Model Driven Extraction of NoSQL Databases Schema: Case of MongoDB

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

Keywords: Schema Less, NoSQL, Model Extraction, Big Data, Model Driven Architecture.

Abstract: Big Data have received a great deal of attention in recent years. Not only the amount of data is on a completely different level than before but also, we have different type of data including factors such as format, structure, and sources. This has definitely changed the tools we need to handle Big Data, giving rise to NoSQL systems. While NoSQL systems have proven their efficiency to handle Big Data, it's still an unsolved problem how the extraction of a NoSQL database model could be done. This paper proposes an automatic approach for extracting a physical model starting from a document-oriented NoSQL database, including links between different collections. In order to demonstrate the practical applicability of our work, we have realized it in a tool using the Eclipse Modeling Framework environment.

1 INTRODUCTION

Big data have received a great deal of attention in recent years. Not only the amount of data is on a completely different level than before but also, we have different type of data including factors such as format, structure, and sources. In addition, the speed at which these data must be collected and analyzed is increasing (Chen, 2014). This has definitely impacted the tools required to store Big Data, and new kinds of data management tools i.e. NoSQL systems have arisen (Han, 2014). Compared to existing systems, NoSQL systems are commonly accepted to support larger volume of data, provide faster data access, better scalability and higher flexibility (Angadi, 2013).

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 types 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 schema 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 of study detailed in Section 2 – 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. The importance and the necessity of the database model are widely recognized. There is still a need for this model to know how data is structured and related in the database; this is particularly necessary to write declarative queries where tables and columns names are specified.

On the one hand, NoSQL systems have proven their efficiency to handle Big Data. On the other hand, the needs of a the NoSQL database physical model remain up-to-date. Therefore, we are convinced that it's important to provide a precise and automatic approach that guides and facilitates the Database model extraction task within NoSQL systems. This approach will assist the user to express his queries.

For this, we propose the "ToNoSQLModel" MDA-based approach. The Model Driven Architecture (MDA) is well-known as a framework

Brahim, A., Ferhat, R. and Zurfluh, G.
Model Driven Extraction of NoSQL Databases Schema: Case of MongoDB.
DOI: 10.5220/0008176201450154
In *Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2019)*, pages 145-154
ISBN: 978-989-758-382-7

for models automatic transformations. Our approach starts from a document-oriented NoSQL database and extracts automatically its physical model. As discussed in the related work, few solutions have dealt with the NoSQL database model extraction. To the best of our knowledge, none of the existing contribution has treated the links between collections.

The remainder of the paper is structured as follows. Section 2 motivates our work using a case of study in the healthcare field. Section 3 introduces our NoSQL database model extraction process. Section 4 reviews previous work. Section 5 details our experiments as well as the validation of our process. Finally, Section 6 concludes the paper and announces future work.

2 ILLUSTRATIVE EXAMPLE

To motivate and illustrate our work, we relied on a case study in the healthcare field that we have used in previous work (Abdelhedi, 2017). 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): **Volume:** the amount of data collected from all the establishments in three years can reach several terabytes. **Variety:** data created while monitoring patients come in different types; it could be (1) structured as the patient’s vital signs (respiratory rate, blood pressure, etc.), (2) semi-structured document such as the package leaflets of medicinal products, (3) unstructured such as consultation summaries, paper prescriptions and radiology reports. **Velocity:** some data are produced in continuous way by sensors; it needs a [near] 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 for this kind of applications that use large amounts of disparate data. Therefore, we are convinced that a NoSQL DBMS, like MongoDB, is the most adapted system to store the medical database.

As mentioned before, this kind of systems operate on schema-less data model. Nevertheless, there is still a need for the database model in order to know how data is structured and related in the database and then to express queries. Regarding the medical application, doctors enter measures regularly for a cohort of patients. They can also record new data in cases where the patient's state of health evolve over time. Few months later, they will analyze the entered data in order to follow the evolution of the pathology. For this, they need the database model to express their queries.

In our view, it’s important to have a precise and automatic solution that guides and facilitates the database model extraction task within NoSQL systems. For this, we propose the ToNoSQLModel process presented in the next section that extracts the physical model of a database stored in MongoDB. This model is expressed using the JSON format.

