicine, e-commerce, manufacturing, etc…).

3.1 Principles of MDA

MDA's proposal is to define system functionality in a Platform Independent Model (PIM) using a specification language, and translate it into a platform-specific model. -form (PSM: Platform Specific Model), to finally generate the compliable (or compiled) code for this platform. The objective is, from a PIM, to obtain several PSMs for several platforms.

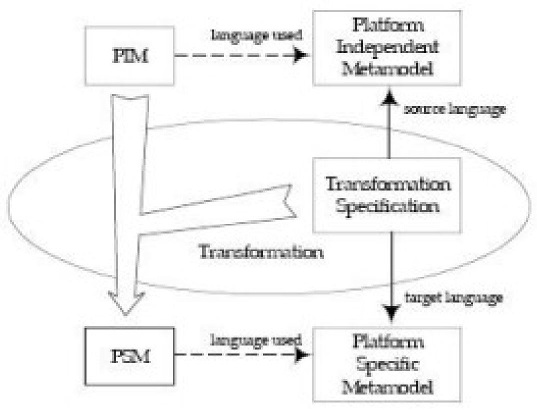

PIM
language used
Platform Independent Metamodel
source language
Transformation Specification
Transformation
target language
PSM
language used
Platform Specific Metamodel

Figure 2: Diagram representing the MDA approach

The MDA concept therefore proposes an open approach, which can partly allow the interoperability of systems and the management of the plasticity of interfaces. Likewise, working on a PIM before moving on to PSM makes it easier to validate a model without being hampered by platform-specific technical details. Moreover, it is easier and faster to manage the different implementations of a system from abstract models than to work on the code of the implementation. For the different models of MDA, there are several standards: the different specification languages are UML, MOF, XMI, CWM and OCL.

3.2 Model transformation

Model-driven engineering serves as a source for the automatic generation of all or part of the modeled system. Model transformation techniques ensure this generation. Classically, the transformation follows the diagram shown in Figure below. A transformation can be seen as a program generating a model Mb conforming to a metamodel MMb from a model Ma conforming to a metamodel MMa. The transformation therefore specifies a set of relationships between the source and target metamodels. When metamodels MMa and MMb refer to two different metamodels, it is called exogenous transformation. For example, transforming an XML file into a JSON file. Otherwise, when MMa and MMb refer to the same metamodel, the transformations are qualified as endogenous. For example, the transformation of a UML model into another UML model.

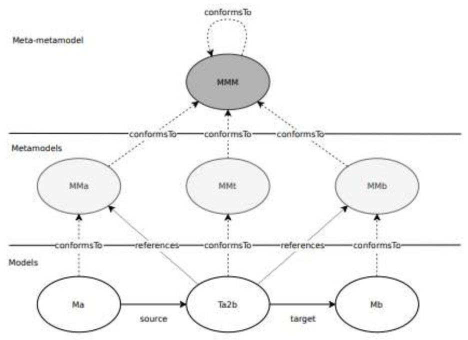


Figure 3: structure of a model transformation

Moreover, during an endogenous transformation, it is possible that the Ma and Mb models are identical. There is therefore no creation of the target model, but updating of the source model. These transformations are referred to as "in-place", "update" or "refining mode" transformations.

3.3 ATLAS Transformation Language

ATL is a hybrid transformation language based on the concept of rules. ATL rules do not rely on graph patterns for rule activation and effects.ATL rules declare a set of input elements along with optional guards on those input elements. This mechanism allows the fine selection of input elements. Output elements are declared in a second section of the rule. This is where the relationships (or bindings) are declared. These bindings are strongly inspired by OCL expressions. Bindings are used to calculate the values of the properties of the target model from the properties of the source model. Except in special cases, the source model is only accessible in read-only mode and the target model is only accessible in write-only mode. In ATL there are two main types of rules. Standard rules are automatically executed for all input combinations present in the source model. Lazy rules are only executed when requested by another rule. There are two types of lazy rules, rules with cache (unique) and without. A lazy rule's cache allows the same output item to be returned if the same rule is called with the same input items multiple times. In addition to this system of declarative rules, ATL allows the execution of blocks of imperative code. These imperative blocks make it easy to specify transformations that are difficult to express in pure declarative ATL. The execution of a transformation is done by the optional call to a special rule, the entry point. The algorithm then searches for the rules applicable to the provided input elements. To do this, it compares the input elements of the rules as well as any guards to the elements of the model and saves the valid combinations. For each of these combinations, the algorithm creates traceability links, used to associate each output element with the corresponding input element(s). Then, the algorithm creates the corresponding output elements and executes the bindings to calculate the value of their properties. Note that the order of application of the rules is not guaranteed.

