



# Automatic Extraction of a Document-oriented NoSQL Schema

Fatma Abdelhedi<sup>1,2</sup> <sup>a</sup>, Amal Ait Brahim<sup>1</sup>, Hela Rajhi<sup>1</sup>, Rabah Tighilt Ferhat<sup>1</sup>  
and Gilles Zurfluh<sup>1</sup> <sup>b</sup>

<sup>1</sup>IRIT, Toulouse Capitole University, Toulouse, France

<sup>2</sup>CBI<sup>2</sup> - TRIMANE, Saint Germain-En-Laye, France

**Keywords:** Big Data, NoSQL, MDA, QVT, Schemaless, Reverse-engineering, Schema-extraction.

**Abstract:** The NoSQL systems make it possible to manage Databases (DB) verifying the 3Vs: Volume, Variety and Velocity. Most of these systems are characterized by the property schemaless which means absence of the data schema when creating a DB. This property provides undeniable flexibility by allowing the schema to evolve while the DB is in use; however, it is a major obstacle for developers and decision makers. Indeed, the expression of queries (SQL type) requires precise knowledge of this schema. In this article, we provide a process for automatically extracting the schema from a NoSQL document-oriented DB. To do this, we use the MDA (Model Driven Architecture). From a NoSQL DB, we propose transformation rules to generate the schema. An experiment of the extraction process was carried out on a medical application.

## 1 INTRODUCTION


The last few years have seen the explosion of data generated and accumulated by increasingly lot of computing devices. The DBs thus formed are designated by the expression "Big Data" which refers to the so "3V" minimum rule: volume, variety and velocity (Chen & Zhang, 2014). This rule characterizes a DB of several terabytes containing data of various types and formats such as texts, tables, highly structure documents. In addition, this data is often captured at very highly frequency and must therefore be filtered and aggregated in real time to avoid necessary saturation of storage space.


Classical implantation of techniques based mainly on the relational principle, have limitations lot of managing massive DB (Angadi et al., 2013). Thereby, new data storage and manipulation systems have been appeared. Grouped under the term NoSQL (Han et al., 2011), these systems are much greeted to manage large volumes of data with flexible schemas. They also provide great scalability and good response of time performance (Angadi et al., 2013).

Most NoSQL DBMS are characterized by the property of the schemaless which corresponds to the absence of the data schema when doing a DB. This property appears in all NoSQL systems such as MongoDB, CouchDB, HBase and Neo4j. Note, however, that it is absent in some systems as Cassandra and Riak TS. The property schemaless offers undeniable flexibility by allowing the schema to evolve readily. For example, adding new attributes to an existing line is done without modifying other lines of the same type previously stored.

In this article, we study the process of extracting the schema from a schemaless NoSQL document-oriented DB. We have chosen the specific vocabulary of the document-oriented NoSQL system MongoDB which is the most used in the industry<sup>1</sup>. Our process uses MDA<sup>2</sup> (Model Driven Architecture) which provides a formal frame for model transformations (Bézivin & Gerbé, 2001).

The rest of the paper is structured as follows. Section 2 presents the medical application which justifies the interest of our work. Section 3 reviews the state of the art. Section 4 presents our contribution which consists in formalizing with MDA the process

<sup>a</sup>  <https://orcid.org/0000-0003-2522-3596>

<sup>b</sup>  <https://orcid.org/0000-0003-3570-9792>

<sup>1</sup> <https://db-engines.com/en/ranking/document+store>

<sup>2</sup> <https://www.omg.org/mda/>

of extracting the schema from a schemaless NoSQL DB. Section 5 details the development of our prototype and describes the experimentation and validation of our process. Finally, section 6 concludes the paper and announces future works.

## 2 CASE STUDY

In this section, we present the case study which motivated our work as well as its development.

### 2.1 NoSQL Database

Our work is motivated by the development of a medical application within the framework of an industrial contract; this application will also allow us to illustrate the concepts and mechanisms used. This involves the establishment of scientific programs dedicated to the following of a specific pathology. Each program can be together around fifty European hospitals (hospitals, clinics and specialized care centers).