3 ToNoSQLModel PROCESS

This article focuses on extracting the model from a NoSQL database with the "schema less" property. We limit ourselves to the document-oriented type which is the most complete in terms of expression of links (use of references and nesting). For this, we propose the ToNoSQLModel process which automatically extracts the model from a document-oriented NoSQL database.

The ToNoSQLModel process is based on OMG's Model Driven Architecture (Hutchinson, 2011). We recall below the outlines of this model transformation approach. MDA is a formal framework for formalizing and 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. For this, MDA uses three models representing the abstraction levels of the application. These are (1) the Computational Independent Model (CIM) describing the services that the application must provide to meet the needs of users, (2) the analysis and design model (PIM for Platform Independent Model) which defines the structure and the behavior of the system without indicating the execution platform and (3) the model of code (PSM for Platform Specific Model) which is the projection of a PIM on a particular technical platform. Since the input of our process corresponds to a NoSQL database and its output is a physical model, we retain only the PSM level.

The extraction of the model from a NoSQL database is done via a sequence of transformations. We will formalize these transformations using the QVT standard (Query View Transformation) defined by the OMG (§ Experiments). Figure 1 shows an overview of our process.

In the following sections, we detail the components of ToNoSQLModel by specifying the inputs / outputs as well as the transformation rules.

3.1 Input

In the following sections, we detail the components of ToNoSQLModel by specifying the inputs / outputs as well as the transformation rules.

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

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

- cll_i.N the collection name,

- $cll_i.\ FL^{IN}$ = $AFL^{IN} \cup CFL^{IN}$, 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:

$\forall$ i ∈ [1..k], $afl_i^{IN} \in AFL^{IN}$ 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:

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

- cfl_i^{IN}.N is the name of cfl_i^{IN},

- $cfl_i^{IN}\ .\ FL^{IN\prime} \in FL^{IN}$ is the set of fields that cfl_i^{IN}contains.

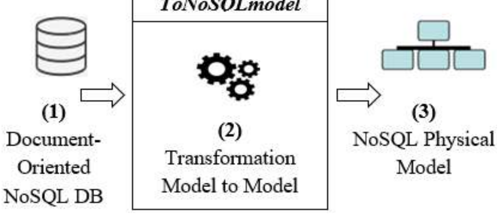


Figure 1: Overview of ToNoSQLModel process.

To express a link between the collections, we used a field called: reference field, denoted by ch^{ref} (Mongo, 2019). 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 2. Note that all the meta-models presented in this article are formalized with the standard Ecore language (EMF, 2018).

3.2 Output

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.

$\forall$ 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:

$\forall$ i ∈ [1..k], $afl_i^{OUT} \in AFL^{OUT}$ 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:

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

- cfl_i^{OUT}.N is the name of cfl_i^{OUT},

- $cfl_i^{OUT}\ .\ FL^{OUT\prime} \in FL^{OUT}$ is the set of fields that cfl_i^{OUT}contains.

