

# Extraction Process of the Logical Schema of a Document-oriented NoSQL Database

Fatma Abdelhedi<sup>1</sup><sup>a</sup>, Hela Rajhi<sup>2</sup><sup>b</sup> and Gilles Zurfluh<sup>2</sup><sup>c</sup>

<sup>1</sup>*CBI<sup>2</sup> - TRIMANE, Saint Germain-En-Laye, France*

<sup>2</sup>*IRIT, Toulouse Capitole University, Toulouse, France*

**Keywords:** NoSQL DB, OrientDB DB, Schemaless, Logical Schema, Models Transformation, ATL, Metamodels.

**Abstract:** The "schemaless" property, common to most NoSQL systems, means the absence of a data schema when creating a database (DB). This property brings an undeniable flexibility by allowing the schema to evolve during the use of DB. However, the absence of a schema is a major obstacle for developers and decision makers. Indeed, the expression of queries (of SQL type) requires a precise knowledge of this schema. In this paper, we propose an automatic process to extract the logical schema of document-oriented NoSQL DBs. We chose the OrientDB NoSQL system which appeared to be the most suitable for the application in our project, because of its ability to express rich data structures and a diversity of links between data: association, composition and inheritance links. Our solution, based on the MDA architecture, proposes to metamodel a NoSQL DB and its schema. From these metamodels, transformation rules allow to extract the schema of the DB. The implementation of this process on an OrientDB DB allows users to have all the necessary elements (class names, properties, data types and links) for the elaboration of queries. An experimentation of the process was carried out on three test-DBs as well as on two massive industrial DBs.


## 1 INTRODUCTION


For several decades, the volume of digital data has increased dramatically due to the multiplicity of computing devices present in all areas of our professional, public and personal lives. Massive DBs or "Big Data" contain several terabytes of data from different sources and in various formats such as text, tables, documents... Currently, relational DBMS dominate the storage market; they require that data respect a schema provided before that is fed (Elmasri & Navathe, 2011). Big Data has favored the emergence of NoSQL systems that provide great flexibility in data management while offering good access performance to large volumes of data. This flexibility is notably allowed by the Schemaless property which does not require schema specification before data entry; thus, the rows of a table can contain different attribute names and values of various types. Thus, most NoSQL systems of document, column or


graph-oriented types (MongoDB, CouchDB, OrientDB, HBase, Neo4j) are schemaless.

In the absence of a schema, writing queries on a NoSQL DB needs to be entrusted to a developer: either who participated in the creation of the DB and therefore who implicitly knows its schema, or who will manually extract the schema by an often-random consultation of the content. A more rational solution is to use a process that automatically extracts the schema from the NoSQL DB and, in recent years, several mechanisms have been proposed to extract the schema from a NoSQL DB (Frozza, Jacinto and al., 2020), (Wang and al., 2015), (Cánovas Izquierdo and Cabot, 2016) and (Frozza, Defreyne and al., 2020).

In this paper, we propose a new process for extracting the logical schema of a document-oriented NoSQL DB called ToOrientDBSchema; the originality of our proposal lies mainly in the diversity of the semantic links considered. The choice of a document-oriented DB is related to the requirements of our case study presented in section 2. Once

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

<sup>b</sup> <https://orcid.org/0000-0001-8538-6229>

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

extracted, the schema will allow users (developers or decision makers) to easily formulate queries on the DB. This paper is organized as follows: Section 2 presents the medical application that justifies the interest of our work. Section 3 presents an overview of our solution and Section 4 reviews the state of the art. Section 5 presents our process by highlighting (i) the source metamodel, (ii) the target metamodel and (iii) the transformation rules. Section 6 describes the experimentation of our process on 3 test-DBs based on the medical application and on two massive industrial DBs. Section 7 positions our process to those proposed in related works. Finally, the conclusion in Section 8 presents perspectives to our work.

## 2 MOTIVATION

Our work is part of a medical application developed for an industrial project.

### 2.1 Medical Data

The application we are interested in concerns the implementation of scientific programs dedicated to the follow-up of rare pathologies on hospitalized patients. Each program may involve up to 50 European institutions (hospitals, clinics and specialized care centers). The main objective of a program is to collect significant data on the evolution of the disease over time, to study its interactions with other relevant diseases and to evaluate the influence of its treatments in the short and medium term. The duration of a program is determined when it is launched and can be between three and ten years. Data collected by multiple institutions in a multi-year program have generally the accepted characteristics of Big Data (the 3 Vs) (Laney, 2001). Indeed, the volume of medical data collected daily from patients can reach, for all establishments and over three years, several terabytes. On the other hand, the nature of the data entered (constant measurements, radiography, scintigraphy, etc.) is diverse and may be different from one patient to another depending on his state of health. Finally, some data are produced continuously by sensors; it must be processed in real time (measurements crossing a threshold that would involve the urgent intervention of a practitioner, for example). Patients follow-up requires the storage of various data such as the record of consultations carried out by practitioners, the results of analyses, prescriptions for medicine and specific treatments. We therefore stored all this data in the multi-model

OrientDB NoSQL system. Due to the nature of data to be stored, we use the schemaless document-oriented model of OrientDB (OrientDB, 2021 April).

One of the problems we are facing in this project is related to the manipulation of data stored in NoSQL schemaless systems. Indeed, the absence of a clearly identified data schema constitutes a major difficulty for writing queries. Thus, in order to formulate their queries, developers (computer scientists) and decision-makers (doctors, managers, etc) have to search empirically for the schema that is integrated in the stored data.