The primary objective of such a program is to collect significant data on the course of the disease over time, to study its interactions with opportunistic diseases and to assess the influence of its treatments in the short and medium term. The duration of a program is done when it is launched and can be between three and ten years. The data collected by several establishments as part of a multi-year program present the characteristics generally accepted for Big Data (the 3 Vs), (Douglas, 2001). Indeed, the volume of medical data collected from daily patients can reach, for all establishments and over three years, several terabytes. On the other hand, the nature of the data entered (measurements, radiography, scintigraphy, etc.) is diverse and may vary from one patient to another depending on their status health. Finally, some data is produced continuously by sensors; they must be processed almost in real time because they can be integrated into time-critical processes (measures crossing a threshold that would involve the intervention of an emergency practitioner, for example). Patient following requires the storage of various data such as the recording of consultations carried out by practitioners, test results, prescriptions for drugs and specific treatments. We therefore stored all of this data in a schemaless NoSQL system.

Thus, to develop requests corresponding to this type of analysis over time, doctors need to know the schema of the DB.

### 2.2 Development Framework

To meet the needs of this medical application, a software has been developed in Java language and uses MongoDB NoSQL system. This DBMS offers stability and provides satisfactory data access performance for the application. The DB has been designed by developers; it is fed from predefined transactions that are implemented by health personnel from touch tablets.

## 3 RELATED WORK

Several research works have proposed processes to extract the schema from a NoSQL schemaless DB, mainly for document DBs like MongoDB. Thus, a process has been proposed in (Klettke et al., 2015) to extract the schema from a collection of JSON documents stored on MongoDB. The returned schema is itself in JSON format; it is obtained by browsing the DB and identifying the names of the attributes that appear in the documents. Attributes are associated with atomic types, lists, or structured documents.

A similar work (Izquierdo & Cabot, 2016) proposes a process for generating a NoSQL model from a collection of JSON documents. This process consists in (1) first extracting the model from each document by replacing pairs (Key, Value) by pairs (Key, Type) and (2) subsequently unifying all the models obtained to have a single model for the whole collection. The generated model is in JSON format.

In the article (Sevilla Ruiz et al., 2015), the authors propose another process of extracting the schema in a document NoSQL DB. The returned result is not a unified schema for the whole DB, but it gives the different versions of document schemas (according to the names and types of the attributes) for each collection. The extraction process is composed of two successive steps. The first one browses the DB and, for each distinct schema version, generates a document in a collection called "Model". In the second step, the process provides a model of each version by instantiating the JSON metamodel.

We can also cite the work of (Gallinucci et al., 2018) which proposes a process called BSP (Build Schema Profile) to classify documents in a collection by applying rules corresponding to user requirements. 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 presence of an attribute in a

document or its value. As in the previous article (Sevilla Ruiz et al., 2015), the result returned by this approach is not a unified schema but a set of schema versions; each one is common to a group of documents.

In addition, the article by (Comyn-Wattiau & Akoka, 2017) proposes a process for extracting a schema from requests for inserting objects and relations in a graph NoSQL DB. The proposed process is based on an MDA architecture and applies two types of transformations to the queries. The process consists of two steps. The first one consists in building a graph (Nodes + Edges) from Neo4J queries. The second step consists in extracting from the graph an Entity-Relationship model by transforming nodes with the same label into entity classes and edges into association.

In (Maity et al., 2018), the authors describe the transition from a document or graph NoSQL DBs to a relational schema. The process groups together all documents (or objects) that have the same field names. For each class of objects thus obtained, it generates a table with the field names as attributes and the field values as lines.

On the other hand, in (Baazizi et al., 2017), the authors propose a process for extracting the schema from a large collection of JSON documents using the MapReduce system. The Map phase consists in extracting the schema of each document in the collection by inferring pairs (key, type) from pairs (key, value). The Reduce phase consists in unifying all the schemas produced in the Map phase in order to provide a global schema of all the documents in the collection. In another article (Baazizi et al., 2019), the same authors proposed to extend this process by integrating the parameterization of the extraction at the Reduce phase. Thus, the user can choose either to unify all the schemas of the documents in the collection, or to unify only the schemas having the same fields (names and types).

