Discovering of a Conceptual Model from a NoSQL Database

Fatma Abdelhedi[1], Amal Ait Brahim[2], Rabah Tighilt Ferhat[2] and Gilles Zurfluh[2]
[1]*CBI2, TRIMANE, Paris, France*
[2]*Toulouse Institute of Computer Science Research (IRIT), Toulouse Capitole University, Toulouse, France*

Keywords: NoSQL, Big Data, Schema-less, Model Extraction, MDA.

Abstract: NoSQL systems have proven effective to handle Big Data. Most of these systems are schema-less which means that the database doesn't have a fixed data structure. This property offers an undeniable flexibility allowing the user to add new data without making any changes on the data model. However, the lack of an explicit data model makes it difficult to express queries on the database. Therefore, users (developers and decision-makers) still need the database data model to know how data are stored and related, and then to write their queries. In a previous work, we proposed a process to extract a physical model from a NoSQL database. In this article, we propose to extend this process by leading to the extraction of a conceptual model that provides an element of semantic knowledge close to human understanding. To do this, we use the Model Driven Architecture (MDA) that provides a formal framework for automatic model transformation. From a NoSQL physical model, we propose formal transformation rules to generate a conceptual model in the form of a UML class diagram. An experimentation of the extraction process was carried out on a medical application.

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. This has impacted the tools required to store Big Data, and new kinds of data management tools, i.e. NoSQL systems have arisen (Chen, 2014). Compared to existing DBMS, NoSQL systems are commonly accepted to support larger volume of data and to 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. This property offers an undeniable flexibility that facilitates the data model evolution and allows end-users to add new information without the need of database administrator; but, at the same time, it makes the database manipulation more difficult. Indeed, even in Big Data context, the user still needs a data model that offers a visibility of how data is structured in the database (table names, attribute names and types, links, etc.)

In practice, the developer that has created the database, is also in charge of writing queries. Thus, he already knows how data is stored and related in the database; so, he can easily express his requests. However, this solution cannot be applied to all cases; for instance, the developer who is asked for doing the application maintenance, does not know the data model. It is the same for a decision maker who needs to query a database while he was not involved in its creation.

On the one hand, NoSQL systems have proven their efficiency to handle Big Data. On the other hand, the needs of a NoSQL database model remain up-to-date. Therefore, we are convinced that it's important to provide to the developer data models describing the database: (1) a physical model that describes the internal organization of data and allows to express queries and (2) a conceptual model that provides a high level of abstraction and a semantic knowledge element close to human comprehension, which guarantees efficient data management (Sevilla, 2015). The physical model is already extracted in a previous work. In this article, we are interested in extending this extraction to the conceptual level by

Abdelhedi, F., Brahim, A., Ferhat, R. and Zurfluh, G.
Discovering of a Conceptual Model from a NoSQL Database.
DOI: 10.5220/0009796100610072
In *Proceedings of the 22nd International Conference on Enterprise Information Systems (ICEIS 2020) - Volume 1*, pages 61-72
ISBN: 978-989-758-423-7

proposing a process which extracts a conceptual model (UML class diagram) from the physical model already extracted.

We should highlight that we formalized and automated our solution using the Model Driven Architecture (MDA) proposed by the Object Management Group (OMG, 2019) and that is well known as a framework for models automatic transformations.

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 reviews previous works. Section 4 describes our model-based process for extracting a conceptual model from schema-less NoSQL databases. Section 5 details our experiments, compares our solution against those presented in Section 3 and validates our solution. Finally, Section 6 concludes the paper and announces future work.

2 ILLUSTRATIVE EXAMPLE

To motivate and illustrate our work, we have used a case study in the healthcare field. 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).

In these programs, one of the benefits of using NoSQL databases is that the evolution of the data (and the model) 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 (1) differs from one patient to another and (2) evolves in unpredictable way over time.

As mentioned before, this kind of systems operate on schema-less data model enabling developers to quickly and easily incorporate new data into their applications without rewriting tables. Nevertheless, there is still a need for a conceptual 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 (Bondiombouy, 2015).

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 ToConceptualModel process presented in section 4 that extracts a conceptual model of a NoSQL database. This model is expressed using a UML class diagram.

3 RELATED WORK

The problem of extracting the data model from schema-less NoSQL databases has been the subject of several research works. Most of these works focus on the physical level (Klettke, 2015), (Sevilla, 2015), (Gallinucci, 2018), (Maity, 2018), (Baazizi, 2017), (Baazizi, 2019) and (Mongo, 2019). For our part, we proposed an extracting process of the physical model from a document-oriented NoSQL database. This process, based on the Model Driven Architecture (MDA), aims to extract a model from the database that allows users to express queries. It generates the physical model of the database by applying a sequence of transformations formalized with the QVT standard. The returned model describes the collections that make up the database. The major contribution of our solution is taking into account the links between these collections.

However, few research works have studied the extraction of a conceptual model from NoSQL databases; here the conceptual term qualifies a semantic model devoid of any technical consideration. Thus in (Comyn-Wattiau, 2017), the authors propose an extraction process of a conceptual model for a graph-oriented NoSQL database (Neo4J). In this particular type of NoSQL databases, the database contains nodes (objects) and binary links between them. The proposed process takes as input

Table 1: Comparative table of extraction works of conceptual model from schema-less NoSQL databases.

	Modeling levels		**Types of NoSQL systems**		**Types of links**	
	Physical	Conceptual	Graph	Document	Association	Composition
(Comyn-Wattiau, 2017)	X	X	X		X	
(Izquierdo, 2016)	X	X		X		X
(Chillón, 2019)	X	X		X	X	X

the insertion requests of objects and links; and then returns an Entity / Association model. This process is based on an MDA architecture and successively applies two transformations. The first is to build a graph (Nodes + Edges) from the Neo4j query code. The second consists of extracting an Entity / Association model from this graph by transforming the nodes with the same label into entities and the edges into associations. These works are specific to graph-oriented NoSQL databases generally used to manage strongly linked data such as those from social networks.

