MDA Process to Extract the Data Model from Document-oriented NoSQL Database

Amal Ait Brahim, Rabah Tighilt Ferhat and Gilles Zurfluh
Toulouse Institute of Computer Science Research (IRIT), Toulouse Capitole University, Toulouse, France

Keywords: Big Data, NoSQL, Model Extraction, Schema Less, MDA, QVT.

Abstract: In recent years, the need to use NoSQL systems to store and exploit big data has been steadily increasing. Most of these systems are characterized by the property "schema less" which means absence of the data model when creating a database. This property brings an undeniable flexibility by allowing the evolution of the model during the exploitation of the base. However, query expression requires a precise knowledge of the data model. In this article, we propose a process to automatically extract the physical model from a document-oriented NoSQL database. To do this, we use the Model Driven Architecture (MDA) that provides a formal framework for automatic model transformation. From a NoSQL database, we propose formal transformation rules with QVT to generate the physical model. An experimentation of the extraction process was performed on the case of a medical application.

1 INTRODUCTION

Recently, there has been an explosion of data generated and accumulated by more and more numerous and diversified computing devices. Databases thus constituted are designated by the expression "Big Data" and are characterized by the so-called "3V" rule (Chen, 2014). This is due to the volume of data that can exceed several terabytes and the variety of these data that are described as complex. In addition, these data are often entered at very high frequency and must therefore be filtered and aggregated in real time to avoid unnecessary saturation of the storage space.

Traditional implantation techniques, based primarily on the relational paradigm, have limitations in managing massive databases (Angadi, 2013). Thus, new data storage and manipulation systems have been developed. Grouped under the term NoSQL (Han, 2011), these systems are well suited for managing large volumes of data with flexible models. They also bring great scalability and good performance in response time (Angadi, 2013).

Most of the NoSQL DBMS are characterized by the "schema less" property which corresponds to the absence of the data schema when creating a database. This property appears in many NoSQL systems such as MongoDB, CouchDB, HBase and Neo4j. Note however that it is absent in some systems such as Cassandra and Riak TS. The "schema less" property offers undeniable flexibility by allowing the model to evolve easily. For example, the addition of new attributes in an existing line is done without modifying the other lines of the same type previously stored; something that is not possible with relational DBMS, where all elements of the model are fixed before data entry. However, the model of a database is an essential knowledge element for data manipulation. Indeed, the knowledge of the model of the base proves necessary, even indispensable, to express a query where appear the names of the tables, the names of the attributes and values compatible with a type. And this is all the more important if the queries are written by decision-makers, who are not supposed to be non-computer scientists.

Currently, NoSQL systems characterized by the property "schema less" do not have a feature to dynamically display the database model. In this article, we propose a process to automatically extract the model from the data stored on a NoSQL DBMS. The goal is to allow users to visualize the data model on demand.

The rest of the paper is structured as follows: Section 2 motivates our work using a case study in the healthcare field; Section 3 reviews previous work on extracting the data model; Section 4 introduces our MDA-based approach; A model to model

Brahim, A., Ferhat, R. and Zurfluh, G.
MDA Process to Extract the Data Model from Document-oriented NoSQL Database.
DOI: 10.5220/0007676201410148
In *Proceedings of the 21st International Conference on Enterprise Information Systems (ICEIS 2019)*, pages 141-148
ISBN: 978-989-758-372-8

transformation is presented in this section to automatically extract the data model from a NoSQL database; Section 5 details our experiments; Section 6 presents the positioning of our work and Section 7 concludes the paper and announces future work.

2 MOTIVATION

To motivate and illustrate our work, we present a case study in the healthcare filed. This case study concerns international scientific programs for monitoring patients suffering from serious diseases. The main goal of this program is (1) to collect data about diseases development over time, (2) to study interactions between different diseases and (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 this kind of program have the features of Big Data (the 3 V) (Doug, 2001). Indeed, the amount of data collected daily from all the establishments in three years can reach several terabytes. Furthermore, data entered while monitoring patients come in different types; it could be structured as the patient's vital signs (respiratory rate, blood pressure, etc.), semi-structured document such as the package leaflets of medicinal products, unstructured such as consultation summaries, paper prescriptions and radiology reports. Finally, some data are produced in continuous way by sensors; it needs a real time process because it could be integrated into a time-sensitive processes (for example, some measurements, like temperature, require an emergency medical treatment if they cross a given threshold).

