Big Data and Knowledge Management: How to Implement Conceptual Models in NoSQL Systems?

Fatma Abdelhedi[2], Amal Ait Brahim[1], Faten Atigui[3] and Gilles Zurfluh[1]
[1]*Toulouse Institute of Computer Science Research (IRIT), Toulouse Capitole University, Toulouse, France*
[2]*CBI[2] – TRIMANE, Paris, France*
[3]*CEDRIC-CNAM, Paris, France*

Keywords: Big Data, NoSQL, Knowledge, MDA, QVT Transformation.

Abstract: In 2014, Big Data has passed the top of the Gartner Hype Cycle, proving that Big Data technologies and application start to be mature, becoming more realistic about how Big Data can be useful for organizations. NoSQL data stores are becoming widely used to handle Big Data; these databases operate on schema-less data model enabling users to incorporate new data into their applications without using a predefined schema. But, there is still a need for a conceptual model to define how data will be structured in the database. In this paper, we show how to store Big Data within NoSQL systems. For this, we use the Model Driven Architecture (MDA) that provides a framework for models automatic transformation. Starting from a conceptual model that describes a set of complex objects, we propose transformation rules formalized with QVT to generate a column-oriented NoSQL model. To ensure efficient automatic transformation, we use a logical model that limits the impacts related to technical aspects of column-oriented platforms. We provide experiments of our approach using a case study example taken from the health care domain. The results of our experiments show that the proposed logical model can be effectively implemented in different column-oriented systems independently of their specific technical details.

1 INTRODUCTION

The number of digital devices that we use nowadays produces a huge amount of data that need to be exploited. The volume of data exceeds many terabytes and we have different type of data including factors such as format, structure, and sources. Furthermore, there is a need for loading and processing of such a huge amount of heterogeneous data in real-time or near real-time; these data need to be used quickly. Volume, Variety and Velocity, often referred to as the three V's, capture the real meaning of Big Data (Chen, 2014). In 2012, Gartner retrieved and gave a more detailed definition as: "Big Data are high-volume, high-velocity, and high-variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization".

Big Data bring many attractive opportunities to knowledge management (Fredriksson, 2015). Simultaneously, to extract knowledge from Big Data, we have to face a lot of challenges mainly related to Big Data storage and process; our focus in this paper is only on Big Data storage. Using relational databases proves to be inadequate for all applications, particularly ones involving large volumes of data (Abello, 2015). As a result, a new kind of databases has appeared, known as "NoSQL" data stores, that are able to handle Big Data with high performance (Angadi, 2013). The key feature of NoSQL databases is that they are schema-less, meaning that data can be inserted in the database without upfront schema definition. This feature enables applications to quickly and easily modify data without rewriting tables when new data are encountered. Nevertheless, there is still a need for a semantic data model to define how data will be structured and related in the database (Daniel, 2016); it is generally accepted that UML meets this requirement (Abello, 2015). Therefore, the purpose of this paper is to present how to store Big Data in NoSQL databases . For this, we propose a MDA-based approach that transforms an UML conceptual model describing Big Data into a NoSQL model. The rest of the paper is structured as follows. Section 2, we motivate our work using a case study taken from the healthcare field. Section 3 reviews previous work on models transformation. Section 4

Abdelhedi, F., Brahim, A., Atigui, F. and Zurfluh, G.
Big Data and Knowledge Management: How to Implement Conceptual Models in NoSQL Systems?.
DOI: 10.5220/0006082302350240
In *Proceedings of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2016) - Volume 3: KMIS*, pages 235-240
ISBN: 978-989-758-203-5

shows our MDA-based approach that aims to map an UML conceptual model into NoSQL model. Section 5 details our experiments. Finally, section 6 ends up with the conclusion and future work.

2 RESEARCH PROBLEM AND MOTIVATION

In this paper, our focus is on the implementation of Big Data described by a conceptual model (UML class diagram) in a NoSQL system; this involves transforming the conceptual model into a data model compatible with NoSQL systems.