Furthermore, another work (Izquierdo, 2016) proposes a process to extract a conceptual model (UML class diagram) from a JSON document. This process consists of 2 steps. The first step is to extract a physical model in JSON format. The second step generates the UML class diagram by transforming the physical model into a root class RC, then the primitive fields (Number, String and Boolean) into RC attributes and the structured fields into component classes CC linked to RC by composition links. Thus, this work only considers the composition links and ignores association links.

On the other hand, in (Chillón, 2019), the authors propose a process to transform a document-oriented database into a conceptual model. It consists of entities with one or more versions according to the attributes they contain. An attribute can be atomic or multivalued with atomic elements and a relationship can be of type association or composition. An association relationship between two entities is obtained by transforming a reference field (using a specific syntax proposed by the authors). In addition, any structured field in the physical model is systematically transformed into a composition link. Thus, these works do not consider either structured attributes or association classes in the conceptual schema.

In Table 1, we summarize the three previous works by considering modeling levels, types of NoSQL systems as well as types of links.

This state of the art shows that the proposed solutions only partially answer our problem. Indeed, the works in (Comyn-Wattiau, 2017) concern only graph-oriented systems that do not take into account either structured attributes or composition links. On the other hand, in the article (Izquierdo, 2016), the authors do not consider association links that are the most common in our case study. Finally, in (Chillón, 2019), the authors do not take into account structured attributes in a class as well as the concept of association classes that makes it possible to characterize a relation by attaching attributes to it.

4 EXTRACTION PROCESS OF THE CONCEPTUAL MODEL

Our work aims to provide users with models to manipulate NoSQL databases. Two models are proposed: (1) the physical model to write queries on this database and application code and (2) the semantic model to give the meaning of the data contained in the database. When data structures are complex, these two models are essential to enable users (usually decision-makers) to understand and manipulate data. As part of this work, we proposed mechanisms to discover a physical model from a NoSQL database in a previous article. This work is based on the MDA architecture and applies a set of transformation rules formalized in QVT. The result model describes the internal organization of the NoSQL database, taking into account the technical details. This model shows the names of collections, attributes and links between collections. The current paper completes the latter and focuses on thetransformation of the physical model into a conceptual model represented by using a UML class diagrams (red circle in Figure 1) and which provides users with the semantics of the data. Note that we limit our study to document-oriented NoSQL databases that are the most complete to express links between objects (use of referenced and nested data).
We propose the ToConceptualModel process which applies a set of transformations ensuring the passage of a NoSQL physical model towards a UML class

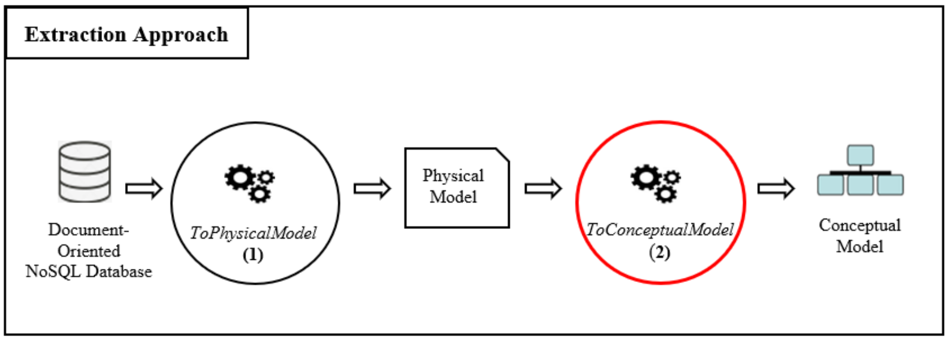


Figure 1: Overview of ToConceptualModel process.

diagram. To formalize and automate our process, we use the Model Driven Architecture (MDA) proposed by the OMG (OMG, 2019), 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. For this, it uses three models representing the abstraction levels of the application. These are (1) the requirements model CIM (Computation Independent Model) in which no IT considerations appear, (2) the independent Platform Independent Model (PIM) independent of technical details and (3) Platform Specific Model (PSM) specific to a particular platform. The transition between the different models is done through a succession of transformations essentially between PIM and PSM. These transformations can be classified according to two categories: M2M (Model-To-Model) when the result of the transformation is a model and M2T (Model-To-Text) when the result is a code.

Since the input of our process is a NoSQL physical model and the output corresponds to the conceptual model (UML class diagram), we retain only the PIM and PSM levels. The transition from the PSM to the PIM is done through a sequence of M2M transformations. We will formalize these transformations using the QVT standard (Query View Transformation) defined by the OMG (Section 5).
In the following subsections, we detail the components of the ToConceptualModel process by specifying the three elements: (a) the source, (b) the target and (c) the transformation rules.

4.1 Source: Physical Model

The physical model is produced by the ToPhysicalModel process shown in Figure 1 was presented in a previous work. In this paper, it is the source of the ToConceptualModel process that we will study here.

This model is composed of collections. Each of them corresponds to the model of a collection of the database and describes its different fields. A field is defined by a name and an atomic, structured or multivalued type. The atomic type is one of the standard data types such as Number, String or Boolean. The structured type (between {}) is composed of one or more fields that can also be atomic, structured or multivalued. The multivalued type is put between []; it consists of atomic, structured or multivalued elements. To express a link between collections, we used a field called DBRef, which is un standard proposed by the MongoDB (MongoDB, 2018). This one is a special case of a structured field. It is composed of two fields; one corresponds to the identifier of the referenced document and the other corresponds to the name of the collection that contains the referenced document.

We present the different concepts of the physical model in the meta-model of Figure 2. All metamodels presented in this article are formalized with the Ecore standardized language that we present in the Experimentation section.