This is a typical example in which the use of a NoSQL system is suitable. On the one hand, in the medical application, briefly presented above, the database contains structured data, data of various types and formats (explanatory texts, medical records, x-rays, etc.), and big tables (records of variables produced by sensors). On the other hand, NoSQL data stores are ideally suited to this kind of applications that need a database which can cope with large amounts of disparate data. Therefore, we are convinced that a NoSQL DBMS, like MongoDB, is the most adapted system to store the medical data.

As an illustration, Figure 1 gives an excerpt from the data model of the medical application. This is the graphical description of the data structures stored in the MongoDB (MongoDB, 2018) system that we used in our experiment. Note now that MongoDB is a "schema less" system, it does not provide this model, either in textual form or in graphical form.

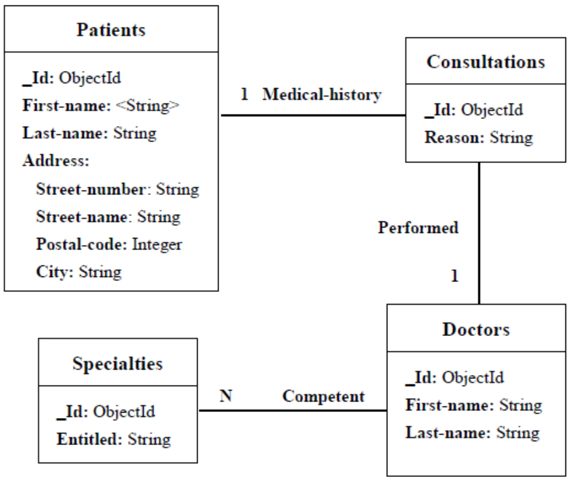


Figure 1: Excerpt from the physical model of data.

This case study is a typical example of applications where users need a tool to display the database model. Indeed, doctors enter measures regularly for a cohort of patients. They can also recording new data in cases where the patient's state of health evolve over time. Few months later, doctors will analyze the entered data in order to follow the evolution of the pathology. For this, they need to use a model to express their queries.

In our view, it's important to have a precise and automatic solution that guides and facilitates the data model extraction task within NoSQL systems. For this, we propose the Query2Model process presented in the next section that extracts the physical model of a database stored in MongoDB.

3 RELATED WORK

In industry, several integration systems and access to heterogeneous data such as Apache Drill (Drill, 2018), CloudMdsQL (CloudMdsQL, 2018) and BigIntegrator (BigIntegrator, 2018), allow to extract the physical model of a NoSQL database, (Bondiombouy, 2015). For example Apache-Drill, which appears as the most successful system, allows to query heterogeneous data stored on different types of systems. The user can obtain the data model by applying a shell script to the NoSQL database.

On the other hand, research work has been proposed in order to extract a physical model from a NoSQL database of type "schema less", mainly for document-oriented databases such as MongoDB.

Thus, a process has been proposed in (Klettke, 2015) to extract the model from a collection of JSON documents stored on MongoDB. The model returned by this process is in JSON format; it is obtained by capturing the names of the attributes that appear in the input documents and replacing their values with their types. Attribute values can be atomic type, lists or nested documents.

In the article of (Sevilla, 2015), the authors propose another process of extraction of the model of a document-oriented NoSQL database which can include several collections. The returned result is not a unified model for the whole database but it gives the different versions of models for each collection. The extraction process is composed of two successive steps. The first one runs through the database and, for each distinct template version, generates a document in a collection called "Template". In the second step, the process provides a model of each version by instantiating the JSON meta-model.

We can also mention the work of (Gallinucci, 2018) which proposes a process called BSP (Build Schema Profile) to classify the documents of a collection by applying rules corresponding to the requirements of the users. These rules are expressed through a decision tree whose nodes represent the attributes of the documents; the edges specify the conditions on which the classification is based. These conditions reflect either the absence or the presence of an attribute in a document or its value. As in the previous article (Sevilla, 2015), the result returned by this approach is not a unified model but a set of version models; each of them is common to a group of documents.

Regarding the state of the art, the solutions proposed to extract the model of a NoSQL database, only partially answer our problem. Indeed, in (Klettke, 2015) and (Gallinucci, 2018), the authors propose processes that take as input a single collection of documents. As a result, the links between the collections are not studied. Similarly, the work of (Sevilla, 2015) does not deal with links although they consider several collections. In our process, we propose a solution to take into account the links between the collections.

4 ToNoSQLmodel PROCESS

The purpose of this article is to automate the extraction of the model from NoSQL databases of type "schema less" which generally divided into four categories: key / value, columns, documents and graphs. We limit ourselves to the type documents which is the most complete in terms of expression of the links (use of references and nestings). ToNoSQLmodel process that we propose, automatically extracts the model from a document-oriented NoSQL database.