3.4 Yet Another Model Transformation Language

Still based on the concept of rules, Yet Another Model Transformation Language (YAMTL) is a recent and powerful approach based on Xtend. Xtend is a general-purpose programming language compiled into Java which notably allows the redefinition of operators that Java does not allow. Inspired by ATL rules, YAMTL rules are composed of two same parts. A first declaring a list of input patterns along with optional filters. A second part contains the output models as well as the expressions used to calculate the values of the properties of the elements of these models. The main difference with ATL is the syntax used. YAMTL is limited on this point by the constraints imposed by Xtend. Indeed, Xtend offers a certain freedom thanks to mechanisms such as the redefinition of operators or extension methods, but does not allow the introduction of new syntax elements. Rules execution follows an order similar to that of ATL. The transformation is first compiled, then an algorithm searches and saves all the rules that can be applied as well as their source elements. Then the rules are executed one after the other creating the elements of the output model.

3.5 QVT language

Query/View/Transformation (QVT) is a standard defined by the OMG to specify transformations between models, whose meta model satisfies the MOF standard. It includes a declarative part and an imperative part. The declarative part consists of two parts: a part carrying out the correspondence between the two models expressed in the standard MOF named QVTr (relations), and a part which makes it possible to evaluate conditions on the elements of our models to make them correspond, named QVTc (Core). These two parts use OCL (Object Constraint Language) to define the matching rules. OCL is a formal language standardized by the OMG for specifying software constraints. The imperative part, consisting of QVTo (operational), makes it possible to extend the declarative language. Constructs such as for loops or if conditions are offered there. QVTo also introduces the use of imperative OCL rules.

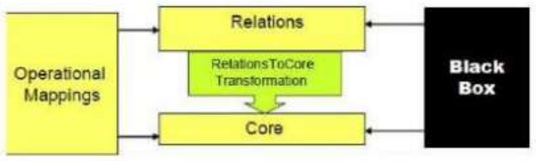


Figure 4: Relationships between QVT metamodels

3.6 Incremental transformation of models

Classic transformations process the entire source model to create a target model that conforms to the transformation specification. This operation can be costly when the source model is composed of a large number of elements, or when the transformation requires costly operations in terms of computation time. This computational cost is not problematic in the case of models that change little. However, if many modifications are applied to elements of the source model, then it may be worth using an incremental transformation approach. The idea behind incremental model transformation is to only apply the transformation to source elements that have changed since the last run. This reduces the time required to apply the transformation to the updated model. Moreover, updating the target model when modifications are detected on the source model allows to preserve the identity of the elements of the target model. This is particularly the case when the target model is a graphical model presented to the user. In this case, it is important that the changes made to the source model are applied quickly to the target model. Moreover, here it is better to update the visual elements presented to the user rather than having to completely replace them with each change.

3.7 Bidirectional model transformation

During a model transformation, the target model is calculated from the source model. Ideally only one model is modified (the source), so recomputing a new target (or calculating a set of changes to apply) is possible using the classical or incremental approaches. A classic example of bidirectional transformation is generating code from a model. It is not uncommon for the generated code to have to be manually modified to add missing features or optimize the code. However, if the model used to generate the code is modified, it is necessary to regenerate the code. A classic approach would ignore code changes and overwrite them with the new version. This behavior is not satisfactory. Another example is the problem of updating views in a database. A view in a database is a virtual table corresponding to the result of a query. These views are useful for more finely controlling access to data. Most database management systems (DBMSs) only allow read access to views. Some DBMS allow inserting, deleting or modifying records of the view, but in very limited contexts.