To motivate and illustrate our work, we present here a case study in the healthcare field. The case study concerns national or international scientific programs for monitoring patients having serious diseases. The main goal of this program is (1) to collect data about the disease development over time, (2) to study interactions between different diseases (3) to evaluate the short and medium-term effects of their treatments. The medical program can last up to 3 years. Data collected from the establishments involved in such a program have the characteristics 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 ; they can be (1) structured like patient's vital signs (respiratory rate, blood pressure, temperature, etc.), patient name, diagnosis codes, etc. (2) unstructured such as patient histories, consultation summaries, paper prescriptions and radiology reports, and (3) semi-structured document such as the package leaflets of medicinal products that provide a set of comprehensible information enabling the use of the medicinal product safely and appropriately. **Velocity:** Some data are produced in continuous flow by sensors; it must be processed in near real time because it can be integrated into time-sensitive processes (for example, some measurements, like temperature, require an emergency medical treatment if they cross a given threshold).

3 OBJECTIVE AND RELATED WORK

3.1 Objective

Our purpose is to implement a conceptual model describing Big Data into NoSQL database. There are four basic types of NoSQL databases: key-value, document-oriented, column-oriented and graph-oriented. In this paper, we choose to focus on column-oriented NoSQL model. This model is considered to be the most efficient in terms of performance, for multi-criteria access queries (vertical data organization with columns-families) (Abadi, 2012).

An overview of our approach is illustrated in figure 1. Starting from a conceptual model, we propose a transformation process to automatically generate a NoSQL logical model. We introduce this logical-level model between conceptual level (business description) and physical level (technical description) to limit the impacts related to technical details of NoSQL platforms. Indeed, there are many column-oriented NoSQL databases available, such as HBase, Cassandra, Accumulo, etc. Each one has its own technical aspects. For example, for the medical program, several column-oriented NoSQL databases can be used in the same inter-establishment software architecture; each establishment may have a specific column-oriented database. To overcome this situation, we propose a logical model independent of a particular NoSQL platform and can be easily implemented in several column-oriented databases independently of their specific technical details.

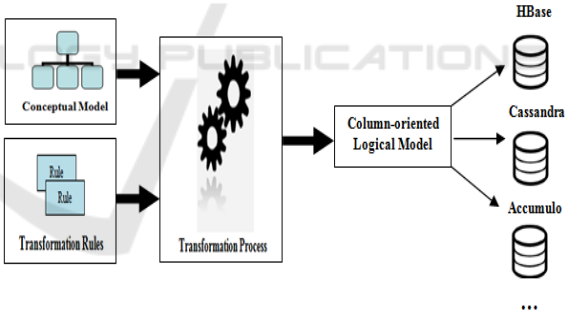

Figure 1: Overview of our approach.

3.2 Related Work

To the best of our knowledge, there are only few solutions that have dealt with NoSQL databases conceptual modeling. Chevalier et al. (Chevalier, 2015) defined a set of rules to map a multidimensional model into two NoSQL models: column-oriented and document-oriented. The links between facts and dimensions have been converted using imbrications. Although the transformation process proposed by authors start from a conceptual level (multidimensional model), this specific model is different from the UML standard; it contains facts,

dimensions and one type of links only. Other studies investigate the process of transforming relational databases into a NoSQL model. Li (Li, 2010) proposed an approach for transforming a relational database into HBase; the relationships between tables (foreign keys) are converted by adding new columns-families that contain references. Vajk et al. (Vajk, 2013) propose a mapping from a relational model to document-oriented model using MongoDB. However, the relational model does not present the semantic richness of UML class diagram (especially through the several types of relationships that exist between classes: association, aggregation, composition, generalization, etc.).

Only few works have presented approaches to implement UML conceptual models in NoSQL databases. Li et al. (Li, 2014) propose a MDA-based approach to transform UML class diagram into HBase. After building the meta-models of UML class diagram and HBase, the authors have proposed mapping rules to realize the transformation from the conceptual level to the physical level. These rules are applicable to HBase, only. Gwendal et al. (Daniel, 2016) describe the mapping between UML conceptual models and graph databases via an intermediate graph meta-model. These rules are specific to graph databases used as a framework for storing, managing and querying complex data with many connections. Generally, this kind of NoSQL databases are used in social networks where data are highly connected.