To formalize and automate our process, we use OMG's Model Driven Architecture (OMG, 2018), which provides a formal framework for automating model transformations. The purpose of this architecture is to describe separately the functional specifications and implementation specifications of an application on a given platform (Hutchinson, 2011). For this, it uses three models representing the abstraction levels of the application. These are (1) the Computational Independent Model (CIM) in which no IT considerations appear, (2) the Independent Platform Independent Model (PIM) model of analysis and design. Execution platforms and (3) Platform Specific Model (PSM) specific to a particular platform. Since the input of our process is a NoSQL database and its output is a physical model, we only retain the PSM level.

The passage of a NoSQL database to its model is done via a sequence of transformations. We will formalize these transformations using the standard QVT (Query View Transformation) defined by the EMF. Figure 2 shows an overview of our process.

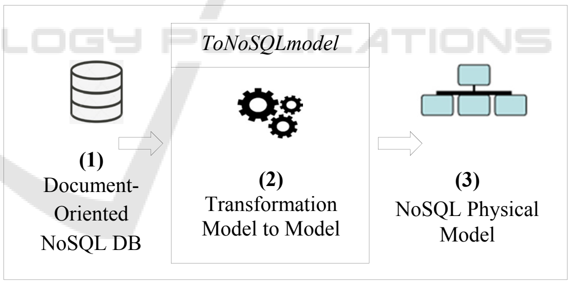


Figure 2: Overview of ToNoSQLmodel process.

In the following sections, we detail the components of our process by specifying the following three elements: (a) the source, (b) the target, and (c) the transformation rules.

4.1 Source

A document-oriented NoSQL database (DB) is defined as a pair (N, CLL), where:

- N is the DB name,
- CLL = $\{cll_1, \ldots, cll_n\}$ is a set of collections

∀ i ∈ [1..n], cll_i ∈ DB. CLL is a pair (N, FL^{IN}), where:

- cll_i.N the collection name,
- $cll_i.FL^{IN}$ = AFL^{IN} ∪ CFL^{IN}, is a set of input fields of cll_i , where:
  - AFL^{IN} = $\{afl_1^{IN}, \ldots, afl_k^{IN}\}$ is a set of atomic fields, where:

∀ i ∈ [1..k], afl_i^{IN} ∈ AFL^{IN} is defined as a pair (N, V), where:

- afl_i^{IN}.N is the name of afl_i^{IN},
- afl_i^{IN}.V is the value of afl_i^{IN},
- CFL^{IN} = $\{cfl_1^{IN}, \ldots, cfl_l^{IN}\}$ is a set of complex fields, where:

∀ i ∈ [1..l], cfl_i^{IN} ∈ CFL^{IN} is defined as a pair (N, $FL^{IN\prime}$), where:

- cfl_i^{IN}.N is the name of cfl_i^{IN},
- $cfl_i^{IN}.FL^{IN\prime}$ ∈ FL^{IN} is the set of fields that cfl_i^{IN}contains.

To express a link between the collections, we used a field called: reference field, denoted by ch^{ref} (MongoDB, 2018). This one is a special case of a complex field. ch^{ref} is composed of two atomic fields ch_1^{ref} and ch_2^{ref}, each of them is defined as a pair (N, V), where:

- ch_1^{ref}.N = $id
- ch_1^{ref}.V : corresponds to the identifier of the referenced document

And,

- ch_2^{ref}.N = $ref
- ch_2^{ref}.V : is the name of the collection that contains the referenced document.

We present these different concepts through the meta-model of Figure 3. Note that all the meta-models presented in this article are formalized with the standard Ecore language (Ecore, 2018).

4.2 Target

The NoSQL model noted M generated by our process, is stored in a collection cll^{Model}. This is defined as a pair (N, D), where:

- cll^{Model}. N is the model name,
- cll^{Model}. D = $\{d_1, \ldots, d_n\}$ is a set of documents that cll^{Model} contains.