Through this related work section, it appears that these solutions only partially answer our problem.

The process we propose aims at extracting a schema from a document NoSQL DB. We chose the MongoDB system that was used in the development of the medical application presented in section 2. In addition, MongoDB is the most widely used NoSQL DBMS in the world as shown by a recent study by DB-Engines (*DB-Engines Ranking*, n.d.). Our process extracts the descriptions of the collections as well as some types of links present in the application.

## 4 EXTRACTING THE NoSQL SCHEMA

The goal of this article is to automate schema extraction from a document-oriented schemaless NoSQL DB. This schema indicates the names of the collections, the names of the fields as well as their types as is done in the relational DBs where the schema is provided by the system. To do this, we propose the ToNoSQLSchema process that allows to automatically extract the schema after a scan of the DB. It is about a "cold extraction", i.e., the process is the totality of the DB at time  $t$  and the schema describes the data structures at time  $t$ . A complementary "hot extraction" has been developed and allows the schema to be updated to consider the development of the DB.

To develop and automate our ToNoSQLSchema process, we use the OMG<sup>3</sup> Model Driven Architecture (MDA) which provides a formal frame for the automation of schema processing (Hutchinson et al., 2011). The objective of this design is to separately describe the specifications functional and the specifications implementation of an application on a given platform; to do this, it uses three sitters representing the levels abstract of the application. These are (1) the requirements schema (CIM for Computation Independent Model) in which IT considerations appear, (2) the analysis and design schema (PIM for Platform Independent Model) independent of aspects technical execution platforms and (3) the code model (PSM for Platform Specific Model) specific to a specific platform. Since the input to our process is a NoSQL DB and its output corresponds to a physical schema, we only retain the PIM level. We develop the inputs / outputs with the Ecore language and the model transformations using the QVT (Query View Transformation) standard defined by the OMG.

The advantage of using the MDA architecture lies mainly in the generalization of our process ToNoSQLSchema. Indeed, the formalization of inputs / outputs by meta models (sections 4.1 and 4.2) as well as the use of QVT notations, ensure a certain independence from the specificities of NoSQL systems, current and future. Warning: for the sake of consistency of the article, we have used the term model from the MDA vocabulary to denote the result of a modeling process; we use the term schema.

By applying a series of QVT transformations on an existing schemaless DB, our ToNoSQLSchema

<sup>3</sup> Object Management Group (OMG): <http://www.omg.org/>

process (Figure 1) produces a schema explicitly describing the structured data.

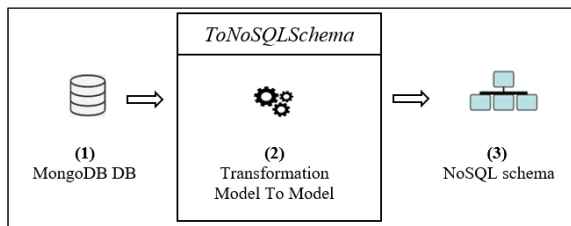


Figure 1: Our ToNoSQLSchema process.

In the following sections, we define the ToNoSQLSchema process by presenting successively the source, the target, and the transformation rules.

#### 4.1 The Source: A Document-oriented NoSQL Database

A document-oriented DB is made up of collections; each of which contains together of documents. A document consists of an aggregate of fields in the form of couple (Field-name, Value). A value can be atomic, multivalued, or structured (i.e., composed of other). To make a link between the collections, the developer can use a field called field reference to a principle proposed in (MongoDB, 2018). This field is a structure of the form: (identifier of the referenced document, name of the targeted collection) if the link is monovalued. On the other hand, to express a multivalued link, the reference field takes the form ([identifiers of the referenced documents], name of the targeted collection). Note, however, that the MongoDB system does not check referential integrity when feeding the DB; this verification remains the responsibility of the data entry software.