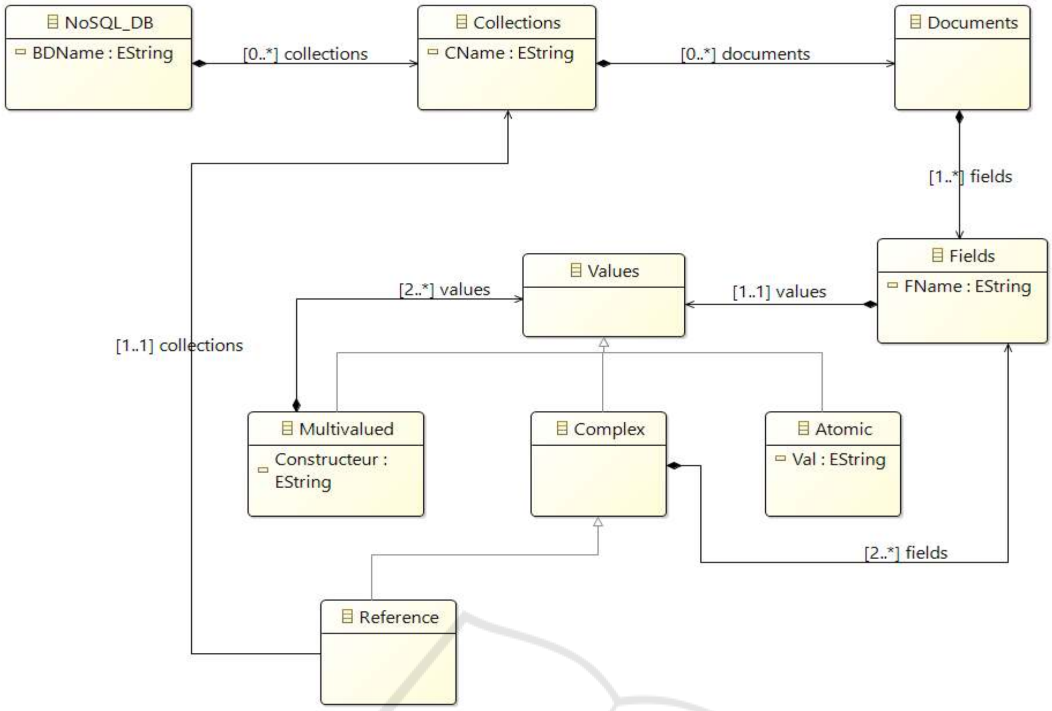


Figure 2: Input metamodel.

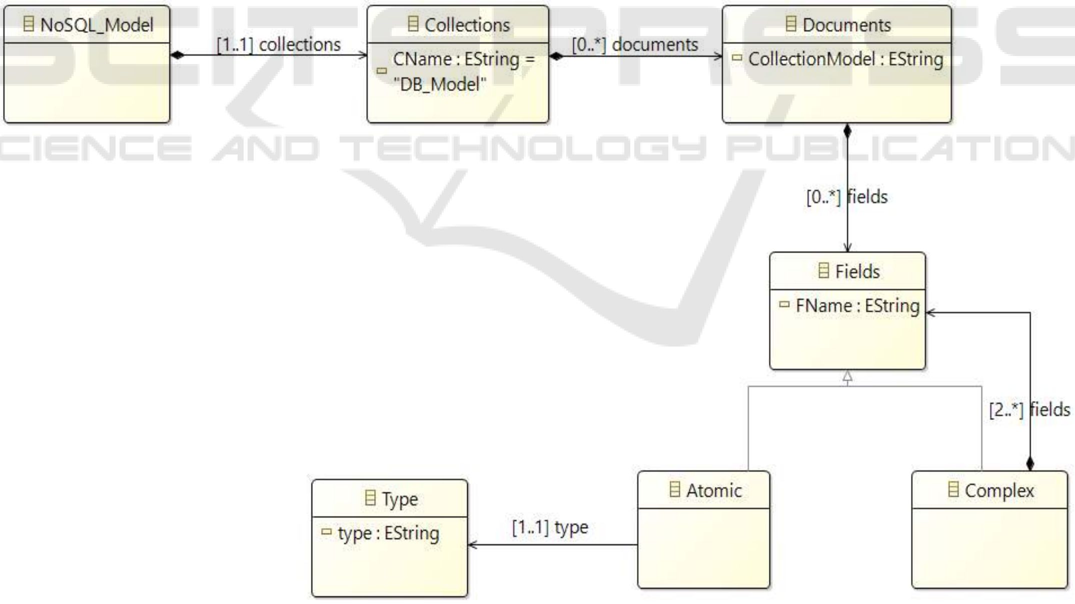


Figure 3: Output metamodel.

3.3 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

4 RELATED WORK

Several research works have been proposed to extract a NoSQL databases model, mainly for document-oriented databases such as MongoDB. In (Klettke, 2015), the authors present a process to extract a 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, lists, or nested documents.

Authors in (Sevilla, 2015) propose a model extraction process from a document-oriented NoSQL database that can include several collections. The returned result is not a unified model for the whole database but it is a set of model versions. These versions are stored in JSON format.

More specific to document-oriented databases, we can mention (Gallinucci, 2018) where authors describe a process called BSP (Build Schema Profile) to classify the documents of a collection by applying a set of rules that correspond to the user requirements. These rules are expressed through a decision tree where nodes represent the attributes of the documents and 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 model versions; each of them is common to a group of documents.