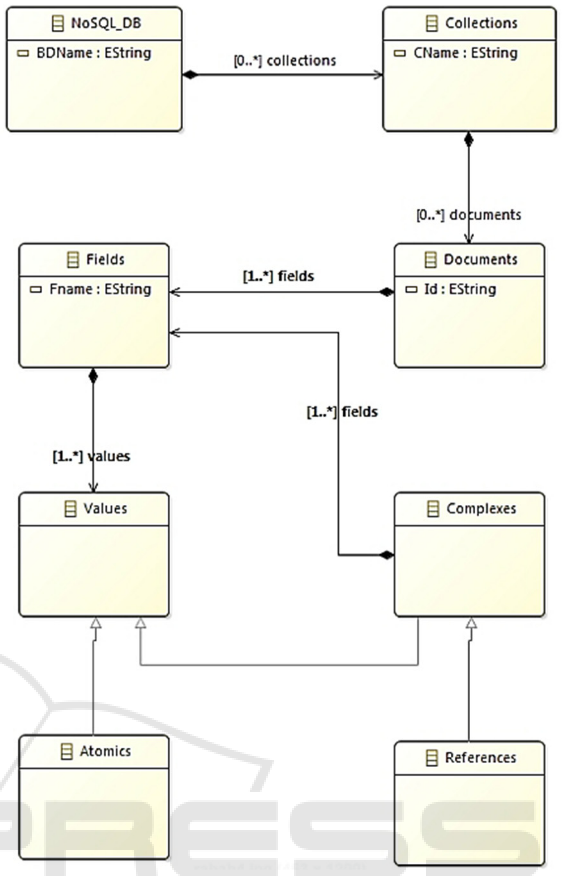


Figure 3: Source Metamodel.

∀ i ∈ [1..n], d_i is defined as a pair (Id, FL^{OUT}), where

- d_i. Id is the identifier of d_i,
- $d_i.FL^{OUT}$= $\{AFL^{OUT}, \ldots, CFL^{OUT}\}$ is a set of imput fields of d_i, where :
  - AFL^{OUT} = $\{afl_1^{OUT}, \ldots, afl_k^{OUT}\}$ is a set of atomic fields of d_i, where:

∀ i ∈ [1..k], afl_i^{OUT} ∈ AFL^{OUT} is defined as a pair (N, Ty), where:

- afl_i^{OUT}.N is the name of afl_i^{OUT},
- afl_i^{OUT}.Ty is the type of afl_i^{OUT}.

Note that the type of afl_i^{OUT} can be either predefined (for example: String, Boolean, Integer, ...) or defined by the user (for example: Patient, Doctors, ...).

- CFL^{OUT} = $\{cfl_1^{OUT}, \ldots, cfl_l^{OUT}\}$ is a set of complex fields of d_i, where:

∀ i ∈ [1..l], cfl_i^{OUT} ∈ CFL^{OUT} is defined as a pair (N, $FL^{OUT\prime}$), where:

- cfl_i^{OUT}.N is the name of cfl_i^{OUT},
- $cfl_i^{OUT}.FL^{OUT\prime}$ ∈ FL^{OUT} is the set of fields that cfl_i^{OUT}contains.

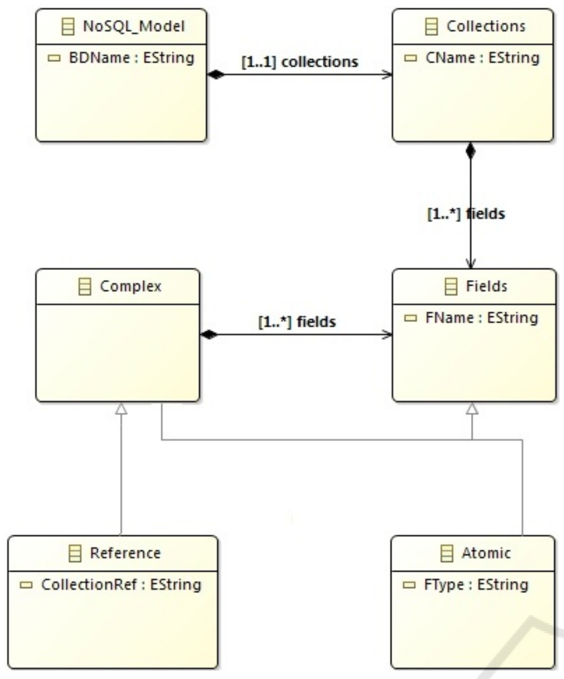


Figure 4: Target Metamodel.

4.3 Transformation Rules

We have formalized the concepts present in the source (document-oriented database) and in the target (NoSQL physical model). In this section, we present our process as a sequence of transformation rules described below.

R1: The DB model is stored in a collection cll^{model}. This is defined as a pair (N,D), where:

- cll^{model}.N= DB.N,
- cll^{model}.D is generated by applying R2.

R2: For each collection cll_i ∈ DB. CLL with i ∈ [1..n], we create a document d_i, where:

- d_i.N = $cll_i.N$
- $d_i.FL^{OUT}$ is generated by applying R3 or R4.

Note that $d_i$contains a unified template for all documents that cll_i contains. This means that our process generates a unique collection model grouping all the fields of the documents. We therefore do not consider several versions of models for the same input collection.