Our problem consists in developing a schema extraction process from a massive DB managed by a NoSQL schemaless system.

### 2.2 Application Development

Our application aims to develop a software environment for medical staff to (1) collect patients 'data and (2) query and analyze the history of this data.

To store this large and complex data of varying types and formats, the NoSQL OrientDB system offers advanced functionalities that are well suited to our application. However, the lack of data schema in this system represents a major obstacle for writing queries on the DB. Indeed, the expression of a query requires knowing the names of the classes as well as the names of the properties and their types. For example, let's consider the query asked by a doctor participating in a program: Obtain the reference of medicines prescribed by doctors to patients suffering from Creutzfeldt-Jacob disease. The translation of this query with an SQL type language could be as follows (extract):

```
Select tuple (x.name, m.ref)
From d in Diseases, x in Doctors,
p in Prescribe, m in Medicines
Where d.name = "Creutzfeldt-Jacob"
and ...
Group By x.name;
```

We can see in this example of a query expressed by a doctor (through an appropriate graphical interface) the need to know the schema of the DB, mainly the names of the classes and the names and types of the properties. We therefore developed a process to extract a logical data schema from an existing DB.

### 3 SOLUTION OVERVIEW

We have a DB managed by the NoSQL OrientDB system. Our process consists of extracting the logical schema from the DB and presenting it in a form that can be read by users (developers and decision-makers). We used the ModelToModel transformation approach of MDA (OMG, 2021 April) to generate the logical schema. We therefore present successively the characteristics of the OrientDB system, the principles of MDA and an overview of our process.

OrientDB is a multi-model NoSQL data storage and manipulation system in the sense that it supports several data organizations. Given the specificities of our application, we chose the document-oriented model whose records (i.e. objects) contain a set of properties (Key, Value); the values of the properties can belong to all types of data (atomic, structured and multivalued). One of the particularities of OrientDB system is the possibility of expressing association links in the form of pointers (reference values) according to the ODMG DB standard (ODMS, 2021 April). In addition, OrientDB is schemaless because, for a given class, the schema of the records is not provided when the class is created.

MDA is a branch of model-driven engineering (MDE) proposed by the OMG (OMG, 2021 June). It is a software development architecture that distinguishes several levels of description making it possible to disregard the technical characteristics (PIM, CIM, PSM) of an application. Thus, the PSM (Platform Specific Model) corresponds to descriptions taking into account the technical characteristics of an implementation platform. In addition, MDA offers model transformation principles and techniques for generating code or, inversely, extracting the model from existing code. This involves applying transformation rules on metamodels describing the starting point (the source) and the arrival (the target). The Eclipse Foundation (Eclipse, 2021 April) has developed implementation tools in accordance with MDA. The objective of our work is to obtain the (unique) schema of an OrientDB schemaless DB. MDA offers us extraction principles consisting in metamodeling the source (DB) and the target (schema) and then applying rules of passage from the source to the target.

This solution has the advantage of being able to be applied to different document-oriented NoSQL DBs (managed by OrientDB or by other systems accepting this model). However, it faces some difficulties related mainly to the detection of data types and links; we therefore made some initial assumptions in section 5.1.

### 4 RELATED WORKS

Several NoSQL DB schema extraction softwares have been proposed by software publishers such as "Spark Dataframe" (Apache Spark, 2021 Oct), "Schema-guru" (SnowPlow Analytics, 2021 Oct) and "Mongodb-schema" (Peter Schmidt, 2021 Oct). These softwares extract the class schema (designated also by tables or collections) from a DB in JSON format; but these softwares do not extract the semantic relationships between objects.

In addition, research works proposed extracting more complete schemas from NoSQL DB. In (Baazizi and al., 2017), the authors propose a process of schema extraction from a JSON dataset using the Map-reduce system. This process can be summarized in 2 phases: the first consists in applying the Map transformation to each record of a class in order to deduce the pairs (key, type) from the pairs (key, value). The result of this step allows to obtain several schemas specific to each record. The Reduce phase consists of merging these schemas in order to provide a global schema for each class. This process was extended by the same authors by integrating the parameterization in the 2nd phase Reduce (Baazizi and al., 2019b); this allows the user to infer the schemas produced in the Map phase at different levels of abstraction.

Another process of schema extraction from an extended JSON dataset has been proposed in (Frezza and al., 2018). Extended JSON records support, in addition to standard types, other data types like the DBRef type allowing to express links between objects, Date, Long, Timestamp, Binary... The extraction process consists in realizing 3 successive steps: i) creation of schemas for each record, ii) grouping of raw schemas in order to obtain a unique class of JSON objects, iii) unification of schemas and iv) construction of the global schema for all records of a class. The processes presented in (Frezza and al., 2018), (Baazizi and al., 2019b) and (Baazizi and al., 2017) provide some answers to our problem. However, the DBs to which they apply to contain a unique class of objects; they therefore do not deal with the links between classes.

In (Aftab et al., 2020), an automatic process for transforming document-oriented NoSQL DB (MongoDB) into relational DB has been presented. It is summarized in three steps: extracting the schema from the source DB, analyzing and converting it into SQL query according to the format of the target DB and finally launching ETL processes. The latter extract the data from the NoSQL DB, process it to create the SQL queries and then load it into the target

DB. The presented process extracts the schemas of the classes with the names and types of the properties however it does not identify the semantic links between the classes.