4 MDA-BASED TRANSFORMATION PROCESS

4.1 MDA Formalism

To address the complexity of applications, model-driven engineering (MDE) approach considers models as the central artifacts in the software engineering process. MDA (Model Driven Architecture) proposed by Object Management Group (OMG) is a mechanism derived from MDE. This architecture defines a hierarchy of models from three points of view: Computation Independent Model (CIM), Platform Independent Model (PIM), and Platform Specific Model (PSM) (Bézivin, 2001). Among this proposed models, we use: (1) **PIM:** to describe data without showing aspects which are specific to the implementation platforms. In this paper, we consider two PIM: conceptual PIM (UML class diagram) that describes data taking into account only its own business aspects, and logical PIM that describes how to organize data (in our case, we use the column-oriented data organization). (2) **PSM:** to represent data taking into account the characteristics of a particular technical platform. At this level, we consider two physical models that correspond to Cassandra and HBase platforms to show that our approach does not restrict us to one implementation platform (figure 2). The mapping between two MDA models provides a succession of transformation rules translating a source model into a target model. OMG has defined a standard called QVT for expressing models transformation.

Note that due to the lack of space, we do not present the process that transforms the logical PIM into PSMs. Only the transformation of conceptual PIM to logical PIM will be presented.

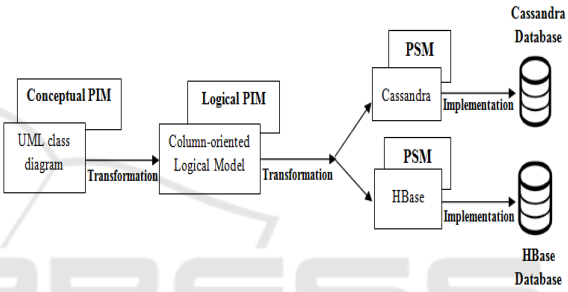

Figure 2: Modeling levels.

4.2 Source: UML Class Diagram

An UML class diagram contains a set of classes $\{C_1, \ldots, C_p\}$. Each class is composed from structural and behavioral constituents. In this paper, we consider only the structural part. Since the operations are linked to the behavior, we will not take them into account. The schema of each class C is a tuple (**N, A, Ident**) where: **C.N** is the class name; **C.A** = $\{a_1,\ldots,a_q\}$ is a set of q attributes. The schema of each attribute is a pair (N:C) where "a.N" is the attribute name and "a.C" the attribute type; C can be a predefined class, i.e. a standard data type (String, Integer, Date ...) or a business class (class defined by user); **C.Ident** is an object identifier whose type is called "Oid"; this identifier is automatically managed by the system for each class. In an UML class diagram, there are essentially four types of relationships between classes: Association, Aggregation, Composition and Generalization. In order to represent these concepts, we propose the following meta-model (figure 3) that is adapted from the one proposed by OMG (OMG, 2011).

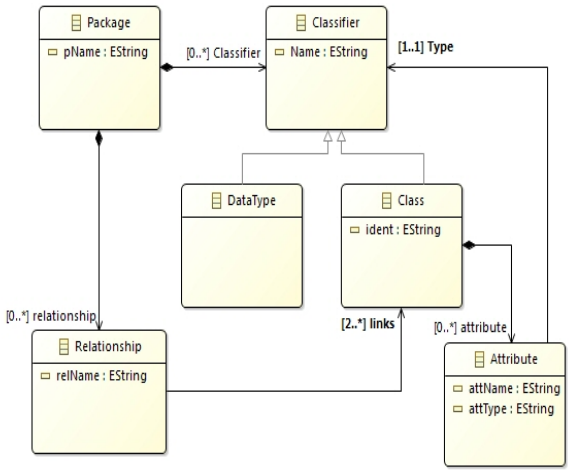

Figure 3: Source meta-model.