R3: Each atomic field afl_j^{IN} ∈ cll_i.AFL^{IN} is transformed into a field afl_j^{OUT} with i ∈ [1..n] and j ∈ [1..k], where:

- $afl_j^{OUT}.N = afl_j^{IN}$.N
- $afl_j^{OUT}.Ty$ is generated according to the form of the value of afl_j^{IN}.

For example, if $afl_j^{IN}.V$ = " ", then $afl_j^{OUT}.Ty$ = String. And, If $afl_j^{IN}.V$ = {" "," ", ... " "}, then $afl_j^{OUT}.Ty$ = Set (String).

R4: Each complex field cfl_j^{IN} ∈ cll_i.CFL^{IN} is transformed into a field cfl_j^{OUT} with i ∈ [1..n] and j ∈ [1..l], where:

- $cfl_j^{OUT}.N = cfl_j^{IN}$.N
- $cfl_j^{OUT}.FL^{OUT'}$ is generated as follows:

  - Apply R3 for each atomic field afl^{IN} ∈ $cfl_j^{IN}.AFL^{IN'}$.

  - Apply the R4 for each complex field cfl^{IN} ∈ $cfl_j^{IN}.CFL^{IN'}$

R5: A reference field ch^{ref} is transformed into a complex field cfl_j^{OUT} with j ∈ [1..2], where :

- cfl_1^{OUT}. N = ch_1^{ref}.N
- cfl_1^{OUT}. Ty = ObjectID
- cfl_2^{OUT}. N = ch_2^{ref}.N
- cfl_2^{OUT}. Ty = ch_2^{ref}.V

5 EXPERIMENTS

5.1 Technical Environment

We briefly describe the techniques we used to implement the approach presented in Figure 2. Since our approach is model driven, we used a technical environment suitable for modeling, meta-modeling and model transformation. We used the Eclipse Modeling Framework (EMF) (EMF, 2018). EMF provides a set of tools for introducing a model-driven development approach within the Eclipse environment. These tools provide three main features. The first is the definition of a meta-model representing the concepts used by the user. The second is the creation of the models instantiating this meta-model and the third is the transformation from model to model and from model to text (Budinsky, 2004). Among the tools provided by EMF, we used: (1) Ecore: a meta-modeling language used for the creation of our meta-models. Figure 3 and Figure 4 illustrate the source and target Ecore meta-models used by our ToNoSQLmodel process. (2) XML Metadata Interchange (XMI): which is a standard used to represent models in XML. (3) QVT (Query, View, and Transformation): which is a standardized language for expressing model transformations. The choice of QVT was based on criteria specific to our approach. Indeed, the transformation tool must be integrated into the EMF environment so that it can be easily used with modeling and meta-modeling tools. Thus, we used the operational QVT language.

We have implemented our process on medical data presented in section 2. The essential aspect, showing the interest of our process, lies in the variety of data used: text, multivalued data, and structured documents. Since we did not study the performance of our prototype, the Volume dimension of the data was not significant; our experiment was therefore limited to a BD of about 500GB.

5.2 Implantation of the ToNoSQLmodel Process

Our ToNoSQLmodel process is expressed in the form of a sequence of elementary steps that build the resulting model (NoSQL physical model) step by step from the source model (document-oriented NoSQL database):

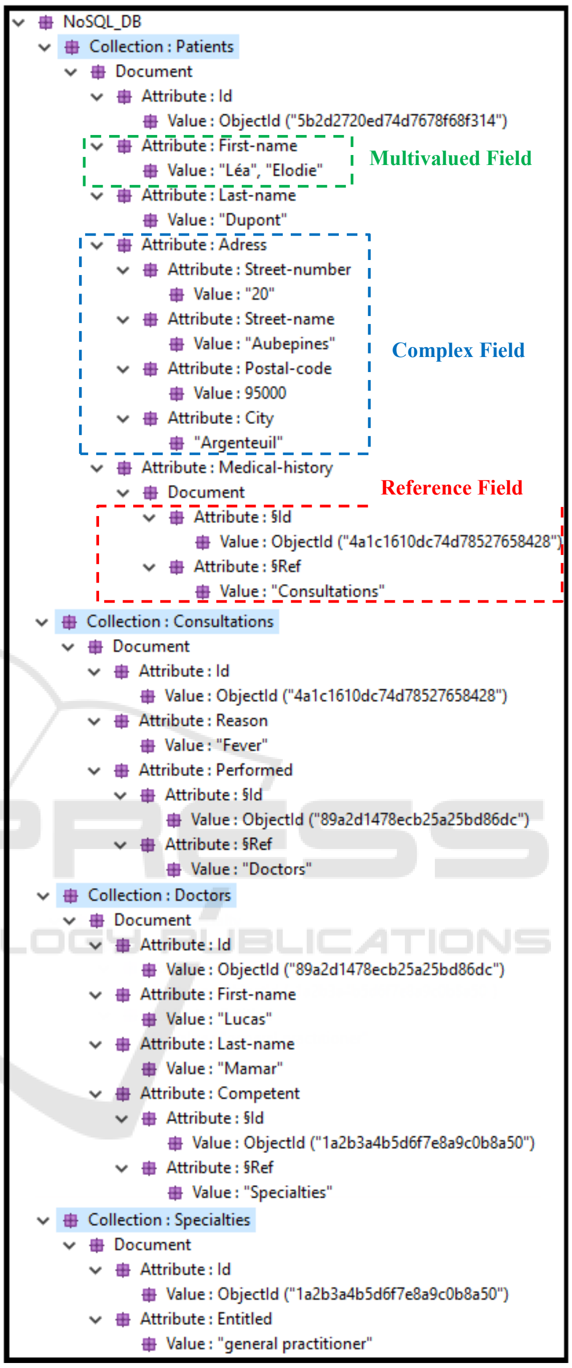
Step 1: We create Ecore meta-models corresponding to the source (Figure 3) and the target (Figure 4).