Our work aims to provide a process for extracting the schema from a document-oriented NoSQL DB; this process is able not only to extract the descriptions of the objects but also to identify the links between these objects. In the OrientDB system, a DB consists of classes containing records describing objects. Each record consists of a set of properties (key, value). A property can represent either an attribute characterizing an object or a link to another record.

In section 5 we present our process and we detail the steps for extracting the schema.

## 5 EXTRACTION OF LOGICAL SCHEMA

In this section, we present our process ToOrientDBSchema which aims to extract the schema from an OrientDB DB. This schema describes the structure of the stored data, including the name of each class, the names of its properties and their types. The development of our process is based on the MDA architecture (Model Driven Architecture) (Bézivin and Gerbé, 2001) defined previously in section 3. The advantage of using this architecture lies in the generalization of our process. Indeed, the formalization of the input and the output by metamodels (sections 5.1 and 5.2) ensures that our process is applicable to any NoSQL DB of document type.

Figure 1 describes the inputs/outputs of our process. Based on the source and target metamodels and by applying transformation rules to an OrientDB DB, the process produces a schema that conforms to the target metamodel describing the structure of the data stored in this DB.

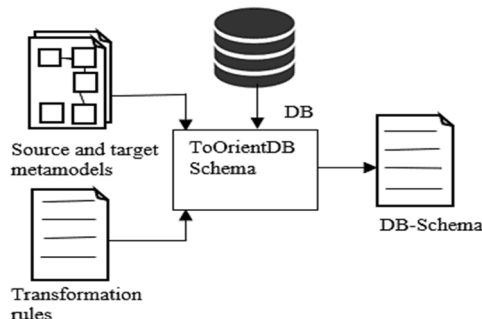


Figure 1: The ToOrientDBSchema process of schema extraction.

### 5.1 Source Metamodel

We have an existing document-oriented NoSQL DB. Many works studied design “drifts” in such DBs, for example the existence of a single unique class grouping together different types of entities or the presence of synonymous property names in records of the same class (Ruiz and al., 2015), (Klettke and al., 2015) and (Baazizi and al., 2019a). These issues have been addressed elsewhere; we made the following starting assumptions to focus on other issues.

H1: a class of the DB represents entities semantically homogeneous having a unique identifier; for example, employees and cars are stored in distinct classes and have their own identifiers. However, entities can also be embedded in a class in the form of a property; it is a choice of conceptual representation of reality. For example, cars can be seen as properties characterizing objects used; in this case, only "Employees" appears as a class.

H2: Records within the same class can contain a variable number of properties. The extraction of a unique schema for a class involves grouping together properties with the same name and type. For example, the two properties phone and telephone of type String will not be grouped together; they will generate two distinct properties.

H3: In the records of a class, properties of the same name and of different types have been previously processed by a specific process applied to the source DBs. This processing detects inconsistencies in the format of values and then harmonizes properties of the same name, either automatically or with the intervention of an administrator. Thus, in two records of the same class, we cannot find a client# property of type Number and a client# property of type String.

These three assumptions allow to obtain schemas excluding certain modelling anomalies of reality; indeed, these could compromise the validity of the extracted schemas. However, it should be noted that these three reducing assumptions do not alter the interest of our proposals since, on the one hand, processing solutions exist elsewhere and, on the other hand, our process and our models are not impacted. However, the non-respect of these assumptions would lead to the production of a schema that does not conform to reality and could cast doubt on the validity of our process.

To describe the source metamodel presented in Figure 2, we used the Ecore language (Eclipse, 2021 Oct); it is a formalism close to UML with which we implemented our solution (section 6: Experimentation).

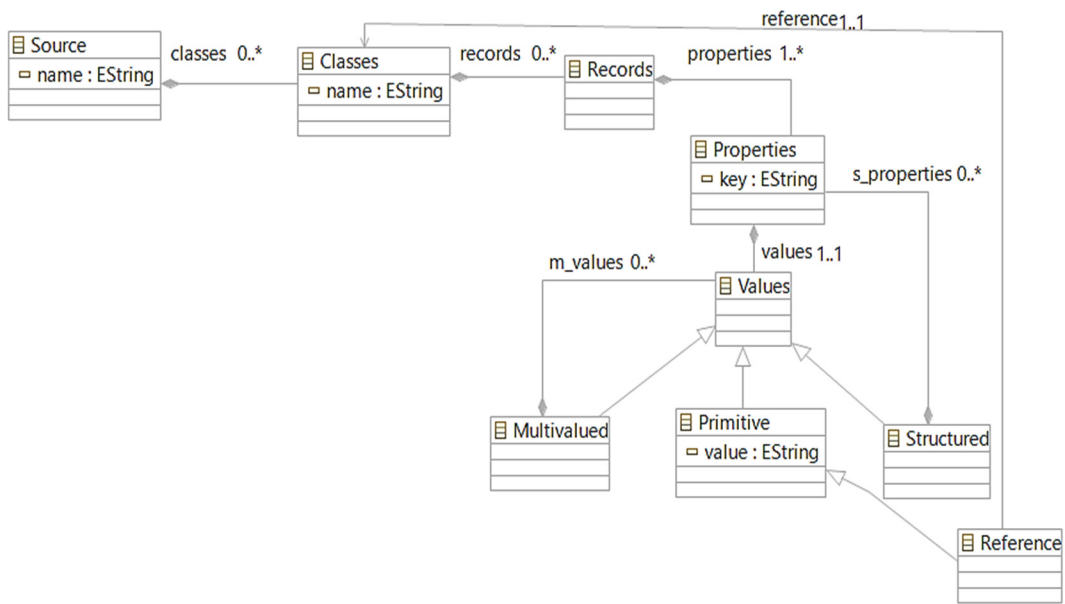


Figure 2: Metamodel describing the source of our process.