3.7 NoSQL databases

The term NoSQL refers to a type of database management systems that goes beyond the relational systems associated with the SQL language by accepting more complex data structures. According to their physical models, the DBs managed by these systems fall into four categories: columns, documents, graphs and key-value. Each of them offering specific features. For example, in a document-oriented DB like MongoDB, data is stored in tables whose rows can be nested. This data organization is coupled with operators that allow access to nested data. The choice of the DBMS category most suited to a given application is linked to the nature of the processing (queries) applied to the data. But this choice is not exclusive since, in each category, DBMSs can provide all types of processing, sometimes at the cost of a certain heaviness or more extensive programming.

3.7.1 Column-oriented model

The column-oriented model is a structured model where data is organized into families of columns, which is equivalent to the concept of a table in the relational model. Rows have an identifier called row key and are composed of a set of values; each is associated with a column. Thus, the search for a value amounts to going through the sequence: line key -> family of columns -> column.
Although the column-oriented model is close to the relational model, the organization of data in the two models is different. In contrast to what is found in a relational DB where the columns are static and present in each row, in a column-oriented DB the columns are dynamic and appear only in the rows concerned. In other words, each line has a different number of columns and new columns can be added to it at any time, thus gaining in extensibility at the data model level.In addition, the column-oriented model has the advantage of improving storage efficiency and avoiding space consumption compared to the relational model. Indeed, due to their design by block allocation, in a relational DBMS, an empty column will still consume space. In a column-oriented DBMS, the storage cost of an empty column is 0. Cassandra, HBase and Accumulo are examples of DBMSs where data is stored in a column-oriented model.

3.7.2 Document-oriented model

Data in a document-oriented model is organized into collections of documents. A document is identified by a key to which corresponds an aggregate of key-value pairs which can be ranked. This means that the value can itself contain one or more key-value pairs. Within the same collection, the documents can be of different structures. In other words, the pairs used to define the documents of a collection are not necessarily the same. In addition, like other NoSQL models, the document-oriented model is flexible, we can add pairs each time a new document is inserted.

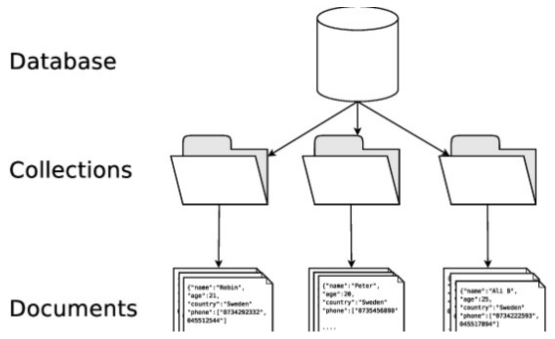


Figure 5: Document Oriented NoSQL Databases

Document-oriented DBMSs, such as MongoDB, couchDB, and couchDB server, provide advanced functionality for document manipulation. With the data model nesting principle they adopt (the document-oriented model), we can model the data in such a way that it supports more advanced querying features, such as the ability to manipulate the document content (full-text search).

3.7.3 Graph-oriented model

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The graph-oriented model is not new because it uses a model called "network model" created in the 1970s by Charles Bachman within the CODASYL consortium which also led to the creation of the COBOL language. We can also consider that knowledge representation models, thesauri and ontologies defined by different formats such as OWL (Ontology Web Language) and RDF (Resource Description Framework) are specialized graph-oriented data models. This model is based on the notion of graph in the mathematical sense to store and process information. This model structures the data in the form of a graph. That is to say a set of nodes that can be linked together by arcs. The nodes represent the entities, they can be compared to the documents of the document-oriented model. A node is defined by one or more labels, a sort of type if we can say so, and includes properties in the form of key/value pairs.