4.2 Target: UML Class Diagram

A UML Class Diagram is defined by a set of classes linked together by relationships. Each class is defined by a name, attributes, and operations. Each attribute is defined by a pair (name, type). Note that an attribute can be structured, i.e. it is composed of other attributes like Address which is composed of: StreetName, ZipCode and City.

In this article, we do not consider operations. A relationship is a semantic connection between two or more classes. The most common links are association, composition, aggregation and inheritance. In this article, we restrict ourselves only to links used in our case study: association and composition. An association link is defined by a name and its extremities; a composition link is defined by its two

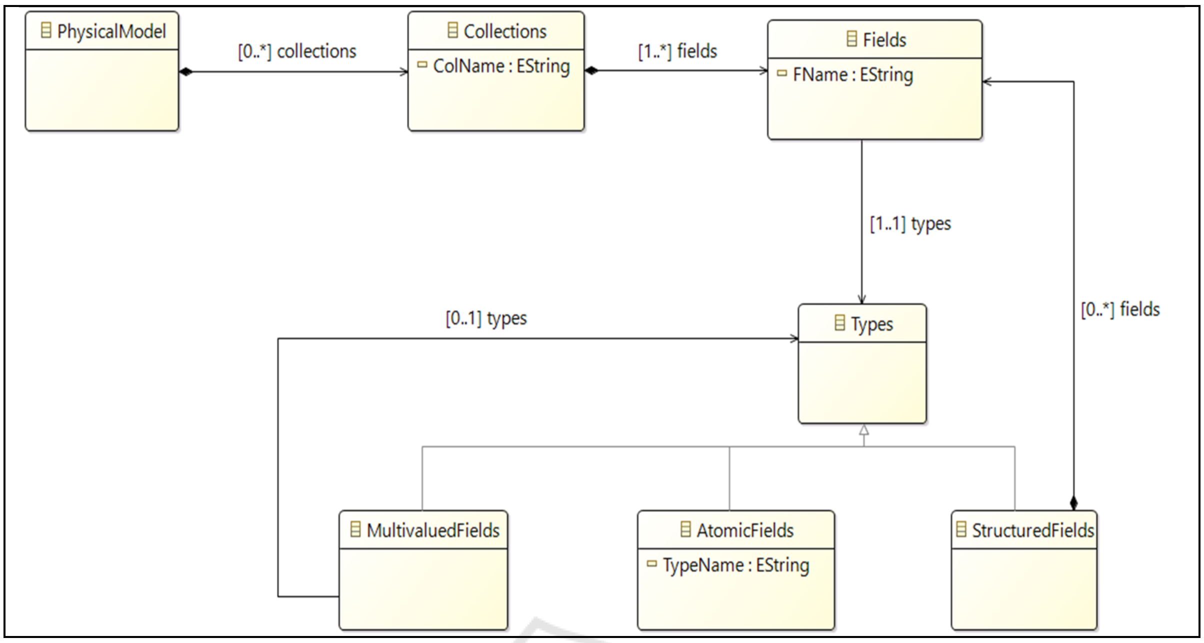


Figure 2: Source Metamodel.

extremities and its meaning (composite class and component class). An extremity can be characterizedby a special constraint called multiplicity that are defined by a lowerCardinality and an upperCardinality. Note that an association link can be characterized by attributes; in this case, it is an association class. We formalize all these concepts through the Ecore meta-model of Figure 3.

4.3 Transformation Rules

After having formalized the concepts present in the source model (Physical Model) then in the target model (UML Class Diagram), we will describe the automatic transition between the two models as a set of transformation rules. These rules will be formalized later in the Experimentation section using the QVT standard. Links between collections (DBRef fields) will be taken into account from rule R5.

R1: A collection is transformed into a class with the same name.

R2: With the exception of the _id field, an atomic field in a collection is transformed into an atomic attribute with the same name and type in the corresponding class.

R3: A multivalued field whose elements are atomic is transformed into a multivalued attribute with the same name and type.

R4: A structured field is transformed into a structured attribute with the same name. The elements constituting the input structured field are transformed by applying R2 if it is an atomic element, R3 for a multivalued element or R4 for a structured element.

R5: A DBRef field is a structure of the following form:

<DBRef_Name>: {$_id: ObjectId,
$Ref: <Collection_Name>}.

A DBRef field in a collection *A* referencing a collection *B* is transformed into an association relationship connecting the two corresponding classes *X* and *Y*; this association relationship has the same name as the DBRef field and has the following cardinalities:

0..1 for the class Y and 0..* for the class X if the field DBRef is monovalued,

0..* for the class Y and 0..* for the class X if the field DBRef is multivalued.

R6: A DBRef field may consist of additional fields as follows:

<DBRef_Name>: {$_id: ObjectId,
$Ref: <Collection_Name>,
<Champ1>: Type,
<Champ2>: Type,
...},

In this case, a DBRef field in a collection A referencing a collection B is transformed into an association class between the corresponding classes X and Y; this association class has the same name as the DBRef field and contains the additional fields as attributes. If these fields are of type DBRef, they are