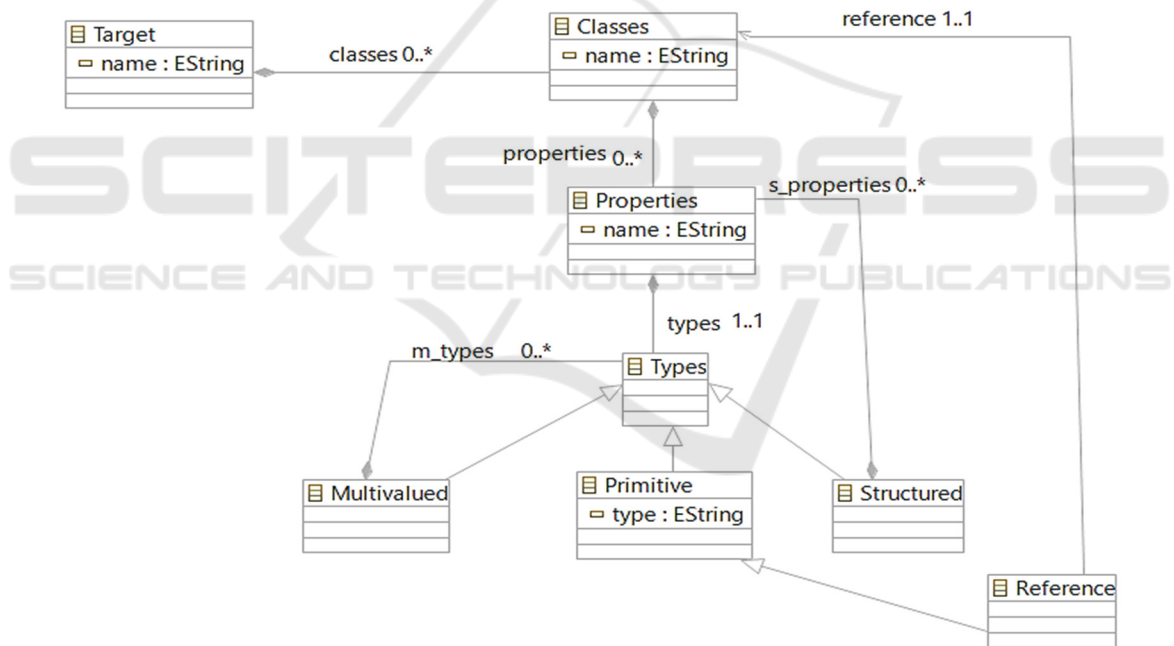


Figure 3: Metamodel describing the resulting schema of our process.

<pre>--R3.1 rule toPrimitiveString {   from S : DBOrientDB!Primitive(     S.value.matches("[0-9a-zA-Z .,:@?!]+"))   to T : SchemaOrientDB!Primitive(     type &lt;- 'EString')}</pre>	<pre>--R3.2 rule toPrimitiveInteger {   from S : DBOrientDB!Primitive(     S.value.matches('[0-9]+'))   to T : SchemaOrientDB!Primitive(     type &lt;- 'EInt')} ...</pre>
---	--

Figure 4.a: Extract from the ATL translation of R3.



According to Ecore, a rectangle represents an object class and an arc corresponds to an association, composition or inheritance relationship. Thus, in Figure 2, an OrientDB DB (Source) is identified by a name and contains a set of classes. Each class is referred by a name and groups together a set of records. Each record contains a set of properties (key, value). Note that two records belonging to the same class can contain different properties. The value of a property can be primitive or complex (structured or multivalued). A primitive value of reference ensures a link between two classes; this value contains the rid ("record identifier") of a record of the referenced class.

This metamodel allows to describe any DB that conforms to a document-oriented NoSQL model, i.e., supported by a system such as MongoDB, OrientDB or CouchDB. The ToOrientDBSchema process will analyze the DB by applying to it the metamodel of figure 2 and will extract a schema that conforms to the target metamodel presented in the next section.

### 5.2 Target Metamodel

The target metamodel formalized with the Ecore language is illustrated in Figure 3. It represents the structural characteristics of an OrientDB DB: the existing classes and their properties, the data types as well as the links (monovalued and multivalued) between the classes.

The root element "Target" corresponds to the schema of an OrientDB DB grouping together a set of class descriptions. Each class contains properties; each of them is associated with a couple (Name, Type). The type of a property can be primitive, structured (i.e. made up of other properties), or multivalued (made up of several values). In the OrientDB system, the links between classes are expressed by object identifiers (references); the value of a link-property points to a record in a class.

### 5.3 Schema Extraction by Transforming the DB

After formalizing the source and target metamodels, we describe how the extraction of logical schema is performed through the use of transformation rules. The rules are first expressed in natural language and then in ATL language (Eclipse, 2021 July). They are considered as a function which, applied to a DB, produces the DB schema: ATL-Rules (DB) = DB-Schema.