3.7.4 NoSQL key-value database

This specific type of NoSQL database uses the key-value method and stores collections of many key-value pairs in memory, on hard disk or on SSDs. Each record corresponds to a set of key-value pairs. A key is made up of a primary key, the unique identifier of the record (or of a row), then secondary keys or attributes associated with values. Values can be any kind of object, a file, a number or string, or even some other key-value pair, in which case the database structure becomes more complex. Unlike relational databases, key-value DBMSs do not have a specific structure. RDBMSs store data in tables where each column is linked to different variables. Keys can have any name, but since this is the only way to retrieve the values associated with them, they should be named strategically. Key names can range from simple numbering to specific descriptions of the value that will follow. A key-value database can be compared to a dictionary or a directory. Dictionaries have words acting as keys and their meanings acting as values.

3.8 Source and Target Metamodels

In our MDA approach, we opted for modeling approaches to generate the document oriented NoSQL database. These approaches require a source meta-model and a target meta-model. In this section, we present the different meta-classes that make up the UML class diagram source meta-model and the document-oriented NoSQL target meta-model. The process of transforming the UML source model into a document-oriented target model.

3.8.1 UML source Metamodel

Figure 6. illustrates the simplified UML source meta-model based on packages including operations, associations and classes. Those classes are composed of properties with parameters.

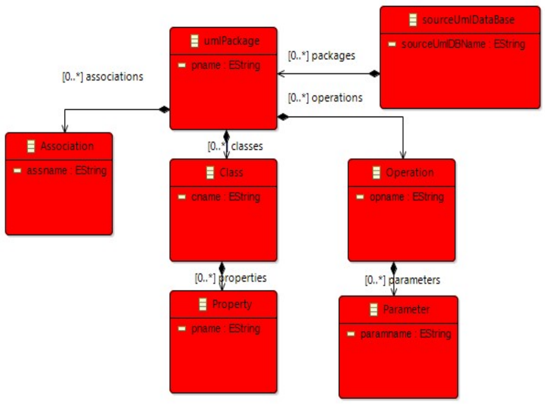


Figure 6: UML source meta-model

3.8.2 Document oriented target Metamodel

Figure 7. Illustrates the simplified Document target meta-model:

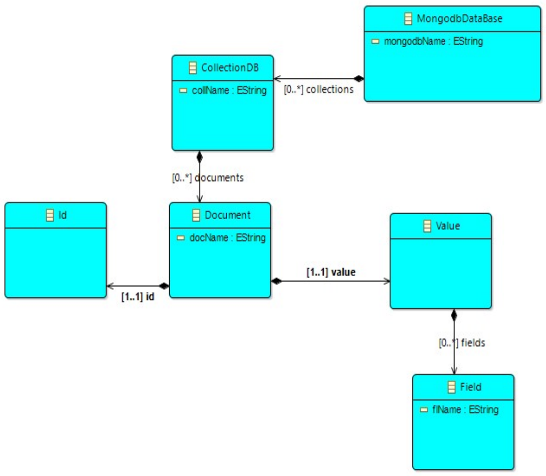
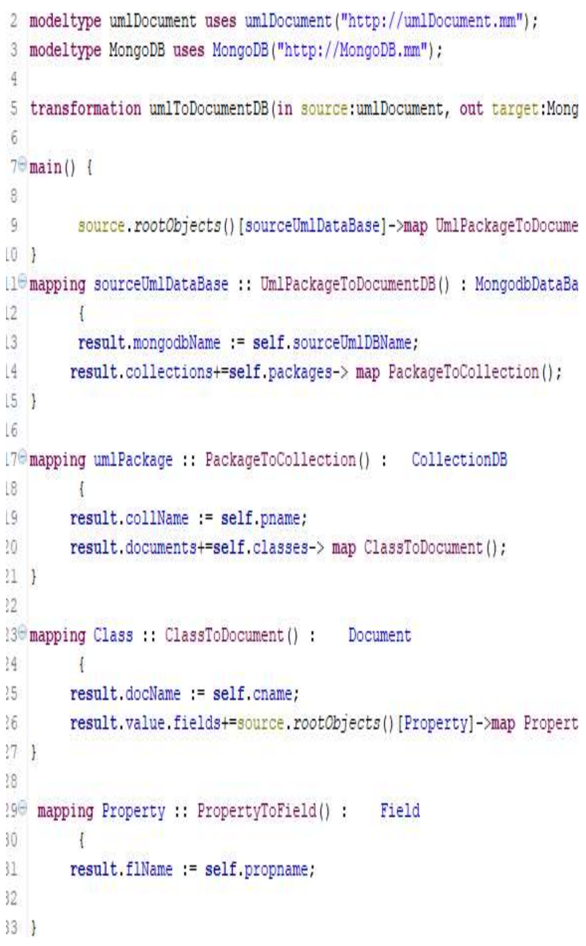


