Towards Automatic Generation of NoSQL Document-Oriented Models

F.ABDELHEDI[2], A.AIT BRAHIM[1], F.ATIGUI[3] and G.ZURFLUH[1]
[1]IRIT, Toulouse Capitole University, Toulouse, France
[2]CBI2 – TRIMANE, Paris, France
[3]CEDRIC- CNAM Paris, France

Abstract - *Volume, Variety and Velocity are the three dimensions that have definitely impacted the tools required to store Big Data. Adapted data management tools have arisen, i.e. NOSQL systems. Compared to existing DBMS, NoSQL systems are commonly accepted to support larger volume of data, provide faster data access, better scalability and higher flexibility. While NoSQL solutions have proven their efficiency to handle Big Data, it's still an unsolved problem how the automatic storage of Big Data in these systems could be ensured. The aim of this paper is to propose a precise and automatic approach that guides and facilitates the Big Database implementation task within document-oriented systems considered as the new generation of DBMS technology. Our approach will assist the developers to map Big Database UML conceptual model into document-oriented physical models; it relies on a unified model designed for document-oriented systems. The instances of this model can be generated to target specific NoSQL platform.*

Keywords: Big Data storage; NoSQL; UML conceptual model; document-oriented model; MDA.

1 Introduction

In the Big Data era, there must be DBMS able to store large datasets effectively with high performance. Not only the amount of data is on a completely different level than before, but also we have various types of data including factors such as format, structure, and sources. Furthermore, the speed at which these data must be collected and analyzed is increasing. In such scenario, Big Data applications need a database solution that integrates easily all possible data structures while offering lower latency, better scalability as well as higher flexibility.

Relational systems are mature data management technology. However, with the rise of Big Data, these systems became unfit for large and distributed data management. The major problems of relational technologies are: (1) the horizontal scale: relational databases are mainly designed for single-server configurations; to scale it, it has to be distributed on multiple powerful servers, which seems really expensive. Furthermore, handling tables across different servers is a complex task, (2) a strict data model to design before data processing: in the context of Big Data, it should be easy to add new data; but the problem is that relational models are hard to change incrementally without impacting performance or making the database offline.

Compared to existing DBMS, NoSQL systems are commonly accepted to support larger volume of data and provide flexible data models, low latency at scale and faster data access [7]. NoSQL covers a wide variety of different systems that can be classified into four basic types: key-value, column-oriented, document-oriented and graph-oriented. In this paper, we focus on the document-oriented. This is justified by the fact that our case study, detailed in section "Motivation", requires processing operations that access to hierarchically structured data, and that document-oriented systems have proven to be the most adapted solution for this kind of operations [13].

Today, the developers have to deal with the problem of storing Big Data in NoSQL systems. New approaches, guidelines, and techniques are required in order to overcome the complexity of this task. In this context, a new automatic approach that guides and facilitates the Big Database implementation task within document-oriented systems will be presented in this paper.

The remainder of the paper is structured as follows. Section 2 motivates our work using a case study in the healthcare field. Section 3 introduces our approach. Sections 4 and 5 detail our contributions. Section 6 presents our experiments. Section 7 reviews previous work. Finally, Section 8 concludes the paper and announces future work.

2 Motivation

2.1 Case study

To motivate and illustrate our work, we present a case study in the healthcare filed. This case study concerns national or international scientific programs for monitoring patients having serious diseases. The main goal of this program is (1) to collect data about disease development over time, (2) to study interactions between different diseases (3) to evaluate the short and medium-term effects of their treatments. The medical program can last up to 3 years. Data collected from establishments involved in such a program have the characteristics of Big Data (the 3 V):
Volume: The amount of data collected from different health care services during a time period can reach several terabytes.

Variety: The amount of data created while monitoring patients come in different types and formats. Therefore, the DW used for this application will contain: (1) structured data (respiratory rate, blood pressure, temperature, patient name, diagnosis codes, etc.), (2) unstructured data (patient histories, visit summaries, paper prescriptions, radiology reports…) and (3) semi-structured document (such as the package leaflets of medicinal products that provide a set of comprehensible information enabling the use of the medicinal product safely and appropriately).

Velocity: Some data are produced in continuous flow by sensors; it must be processed in near real time because it can be integrated into time-sensitive processes. For example, some measurements, like temperature, require an emergency medical treatment if they cross a given threshold).

2.2 Necessity of conceptual model for Big Data applications

One of the NoSQL key features is that databases can be schema-less. This means, in a table, meanwhile the row is inserted, the attributes names, and values are specified. Unlike relational systems - where first, the user defines the schema and creates the tables, second he inserts data -, the schema-less property offers undeniable flexibility that facilitates the physical model evolution. End-users are able to add information without the need of database administrator. For instance, in the medical program that follows-up patients suffering from a chronic pathology – case study presented in the previous section – one of the benefits of using NoSQL databases is that the evolution of the data (and schema) is fluent. In order to follow the evolution of the pathology, information is entered regularly for a cohort of patients. But the situation of a patient can evolve rapidly which needs the recording of new information. Thus, few months later, each patient will have his own information, and that’s how data will evolve over time. Therefore, the data model (i) differs from one patient to another and (ii) evolves in unpredictable way over time. We should highlight that this flexibility concerns the physical level i.e. the stored database exclusively [2].