**R1:** An OrientDB DB from the source is transformed into a schema with the same name in the target.

**R2:** Each input class is transformed into a class type with the same name.

**R3:** For all the records of a class, each property of the form (Name, Value) is transformed into a couple (Name, Type). For an atomic value (other than a link), we associate the "Primitive" constructor with the type of the value. For example, if the value of a property is a string surrounded by the characters "", then the generated type is "Primitive EString". Our process applies R3 on all the records of the same class and generates a unique model for this class.

In figure 4.a, we expressed the rule R3 using the ATL language and we present an example in figure 4.b. Note that R3 is made up of a number of ATL sub-rules equal to the number of primitive types.

**R4:** A value corresponding to a link of the form "#xx:xx" (prefixed by the xmi tag "Reference") is transformed into type "ERef". Figure 5.b shows an example of the application of R4 to a record containing a link to the Doctors class.

**R5:** A structured value is transformed into a structured type; thus the "Structured" constructor is associated with all the sub-properties and their respective types. The previous rules are applied recursively on each sub-property. Figure 6.a and 6.b shows the ATL expression of R5 and an example of application.

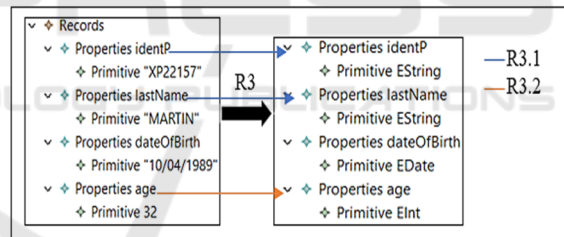


Figure 4.b: Example of application of R3.

```
--R4
rule toPrimitiveReference{
  from S : DBOrientDB!Reference(
    S.value.matches("#[0-9][0-9]:[0-9][0-9]"))
  to T : SchemaOrientDB!Reference(
    type <- 'ERef',
    reference<-S.reference)}
```

Figure 5.a: The rule R4 formalized in ATL language.

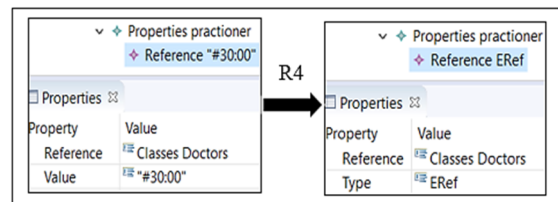


Figure 5.b: Example of application of R4.

```
--R5
rule toStructured{
  from S : DBOrientDB!Structured
  to T : SchemaOrientDB!Structured(
    s_properties<-S.s_properties)}
```

Figure 6.a: The rule R5 formalized in ATL language.

**R6:** The value of a multivalued property is transformed by associating the "Multivalued" constructor with the type of the component values. The determination of the type results from the application of one of the previous rules. We present an example of application of this rule in figure 7.

We presented in this section the 6 rules of our transformation process expressed in ATL language. In the next section, we will show how this process has been experimented.

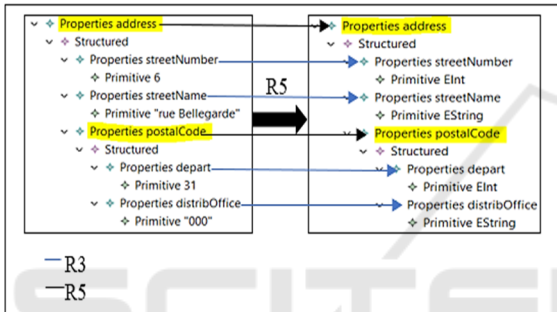


Figure 6.b: Example of application of R5.

```
--R6
rule toMultivalued{
  from S : DBOrientDB!Multivalued
  to T : SchemaOrientDB!Multivalued(
    m_types<-S.m_values)}
```

Figure 7.a: The rule R6 formalized in ATL language.

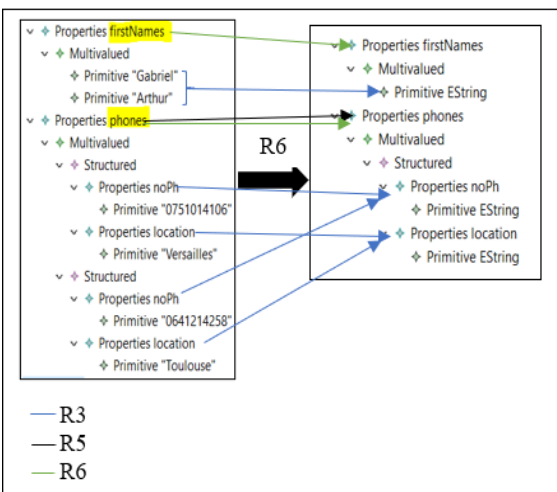


Figure 7.b: Example of application of R6.

## 6 EXPERIMENTATION

We present the experimentation of our process on three test-DBs based on the medical application described in section 2.1 and on two massive industrial BDs. First, we implemented our process on a dataset with twelve classes taken from the DB of scientific programs and representing a little less than a gigabyte. The purpose of this test is to verify the proper functioning of our process without taking into account the performance of extracting a schema from big data. Then, we applied our process on three test-DBs created from their previously known schemas. The goal is to ensure that for each test-DB, the schema generated by our process correctly describes the data stored in the DB. Finally, to confirm the Big Data aspect of our work, we experimented our process on two massive industrial DBs.