4.3 Target: Column-Oriented Logical Model

A column-oriented database consists of a set of tables. Each table is a container of a collection of rows with variable length; each row is identified by a unique identifier called "Row-Key". By default, we store the database in a single table that we call T. T is comprised of a set of column-families {f1,…fp}. The schema of a column-family f is a tuple (**N, COL, Id**) where: **f.N** is the column-family name; **f.COL** = $\{col_1,\ldots,col_q\}$ is a set of q columns. The schema of each column is a triplet (N, T, TS) where "col.N" is the column name, "col.T" the column type and "col.TS" the TimeStamp. In this paper, we do not consider the TimeStamp parameter; **F.Id** is a unique column-family identifier whose type is called "Row-Key". We present these concepts through the meta-model of figure 4.

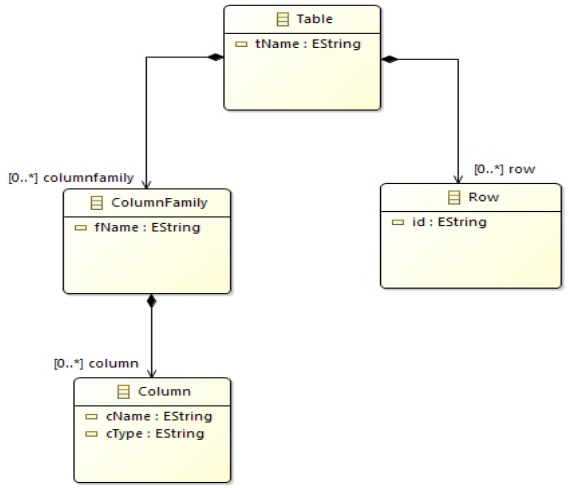

Figure 4: Target meta-model.

4.4 Transformation Rules

For each transformation rule, we try to justify our choice based on the features and limits of the column-oriented storage. Note that at this stage of our work, we do not take into account the optimized solutions obtained through analysis queries.

- **R1: Package to Table**

NoSQL systems are not designed for the purpose of joining data from multiple tables. Even though, it is possible to support the join operation in some NoSQL systems, it's still a complicated process (Cattell, 2011). Therefore, we chose to store the database (the package) in a single target table.

- **R2: Class to Column-Family**

All the attributes of a class are grouped into the same column-family. Indeed, all the data in a single column-family will be stored in the same file on the disk. That will enable the possibility of processing large amount of data faster and more cost effectively (Abadi, 2008). An identifier whose type is "Oid" is transformed into an identifier whose type is "Row-Key". Indeed, each row described by a column-family represents an instance of the corresponding class. Therefore, an identifier of type "Oid" used to identify an instance of a class is transformed into an identifier of type "Row-Key" used to identify the row described by the corresponding column-family.

- **R3: Association class to Column-Family**

Like any other class, each association class is transformed into a column-family where each column is either a class attribute or an attribute whose type is "Oid" used to reference the target column-families (related classes).

- **R4: Association relationship to Column-Family**

This rule transform each n-ary association into a new column-family composed of n columns, where each column has the type "Row-Key": these columns are used to reference the target columns families (linked classes). This rule generalizes the process of transforming association links; it applies to any association regardless of its degree (binary, ternary, quaternary, etc.) and cardinalities.

- **R5: Composition/aggregation relationship**

Composition relationship can be represented as references or nested data. As the columns oriented databases don't support nested data (column containing other columns), we use references to tackle this issue. Therefore, each composition/aggregation relationship is transformed by creating a new column whose type is called "set Row-Key" in the column-family corresponding to the container class; this column is used to reference the

column-family (ies) corresponding to the contained class (es). We note that number of values that will be contained in this new column depends on the maximum cardinality of composition relationship.

- **R6: Generalization relationship**

We propose to transform each generalization relationship by creating a new column which the type is "Row-Key" in the column-family corresponding to the subclass; this column is used to reference the column-family corresponding to the superclass.