Figure 7: Document target meta-model

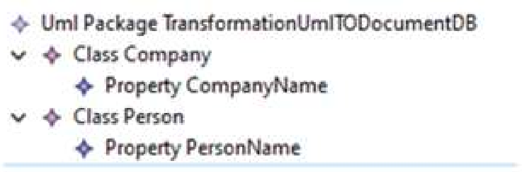
4. RESULTS AND DISCUSSIONS

4.1 Transformations rules M2M and M2T

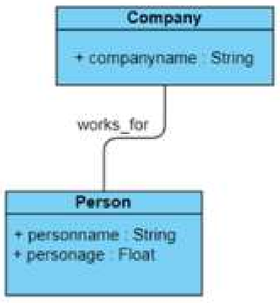
Figure 8. Illustrates the transformation rules M2M and M2T:

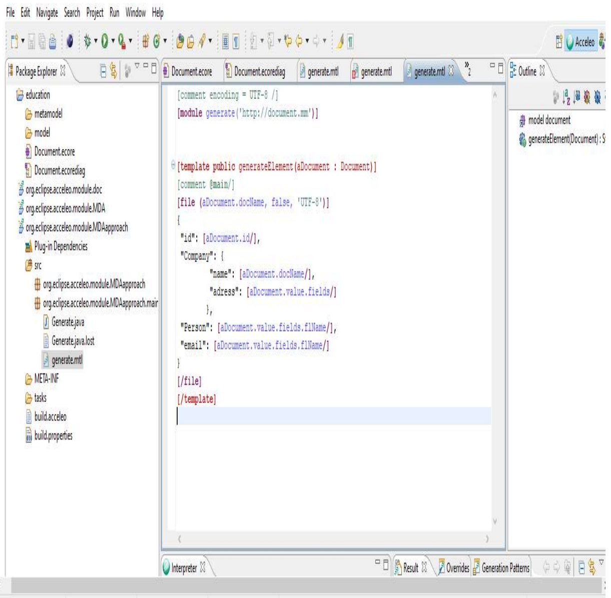


```
modeltype umlDocument uses umlDocument("http://umlDocument.mm");
modeltype MongoDB uses MongoDB("http://MongoDB.mm");

transformation umlToDocumentDB(in source:umlDocument, out target:Mong

main() {

      source.rootObjects()[sourceUmlDataBase]->map UmlPackageToDocume
}
mapping sourceUmlDataBase :: UmlPackageToDocumentDB() : MongodbDataBa
      {
      result.mongodbName := self.sourceUmlDBName;
     result.collections+=self.packages-> map PackageToCollection();
}

mapping umlPackage :: PackageToCollection() :   CollectionDB
      {
     result.collName := self.pname;
     result.documents+=self.classes-> map ClassToDocument();
}

mapping Class :: ClassToDocument() :    Document
      {
     result.docName := self.cname;
     result.value.fields+=source.rootObjects()[Property]->map Propert
}

 mapping Property :: PropertyToField() :    Field
      {
     result.flName := self.propname;

}
```