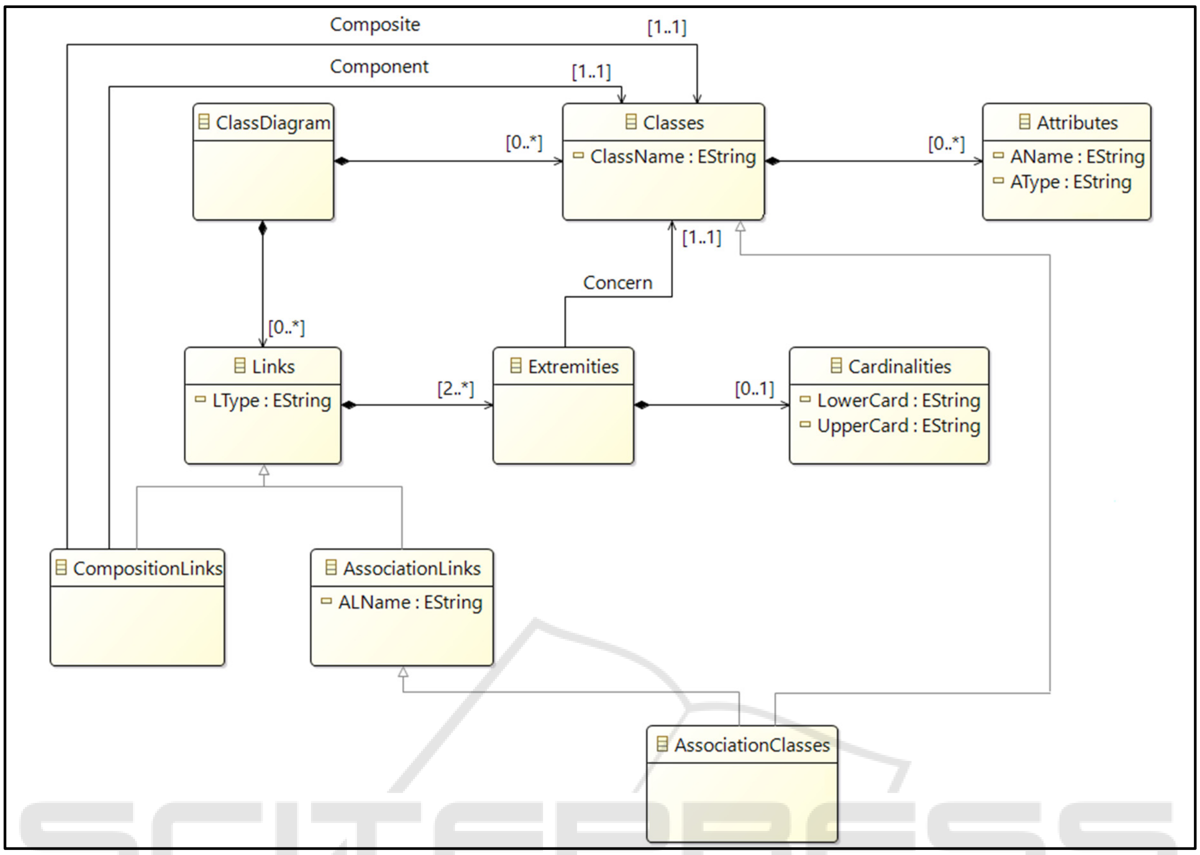


Figure 3: Target Metamodel.

transformed into association relationships by applying R5. The association class obtained has the same cardinalities as the association relationship (R5).

R7: In a collection A, if a field is structured or multivalued of structured elements and the structure consists of at least one field DBRef, then this structured field is transformed into a component class Z linked to the class X (corresponding to A) with a composition link. This one has has the following cardinalities:

0..1 for the class Z if the input field is structured,

0 ..* for the class Z if the input field is multivalued of structured elements.

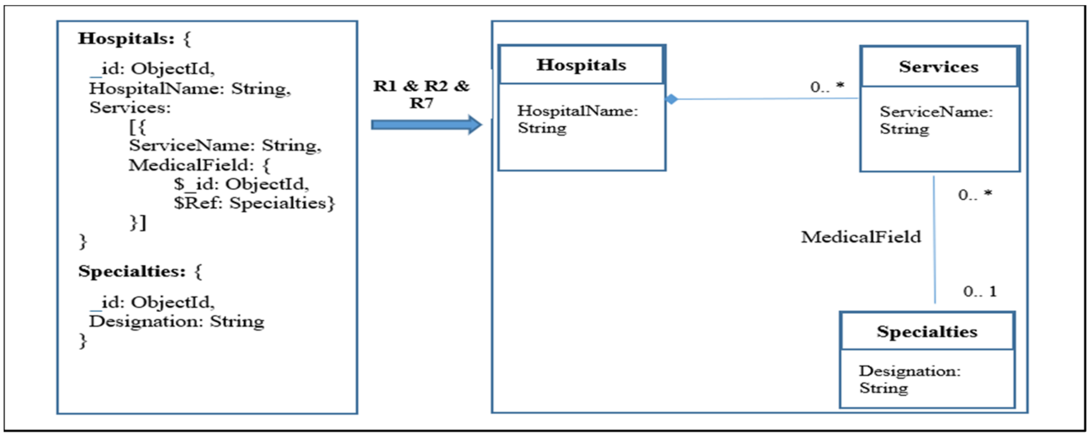
The component class Z is linked with another class T by applying the rule 5.

To illustrate the implementation of our process, Figure 4 shows the transformation of a physical model to a UML class diagram by applying three rules; other examples are presented in the Experiment section. Thus, the classes Hospitals and Specialties are obtained by applying R1 on the collections that have the same names, respectively on the physical model. The atomic Attributes HospitalName and Designation result from the application of R2. Finally, the component class Services as well as the relationship MedicalField are obtained by applying R7.

5 EXPERIMENTS

5.1 Technical Environment

In this section, we describe the techniques used to implement the *ToConceptualModel* process. 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) (Budinsky, 2004). 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 handled by the user. The second is the creation of the models instantiating this meta-model and the third one is the transformation



```
Hospitals: {
  _id: ObjectId,
  HospitalName: String,
  Services:
     [{
     ServiceName: String,
     MedicalField: {
          $_id: ObjectId,
          $Ref: Specialties}
     }]
}
Specialties: {
  _id: ObjectId,
  Designation: String
}
```

Figure 4: Transformation example.

from model to model and from model to text. Among the tools provided by EMF, we used:

Ecore: a metamodeling language used to create our metamodels. Figure 2 and Figure 3 illustrate the source and target Ecore meta-models used by *ToConceptualModel* process.

XML Metadata Interchange (XMI): the XML based standard that we use to create models.