In information systems, the importance and the necessity of conceptual models are widely recognized. The conceptual model provides a high level of abstraction and a semantic knowledge element close to human comprehension, which guarantees efficient data management [1]. Furthermore, this model is a document of interchange between end-users and designers, and between designers and developers. Also, the conceptual model is used for system maintenance and evolution that can affect business needs and/or deployment platform. The Unified Modeling Language (UML) is widely accepted as the standard of information system modeling [1].

2.3 Objective & utility

On the one hand, NoSQL systems have proven their efficiency to handle Big Data. On the other hand, the needs of a conceptual modeling and design approach remain up-to-date. Therefore, we are convinced that it’s important to provide a precise approach that guides and facilitates the Big Database implementation task within NoSQL systems. This approach will assist the developers to map Big Database UML conceptual model into NoSQL physical models.

For this, we propose the "Object2NoSQL" MDA-based approach presented in the following section.

3 Object2NoSQL approach

3.1 Three-Level architecture of Object2NoSQL process

In this paper, we propose the Objetc2NoSQL approach that starts from a UML class diagram and generate NoSQL physical models. We introduce a logical level between conceptual (business description) and physical (technical description) levels in which a document-oriented logical model is developed.

The need for this intermediate (logical) layer is justified by referring to the ANSI/SPARC architecture. This architecture shows conceptual (business description) and internal (technical description) levels. The latter may be decomposed into two levels: logical and physical [18]. The logical level aims to provide an intermediate model that describes the structure of the data, without itemizing the specific features of each system. This ensures data independence meaning that upper level is isolated from changes to lower level.

In our scenario, the logical model corresponds to a NoSQL model that can be implemented on different document-oriented platforms. The advantage of using this unified model is to limit the impacts related to technical aspects of NoSQL systems. Thus, technological changes of the NoSQL system (or even its replacement by another system) will appear in the physical model, but would not affect the logical model. Only the transformation process “Logical Model -> Physical Model” will be adapted and restarted; there will be no impact on “Conceptual Model -> Logical Model” transformation. This simplifies the transformations and saves developers efforts and time.

3.2 Formalization tool

To formalize and automate the Object2NoSQL process, we use the Model Driven Architecture (MDA) [4]. MDA is well-known as a framework for models automatic transformations; One of its main aims is to separate the functional specification of a system from the details of its implementation in a specific platform. This architecture defines a hierarchy of models from three points of view: Computation Independent Model (CIM), Platform Independent Model (PIM), and Platform Specific Model (PSM) [5]. Among these models, we use the PIM to describe data hiding all aspects related to the implementation platforms, and the PSM to represent data using a specific technical platform.

3.3 Component of Object2NoSQL process

In our scenario, the UML class diagram and the document-oriented logical model belong to the PIM level. At the PSM level we consider NoSQL physical models that correspond to

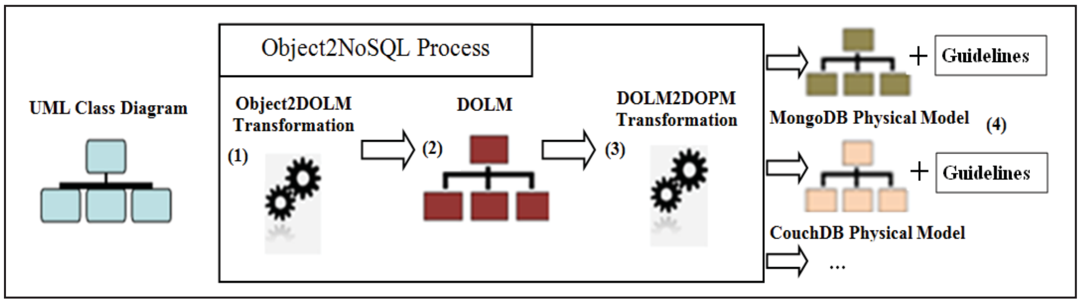


Fig.1. Object2NoSQL Approach

document-oriented platforms. Switching between models is ensured by M2M (Model-To-Model) transformations; such transformations describe a mapping between source and target models elements. Figure 1 shows the different component of Object2NoSQL process.

Object2DOLM (1) is the first step in the process. It transforms the input UML Class Diagram into the Document-Oriented Logical Model (DOLM) presented in section IV. DOLM2DOPM (3) is the second step that generates Documents-Oriented Physical Models (DOPMs) (4) starting from the DOLM, as well as a set of guidelines necessary for the developer to elaborate the application interface. This interface will be used by doctors to perform some tasks such as data entry.