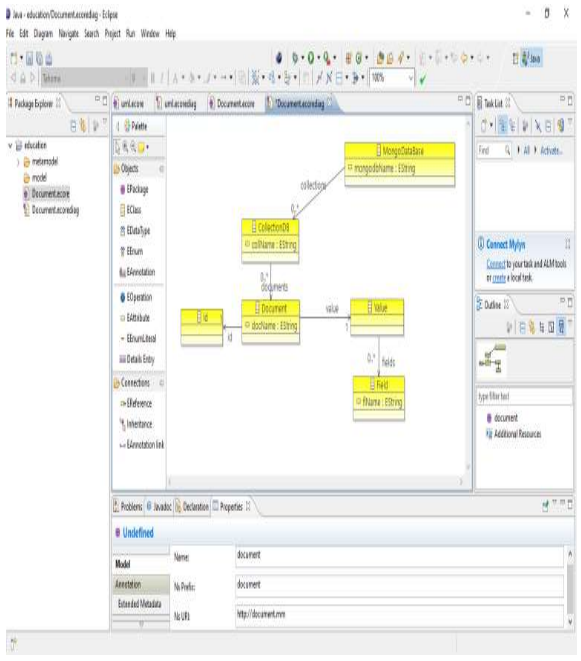


```
Uml Package TransformationUmlTODocumentDB
  Class Company
    Property CompanyName
  Class Person
    Property PersonName
```





```
[comment encoding = UTF-8 /]
[module generate('http://document.mm')]


[template public generateElement(aDocument : Document)]
[comment @main/]
[file (aDocument.docName, false, 'UTF-8')]
{
 "id": [aDocument.id/],
 "Company": {
         "name": [aDocument.docName/],
         "adress": [aDocument.value.fields/]
        },
 "Person": [aDocument.value.fields.flName/],
 "email": [aDocument.value.fields.flName/]
}
[/file]
[/template]
```

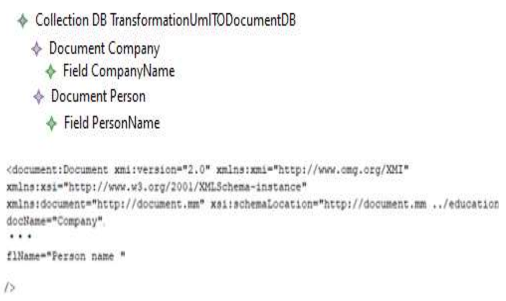


Name: document
Ns Prefix: document
Ns URI: http://document.mm

Figure 8: Transformation rules M2M and M2T

To validate our transformation rules, we conducted several tests. After applying the transformation on the UML source model, we generated the document-oriented PSM target model (see Figure 9).

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195



```
Collection DB TransformationUmlTODocumentDB
  Document Company
    Field CompanyName
  Document Person
    Field PersonName

<document:Document xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:document="http://document.mm" xsi:schemaLocation="http://document.mm ../education
docName="Company".
...
flName="Person name "

/>
```

Figure 9: Document-Oriented MongoDB PSM

5. CONCLUSION

In this article, we proposed an MDA approach to migrate a UML class diagram representing a relational database to a document-oriented database. The transformation rules were developed using QVT. This work should be extended to allow the generation of other NoSQL solutions such as key-value-oriented and column-oriented solutions. Through this work we have validated the validity and performance of the MDA approach today, we can say that the application of such an approach in the context of Big Data is very important and also a current research axis. Our contribution in this work concerns the two types of model transformations, a model to model transformation with the QVTo model transformation language, this transformation represents a vertical transformation from the PIM to the PSM. Another important transformation that we carried out is the model to text transformation, this transformation was provided by the Acceleo tool, it is a transformation from a PSM to code. The contribution we presented is rich and represents a basis for other authors but for the moment does not cover all aspects of NoSQL databases, it remains as a perspective for future work.

REFERENCES:

[1] A. Srai, F. Guerouate, N. Berbiche, H. Drissi, “Generated PSM Web Model for E-learning Platform Respecting n-tiers Architecture,” International Journal of Emerging Technologies in Learning (iJET), vol. 12, no. 10, pp. 212-220, 2017.