QVT (Query, View, Transformation): the OMG standard language for specifying model transformations. The choice of the QVT standard 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.

5.2 Implementation of the ToConceptualModel Process

ToConceptualModel process is expressed as a sequence of elementary steps that build the resulting model (UML class diagram) step by step from the source model (physical model).

Step1: we create a source and a target metamodel to represent the concepts handled by our process.

Step2: we build an instance of the source metamodel. For this, we use the standard based XML Metadata Interchange (XMI) format. This instance is shown in Figure 5

Step3: we implement the transformation rules by means of the QVT language provided within EMF.

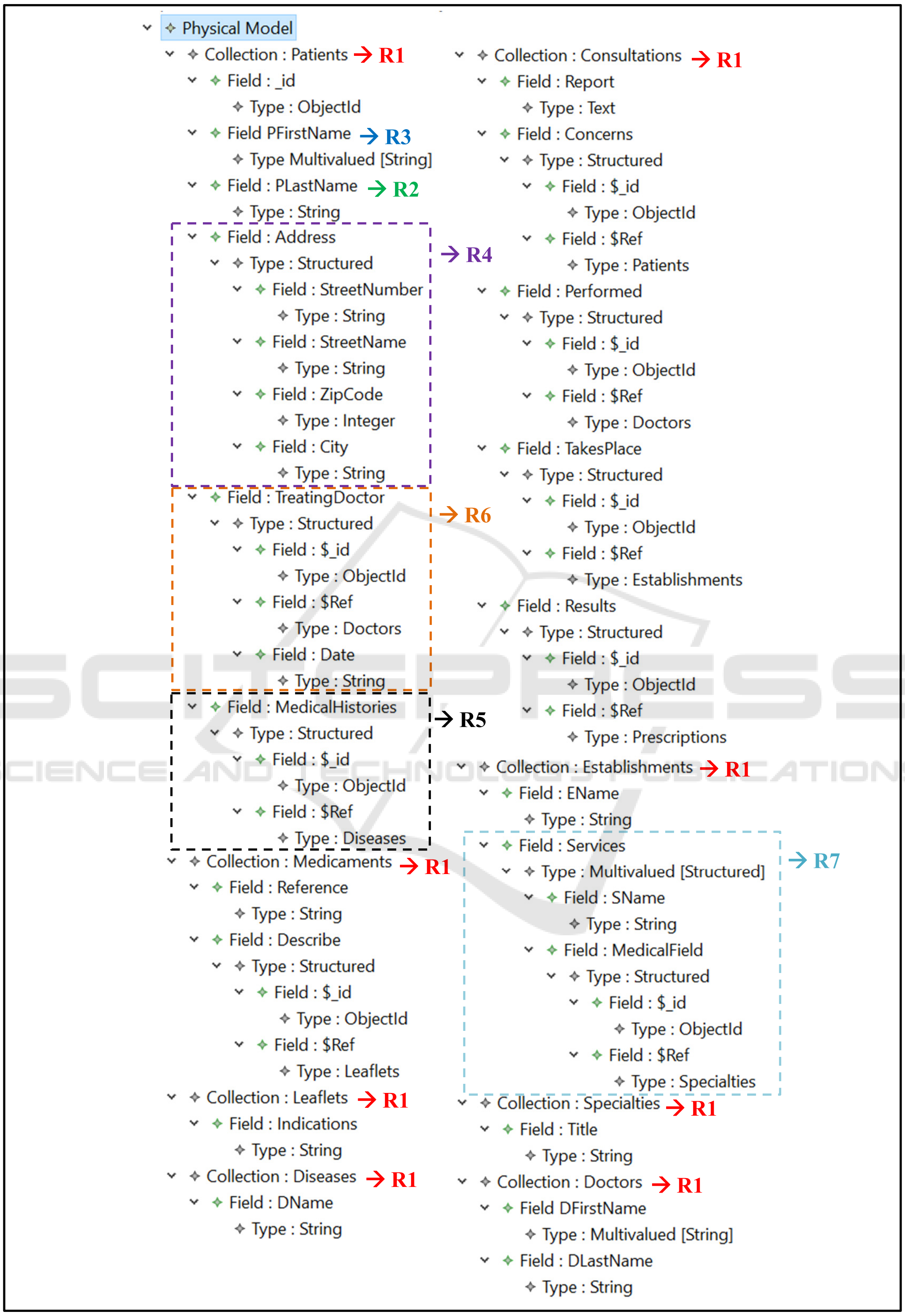
Step4: we test the transformation by running the QVT script created in step 3. This script takes as input the source model built in step 2 (physical model) and returns as output a UML class diagram. The result is provided in the form of XMI file as shown in figure 6.

5.3 Comparison

The aim of this section is to compare our solution with the three works (Comyn-Wattiau, 2017), (Izquierdo, 2016), (Chillón, 2019) presented in section 3 and that have investigated the process of extracting a NoSQL database conceptual model.