Indeed, depending on how we define their physical data model, NoSQL systems can be classified into three categories:

(1) Systems where the data model is previously fixed. In other words, like relational systems, attributes names and types must be defined when creating each table. Data entry is only possible after the model has been fully defined. Example: Cassandra (column-oriented system) and Riak TS (key-value system).
(2) Systems where only part of the model is defined before the user inserts data; it usually concerns specifying tables names. This is the case in MongoDB (document-oriented system) and HBase systems (column-oriented system).
(3) Systems where the data model is specified as and when the row is inserted. The user inserts each row specifying the table name as well as the attributes names. Example: Neo4j (graph-oriented system) and Redis (key-value system).

In the last two categories, systems do not require the full definition of the physical model before data entry. However, the developer must have knowledge on the attributes to use and guidelines on how to implement relationships. Thus, in addition to generating the elements needed for creating the NoSQL physical model, our approach provides details and guidelines that allow developer to elaborate the application interface.

4 Object2DOLM transformation

In this section we present the Object2DOLM transformation, which is the first step in our approach as shown in figure 1. We first define the source (UML Class Diagram) and the target (Document-Oriented Logical Model). After that, we focus on the transformation itself.

4.1 Source: UML class diagram

A Class Diagram (CD) is defined as a tuple (N, C, L), where:

N is the class diagram name,
C is a set of classes. Classes are composed from structural and behavioral features. In this paper, we consider the structural features only. Since the operations describe the behavior, we do not consider them. For each class c ∈ C, the schema is a tuple (N, A, IdentO^c), where:

- c.N is the class name,
- c.A = $\{a_1^c, \dots, a_q^c\}$ is a set of q attributes. For each attribute a^c ∈ A, the schema is a pair (N,C) where "a^c.N" is the attribute name and "a^c.C" the attribute type; C can be a predefined class, i.e. a standard data type (String, Integer, Date ...) or a business class (class defined by user),
- c. IdentO^c is a special attribute of c; it has a name IdentO^c.N and a type called "Oid". In this paper, an attribute which type is "Oid" represents a unique object identifier, i.e. an attribute which value distinguishes an object from all other objects of the same class [6].

L is a set of links. Each link l between n classes, with n>=2, is defined as a tuple (N, Ty, Pr^l), where:

- l.N is the link name,
- l.Ty is the link type. In this paper, we consider the three main types of links between classes: Association, Composition and Generalization,
- l.$\text{Pr}^l = \{\text{pr}_1^l, \dots, \text{pr}_n^l\}$ is a set of n pairs. ∀ i ∈ {1,..,n}, pr_i^l = (c, cr^c), where pr_i^l.c is a linked class and $\text{pr}_i^l.\text{cr}^c$ is the multiplicity placed next to c. Note that $\text{pr}_i^l.\text{cr}^c$ can contain a null value if no multiplicity is indicated next to c (like in generalization link).

4.2 Target: Document-Oriented logical model

The Document-Oriented Logical Model (DOLM) mainly shows tables and their interrelationships (binary relationships). In this section, we present this model.

In DOLM, DataBase (DB) is the top level container that owns all the elements. It's defined as a tuple (N, T, R), where:
N is the database name,
T is a set of tables. The schema of each table t ∈ T is a tuple (N, A, IdentL^t), where:

- t.N is the table name,
- t.A = $\{a_1^t, \dots, a_q^t\}$ is a set of q attributes that will be used to define rows of t; each row can have a variable number of attributes. The schema of each attribute a^t ∈ A is a pair

(N,Ty) where “a^t.N” is the attribute name and “a^t.Ty” the attribute type,

- t. $IdentL^t$ is a special attribute of t; it has a name $IdentL^t$.N and a type called “Rid”. In this paper, an attribute which type is “Rid” represents a unique row identifier, i.e. an attribute which value distinguishes a row from all other rows of the same table.

R is a set of binary relationships. Each relationship r $\in$ R between t_1 and t_2 is defined as a tuple (N, Pr^r), where:

- r.N is the relationship name,
- r.Pr^r = $\{pr_1^r, pr_2^r\}$ is a set of two pairs. $\forall$ i $\in$ {1,2}, pr_i^r = (t, cr^t), where pr_i^r.t is a related table and $pr_i^r.\, cr^t$ is the multiplicity placed next to t.

4.3 Transformation Rules

R1: each class diagram CD is transformed into a database DB, where DB.N = CD.N.

R2: each class c $\in$ C is transformed into a table t $\in$ DB, where t.N = c.N, $IdentL^t$.N = $IdentO^c$.N.

R3: each class attribute $a^c \in$ c.A is transformed into a table attribute a^t, where a^t.N = a^c.N, a^t.Ty = a^c.C, and added to the attribute list of its transformed container t such as $a^t \in$ t.A.