[2] A. Srai, F. Guerouate, N. Berbiche, H. Drissi, “MDA Approach for EJB Model,” 6th IEEE International Conference on Multimedia Computing and Systems (ICMCS’18). DOI:10.1109/ICMCS.2018.8525924.

[3] Gharavi, V., Mesbah, A., Deursen, A. V., “Modelling and Generating AJAX Applications: A Model-Driven Approach,” Proceeding of the7th International Workshop on Web-Oriented Software Technologies, New York, USA (Page: 38, Year of publication: 2008, ISBN: 978-80-227-2899-7).

[4] J. Bezivin; S. Hammoudi; D. Lopes; F. Jouault, Applying MDA approach for Web service platform, Proceedings. Eighth IEEE International Enterprise Distributed Object Computing Conference, 2004. EDOC 2004, Monterey, CA, USA, DOI: 10.1109/EDOC.2004.1342505.

[5] Kurillova, Model Driven E-Learning Platform Integration, Proceedings of the EC-TEL 2007 PROLEARN Doctoral Consortium, Crete, Greece, September 18, 2007.

[6] X. Zhang et al., A Model Driven Architecture Approach for Developing E-Learning Platform, (Eds.): Edutainment 2010, LNCS 6249, pp. 111-122,2010.

[7] Aziz Srai, Fatima Guerouate, Hilal Drissi Lahsini, Generated Psm Multi-Layered Model Using Mda Approach, International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-8 Issue-4, April 2019.

[8] Essebaa, I. , Chantit, S. QVT Transformation Rules to Get PIM Model from CIM Model, Europe and MENA Cooperation Advances in Information and Communication Technologies, pp.195-207, January 2017. DOI: 10.1007/978-3-319-46568-5_20

[9] Rhazali, Y. , Hadi, Y. and Mouloudi A. , Model Transformation with ATL into MDA from CIM to PIM Structured through MVC, Procedia Computer Science 83:1096-1101 December 2016. DOI: 10.1016/j.procs.2016.04.229.

[10] Roubi, S. , Erramdani, M. and Mbarki, S. Model Driven Architecture as an Approach for Modeling and Generating Graphical User Interface. Proceedings of the Mediterranean Conference on Information & Communication Technologies 2015, pp.651-656. DOI: 10.1007/978-3-319-30298-0_72

[11] Mbarki, S. and Rahmouni, M. Combining UML class and activity diagrams for MDA generation of MVC 2 web applications. International Review on Computers and Software (I.RE.CO.S), 8(4):949-957 · April 2013.

[12] rédéric J., & Ivan, K. (2006). Transforming models with ATL. Proceedings of MoDELS 2005 Workshops, LNCS 3844, (pp. 128 – 138), Springer-Verlag Berlin Heidelberg.

[13] Czarnecki, K., Helsen, S., Classification of Model Transformation Approaches, in online proceedings of the 2nd OOPSLA'03 Workshop on Generative Techniques in the Context of MDA. Anaheim, October, 2003.

[14] Li, Y., Gai, K., Qiu, L., Qiu, M., & Zhao, H. (2017). Intelligent cryptography approach for secure distributed big data storage in cloud computing. Information Sciences, 387, 103-115.

[15] Mpinda, S. A. T., Maschietto, L. G., & Bungama, P. A. (2015). From relational database to columnoriented nosql database: Migration process. International Journal of Engineering Research & Technology (IJERT), 4, 399-403.

[16] Karnitis, G., & Arnicans, G. (2015, June). Migration of relational database to document-oriented database: structure denormalization and data transformation. In Computational Intelligence, Communication Systems and Networks (CICSyN), 2015 7th International Conference on (pp. 113-118). IEEE.

[17] Daniel, G., Sunyé, G., & Cabot, J. (2016, November). UMLtoGraphDB: mapping conceptual schemas to graph databases. In International Conference on Conceptual Modeling (pp. 430-444). Springer, Cham