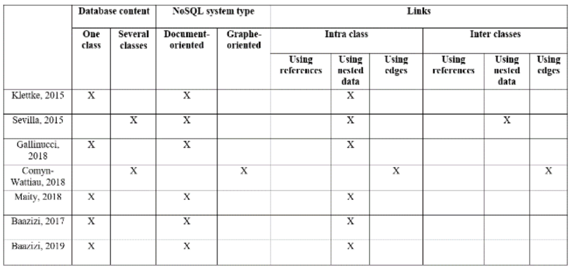
We can also mention (Maity, 2018) that describes a mapping from a document-oriented NoSQL database to a relational model. The process groups together all documents that have the same fields name. For each class of documents, it generates a table that have as columns the fields names and as rows the fields values.

Another study (Baazizi, 2017) have proposed a model extraction process from a collection of JSON documents. This process is based on the use of MapReduce. The Map step consists of extracting the schema of each document in the collection by mapping each couple (field, value) into another couple (field, type). The Reduce step consists of unifying all the schemas produced in the Map step in order to provide an overall schema for the input collection. The same authors have proposed in another paper (Baazizi, 2019) an extension of the process prposed in (Baazizi, 2017) in order to take into account the parameterization of the extraction at the Reduce step. Thus, the user can choose either to unify all the schemas of the collection, or to unify only the schemas having the same fields (same names and types).

On the other hand, (Comyn-Wattiau, 2017) proposes a process for extracting a model from object insertion queries and relations in a graph-oriented databases. The proposed process is based on an MDA architecture and applies two treatments. The first one build a graph (Nodes + Edges) starting from Neo4j queries. The second one consists of extracting an Entity / Association model from the graph returned by the first treatment.

In Table 1, we summarize the previous works using three criteria: the database content (one or several classes), the considered NoSQL system type (document or graph) and the way used to implement links (references, nested data or edges).

Table 1: Comparative table of previous works.



	Database content		NoSQL system type		Links					
	One class	Several classes	Document-oriented	Graphe-oriented	Intra class			Inter classes		
					Using references	Using nested data	Using edges	Using references	Using nested data	Using edges
Klettke, 2015	X		X			X				
Sevilla, 2015		X	X			X			X	
Gallinucci, 2018	X		X			X				
Comyn-Wattiau, 2018		X		X			X			X
Maity, 2018	X		X			X				
Baazizi, 2017	X		X			X				
Baazizi, 2019	X		X			X				

Regarding the state of the art, the solutions proposed in (Gallinucci, 2018), (Klettke, 2015), (Maity, 2018), (Baazizi, 2017) and (Baazizi, 2019) start from a single collection of documents and take into account only the links implemented using nested data ; the links presented using references are not considered. The process proposed in (Sevilla, 2015) takes as input a set of collections ; however, only the use of nested data to express links is considered. On the other hand, authors in (Comyn-Wattiau, 2017) have worked on graph-oriented systems. This kind of NoSQL systems does not offer many solutions to implement links as like document-oriented systems ; it expresses explicitly links between data using edges.

To overcome these limits, we define an automatic process to extract the database model within documents-oriented NoSQL systems. This process takes into account the links between collections.

5 EXPERIMENTS AND VALIDATION

5.1 Experiments

We have formalized this mapping using the QVT (Query / View / Transformation) language, which is the OMG standard for models transformation. We carry out the experimental assessment using a model transformation environment called Eclipse Modeling Framework (EMF). It's a set of plugins which can be used to create a model and to generate other output based on this model. Among the tools provided by EMF we use:

(1) Ecore: the metamodeling language that we used to create our metamodels,
(2) XML Metadata Interchange (XMI): the XML based standard that we use to create models,
(3) Query / View / Transformation (QVT): the OMG language for specifying model transformations.

ToNoSQLModel transformation is expressed as a sequence of elementary steps that builds the resulting model step by step from the source (NoSQL database):
Step 1: we create Ecore metamodels corresponding to the source (Figure 2) and the target (Figure 3).
Step 2: we build an instance of the source metamodel. For this, we use the standard-based XML Metadata Interchange (XMI) format (Figure 4).
Step 3: we implement the mapping by means of the QVT plugin provided within EMF. An excerpt from the QVT script is shown in Figure 5.
Step 4: we test the transformation by running the QVT script created in step 3. This script takes as input the source database builded in step 2 and returns as output the NoSQL physical model. The result is provided in the form of XMI file as shown in Figure 6.