R4: each binary link l $\in$ L (regardless of its type: Association, Composition or Generalization) between two classes c_1 and c_2 is transformed into a relationship r $\in$ R between the tables t_1 and t_2 representing c_1 and c_2, where r.N = l.N, r.Pr^r = $\{(t_1, cr^{c_1}),(t_2, cr^{c_2})\}$, cr^{c_1} and cr^{c_2} are the multiplicity placed respectively next to c_1 and c_2.

R5: each link l $\in$ L between n classes $\{c_1, \dots, c_n\}$ (n>=3) is transformed into (1) a new table t^l, where t^l.N = l.N and t^l.A = $\emptyset$, and (2) n relationships $\{r_1, \dots, r_n\}$, $\forall$ i $\in$ {1,..,n} r_i links t^l to another table t_i representing a related class c_i, where r_i.N = (t^l.N)_(t_i.N) and r_i. Pr^r = $\{(t^l, null), (t_i, null)\}$.

R6: each association class c_{asso} between n classes $\{c_1, \dots, c_n\}$ (n>=2) is transformed like a link between multiple classes (R5) using (1) a new table t^{ac}, where t^{ac}.N = l.N, and (2) n relationships $\{r_1, \dots, r_n\}$, $\forall$ i $\in$ {1,..,n} r_i links t^{ac} to another table t_i representing a related class c_i, where r_i.N = (t^{ac}.N)_(t_i.N) and r_i. Pr^r = $\{(t^{ac}, null), (t_i, null)\}$. Like any other table, t^{ac} contain also a set of attributes A, where t^{ac}.A = c_{asso}.A.

We have formalized these transformation rules using the QVT (Query / View / Transformation), which is the OMG standard for models transformation. An excerpt from QVT rules is shown in figure 3.

5 DOLM2DOPM transformation

In this section we present the DOLM2DOPM transformation that generates (1) the physical model of the adopted document-oriented system and (2) the guidelines that assist developers to implement the logical relationships and indicate the attributes that can be used.

5.1 Source: Document-Oriented Logical Model

The source of DOLM2DOPM transformation is the target of the previous transformation Object2DOLM. It’s a document-oriented NoSQL logical model.

5.2 Target: Document-Oriented physical model

To illustrate our approach, we have chosen MongoDB and CouchDB systems. For MongoDB, the corresponding mapping is available in [17].

5.2.1 couchDB

CouchDB database (DB^{CH}) doesn't have tables [16]. It contains a set of documents that are the containers of data. Thus, DB^{CH} is defined as a tuple (N, D), where:

N is the database name,

D is a set of documents. Each document has: (1) an identifier which allows to uniquely reference it in the database, and (2) a set of key-value pairs called properties. Each property is composed of a key that represents its name, and a value that can be atomic or complex (composed of other properties). Formally, the schema of each document d $\in$ D is a tuple (Id^d, PR), where:

- d. Id^d is a unique identifier of d. It has a name Id^d.N and a type Id^d.Ty.
- d.PR = $PR^A \cup PR^{CX}$ is a set of atomic and complex properties that will be used to fill d. The schema of an atomic property $pr^a \in PR^A$ is a pair (Key,Ty) where “pr^a.Key” is the property name and “pr^a.Ty” is the property type. The schema of a complex property $pr^{cx} \in PR^{CX}$ is also a tuple (Key, PR’) where pr^{cx}.Key is the property name and pr^{cx}.PR’ is a set of properties where PR’$\subset$ PR.

5.3 Target: Document-Oriented physical model

5.3.1 couchDB physica model

R1: the logical database DB is transformed into a CouchDB database DB^{CH}, where DB^{CH}.N = DB.N.

5.3.2 Guidelines

CouchDB database (DB^{CH}) doesn't have tables [16]. It contains a set of documents that are the containers of data. Thus, DB^{CH} is defined as a tuple (N, D), where:

N is the database name,

R2: As mentioned before (Section V.B), a CouchDB database contains a set of documents; the concept of "collections" that allows to classify these documents don’t exist. Each row in a logical table will correspond to a document in CouchDB. This document needs to be explicitly associated to the corresponding table. For this, our process creates, for each document d corresponding to a row in a logical table t, a complex property pr_t^{cx} where:

- pr_t^{cx}.Key is composed of (1) the table name and (2) a sequential number,
- pr_t^{cx}.Value contains the property list of d.

For example, for a row in the logical table “Patient”, we have the following CouchDB document:

{

```
Patient_1 : { name : " ",
              profession : " ",
              … }
}
```

Thus, all documents having the property key XXXX_i, will be considered to belong to the logical table XXXX.

Formally, ∀ d ∈ DB^{CH}.D a document corresponding to a row in a logical table t, we create a complex property pr_t^{cx} where:

- pr_t^{cx}.Key = [t.N]_i ; i is a sequential number referring a row in t (example : Patient_1, Doctor_5 …),
- Each attribute $a^t \in t.A^t$ is transformed into a property pr^a, where pr^a.Key = a^t.N, and pr^a.Ty = a^t.Ty, and then added to the property list of its container pr_t^{cx} such as $pr^a \in pr_t^{cx}$.Value.