### 6.1 Technical Environment

For our test, we used a metamodeling and model transformation platform conforming to the MDA architecture. The Eclipse Modeling Framework platform (EMF) (Eclipse, 2021 June) includes a set of tools among which we implemented:

- Ecore: a metamodeling language (Eclipse, 2021 Oct) that allowed to define the source and target metamodels (figure 2 and 3).

- XMI: a format used to present instances of metamodels (OMG, 2021 July).

- ATL: a model transformation language that provides a high level of abstraction and expressiveness by describing the transformation of elements from the source model to the target model (Allilaire and al., 2006). This language, inspired by the QVT formalism, has a level of execution performance suitable for large volumes of data (Van Amstel and al., 2011).

### 6.2 Test Case

To test our process on the medical DB and generate its schema, we implemented the following steps.

**Step1:** Conversion of the OrientDB DB into the XMI format required by the Eclipse platform. We developed a software in Java language to convert the Json code of a DB by introducing the specific XMI tags of the OrientDB syntax such as <classes>, <records> and <properties>. Figure 8 shows an example of the input/output of our software. This software scans the entire source DB and creates a new DB in XMI format that will serve as the source for our ToOrientDBSchema process. The execution time

of the conversion is important and proportional to the volume of the DB. At the present stage of our work, the conversion software must be executed each time a new schema extraction is requested (due to data update). But we are currently studying the possibility to reflect each update of the source DB in the XMI DB; in this case, only one conversion of the DB into XMI will be necessary.

**Step 2:** Application of the transformation rules defined in section 5.3 on the instance of the source metamodel created in step 1 (BD/XMI). Figure 9 shows an extract of the transformation rules expressed with ATL. This operation automatically generates the DB schema according to the concepts of the target metamodel. In figure 10, we present an extract of the generated schema. It describes the structure of each class ("Patients" for example): the properties it contains and their types. The schema shows monovalued and multivalued links. Figure 10 contains the two properties "practioner" and "antecedents" which are of type "ERef" and represent respectively a monovalued link to the class "Doctors" and a multivalued link to the class "Pathologies".

Now, it is necessary to verify that the schema generated by our process corresponds to the description of the data contained in the DB. To do this, we manually created three separate test-DBs on which we applied our extraction process. Specifically, the approach of verification used for each DB was as follows.

1. Manual elaboration of an (initial) DB schema inspired by the application of medical programs (cf. section 2). Each schema had between 8 and 10 classes and between 4 and 7 inter-class links.

2. Creation of a DB under the OrientDB system following the previous schema. This was done using suitable software for entering attribute values and storing them under OrientDB.

3. Implementation of our automatic schema extraction process on the DB.

4. Visual comparison of the extracted schema and the initial schema.

The experimentation carried out on the three test-DBs made it possible to verify the correct functioning of the ToOrientDBSchema process. Thus, Each DB has been developed to take into account (and test) the full diversity of types of data and links that a DB may contain. Figures 11 and 12 show an extract from a test-DB and its generated schema by our process.

We completed this experiment by applying the ToOrientDBSchema process on two massive industrial DBs (with the help of the company Trimane), one used in a legal application and the other containing financial data. Each of these OrientDB DBs has a volume between one and two terabytes. The extracted schemas have been visually validated by developers with in-depth knowledge of these DBs.