5.2 Validation

5.2.1 Experimental Environment

Our problem is to extract the model of a database managed by a NoSQL system. Such a feature is intended for users who do not know the data structure (developer who has not created the database, decision makers, etc.); its major interest is to allow the expression of queries as can be done in relational systems.

The experiments of our proposal were carried out on a cluster composed of 3 machines. Each machine has the following specifications: Intel Core i5, 8 GB of RAM and 2 TB of disk. One of these machines is configured to act as a master; the other two machines have slave status.

To implement our solution, we used the tools JSON Generator (JSON Generator 2018) and Generate Test Data data generation tools (Generate Test Data 2018). We produced a 3TB dataset in the form of JSON files. These files were loaded into MongoDB using shell commands.

5.2.2 Query Set

For our experiment, we have considered four kinds of queries: **(1)** those using one collection (example : select the patients whose age is between 10 and 70), **(2)** queries that use two related collections with the link is expressed using a monovalued reference field (example: we want the name of doctor who has performed the consultation number 41), **(3)** queries that use two related collections with the link is expressed using a multivalued reference field (example: select the antecedents of patient ”DUPONT David”), **(4)** queries that use two related collections with the link is expressed using nested data.

Table 2 shows the comparison results between our solution and those proposed in (Klettke, 2015), (Sevilla, 2015), (Gallinucci, 2018), (Maity, 2018), (Baazizi, 2017) and (Baazizi, 2019) regarding the expression of queries. Note, however, that we only consider works that deal with document-oriented NoSQL databases. Thus, we have excluded the work of (Comyn-Wattiau, 2017) which uses a graph-oriented database. For each query we have considered to perform this comparison, we indicate if it can be formulated using the model obtained by each solution proposed in the mentioned works.

Table 2: Comparison results between our solution and state of the art.

Query category	Klettke, 2015	Sevilla, 2015	Gallinucci, 2018	Maity, 2018	Baazizi, 2017	Baazizi, 2019	ToNoSQLModel
(1)	X	X	X	X	X	X	X
(2)							X
(3)							X
(4)		X					X

Table 2 shows that the absence of taking into account the links between collections in the referenced works (Klettke, 2015), (Sevilla, 2015), (Gallinucci, 2018), (Maity, 2018), (Baazizi, 2017) and (Baazizi, 2019), does not make it possible to write complex queries. Considering for example the following query that applies a join between the Patients collection and the Doctors collection:

```
db.Patients.aggregate (
[
{$ lookup: {from: "Doctors", localField: "Treating-Doctors._id", foreignField: "_id", as: "Doctors"}}
])
```

We can see that we can not write this query if we do not visualize the link between Patients and Doctors.

For better readability, we give the equivalent of Figure 6 in the form of a class diagram as shown in Figure 7. This is a graphical description of the data structures stored in the MongoDB system that we used in our experimentation. Note that as MongoDB is a schema less system, it does not provide this model, either in textual form or in graphical form.