R3: In CouchDB, the logical relationships could be converted using two forms: references and nested data. Thus, for each relationship r between two tables $\mathbf{t_1}$ and $\mathbf{t_2}$, the following solutions may be considered:

Solution 1: r is transformed into a property pr^{ref} referencing one or more documents that correspond to rows in t_2, where pr^{ref}.key = (t_2.N)_Ref, and then added to the property list of documents that contain the complex property $pr_{t_1}^{cx}$ where $pr_{t_1}^{cx}$.Key = [t_1.N]_i, such as $pr^{ref} \in pr_{t_1}^{cx}$.Value.

Solution 2: r is transformed into a property pr^{ref} referencing one or more documents that correspond to rows in t_1, where pr^{ref}.key = (t_1.N)_Ref, and then added to the property list of documents that contain the complex property $pr_{t_2}^{cx}$ where $pr_{t_2}^{cx}$.Key = [t_2.N]_i, such as $pr^{ref} \in pr_{t_2}^{cx}$.Value.

Solution 3: r is transformed by embedding one or more documents containing the property $\boldsymbol{pr_{t_2}^{cx}}$.Key = [$\boldsymbol{t_2}$.N]_i in documents containing the property $pr_{t_1}^{cx}$.Key = [t_1.N]_i, where $pr_{t_2}^{cx} \in pr_{t_1}^{cx}$.Value.

Solution 4: r is transformed by embedding one or more documents containing the property $pr_{t_1}^{cx}$.Key = [t_1.N]_i in documents containing the property $pr_{t_2}^{cx}$.Key = [t_2.N]_i, where $pr_{t_1}^{cx} \in pr_{t_2}^{cx}$.Value.

The type of the reference property (monovalued or multivalued) used in solutions 1 and 2, as well as the number of documents (one or many) to be nested in solutions 3 and 4, depend on the relationship cardinalities.

In this section, we have proposed different solutions to transform the logical relationships under CouchDB. In order to choose the most suitable solution, the developer can be well guided thanks to the performance measurement shown in Section "Experiments". We have measured the queries response time using each of the proposed solution. The developer will make his choice according to the queries features he needs to perform, the expected performances as well as the queries frequency of use.

6 Experiments

In this section, we show how to transform a UML conceptual model into a document-oriented NoSQL physical model. As presented in the previous section, several solutions can ensure this transformation. We first detail how we implemented the Object2NoSQL process and then we show the experiment we conducted to study the impact that the choice of solution used to map the logical relationships may have on the execution time of queries.

6.1 Implementation

We have implemented the Object2DOLM and DOLM2DOPM transformations using a set of tools provided by Eclipse Modeling Framework (EMF). It's a models transformation environment that contains a set of plugins which can be used to create a model and generate other outputs based on this model. Each transformation is expressed as a sequence of elementary steps that builds the resulting model step by step from the source model. **Step1:** we create the source and the target metamodels using the metamodeling language Ecore. **Step2:** we build an instance of the source metamodel; for this, we use the standard-based XML Metadata Interchange (XMI) format. **Step3:** we implement the transformation rules by means of the Query / View / Transformation (QVT) language (the OMG standard language for specifying model transformations). **Step4:** we test the transformation by running the QVT script of step3; this script takes as input the source model created in step 2 and returns the resulting model in the form of XMI file.

Object2DOLM is the first step in Object2NoSQL process. It transforms the input UML class diagram (figure 2) into the proposed DOLM (figure 4). Object2DOLM transformation is performed by means of the mapping rules defined in section 4. These rules have been formalized using the QVT language; an excerpt from the QVT script is shown in figure 3. The comments in the script indicate the rules used.

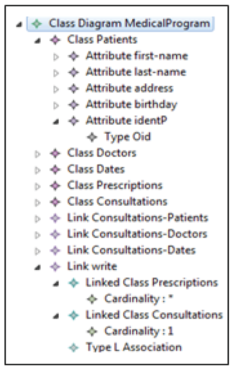
In the second step, the developer indicates the document-oriented system he wants to use (MongoDB for example) and chooses one of the relationship mapping solutions we propose; then the DOLM2DOPM transformation runs (figure 5). Starting from the DOLM created by the previous transformation (Object2DOLM), DOLM2DOPM generates (1) the physical model of the selected system (figure 6) and (2) the associated guidelines (figure 7). We illustrate our experiment using MongoDB.

Note that due to lack of space, we only present excerpts from models and QVT scripts.

6.2 Evaluation

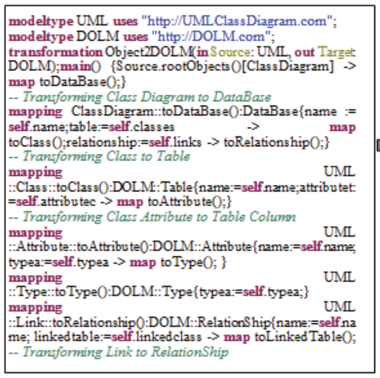
Depending on the functionalities of the document-oriented system selected by the developer, the logical relationships could be mapped into different forms. To assist the developer in choosing the most effective form, we performed an evaluation to study the impact of each mapping solution on the queries execution time.