The source, i.e. the DB, is described by a meta-model (Figure 2) formalized with the standardized language Ecore.

#### 4.2 The Target: A NoSQL Schema

The NoSQL schema generated by our process describes the collections, their fields and the links restrained in the DB. It is saved under MongoDB in a set of documents. Each file contains the model diagram of a collection. The documents are composed of a lot of domains; each of them can be atomic, structured, or multivalued. An atomic field is presented in the form of a couple (Name, Type) where Type contains Integer, Boolean or String. A structured field is made up of an aggregate of fields

of any type. A multivalued field is composed of a set of fields of the same type. We formalize these concepts through the Ecore meta-model of Figure 3.

#### 4.3 Transformations

After having formalized the concepts located at the input of the process (document-oriented NoSQL DB) and at the output (NoSQL model), we describe the switch between automatic the two models in the form of a sequence of transformation rules.

The following present the rules in natural language of ToNoSQLSchema process.

**R1:** A NoSQL document-oriented DB is transformed into a collection called DB\_Schema.

**R2:** Each input collection is transformed into a document. This includes a field noted CollectionSchema which indicates the name of the set concerned and that we find at the top in the target schema. Note that each document contains a unified schema for all that make up the input set. This means that our process generates a single collection schema grouping together all document fields that we have in the collections input. We therefore do not consider grouping together all document fields that we have in the collections input. We therefore do not consider several versions for the same input collection as it is in (Sevilla Ruiz et al., 2015) and (Gallinucci et al., 2018). Indeed, we are dealing with the case of synonymous fields nor the case where a field has different types depending on their values within the same collection at the level of database input.

**R3:** For each atomic field, the pair (Field-name, Value) is transformed into a couple in the set of the target model like (Field-name, Type). In fact, the type of a field is generated according to its value. For example, if the value is of the form "xxx", then its type is String; if the value is of the form [yyy, zzz, etc.], then its type is [Integer], i.e., a set of Integers.

For example, the field Last-NameP: "DUPONT" is transformed into Last-NameP: String.

**R4:** For structured field, the process goes through all the fields that make it up. For each of them, if it is an atomic field, then apply R3; if this is a structured field then apply R4.

For example, the field Address: {Code: 1735, City: "Paris"} is transformed into {Code: Integer, City: String}.

**R5:** For a monvalued reference field, the pair (document identifier, name of the collection) is transformed into a pair (ObjectId, name of collection).