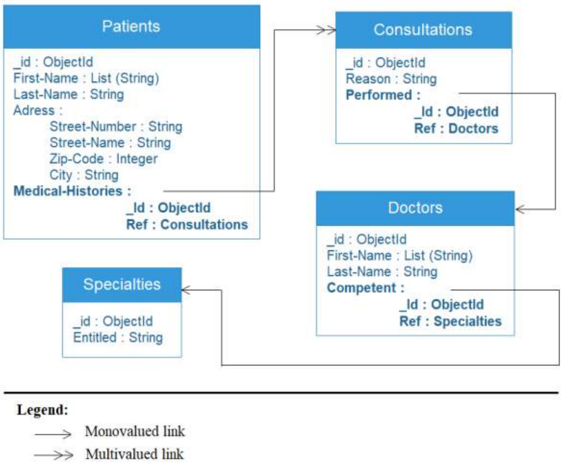


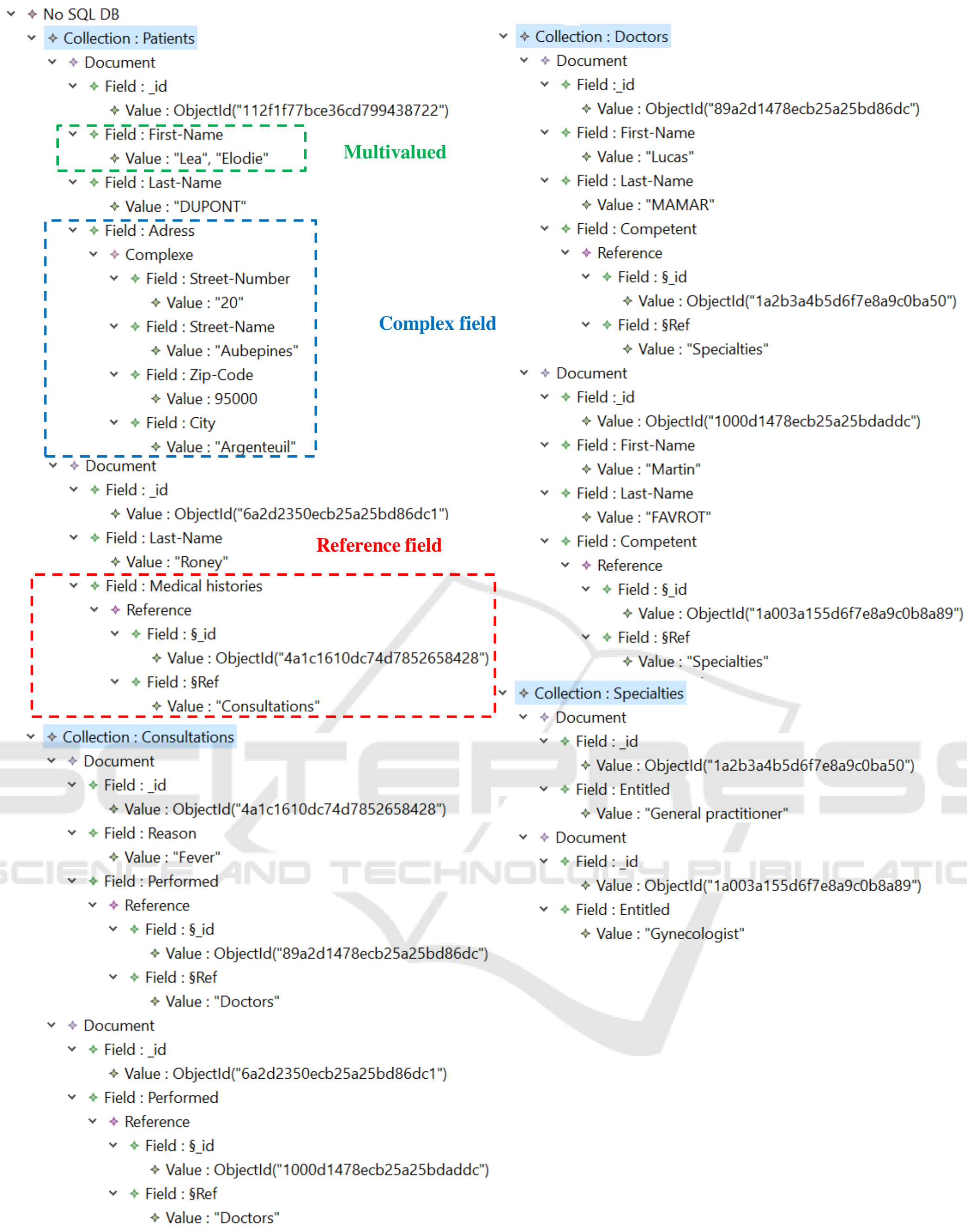
Figure 7: Excerpt from the physical model of data.

6 CONCLUSION AND PERSPECTIVES

Our work is part of the evolution of databases towards Big Data. They are currently focused on the extraction mechanisms of the model of a NoSQL database "schema less" to allow the expression of queries by end-users.

In this article, we have proposed an automatic process that builds the physical model of a NoSQL database as it is used. 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 document-oriented 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 as well as the links between them. We have experimented our process on the case of a medical application that deals with scientific programs for the follow-up of pathologies; the database is stored on the MongoDB system.