Step 2: we build an instance of the source meta-model to produce the document-oriented NoSQL database (see Figure 5); this database is an extract from the medical application data described in section 2 and stored as an XMI file.

Step 3: We implement the transformation rules using the QVT language provided by EMF. An excerpt from the QVT script is shown in Figure 7; the comments in the script indicate the rules used.

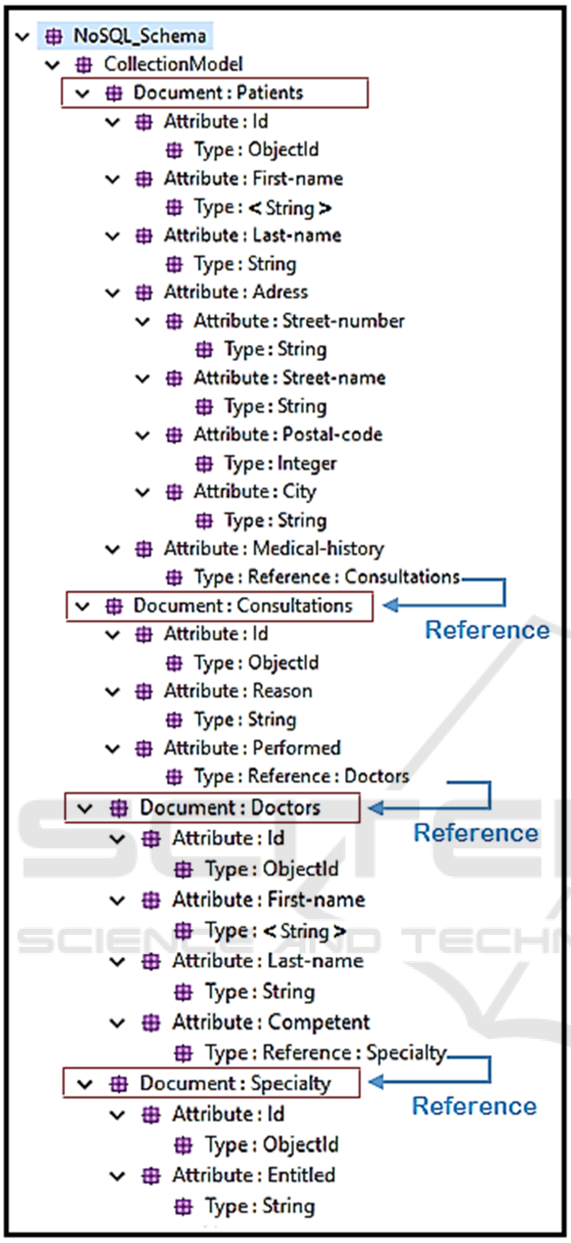
Step 4: We test the transformations by running the QVT script created in step 3. This script takes as input the source model created in step 2 and outputs the NoSQL physical model. The result is provided as an XMI file, as shown in Figure 6.