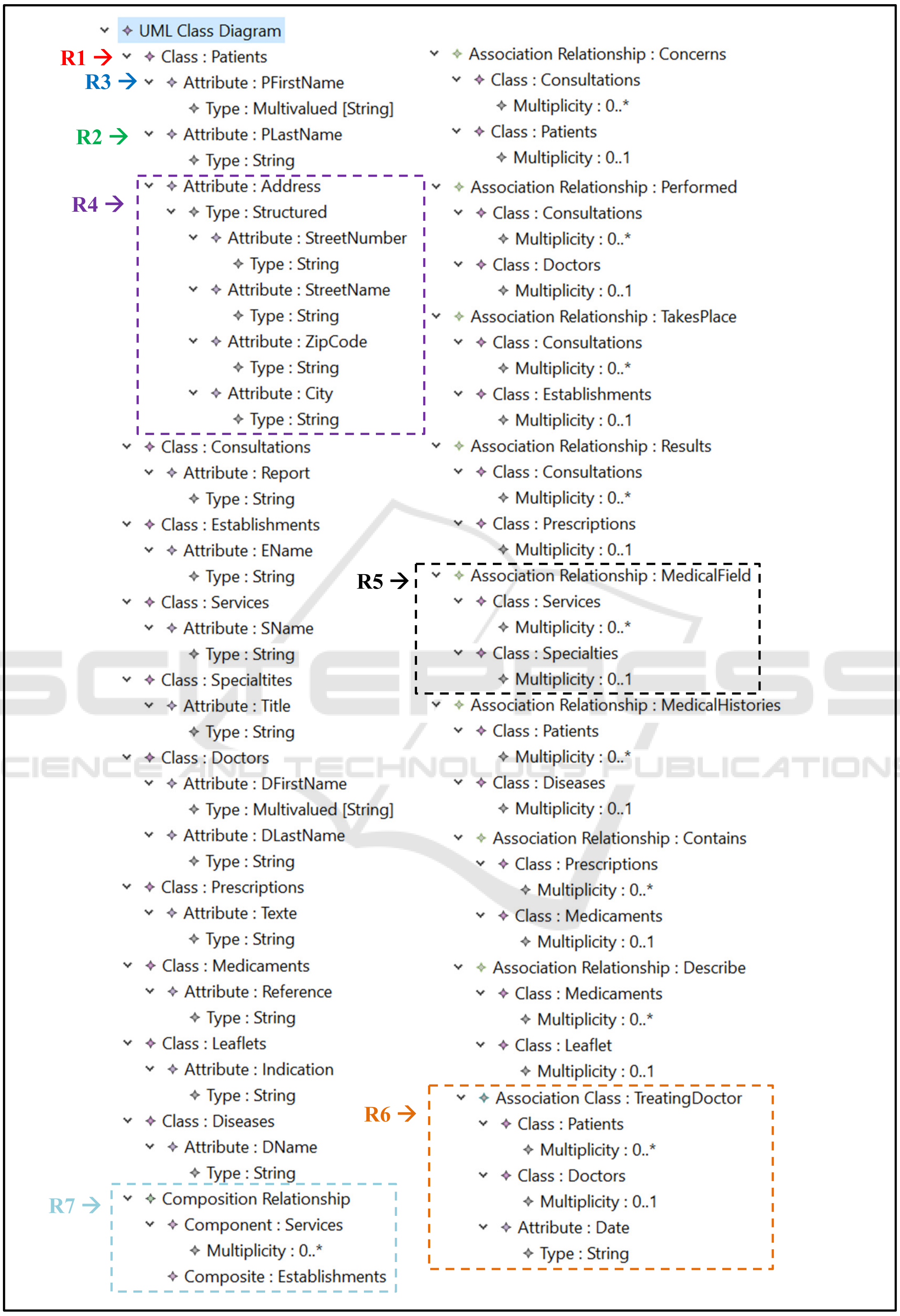
Starting from a graph-oriented NoSQL database, authors in (Comyn-Wattiau, 2017) propose to extract an E/A model based on a set of mapping rules between the conceptual level and the physical one. Obviously, these rules are specific to graph-oriented systems used as a framework for managing complex data with many connections. This kind of NoSQL DBMS lack of ability to define structured attributes and composition links that we need to use in our use case (cf. Section 2). The solution presented in (Izquierdo, 2016) have the advantage to start from a document-oriented NoSQL database. But the proposed mapping doesn't take into account association links between collections; such type of links is a key element in our case to describe relationships between the medical application objects. Other process in (Chillón, 2019) focuses on association links during the extraction of a document-oriented NoSQL database, however it doesn't consider structured attributes and association classes.

To overcome the limits of these works, we have proposed a more complete solution based on the Model Driven Architecture (MDA). Table 2 summarizes the main features of our process and sets them against those of (Comyn-Wattiau, 2017), (Izquierdo, 2016) and (Chillón, 2019) processes.



```
Physical Model
  Collection : Patients → R1
    Field : _id
      Type : ObjectId
    Field PFirstName → R3
      Type Multivalued [String]
    Field : PLastName → R2
      Type : String
    Field : Address                          → R4
      Type : Structured
        Field : StreetNumber
          Type : String
        Field : StreetName
          Type : String
        Field : ZipCode
          Type : Integer
        Field : City
          Type : String
    Field : TreatingDoctor                   → R6
      Type : Structured
        Field : $_id
          Type : ObjectId
        Field : $Ref
          Type : Doctors
        Field : Date
          Type : String
    Field : MedicalHistories                 → R5
      Type : Structured
        Field : $_id
          Type : ObjectId
        Field : $Ref
          Type : Diseases
  Collection : Medicaments → R1
    Field : Reference
      Type : String
    Field : Describe
      Type : Structured
        Field : $_id
          Type : ObjectId
        Field : $Ref
          Type : Leaflets
  Collection : Leaflets → R1
    Field : Indications
      Type : String
  Collection : Diseases → R1
    Field : DName
      Type : String
  Collection : Consultations → R1
    Field : Report
      Type : Text
    Field : Concerns
      Type : Structured
        Field : $_id
          Type : ObjectId
        Field : $Ref
          Type : Patients
    Field : Performed
      Type : Structured
        Field : $_id
          Type : ObjectId
        Field : $Ref
          Type : Doctors
    Field : TakesPlace
      Type : Structured
        Field : $_id
          Type : ObjectId
        Field : $Ref
          Type : Establishments
    Field : Results
      Type : Structured
        Field : $_id
          Type : ObjectId
        Field : $Ref
          Type : Prescriptions
  Collection : Establishments → R1
    Field : EName
      Type : String
    Field : Services                         → R7
      Type : Multivalued [Structured]
        Field : SName
          Type : String
        Field : MedicalField
          Type : Structured
            Field : $_id
              Type : ObjectId
            Field : $Ref
              Type : Specialties
  Collection : Specialties → R1
    Field : Title
      Type : String
  Collection : Doctors → R1
    Field DFirstName
      Type : Multivalued [String]
    Field : DLastName
      Type : String
```