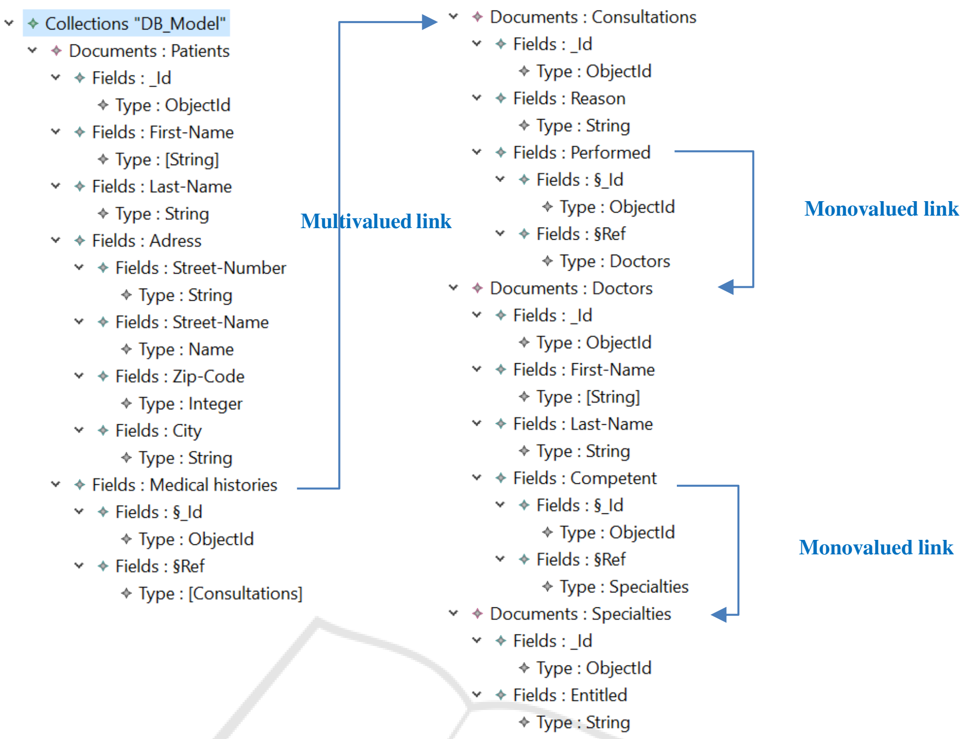
Regarding future work, we aim to enrich our process so that it can take into consideration the diversity of particular cases related to the data entered. In fact, when feeding the database, users can enter incorrect data: misspelled field names, values associated with the same field of different types, etc. The current version of our process is based on consistent strategies, but the result may not be entirely satisfactory to users.



```
No SQL DB
  Collection : Patients
    Document
      Field : _id
        Value : ObjectId("112f1f77bce36cd799438722")
      Field : First-Name
        Value : "Lea", "Elodie"
      Field : Last-Name
        Value : "DUPONT"
      Field : Adress
        Complexe
          Field : Street-Number
            Value : "20"
          Field : Street-Name
            Value : "Aubepines"
          Field : Zip-Code
            Value : 95000
          Field : City
            Value : "Argenteuil"
    Document
      Field : _id
        Value : ObjectId("6a2d2350ecb25a25bd86dc1")
      Field : Last-Name
        Value : "Roney"
      Field : Medical histories
        Reference
          Field : §_id
            Value : ObjectId("4a1c1610dc74d7852658428")
          Field : §Ref
            Value : "Consultations"
  Collection : Consultations
    Document
      Field : _id
        Value : ObjectId("4a1c1610dc74d7852658428")
      Field : Reason
        Value : "Fever"
      Field : Performed
        Reference
          Field : §_id
            Value : ObjectId("89a2d1478ecb25a25bd86dc")
          Field : §Ref
            Value : "Doctors"
    Document
      Field : _id
        Value : ObjectId("6a2d2350ecb25a25bd86dc1")
      Field : Performed
        Reference
          Field : §_id
            Value : ObjectId("1000d1478ecb25a25bdaddc")
          Field : §Ref
            Value : "Doctors"

  Collection : Doctors
    Document
      Field : _id
        Value : ObjectId("89a2d1478ecb25a25bd86dc")
      Field : First-Name
        Value : "Lucas"
      Field : Last-Name
        Value : "MAMAR"
      Field : Competent
        Reference
          Field : §_id
            Value : ObjectId("1a2b3a4b5d6f7e8a9c0ba50")
          Field : §Ref
            Value : "Specialties"
    Document
      Field : _id
        Value : ObjectId("1000d1478ecb25a25bdaddc")
      Field : First-Name
        Value : "Martin"
      Field : Last-Name
        Value : "FAVROT"
      Field : Competent
        Reference
          Field : §_id
            Value : ObjectId("1a003a155d6f7e8a9c0b8a89")
          Field : §Ref
            Value : "Specialties"
  Collection : Specialties
    Document
      Field : _id
        Value : ObjectId("1a2b3a4b5d6f7e8a9c0ba50")
      Field : Entitled
        Value : "General practitioner"
    Document
      Field : _id
        Value : ObjectId("1a003a155d6f7e8a9c0b8a89")
      Field : Entitled
        Value : "Gynecologist"
```