We carry out the experimental assessment using a cluster made up of 3 machines. Each machine has the following specifications: Intel Core i5, 8GB RAM and 2TB disk.



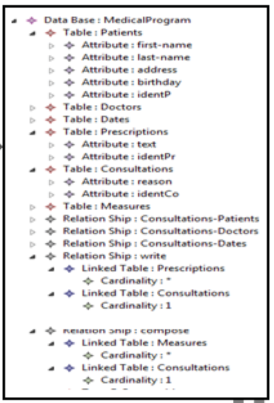
```
Class Diagram MedicalProgram
  Class Patients
    Attribute first-name
    Attribute last-name
    Attribute address
    Attribute birthday
    Attribute identP
      Type Oid
  Class Doctors
  Class Dates
  Class Prescriptions
  Class Consultations
  Link Consultations-Patients
  Link Consultations-Doctors
  Link Consultations-Dates
  Link write
    Linked Class Prescriptions
      Cardinality : *
    Linked Class Consultations
      Cardinality : 1
    Type L Association
```

Fig.2. Source Model



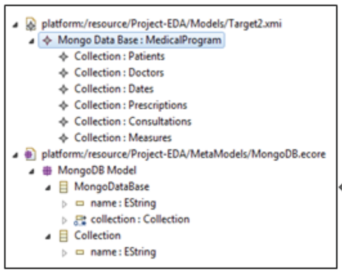
```
modeltype UML uses "http://UMLClassDiagram.com";
modeltype DOLM uses "http://DOLM.com";
transformation Object2DOLM(in Source: UML, out Target:
DOLM);main() {Source.rootObjects()[ClassDiagram] ->
map toDataBase();}
-- Transforming Class Diagram to DataBase
mapping ClassDiagram::toDataBase():DataBase{name :=
self.name;table:=self.classes -> map
toClass();relationship:=self.links -> toRelationship();}
-- Transforming Class to Table
mapping UML
::Class::toClass():DOLM::Table{name:=self.name;attributet:
=self.attributec -> map toAttribute();}
-- Transforming Class Attribute to Table Column
mapping UML
::Attribute::toAttribute():DOLM::Attribute{name:=self.name;
typea:=self.typea -> map toType(); }
mapping UML
::Type::toType():DOLM::Type{typea:=self.typea;}
mapping UML
::Link::toRelationship():DOLM::RelationShip{name:=self.na
me; linkedtable:=self.linkedclass -> map toLinkedTable();
-- Transforming Link to RelationShip
```

Fig.3. Object2DOLM



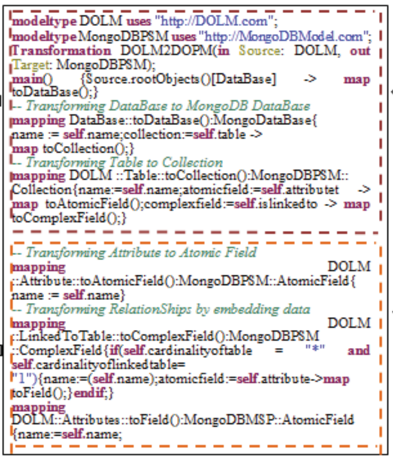
```
Data Base : MedicalProgram
  Table : Patients
    Attribute : first-name
    Attribute : last-name
    Attribute : address
    Attribute : birthday
    Attribute : identP
  Table : Doctors
  Table : Dates
  Table : Prescriptions
    Attribute : text
    Attribute : identPr
  Table : Consultations
    Attribute : reason
    Attribute : identCo
  Table : Measures
  Relation Ship : Consultations-Patients
  Relation Ship : Consultations-Doctors
  Relation Ship : Consultations-Dates
  Relation Ship : write
    Linked Table : Prescriptions
      Cardinality : *
    Linked Table : Consultations
      Cardinality : 1

  Relation Ship : compose
    Linked Table : Measures
      Cardinality : *
    Linked Table : Consultations
      Cardinality : 1
```

Fig.4. Target Model



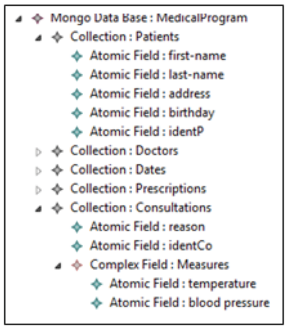
```
platform:/resource/Project-EDA/Models/Target2.xmi
  Mongo Data Base : MedicalProgram
    Collection : Patients
    Collection : Doctors
    Collection : Dates
    Collection : Prescriptions
    Collection : Consultations
    Collection : Measures
platform:/resource/Project-EDA/MetaModels/MongoDB.ecore
  MongoDB Model
    MongoDataBase
      name : EString
      collection : Collection
    Collection
      name : EString
```