Figure 5: Source Model.



```
UML Class Diagram
  Class : Patients
    Attribute : PFirstName
      Type : Multivalued [String]
    Attribute : PLastName
      Type : String
    Attribute : Address
      Type : Structured
        Attribute : StreetNumber
          Type : String
        Attribute : StreetName
          Type : String
        Attribute : ZipCode
          Type : String
        Attribute : City
          Type : String
  Class : Consultations
    Attribute : Report
      Type : String
  Class : Establishments
    Attribute : EName
      Type : String
  Class : Services
    Attribute : SName
      Type : String
  Class : Specialtites
    Attribute : Title
      Type : String
  Class : Doctors
    Attribute : DFirstName
      Type : Multivalued [String]
    Attribute : DLastName
      Type : String
  Class : Prescriptions
    Attribute : Texte
      Type : String
  Class : Medicaments
    Attribute : Reference
      Type : String
  Class : Leaflets
    Attribute : Indication
      Type : String
  Class : Diseases
    Attribute : DName
      Type : String
  Composition Relationship
    Component : Services
      Multiplicity : 0..*
    Composite : Establishments
  Association Relationship : Concerns
    Class : Consultations
      Multiplicity : 0..*
    Class : Patients
      Multiplicity : 0..1
  Association Relationship : Performed
    Class : Consultations
      Multiplicity : 0..*
    Class : Doctors
      Multiplicity : 0..1
  Association Relationship : TakesPlace
    Class : Consultations
      Multiplicity : 0..*
    Class : Establishments
      Multiplicity : 0..1
  Association Relationship : Results
    Class : Consultations
      Multiplicity : 0..*
    Class : Prescriptions
      Multiplicity : 0..1
  Association Relationship : MedicalField
    Class : Services
      Multiplicity : 0..*
    Class : Specialties
      Multiplicity : 0..1
  Association Relationship : MedicalHistories
    Class : Patients
      Multiplicity : 0..*
    Class : Diseases
      Multiplicity : 0..1
    Association Relationship : Contains
      Class : Prescriptions
        Multiplicity : 0..*
      Class : Medicaments
        Multiplicity : 0..1
    Association Relationship : Describe
      Class : Medicaments
        Multiplicity : 0..*
      Class : Leaflet
        Multiplicity : 0..1
    Association Class : TreatingDoctor
      Class : Patients
        Multiplicity : 0..*
      Class : Doctors
        Multiplicity : 0..1
      Attribute : Date
        Type : String
```

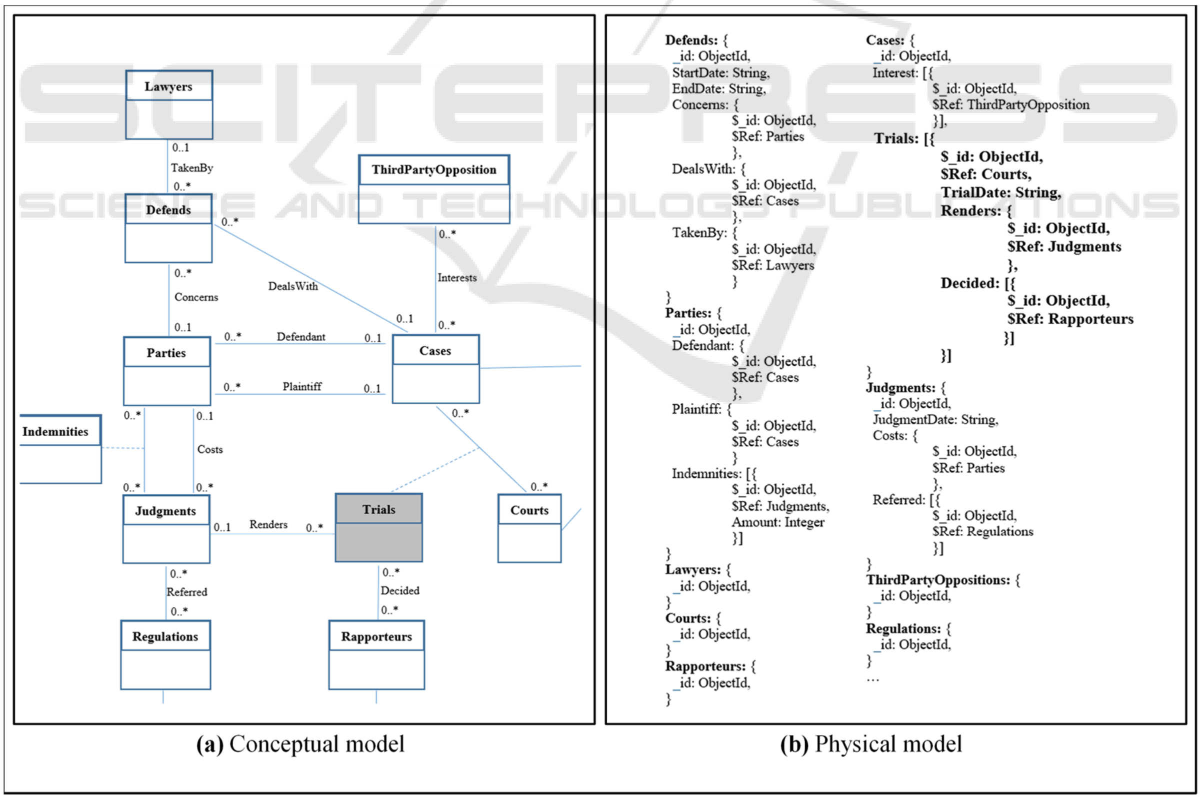
Figure 6: Target Model.

Table 2: Comparative table of solutions.