Figure 4: Document-oriented NoSQL database model.



```
Collections "DB_Model"
  Documents : Patients
    Fields : _Id
      Type : ObjectId
    Fields : First-Name
      Type : [String]
    Fields : Last-Name
      Type : String
    Fields : Adress
      Fields : Street-Number
        Type : String
      Fields : Street-Name
        Type : Name
      Fields : Zip-Code
        Type : Integer
      Fields : City
        Type : String
    Fields : Medical histories
      Fields : §_Id
        Type : ObjectId
      Fields : §Ref
        Type : [Consultations]
  Documents : Consultations
    Fields : _Id
      Type : ObjectId
    Fields : Reason
      Type : String
    Fields : Performed
      Fields : §_Id
        Type : ObjectId
      Fields : §Ref
        Type : Doctors
  Documents : Doctors
    Fields : _Id
      Type : ObjectId
    Fields : First-Name
      Type : [String]
    Fields : Last-Name
      Type : String
    Fields : Competent
      Fields : §_Id
        Type : ObjectId
      Fields : §Ref
        Type : Specialties
  Documents : Specialties
    Fields : _Id
      Type : ObjectId
    Fields : Entitled
      Type : String
```

Figure 5: QVT script.

```
modeltype NoSQL_DB uses "http://nosqldatabaseMM.com";
modeltype NoSQL_Schema uses "http://nosqlschemaMM.com";
transformation NoSQLdb2NoSQLschema(in Source:     NoSQL_DB, out Target: NoSQL_Schema);
main() {
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();
structuredufield:=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::StructuredUField{
```

Figure 6: NoSQL Physical model.

REFERENCES

Abdelhedi, F., Brahim, A. A., Atigui, F., & Zurfluh, G. (2017, August). MDA-Based Approach for NoSQL Databases Modelling. In International Conference on Big Data Analytics and Knowledge Discovery (pp. 88-102). Springer, Cham.

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.

Baazizi, M. A., Lahmar, H. B., Colazzo, D., Ghelli, G., & Sartiani, C. (2017, March). Schema inference for massive JSON datasets. In Extending Database Technology (EDBT).

Baazizi, M. A., Colazzo, D., Ghelli, G., & Sartiani, C. (2019). Parametric schema inference for massive JSON datasets. The VLDB Journal, 1-25.

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.

Comyn-Wattiau, I., & Akoka, J. (2017, December). Model driven reverse engineering of NoSQL property graph databases: The case of Neo4j. In 2017 IEEE International Conference on Big Data (Big Data) (pp. 453-458). IEEE.

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

EMF. https://www.eclipse.org/modeling/emf/. Online; 5 July 2018.

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

Generate Test Data (2018) http://www.convertcsv.com/generate-test-data.htm Online; 5 July 2018.

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.

JSON Generator (2018) http://www.json-generator.com/. Online; 5 July 2018.

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).

Maity, B., Acharya, A., Goto, T., & Sen, S. (2018, June). A Framework to Convert NoSQL to Relational Model. In Proceedings of the 6th ACM/ACIS International Conference on Applied Computing and Information Technology (pp. 1-6). ACM.

MongoDB, The database for modern applications (2019) https://www.mongodb.com/fr Online; 5 July 2019.

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.