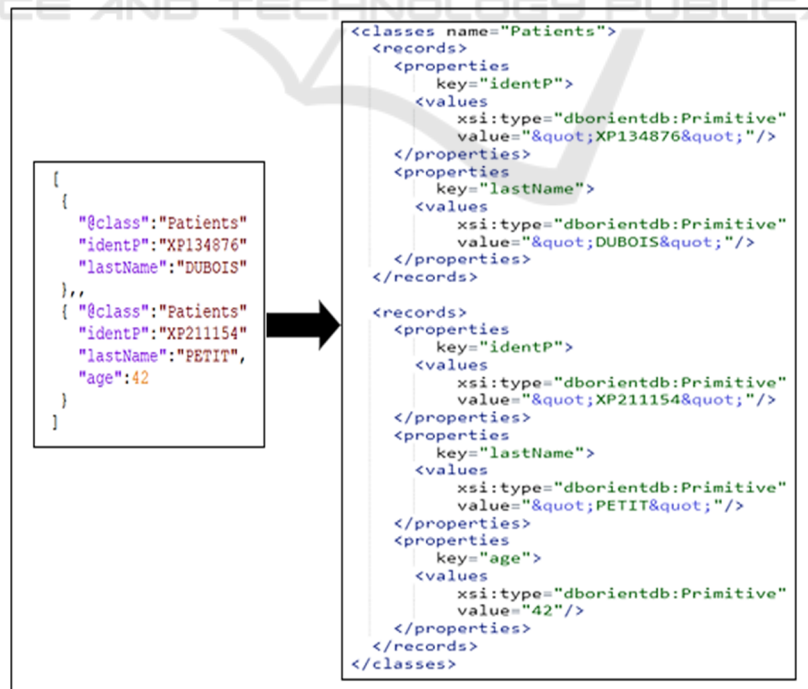


Figure 8: Conversion of OrientDB data to XMI.



```

--@path Source=/medicalApp/Metamodels2/DBOrientDB.ecore --R3.1
--@path Target=/medicalApp/Metamodels2/SchemaOrientDB.ecore rule toPrimitiveString {
module ToOrientDBSchema; --R3.6
create OUT : SchemaOrientDB from IN : DBOrientDB; rule toPrimitiveNull {
--R1 rule toTarget{ from S : DBOrientDB!Primitive(
from S : DBOrientDB!Source S.value.matches("[0-9a-zA-Z .,:@?!]+"))
to T : SchemaOrientDB!Target( from S : DBOrientDB!Primitive(
name<-S.name, S.value.matches('null'))
classes<-S.classes)) to T : SchemaOrientDB!Primitive(
type <- 'EString'))
--R2 rule toClasses { --R3.2 rule toPrimitiveInteger {
from S : DBOrientDB!Classes from S : DBOrientDB!Primitive(
S : DBOrientDB!Classes S.value.matches('[0-9]+'))
to T : SchemaOrientDB!Classes( --R4 rule toPrimitiveReference{
name <- S.name, from S : DBOrientDB!Reference(
properties <- S.getProperties()) S.value.matches("#[0-9][0-9]:[0-9][0-9]"))
to T : SchemaOrientDB!Classes( --R5 rule toStructured{
name <- S.name, from S : DBOrientDB!Structured(
properties <- S.getProperties()) to T : SchemaOrientDB!Structured(
s_properties<-S.s_properties))
--R3 rule toProperties{ --R6 rule toMultivalued{
from S : DBOrientDB!Properties from S : DBOrientDB!Multivalued(
to T : SchemaOrientDB!Properties( S.value.matches('true|false'))
name<-S.key, to T : SchemaOrientDB!Multivalued(
types<-S.values)) m_types<-S.m_values))
type <- 'EBoolean'))

```

Figure 9: Extract from the transformation rules expressed with ATL.

Table 1: Comparison with the processes presented in Related works section.

Process \ Criteria	(Baazizi and al., 2017)	(Baazizi and al., 2019b)	(Frozza and al., 2018)	(Aftab and al., 2020)	Our process ToOrientDBSchema
Dataset format	JSON datasets	JSON datasets	Extended JSON (MongoDB)	JSON (MongoDB)	Extended JSON (OrientDB)
Schema format	JSON	JSON	JSON	JSON	XMI
Number of classes	1	1	1	n	n
Links	No	No	No	No	Yes

## 7 DISCUSSION

In this section, we compare the ToOrientDBSchema process that we proposed to the processes of schema extraction cited in the related works (section 4). This comparison is summarized in Table 1. We focus on the following criteria: i) the format of the data stored in the dataset, ii) the format of the schema generated by the proposed process, iii) the number of classes in the dataset and iv) the existence of association links in the dataset. Table 1 shows that the major contribution of our process is the processing of association links (monovalued and multivalued) in the form of references as they occur in standard object systems (ODMS, 2021 June).

## 8 CONCLUSION

This article proposes the ToOrientDBSchema process for extracting the schema from an OrientDB conforming to the document-oriented model. This process based on the MDA architecture comprises 3 steps: the modeling of the source and the target through metamodels and the formalization of the transformation rules. The source metamodel describes the content of any DB to which our process is applied and the target metamodel models the result of the process, i.e. the schema of the DB. This one describes the classes, properties and their types as well as the semantic links contained in the DB. The transformation rules allow to make a transition from a DB to its schema in accordance with MDA principles. This process has been tested on three test-DBs based on the medical application as well as on

two massive industrial BDs. It should be noted that the proposed solution applies to a massive DB; the execution time of the process, even optimized, can last several minutes. However, as it is the case in our application, the stored data can evolve quickly and make the extracted schema obsolete. We have therefore developed another process to update the schema as the DB evolves; this process is not tackled in this article.

We are currently working to complete the ToOrientDBSchema process. Indeed, the OrientDB system makes it possible to express inheritance links between classes. Consequently, the source and target metamodels could be extended to take into account this type of link (Chillón and al., 2021). On the other hand, our process generates a unique schema for each class, however it does not check whether there are properties of different names with the same semantics (assumption 2 in section 5.1). For example, the "address" and "adr" properties. All of these characteristics could be integrated into our process to generate a more complete schema.

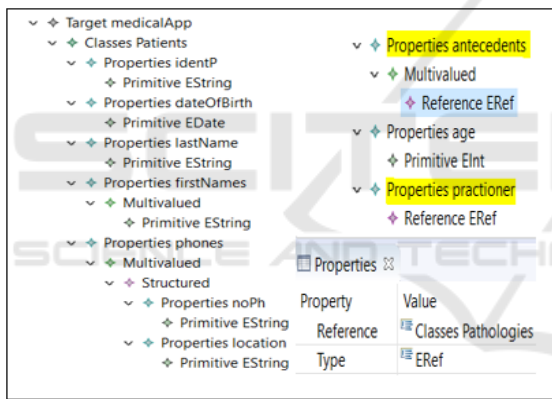


Figure 10: Extract from the medical DB schema generated by our process.

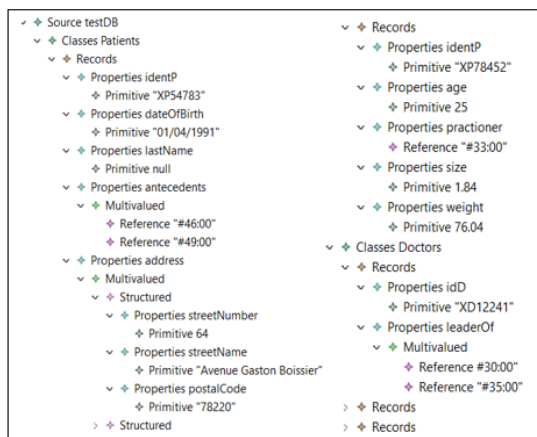


Figure 11: Extract from a test-DB.