Fig.6. MongoDB Model



```
modeltype DOLM uses "http://DOLM.com";
modeltype MongoDBPSM uses "http://MongoDBModel.com";
Transformation DOLM2DOPM(in Source: DOLM, out
Target: MongoDBPSM);
main() {Source.rootObjects()[DataBase] -> map
toDataBase();}
-- Transforming DataBase to MongoDB DataBase
mapping DataBase::toDataBase():MongoDataBase{
name := self.name;collection:=self.table ->
map toCollection();}
-- Transforming Table to Collection
mapping DOLM ::Table::toCollection():MongoDBPSM::
Collection{name:=self.name;atomicfield:=self.attributet ->
map toAtomicField();complexfield:=self.islinkedto -> map
toComplexField();}

-- Transforming Attribute to Atomic Field
mapping DOLM
::Attribute::toAtomicField():MongoDBPSM::AtomicField{
name := self.name}
-- Transforming RelationShips by embedding data
mapping DOLM
::LinkedToTable::toComplexField():MongoDBPSM
::ComplexField{if(self.cardinalityoftable = "*" and
self.cardinalityoflinkedtable=
"1"){name:=(self.name);atomicfield:=self.attribute->map
toField();}endif;}
mapping
DOLM::Attributes::toField():MongoDBMSP::AtomicField
{name:=self.name;
```

Fig.5. DOLM2DOPM Transformation



```
Mongo Data Base : MedicalProgram
  Collection : Patients
    Atomic Field : first-name
    Atomic Field : last-name
    Atomic Field : address
    Atomic Field : birthday
    Atomic Field : identP
  Collection : Doctors
  Collection : Dates
  Collection : Prescriptions
  Collection : Consultations
    Atomic Field : reason
    Atomic Field : identCo
    Complex Field : Measures
      Atomic Field : temperature
      Atomic Field : blood pressure
```

Fig.7. Guidelines

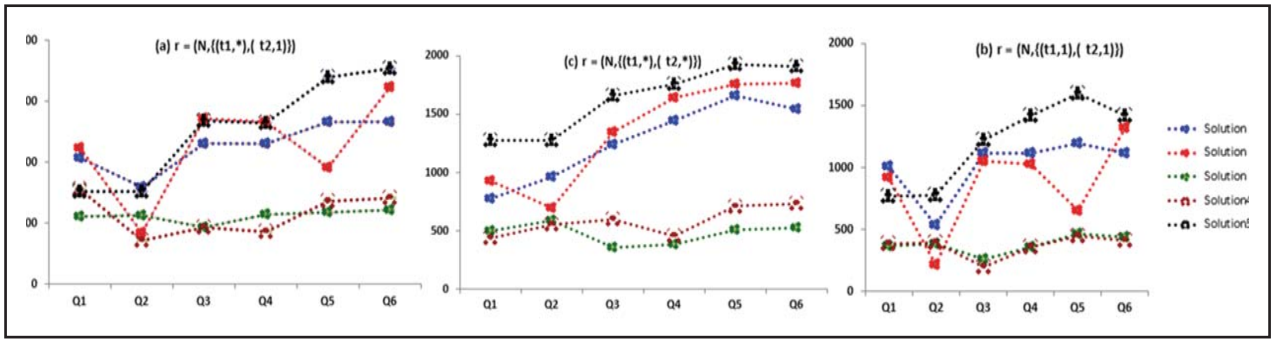


Fig.8. Experimental Results

On the other hand, we have used data generator tools to generate a dataset of about 1TB with JSON format. These files are loaded into the systems using shell commands. For queries set, we have written 6 queries which concern two tables and the relationship between them. These queries involve the two related tables and gradual increase in the number of attributes to return. An excerpt from our experiment results is depicted in figure 8. For each query, we indicate the obtained response time according to (1) the relationship cardinalities and (2) the transformation solution used. Thus, the developer can make his choice using our experiment results. This choice will be based on the following criteria: (1) The features of queries (number of filters, number of attributes to return) and (2) How frequently each query will be used. We note that due to lack of place, only the results obtained under MongoDB are presented.

7 Related work

Big Data applications developers have to deal with the question: how to store Big Data in NoSQL systems? To address this problem, existing solutions propose to model Big Data, and then define mapping rules towards the physical level. In the specific context of a data warehouse, both [12] and [11] have defined a set of rules to transform a multidimensional model into a NoSQL model. Other studies [9] and [8] have investigated the process of transforming relational databases into a NoSQL model. To the best of our knowledge, only few works have presented approaches to implement UML conceptual model into NoSQL systems. Li et al. [10] propose a MDA-based process to transform UML class diagram into column-oriented model specific to HBase. Starting from the UML class diagram and HBase metamodels, authors have proposed mapping rules between the conceptual level and the physical one. Obviously, these rules are applicable to HBase, only. Gwendal et al. [3] describe the mapping between a UML conceptual model and graph databases via an intermediate graph metamodel. In this work, the transformation rules are specific to graph databases used as a framework for managing complex data with many connections. Generally, this kind of NoSQL systems is used in social networks where data are highly connected. In [14] data modeling in MongoDB database has been shown by using class diagram and JSON format to represent the documents. Similarly, Banker [15] provides some tools of data modeling, but limited to MongoDB database and always referring to JSON format as a modeling solution.