Our source database contains various data as shown in Figure 5: multivalued fields (such as First-name in the Patients collection), complex fields (Address in the Patients collection) as well as monovalued (Medical-history) and multivalued links (Competent).



```
NoSQL_DB
  Collection : Patients
    Document
      Attribute : Id
        Value : ObjectId ("5b2d2720ed74d7678f68f314")
      Attribute : First-name
        Value : "Léa", "Elodie"
      Attribute : Last-name
        Value : "Dupont"
      Attribute : Adress
        Attribute : Street-number
          Value : "20"
        Attribute : Street-name
          Value : "Aubepines"
        Attribute : Postal-code
          Value : 95000
        Attribute : City
          "Argenteuil"
      Attribute : Medical-history
        Document
          Attribute : $Id
            Value : ObjectId ("4a1c1610dc74d78527658428")
          Attribute : $Ref
            Value : "Consultations"
  Collection : Consultations
    Document
      Attribute : Id
        Value : ObjectId ("4a1c1610dc74d78527658428")
      Attribute : Reason
        Value : "Fever"
      Attribute : Performed
        Attribute : $Id
          Value : ObjectId ("89a2d1478ecb25a25bd86dc")
        Attribute : $Ref
          Value : "Doctors"
  Collection : Doctors
    Document
      Attribute : Id
        Value : ObjectId ("89a2d1478ecb25a25bd86dc")
      Attribute : First-name
        Value : "Lucas"
      Attribute : Last-name
        Value : "Mamar"
      Attribute : Competent
        Attribute : $Id
          Value : ObjectId ("1a2b3a4b5d6f7e8a9c0b8a50")
        Attribute : $Ref
          Value : "Specialties"
  Collection : Specialties
    Document
      Attribute : Id
        Value : ObjectId ("1a2b3a4b5d6f7e8a9c0b8a50")
      Attribute : Entitled
        Value : "general practitioner"
```

Figure 5: Source Model.



```
NoSQL_Schema
  CollectionModel
    Document : Patients
      Attribute : Id
        Type : ObjectId
      Attribute : First-name
        Type : < String >
      Attribute : Last-name
        Type : String
      Attribute : Adress
        Attribute : Street-number
          Type : String
        Attribute : Street-name
          Type : String
        Attribute : Postal-code
          Type : Integer
        Attribute : City
          Type : String
      Attribute : Medical-history
        Type : Reference : Consultations
    Document : Consultations
      Attribute : Id
        Type : ObjectId
      Attribute : Reason
        Type : String
      Attribute : Performed
        Type : Reference : Doctors
    Document : Doctors
      Attribute : Id
        Type : ObjectId
      Attribute : First-name
        Type : < String >
      Attribute : Last-name
        Type : String
      Attribute : Competent
        Type : Reference : Specialty
    Document : Specialty
      Attribute : Id
        Type : ObjectId
      Attribute : Entitled
        Type : String
```

Figure 6: Target Model.

Figure 6 shows the data model resulting from our ToNoSQLmodel process. This model is generated in the Json formalism but it corresponds to the graphical representation that we gave in Figure 1.

```
    modeltype NoSQL_DB uses
"http://nosqldatabaseMM.com";                    The script input
    modeltype NoSQL_Schema uses
"http://nosqlschemaMM.com";
    transformation NoSQLdb2NoSQLschema(in Source:
NoSQL_DB, out Target: NoSQL_Schema);
    main() {                                     The script output
    Source.rootObjects()[NoSQL_DB] -> map
toNoSQL_Schema();}
    mapping NoSQL_DB
::NoSQL_DB::toNoSQL_Schema():NoSQLSchema::NoSQL
_Schema{
    sName:=self.dbName;
    collection:=self.collections -> map toCollection();}
    -- Transforming Collections
    mapping Insert
::Collections::toCollection():Update::Collection{
    cName:=self.cName;
    atomicufield:=self.atomicifield -> map toAtomicField();
    structurdufield:=self.structuredifield -> map
toStructuredField();}
    -- Transforming Atomic Fields
    mapping Insert
::AtomicIField::toAtomicField():Update::AtomicUField{
    fielduname:=self.fieldiname -> map toFieldName();
    fielduvalue:=self.fieldivalueform -> map
toFieldValue1();
    fielduvalue:=self.fieldivalue -> map toFieldValue2();}
    mapping Insert
::FieldIName::toFieldName():Update::FieldUName{NameU:
=self.NameI;}
    mapping
Insert::FieldIValue::toFieldValue1():Update::FieldUValue{
    if    self FieldIValue             or   self FieldIValue
           FieldUValue
    FieldUValue              endif }
    mapping
Insert::FieldIValueForm::toFieldValue2():Update::FieldUValue
{
    if  self FieldIValueForm         FieldUValue
endif
    if   self FieldIValueForm                    FieldUValue
        endif }
    -- Transforming Structured Fields
     mapping                                          Insert
::StructuredIField::toStructuredField():Update::StructuredUFiel
d{
```

Figure 7: Excerpt of the QVT code.