These rules are formalized using QVT. Note that due to lack of space, we only present in figure 5 the formalization of the main transformation rule (R1).

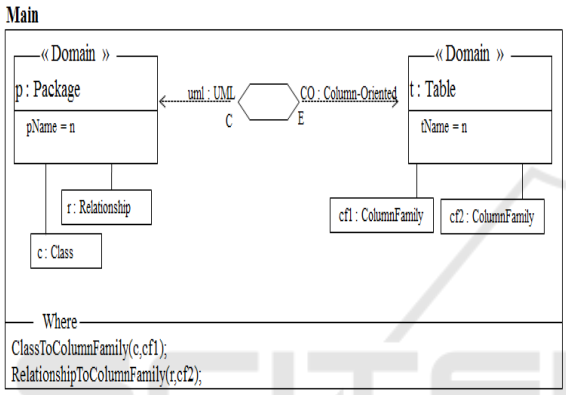

Figure 5: QVT transformation of conceptual PIM into logical PIM.

5 EXPERIMENTS

In this section, we show how to implement an UML conceptual model in two column-oriented platforms. First, we provide the implementation of the QVT transformation process as presented in section 4. Second, we show that the target logical model can be effectively implemented in different column-oriented platforms.

5.1 Experimental Environment

We carry out the experimental assessment using MDE environment that allows us to implement models, meta-models and QVT transformations.

Eclipse Modeling Framework (EMF): is a modeling framework and code generation to support the development of tools and model driven applications; **Ecore:** is a meta-modeling language that we used to create our meta-models; **XML Metadata Interchange (XMI)**: is XML based standard for metadata interchange. We use XMI to create models as instance of meta-models; **Query / View / Transformation (QVT):** is the OMG standard for model-to-model transformation.

5.2 Conceptual PIM to Logical PIM Transformation

Before proceeding to the implementation of the transformation rules, first, we created Ecore meta-models corresponding to the source (figure 3) and the target (figure 4).

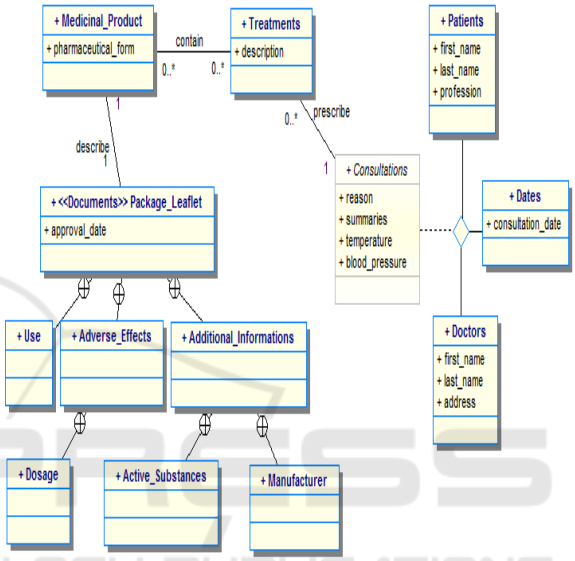

Figure 6: Source model (excerpt).

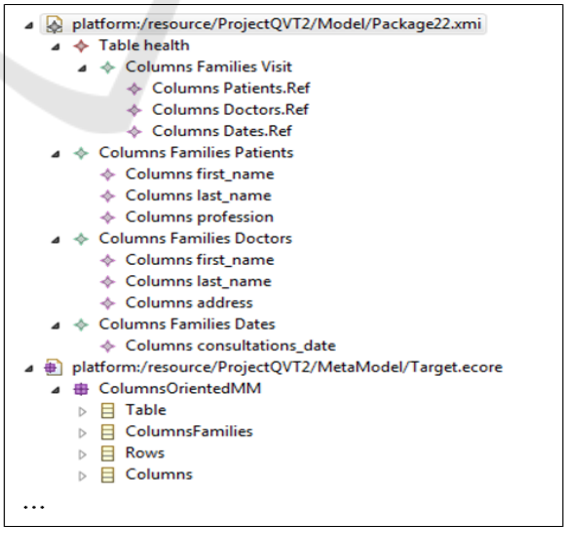

Figure 7: Target model (excerpt).