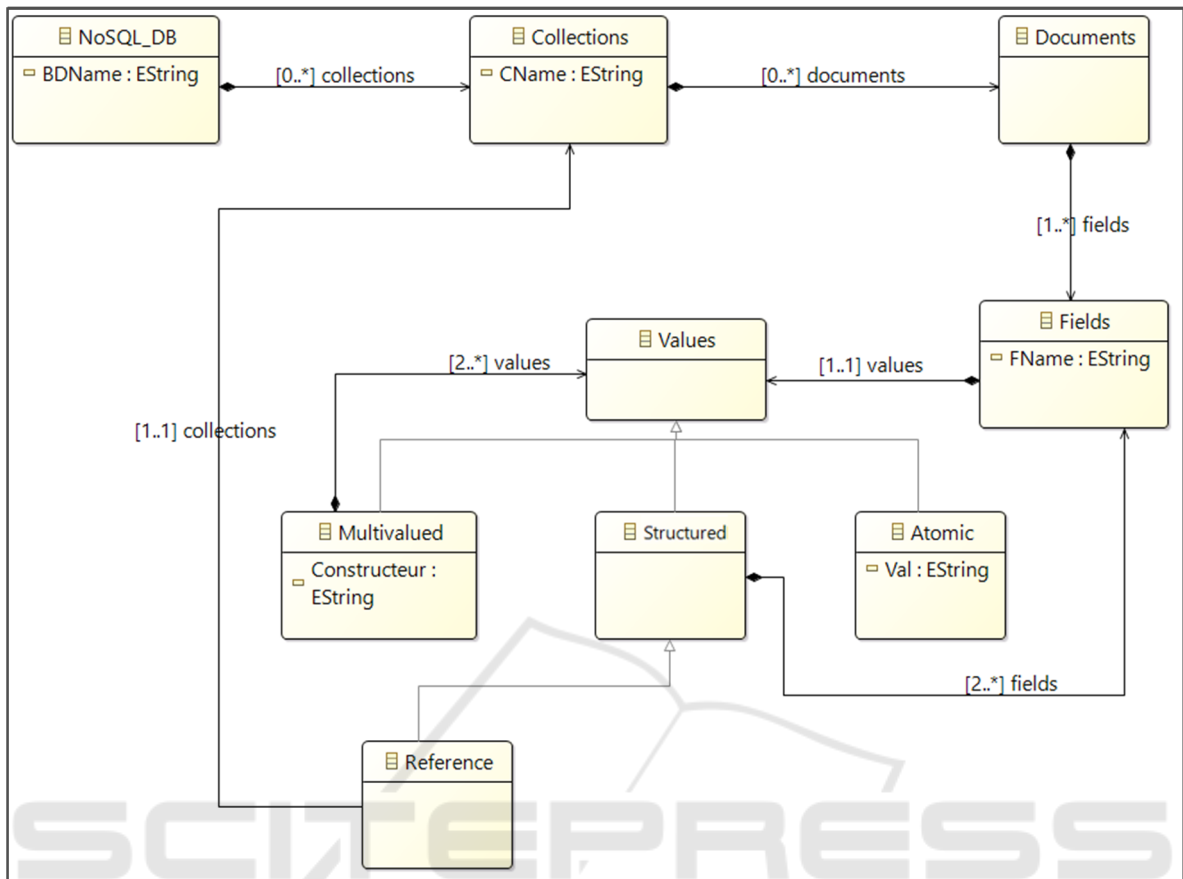


Figure 2: Meta-model describing the source of our process.

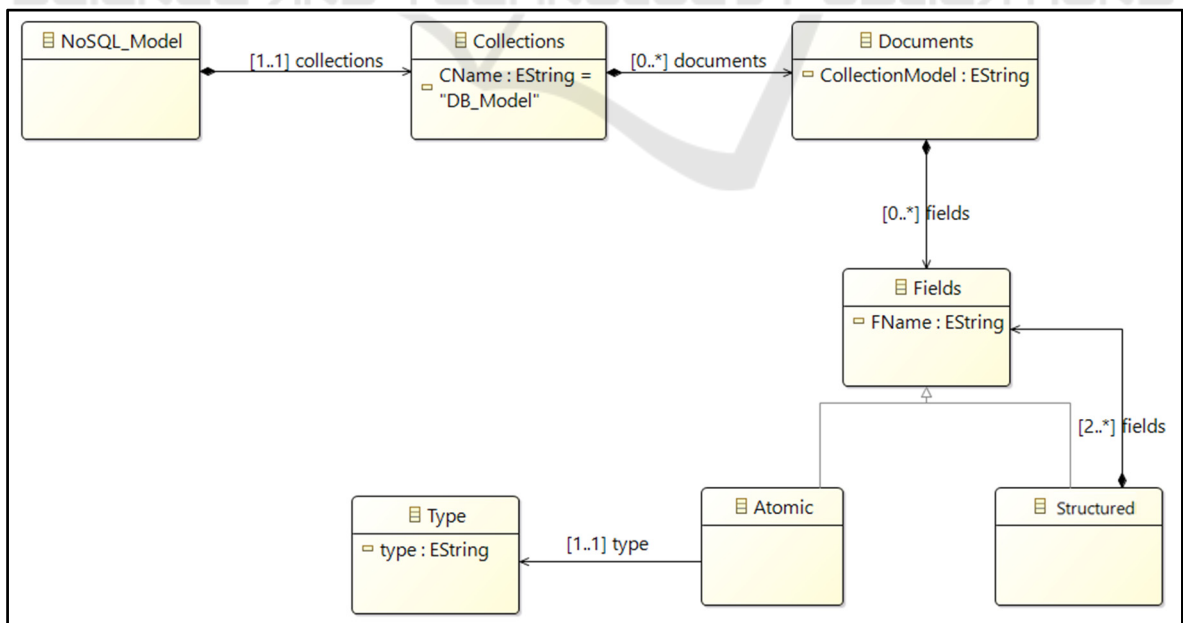


Figure 3: Meta-model describing the target of our process.