Regarding the state of the art, some of the existing works [9] and [8] focus on relational model that, unlike UML class diagram, lacks of semantic richness, especially through the several types of relationships that exist between classes. Other solutions, [12] and [11] have the advantage to start from the conceptual level. But, the proposed models are Domain-Specific (Data Warehouses system), so they consider fact, dimension, and typically one type of links only. Approaches proposed in [10] and [3] are only applicable to column-oriented [10] and graph-oriented [3] data stores. [14] and [15] present a study of techniques and tools for data modeling using MongoDB system; the proposed solutions are not generic, they are restricted to MongoDB document database. However, it makes more sense to generalize the transformation process in order to allow the user to choose the target document-oriented system that suits the best with business rules and technical constraints.

8 Conclusion and future work

This paper provides an automatic MDA approach that generates NoSQL physical models starting from a UML conceptual model. Our approach relies on a pivot logical model that uses tables and binary relationships. This model exhibits a sufficient degree of platform-independency, so that its mapping to one or more NoSQL document-oriented platforms is feasible. The advantage of using a unified (logical) model is that this model remains stable, whenever the NoSQL systems evolve or even if we decide to completely change the used platform. In these two cases, it would be enough to evolve the physical (platform-dependent) models, and of course adapt the transformation rules; this simplifies the transformation process and saves developers efforts and time.

Our approach is based on two main steps. The first one automatically creates the logical model starting from a UML class diagram. In the second step, the developer chooses one of the relationship implementation solutions we propose, and the NoSQL physical models are generated starting from the logical model. Our approach assists the developer to choose the most suited implementation to the project he is working on. We have measured the queries response time using each of the proposed transformation solution. Thus, the developer can choose the most suited solution according to: (1) the queries features (number of filters, number of attributes to return) and (2) the queries frequency of use.

We are currently working on automating the overall process. The choice of the relationship implementation solution will be done by the system itself without requiring developer intervention. We also plan to complete and generalize our transformation process to consider the constraints defined in the conceptual model; once the NoSQL physical model is created, another process has to be performed to check these constraints.

9 References

[1] A. Abelló, “Big data design”, in OLAP, 2015.

[2] V. Herrero, A. Abelló, O. Romero, “NOSQL design for analytical workloads: variability matters”, in ER, 2016.

[3] D. Gwendal, S. Gerson, C. Jordi. “Mapping Conceptual Schemas to Graph Databases”. In ER, 2016.

[4] J. Hutchinson, M.Rouncefield, and J.Whittle. "Model-driven engineering practices in industry", in ICSE, 2011.

[5] J. .Bézivin and O. Gerbé. "Towards a Precise Definition of the OMG/MDA Framework", in ASE, 2001.

[6] “About the Unified Modeling Language Specification Version 2.5.” [Online]. Available: http://www.omg.org/spec/UML/2.5/.

[7] A. Angadi, Ak. Angadi, Karuna. Gull. “Growth of New Databases & Analysis of NOSQL Datastores”, in IJARCSSE, 2013.

[8] T. Vajk, P. Feher, K. Fekete, H. Charaf. “Denormalizing data into schema-free databases”, in CogInfoCom, 2013.

[9] C. Li. "Transforming relational database into HBase: A case study", in ICSESS, 2010.

[10] Yan Li, Ping Gu. "Transforming UML Class Diagrams into HBase Based on Meta-model”, in ISEEE, 2014.

[11] Dehdouh, K., Bentayeb, F., Boussaid, O., Kabachi, N. “Using the column oriented model for implementing big data warehouses”, in PDPTA, 2015.

[12] M. Chevalier, M. El Malki, A. Kopliku, O. Teste, and R. Tournier, “How can we implement a Multidimensional Data Warehouse using NoSQL?”, in ICAIS, 2015.

[13] R.Kumar, S.Charu, S.Bansal. "Effective Way to Handling Big Data Problems using NoSQL Database ", In JoADMS, 2015.

[14] R. Arora, R. Aggarwal. “Modeling and querying data in mongodb”, in IJSER, 2013.

[15] K. Banker. MongoDB in action, Manning Publications Co, 2011.

[16] “Apache CouchDB 2.1 Documentation”. [Online]. Available: http://docs.couchdb.org/en/2.1.1/.

[17] Abdelhedi, F., AIT Brahim, A., Atigui, F., & Zurfluh. “MDA-Based Approach for NoSQL Databases Modelling”, in DaWaK, 2017.