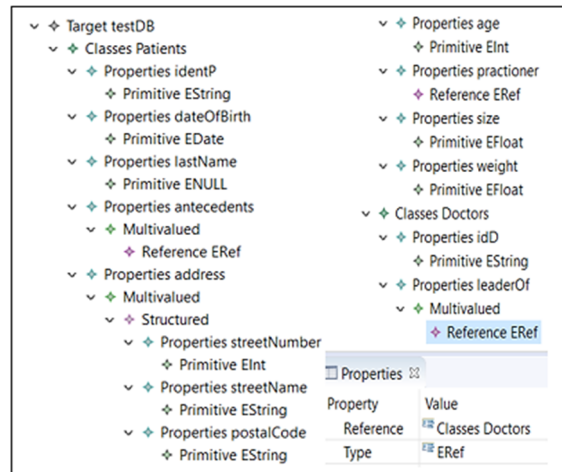


Figure 12: Extract from the logical schema obtained.

## REFERENCES

Elmasri, R., & Navathe, S. (2011). Fundamentals of database systems (6th ed). Addison-Wesley.

Frozza, A. A., dos Santos Mello, R., & da Costa, F. de S. (2018). An approach for schema extraction of JSON and extended JSON document collections. 2018 IEEE International Conference on Information Reuse and Integration (IRI), 356-363.

Baazizi, M.-A., Colazzo, D., Ghelli, G., & Sartiani, C. (2019b). 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, mars 21). Schema Inference for Massive JSON Datasets. Extending Database Technology (EDBT).

Frozza, A. A., Jacinto, S. R., & Mello, R. dos S. (2020). An Approach for Schema Extraction of NoSQL Graph Databases. 2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI), 271-278.

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.

Wang, L., Zhang, S., Shi, J., Jiao, L., Hassanzadeh, O., Zou, J., & Wangz, C. (2015). Schema management for document stores. Proceedings of the VLDB Endowment, 8(9), 922-933.

Laney, D. (2001). 3D data management: Controlling data volume, velocity and variety. META group research note, 6(70), 1.

Ruiz, D. S., Morales, S. F., & Molina, J. G. (2015). Inferring versioned schemas from NoSQL databases and its applications. International Conference on Conceptual Modeling, 467-480.

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 (BTW 2015)*.
- Baazizi, M.-A., Colazzo, D., Ghelli, G., & Sartiani, C. (2019a). A type system for interactive JSON schema inference. *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*.
- Cánovas Izquierdo, J. L., & Cabot, J. (2016). JSONDiscoverer: Visualizing the schema lurking behind JSON documents. *Knowledge-Based Systems*, 103, 52-55.
- Frezza, A. A., Defreyne, E. D., & dos Santos Mello, R. (2020). A Process for Inference of Columnar NoSQL Database Schemas. *Anais do XXXV Simpósio Brasileiro de Bancos de Dados*, 175-180.
- Van Amstel, M., Bosems, S., Kurtev, I., & Pires, L. F. (2011). Performance in model transformations: Experiments with ATL and QVT. *International Conference on Theory and Practice of Model Transformations*, 198-212.
- Chillón, A. H., Hoyos, J. R., García-Molina, J., & Ruiz, D. S. (2021). Discovering entity inheritance relationships in document stores. *Knowledge-Based Systems*, 230, 107394.
- Allilaire, F., Bézivin, J., Jouault, F., & Kurtev, I. (2006). ATL-eclipse support for model transformation. *Proceedings of the Eclipse Technology eXchange workshop (eTX) at the ECOOP 2006 Conference, Nantes, France*, 66.
- Aftab, Z., Iqbal, W., Almustafa, K. M., Bukhari, F., & Abdullah, M. (2020). Automatic NoSQL to Relational Database Transformation with Dynamic Schema Mapping. *Scientific Programming*, 2020, 8813350.
- ODMS. Object Databases. <http://www.odbms.org/free-downloads-and-links/object-databases/>, consulted in (2021, June)
- Apache Spark. Spark SQL Guide. <https://spark.apache.org/docs/latest/sql-programming-guide.html>, consulted in (2021, Oct).
- SnowPlow Analytics. Schema-Guru. <https://github.com/snowplow/schema-guru/releases/tag/0.5.0>, consulted in (2021, Oct)
- Peter Schmidt. *mongodb-schema*. <https://github.com/mongodb-js/mongodb-schema>, consulted in (2021, Oct).
- Eclipse. EMF. <https://www.eclipse.org/modeling/emf/>, consulted in (2021, June).
- Eclipse. Ecore Tools. <https://www.eclipse.org/ecoretools/doc/index.html>, consulted in (2021, Oct).
- OMG. XMI. <https://www.omg.org/spec/XMI/>, consulted in (2021, July).
- Eclipse. ATL. (2021, July). <https://www.eclipse.org/atl/>, consulted in (July, 2021).
- OrientDB. <https://orientdb.org/>, consulted in (2021, April).
- OMG. MDA. <https://www.omg.org/mda>, consulted in (2021, April).
- Eclipse. <https://www.eclipse.org>, consulted in (2021, April).
- ODMS. ODMG standard. <http://www.odbms.org/odmg-standard/>, consulted in (2021, April).
- OMG. <https://www.omg.org/>, consulted in (2021 June).