**R6:** For a multivalued reference field, the pair ([document identifiers], name of the collection) is transformed into a pair ([ObjectId], name of collection).

We present in Figure 4 an example for applying these rules.

## 5 DEVELOPMENT OF A PROTOTYPE AND VALIDATION

### 5.1 Technical Environment

In collaboration with the company TRIMANE, a digital services company, we have developed a schematic extraction software prototype according to the principles outlined above. We briefly describe the mechanisms that in the development we implemented of ToNoSQLSchema. We use a technical middle pertinent of modeling, meta-modeling and transformation in accordance with the MDA architecture. We therefore used the Eclipse Modeling Framework (EMF) platform (Budinsky et al., 2004). EMF provides a set of elements designed un introducing a schema approach to development within the Eclipse environment. Among the tools of EMF, we used:

- Ecore: a meta-modeling language used for the formation of meta models; Figures 2 and 3 illustrate the Ecore source and target meta models.
- XMI (XML Metadata Interchange): a standard used to show schemas in XML format.

- QVT: a standardized language for voicing schema transformations.

First, we tested our prototype on data from the medical application presented in section 2.

### 5.2 Prototype Achievement

The prototype contains, in dedicated files, the transformation rules expressed in QVT as well as the meta- models in format for the source and the target Ecore (Figures 2 and 3). To obtain the resulting schema, it is necessary to apply the following steps:

**Step1:** we create a source and a target metamodel to represent the concepts handled by our process (Figures 2 and 3 respectively).

**Step2:** we build an instance of the source metamodel. For this, we use the standard based XML Metadata Interchange format.

**Step3:** we implement the transformation rules by means of the QVT language provided within EMF as shown in Figure 5.

**Step4:** we test the transformation rules by running the QVT script created in step3. This script takes as input the source model built in step2 and returns as output a physical schema. The result is provided in the form of XMI file.

### 5.3 Experimentation and Validation

To validate our process, it is necessary on the one hand to check the descriptive quality of the generated schema and on the other hand to show the usefulness of this schema for users.

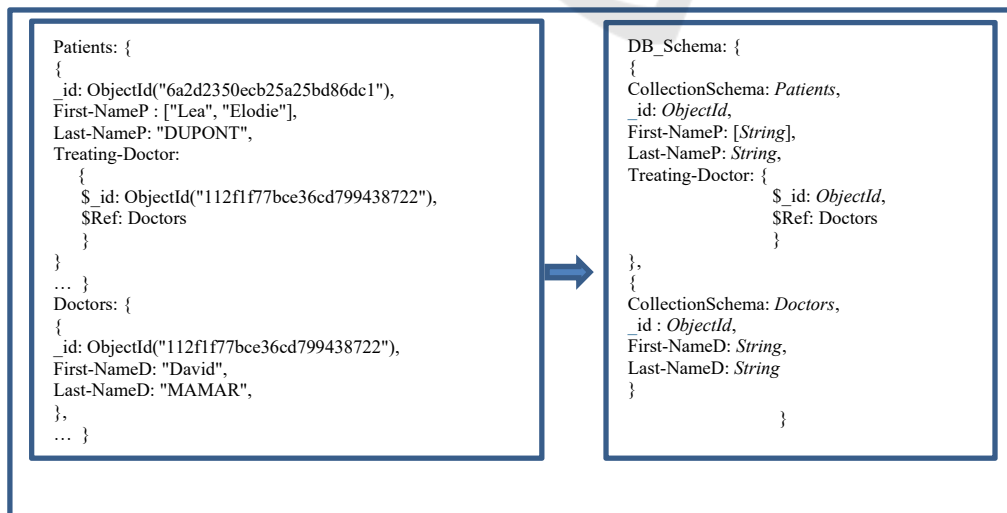


Figure 4: Example of transformations.