6 POSITIONING OF OUR WORK

We position our work against Apache Drill system and the state-of-the-art presented in Section 3.

Apache-Drill system does not display a complete model for data stored under MongoDB. Indeed, only the names of the collections and the fields of the first level are displayed, which means that the nested fields do not appear. However, our process gives, for each collection, its name as well as all the names and types of the fields, that these are atomic or complex.

On the other hand, for the three research works cited in the state of the art, the proposed solutions do not consider the links between collections. However, in the application presented in section 2 (see Figure 1), the links between collections are useful for treatments and requests made by doctors. Thus, our process proposes a solution to take into consideration the links between the collections and formalize them in the resulting data model.

Finally, it should be emphasized that our process is based on the MDA architecture. This brings both a standard formalism of description of the transformation rules and a way of automating the sequences of transformations.

7 CONCLUSION AND PERSPECTIVES

Our work is part of the evolution of databases towards Big Data. Our studies are currently focused on the extraction mechanisms of the data model from a NoSQL database in order to facilitate the expression of queries.

In this article, we have proposed an automatic process to extract the physical model from a document-oriented NoSQL database. This process is based on the Model Driven Architecture (MDA) architecture that provides a formal framework for automating model transformations. Our process generates a NoSQL physical model from a NoSQL database by applying a sequence of transformations formalized with the QVT standard. The returned model describes the structure of the collections that make up the database and their links. We have experimented our process on the case study in healthcare filed. This case study concerns scientific programs for monitoring patients having serious diseases; the database is stored on MongoDB system.

As future work, we plan to study the update of the data model as the database is being exploited. Indeed, the data volume can reach several terabytes, the generation of the model requires the scan of the entire database. It is therefore not possible for a user to restart the process each time he wishes to express a new query.

REFERENCES

Angadi, A. B., Angadi, A. B., & Gull, K. C. (2013). Growth of New Databases & Analysis of NOSQL Datastores. International Journal of Advanced Research in Computer Science and Software Engineering, 3, 1307-1319.

BigIntegrator (2018). IBM BigIntegrate. https://www.ibm.com/us-en/marketplace/ibm-biginsights-bigintegrate; 5 December 2018.

Bondiombouy, C. (2015). Query processing in cloud multistore systems. In BDA : Bases de Données Avancées.

Budinsky, F., Steinberg, D., Ellersick, R., Grose, T. J., & Merks, E. (2004). Eclipse modeling framework: a developer's guide. Addison-Wesley Professional.

Chen, CL Philip et Zhang, Chun-Yang. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. Information Sciences, 2014, vol. 275, p. 314-347.

CloudMdsQL (2018). CloudMdsQL Compiler. http ://cloudmdsql.gforge.inria.fr/ Online ; 5 December 2018.

Douglas, L., 2001. 3d data management: Controlling data volume, velocity and variety. Gartner. Retrieved, 6, 2001.

Drill (2018). Apache Drill. https: //drill.apache.org/ Online ; 5 December 2018.

Ecore (2018). The eclipse modeling framework project. http ://www.eclipse.org/emf Online; 5 December 2018.

EMF (2018). Projets EMF. www.eclipse.org/stp/ and www.eclipse.org/emf/ Online; 5 December 2018.

Gallinucci, E., Golfarelli, M., & Rizzi, S. (2018). Schema profiling of document-oriented databases. Information Systems, 75, 13-25.

Han, Jing, Haihong, E., LE, Guan, et al. Survey on NoSQL database. Pervasive computing and applications (ICPCA), 2011 6th international conference on. IEEE, 2011. p. 363-366.

Harrison, G. (2015). Next Generation Databases: NoSQLand Big Data. Apress.

Hutchinson, J., Rouncefield, M., & Whittle, J. (2011, May). Model-driven engineering practices in industry. In Proceedings of the 33rd International Conference on Software Engineering (pp. 633-642). ACM.

Klettke, M., U. Störl, et S. Scherzinger (2015). Schema extraction and structural outlier detection for json-based nosql data stores. Datenbanksysteme für Business, Technologie und Web (BTW 2015).

MongoDB (2018). Mongodb atlas database as a service. https://docs.mongodb.com/manual/reference/database-references/ Online ; 5 December 2018.

OMG (2018). Object Management Group. http://www.omg.org/ Online ; 5 December 2018.

Sevilla, Diego Ruiz, Severino Feliciano Morales, and Jesús García Molina. "Inferring versioned schemas from NoSQL databases and its applications." International Conference on Conceptual Modeling. Springer, Cham, 2015.