	Types of NoSQL systems		Types of links			Structured attributes	Use Of MDA
	Graph	Document	Association	Composition	Association class		
(Comyn-Wattiau, 2017)	X		X				X
(Izquierdo, 2016)		X		X			
(Chillón, 2019)		X	X	X			
Our Process		X	X	X	X	X	X

Table 3: Query writing time.

	Without model	Physical model alone	Conceptual and physical model
Developer 1	Database 1: 50 minutes	Database 2: 23 minutes	Database 3: 18 minutes
Developer 2	Database 2: 40 minutes	Database 3: 25 minutes	Database 1: 16 minutes
Developer 3	Database 3: 48 minutes	Database 1: 20 minutes	Database 2: 16 minutes
Average	**46 minutes**	**23 minutes**	**17 minutes**



```
Defends: {
  _id: ObjectId,
  StartDate: String,
  EndDate: String,
  Concerns: {
          $_id: ObjectId,
          $Ref: Parties
          },
  DealsWith: {
          $_id: ObjectId,
          $Ref: Cases
          },
  TakenBy: {
          $_id: ObjectId,
          $Ref: Lawyers
          }
}
Parties: {
  _id: ObjectId,
  Defendant: {
          $_id: ObjectId,
          $Ref: Cases
          },
  Plaintiff: {
          $_id: ObjectId,
          $Ref: Cases
          }
  Indemnities: [{
          $_id: ObjectId,
          $Ref: Judgments,
          Amount: Integer
          }]
}
Lawyers: {
  _id: ObjectId,
}
Courts: {
  _id: ObjectId,
}
Rapporteurs: {
  _id: ObjectId,
}

Cases: {
  _id: ObjectId,
  Interest: [{
          $_id: ObjectId,
          $Ref: ThirdPartyOpposition
          }],
  Trials: [{
          $_id: ObjectId,
          $Ref: Courts,
          TrialDate: String,
          Renders: {
                  $_id: ObjectId,
                  $Ref: Judgments
                  },
          Decided: [{
                  $_id: ObjectId,
                  $Ref: Rapporteurs
                  }]
          }]
}
Judgments: {
  _id: ObjectId,
  JudgmentDate: String,
  Costs: {
          $_id: ObjectId,
          $Ref: Parties
          },
  Referred: [{
          $_id: ObjectId,
          $Ref: Regulations
          }]
}
ThirdPartyOppositions: {
  _id: ObjectId,
}
Regulations: {
  _id: ObjectId,
}
…
```

(a) Conceptual model **(b)** Physical model

Figure 7: Screen representing the database of one of the three applications.

5.4 Validation

Concerning the model extraction of schema-less NoSQL databases, our approach allows to display to the developer simultaneously a conceptual model and a physical model; the first to understand the semantics of the database and the second to write queries. To evaluate the relevance of our approach, our prototype (section 4) was implemented by three developers at Trimane, a digital services company specialized in business intelligence and Big Data. The three experienced developers (IT consulting engineers) were tasked with providing maintenance for three separate applications. None of the developers know, previously, the data model of the concerned applications. For each application, each developer writes ten queries that have an increasing complexity according to three different cases: (1) without any data model, (2) with the physical data model or (3) with the both conceptual and physical models. Figures 7(a) and 7(b) show respectively an example of the conceptual and physical models corresponding to one of the three applications. Note that due to lack of place, we present data models (conceptual and physical one) of only one application.

We should also highlight that for reasons of visibility, models are represented to the user in the same screen and with an appropriate format: JSON for the physical model and the graphic format for the conceptual one. Each time we click on a class on the conceptual model, we will have its equivalent on the physical model. For example, the part of the physical model written in bold corresponds to the selected class (Trials).

Each database is associated with a set of queries whose natural language statements are provided to the three developers. In Table 3, we calculated the average time of writing the queries by the three developers in each situation: (1) without any data model, (2) with the physical data model or (3) with the both conceptual and physical models.

Our initial hypothesis was verified in the situations considered. This establishes that a knowledge of semantics and data structure allows the developer to write queries faster on a schema-less NoSQL database. The small difference noted between the use of the single physical diagram and the use of the two models (conceptual and physical), is probably due to the experience of the three developers.

6 CONCLUSION AND FUTURE WORK

Our work is part of Big Data databases. They are currently dealing with the reverse engineering mechanisms of schema-less NoSQL databases to provide users with models to manipulate NoSQL databases.

In this article, we have proposed an automatic process ToConceptualModel which focuses on the transformation of a physical model into a conceptual model represented using a UML class diagrams by applying a set of rules. The resulting conceptual model makes it easier for developers and decision-makers to understand the database and write queries. To formalize and automate our process, we use the Model Driven Architecture (MDA) proposed by the OMG, which provides a formal framework for automating model transformations.

The major contribution of our solution is the consideration of structured attributes, association relationships, composition relationships as well as association classes. We have experimented our process on the case of a medical application which relates to scientific programs of follow-up of pathologies; the database is stored on a document-oriented NoSQL Database.

As future work, we plan to complete our transformation process to have more semantics in the conceptual model by considering other types of links such as inheritance, aggregation and N-ary.

REFERENCES

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.

Extract Mongo Schema https://www.npmjs.com/package/extract-mongo-schema/v/0.2.9 Online; 5 October 2019.

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

Izquierdo, J. L. C., & Cabot, J. (2016). JSONDiscoverer: Visualizing the schema lurking behind JSON documents. Knowledge-Based Systems, 103, 52-55.

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 (2018). Mongodb atlas database as a service. https://www.mongodb.com/. Online; 5 November 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.

Chillón, A. H., Ruiz, D. S., Molina, J. G., & Morales, S. F. (2019). A Model-Driven Approach to Generate Schemas for Object-Document Mappers. IEEE Access, 7, 59126-59142.

Object Management Group (2019) https://www.omg.org/ Online; 5 July 2019.