Regarding the descriptive quality of the schema, our process is based on the MDA architecture. Thus, the use of meta-models to extract the schema from the DB guarantees the correct use of modeling formalisms. We can therefore consider that the schema produced by the ToNoSQLSchema prototype describes well the data structures present in the DB.

Regarding the usefulness of a schema for developers responsible for writing queries on the DB, we implemented our prototype on test data in an industrial environment. To carry out this experimentation, we considered two applications, one belonging to the medical field and the other to the legal field. We sent test sets to sixteen developers (IT consulting engineers) from TRIMANE, a digital services company specializing in business intelligence and Big Data. The aim was to test the hypothesis that it is faster to write queries on a DB when you have the data schema. Although the verification of this hypothesis appears intuitive, the experimentation provided us with indicators corresponding to the query writing times.

Of the 32 test results returned by the 16 developers, the time taken to write queries by category (with or without schema) was relatively homogeneous. Thus, the absence of any schema associated with a DB required the developer to manually search for field properties and justifies the average time of 49 minutes shown in Table 1. On the contrary, the use of a schema facilitated the

understanding of the DB and led to a significant reduction of the query writing time (- 27 minutes).

Table 1: Average time to write queries.

	Without schema	With schema
Average time to write the 10 queries	49 minutes	22 minutes

A schemaless NoSQL DB can contain various data presenting complex structures and whose semantics are often difficult to understand by computer scientists themselves. Our schema extraction system therefore provides an appreciable facility allowing developers to write queries more quickly.

## 5.4 Discussion

In this sub-section, we compare our solution with the work presented in section 3.

In (Klettke et al., 2015), (Izquierdo & Cabot, 2016), (Maity et al., 2018), (Baazizi et al., 2017) (Baazizi et al., 2019), the authors propose processes that take as input a single collection and do not consider the links between objects. Similarly, the works (Sevilla Ruiz et al., 2015) and (Gallinucci et al., 2018) have the advantage of starting from a DB composed by several collections. However, they do not address the links between collections. On the

```

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 NoSQL_DB ::Collections::toCollection():NoSQL_Schema::Collection{
      cName:=self.cName;
      atomicfield:=self.atomicfield -> map toAtomicField();
      structuredfield:=self.structuredfield -> map toStructuredField();
      -- Transforming Atomic Fields
      mapping NoSQL_DB::AtomicField::toAtomicField():NoSQL_Schema::AtomicUField{
        fieldname:=self.fieldname -> map toFieldName();
        fielduvalue:=self.fieldvalueform -> map toFieldValue1();
        fielduvalue:=self.fieldvalue -> map toFieldValue2();
      }
      mapping NoSQL_DB::FieldName::toFieldName():NoSQL_Schema::FieldUName{NameU:=self.NameU};
      mapping NoSQL_DB::FieldUValue::toFieldValue1():NoSQL_Schema::FieldUValue{
        if ((self.FieldUValue = "True") or (self.FieldUValue = "False")) {FieldUValue:= "Boolean";}
        FieldUValue:= "Number"; endif;
      }
      mapping NoSQL_DB::FieldUValueForm::toFieldValue2():NoSQL_Schema::FieldUValue{
        if (self.FieldUValueForm = "") {FieldUValue:= "String";} endif;
        if (self.FieldUValueForm = "--/--/--") {FieldUValue:= "Date";} endif;
      }
      -- Transforming Structured Fields
      mapping NoSQL_DB::StructuredField::toStructuredField():NoSQL_Schema::StructuredUField{

```

Figure 5: Extract of the QVT code.

other hand, the works of (Comyn-Wattiau & Akoka, 2017) do not take into consideration structured attributes because the graphical oriented system Neo4J does not allow to declare this type of attributes.