The next step is to create an XMI instance of the source meta-model (Figure 6). In parallel, we used

Operational QVT plugin provided within EMF to implement the QVT rules. Finally, we tested the transformation by running the QVT script. The execution of this script transforms the UML class diagram (figure 6) into the NoSQL table corresponding to the logical model (figure 7).

5.3 Logical PIM to PSMs Transformation

Choosing column-oriented model at logical level does not imply a specific target platform. Consequently, several implementation platforms could be used. In this paper, we choose to consider two PSMs that correspond to HBase and Cassandra platforms.

- **Cassandra PSM**

Cassandra is a columns-oriented database. It consists of one data container named Keyspace. The Keyspace is associated to a set of columns families; each column-family is identified by a PrimaryKey and contains a set of columns that must be declared up front at schema definition time. We note that the concepts of "Table" and "Row-Key" used at the logical level will be replaced respectively by "Keyspace" and 'PrimaryKey".

- **HBase PSM**

HBase is a column-oriented database built on top of Hadoop (Grover, 2015). HBase database consists of one table named HTable. The HTable is associated with a set of columns families that must be declared up front at schema definition time, whereas columns do not need to be defined at schema time but can be conjured on the fly. Each row in HTable is identified by a RowKey.

Based on Cassandra PSM and HBase PSM, we have created manually Cassandra and HBase databases.

6 CONCLUSION AND PERSPECTIVES

In this paper we have presented a MDA-based approach to implement UML conceptual model describing Big Data in column-oriented NoSQL systems. Our approach consists of a chain of transformations that generate a column-oriented logical model independent of a particular NoSQL platform; this independence makes it easier to implement conceptual models into several column-oriented databases such as Cassandra and HBase regardless of their specific technical details.

As future work, we plan to complete our transformation process and propose a mapping for OCL expressions defined in the conceptual model; queries languages provided by NoSQL databases (such as CQL, HiveQL …) could be used for this. Another ongoing work concerns the validation of the proposed transformation process on scientific medical applications.

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*.

Cattell, R., 2011. Scalable SQL and NoSQL data stores. *Acm Sigmod Record*.

Abelló, A., 2015. Big data design. In *Proceedings of the ACM DOLAP*.

Li, C., 2010. Transforming relational database into HBase: A case study. In *IEEE ICSESS*.

Chen, C. P., Zhang, C. Y., 2014. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information Sciences*.

Bézivin, J., Gerbé, O., 2001. Towards a precise definition of the OMG/MDA framework. In *ASE*.

Chevalier, M., El Malki, M., Kopliku, A., Teste, O., Tournier, R., 2015. Implementing multidimensional data warehouses into NoSQL. In *ICEIS*.

Abadi, D. J., Madden, S. R., Hachem, N., 2008. Column-stores vs. row-stores: How different are they really?. In *Proceedings of the ACM SIGMOD*.

Li, Y., Gu, P., Zhang, C., 2014. Transforming UML class diagrams into HBase based on meta-model. In *ISEEE*.

Grover, M., Malaska, T., Seidman, J., Shapira, G., 2015. Hadoop application architectures. *O'Reilly Media*.

Abadi, D., Boncz, P., Harizopoulos, S., Idreos, S., Madden, S., 2013. The design and implementation of modern column-oriented database systems. Now.

Fredriksson, C., 2015. knowledge management with big data creating new possibilities for organizations. *In NORKOM*.

Daniel, G., Sunyé, G., Cabot, J., 2016. UMLtoGraphDB: Mapping Conceptual Schemas to Graph Databases. In *ER*.

Vajk, T., Fehér, P., Fekete, K., Charaf, H., 2013. Denormalizing data into schema-free databases. In *CogInfoCom, IEEE*

OMG, 2011. MOF 2.0 QVT Specification.