To overcome these limitations, we proposed a more complete solution (ToConceptualSchema). This solution considers two techniques to describe the relationships between objects in the medical application: structured attributes and links. Indeed, our process considers two types of links (monovalued and multivalued) between the collections of the document-oriented database. In addition, it considers all types of attributes, whether atomic, structured, or multivalued.

## 6 CONCLUSION

Our work falls within the framework of Big Data DBs. They currently focus on the schema extraction mechanisms of a schemaless NoSQL database to facilitate query expression.

In this article, we have proposed an automatic process to extract the schema from a document NoSQL DB. This process based on the MDA provides a formal framework for the automation of model transformation. It generates the DB schema by applying a series of transformations expressed in QVT language.

Our ToNoSQLSchema process is based on the MDA architecture, which provides both a formal framework and the ability to evolve DB systems. In addition, it proposes the considering of monovalued and multivalued association links between the documents contained in the DB.

Currently we are studying a functionality to maintain the schema obtained by the ToNoSQLSchema process. It is a question of reflecting in the schema the structural evolutions of the DB throughout its exploitation.

## 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(6).
- Baazizi, M.-A., Colazzo, D., Ghelli, G., & Sartiani, C. (2019). Parametric schema inference for massive JSON datasets. *The VLDB Journal*, 28(4), 497–521.
- Baazizi, M.-A., Lahmar, H. B., Colazzo, D., Ghelli, G., & Sartiani, C. (2017). Schema inference for massive JSON datasets.
- Bézivin, J., & Gerbé, O. (2001). Towards a precise definition of the OMG/MDA framework. *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001)*, 273–280.
- Budinsky, F., Steinberg, D., Ellersick, R., Grose, T. J., & Merks, E. (2004). *Eclipse modeling framework: A developer's guide*. Addison-Wesley Professional.
- Chen, C. P., & Zhang, C.-Y. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information Sciences*, 314–347.
- Comyn-Wattiau, I., & Akoka, J. (2017). Model driven reverse engineering of NoSQL property graph databases: The case of Neo4j. *2017 IEEE International Conference on Big Data (Big Data)*, 453–458.
- DB-Engines Ranking. (n.d.). DB-Engines. Retrieved December 10, 2020, from <https://db-engines.com/en/ranking>
- Douglas, L. (2001). 3d data management: Controlling data volume, velocity and variety. *Gartner*. Retrieved, 6(2001), 6.
- Gallinucci, E., Golfarelli, M., & Rizzi, S. (2018). Schema profiling of document-oriented databases. *Information Systems*, 75, 13–25.
- Han, J., Haihong, E., Le, G., & Du, J. (2011). Survey on NoSQL database. *2011 6th International Conference on Pervasive Computing and Applications*, 363–366.
- Hutchinson, J., Rouncefield, M., & Whittle, J. (2011). Model-driven engineering practices in industry. *Proceedings of the 33rd International Conference on Software Engineering*, 633–642.
- Izquierdo, J. L. C., & Cabot, J. (2016). JSONDiscoverer: Visualizing the schema lurking behind JSON documents. *Knowledge-Based Systems*, 103, 52–55.
- Klettke, M., Störl, U., & Scherzinger, S. (2015). Schema extraction and structural outlier detection for JSON-based NoSQL data stores. *Datenbanksysteme Für Business, Technologie Und Web*.
- Maity, B., Acharya, A., Goto, T., & Sen, S. (2018). A Framework to Convert NoSQL to Relational Model. *Proceedings of the 6th ACM/ACIS International Conference on Applied Computing and Information Technology*, 1–6.
- MongoDB.(2018).<https://docs.mongodb.com/manual/reference/database-references/>
- Sevilla Ruiz, D., Morales, S. F., & Molina, J. G. (2015). Inferring versioned schemas from NoSQL databases and its applications. *International Conference on Conceptual Modeling*, 467–480.