ISSN 2278-3091
Volume 8, No.3, May - June 2019
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse49832019.pdf
https://doi.org/10.30534/ijatcse/2019/49832019

Hadoop Storage Big Data layer: meta-modeling of key concepts and features

Allae Erraissi[1], Abdessamad Belangour[2]
[1,2]Laboratory of Information Technology and Modeling, Hassan II University, Faculty of sciences Ben M'Sik, Casablanca, Morocco, erraissi.allae@gmail.com

ABSTRACT

In the era of information, humanity produces huge quantities of data measured in terms of terabytes or petabytes that is yet growing exponentially with time. This situation led to the emergence of a large number of big data systems and technologies that share similar architectures but with different implementations. The common architecture is composed of Data sources, Ingestion, Visualization, Hadoop Platform management, Hadoop Storage, Hadoop Infrastructure, Security, and Monitoring Layers. In our way for a unified abstract implementation, we proposed in a previous work a meta-model for data sources and ingestion layers. We relied on our previous comparatives studies to define key concepts of storage in Big Data to propose a meta-model for storage layer. Thus, in this paper, we are going to present our meta-model for storage layer. The main goal of this universal meta-modeling is to enable Big Data distribution providers to offer standard and unified solutions for a Big Data system.

Key words: Big Data, Model Driven Engineering, Storage layer, NoSQL databases, HDFS.

1. INTRODUCTION

Today, Big Data, which appears to be a vague term, is in reality, a massive phenomenon that has quickly become an obsession for scientists, entrepreneurs, governments and the media. An indication of the growing concern about this phenomenon is that companies focus their efforts on deploying the most efficient and secure architecture to collect, store and process an abundance of increasingly heterogeneous data, in real time while integrating machine-learning technologies [1].

According to our earlier research studies of the distributions of leading Big Data solution providers [2], we found that each distribution of Hadoop has its own vision for a Big Data system. We also deduce that Programmers do not have the necessary meta-models to create standard applications that can be compatible with each provider because each provider has his own policy for a Big Data system. Indeed, this work comes after our first Meta-modeling of the two layers Data Sources and Ingestion [3]. In this paper, we propose a meta-model for the Storage layer. This meta-model together with previous ones we proposed for the other layers, can be used as an independent cross-platform Domain Specific Language.
Correspondingly, we shall start in this article with the definition of Hadoop and their main components. Then, we shall discuss the Hadoop distributed file system "HDFS" [4] and its architecture as well as the NoSQL databases [5]. Finally, we shall propose a meta-model for the Storage layer.

2. RELATED WORK

This paper is an extension of our earlier works. Previously, we rely on our comparatives studies on the five main suppliers of big data solutions [2] to define the key concepts of storage layer at the level of the global architecture of a big data system. The most obvious finding to emerge from this comparative study is that several distributions like HortonWorks, Cloudera, MapR, Pivotal HD, and IBM's BigInsights in the IT market have the ability to handle Big Data. Hence, rise the need to standardize concepts through the application of techniques related to Model-driven engineering "MDE". Accordingly, this work is a progress report of our first meta-modeling of the two layers: Data Sources and Ingestion [3] [7]. It is also an extended version of our work that has already been published in the Proceeding of a Conference [7].

3. HADOOP

Hadoop is an open source Java framework that was created by Doug Cutting the founder of Apache Lucene. The main aim of this solution is to facilitate the creation and testing of scalable and distributed methods. It brings together a whole set of modules programmed in Java destined to facilitate the distribution and execution of tasks on several thousand nodes. Despite source code written in Java, any programming language can use Hadoop (like Python, C++, etc.). Besides that, Hadoop can provide and manage its own distributed Hadoop Distributed File System (HDFS) [8]. It also offers the implementation of a set of

tools for parallel data manipulation and analysis such as Map/Reduce [6], HBase [9], Hive [10] and Pig [11].

4. META-MODELING OF HADOOP DISTRIBUTED FILE SYSTEM

Hadoop distributed file system (HDFS) is a distributed file system that covers all nodes of a Hadoop cluster. It connects file systems on many local nodes in order to make it a large file system.

The Characteristics of a Hadoop Distributed File System:

- The system manages the location of the data during the distribution of tasks.
- It is fault tolerant, i.e. it automatically manages the failure of these nodes. Indeed, the data is replicated on several different hosts to ensure its reliability.

4.1 HDFS – Master / Slave Architecture

4.1.1. Master: NameNode

Manages the file system's namespace and metadata. The FsImage stores the namespace of the entire file system, with the mapping of blocks to files and properties of the file system [12]. This file is stored in the local file system of the NameNode. It contains the metadata on the disk (not an exact copy of what the RAM contains; but up to a certain point, a copy of control). The NameNode uses a transaction log file called EditLog to keep a record of each change in file system metadata and synchronizes with the RAM metadata after each writes. The NameNode has a knowledge of the DataNodes in which the blocks are stored. Thus, when a client requests Hadoop to recover a file, it is via the NameNode that the information is extracted. This NameNode will tell the client which DataNodes contain the blocks. All that remains for the customer is to recover the desired blocks [13]. In the event of a power failure on the NameNode, you must perform a recovery using FsImage and EditLog.

4.1.2. Slave: DataNode

A cluster has multiple DataNodes. These DataNodes handle the storage attached to the nodes and periodically reports status for the NameNode. A DataNode contains the data blocks. They are under the command of the NameNode and are nicknamed the Workers. On that account, they are solicited by NameNodes during reading and writing operations. In reading, the DataNodes will transmit to the client the blocks corresponding to the file to be transmitted. In writing; the DataNodes will return the location of the newly created blocks [14].
This figure shows the Master/Slave architecture of the Hadoop distributed file system HDFS:

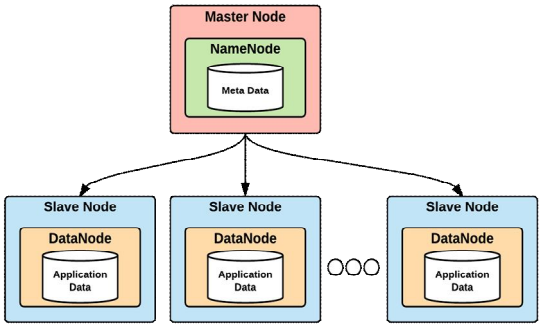

Figure 1: Master/Slave architecture of HDFS [14]

4.2 HDFS – Blocks

The HDFS is developed to support very large files. Data in a Hadoop cluster is divided into smaller pieces, which are distributed throughout the cluster. These smaller pieces are called blocks. HDFS uses much larger block sizes than conventional operating systems. By default, the size is set to 64 MB. However, it is possible to increase to 128 MB, 256 MB, 512 MB or even 1 GB, whereas on conventional operating systems, the size is usually 4 KB. Thus, Interest in providing larger sizes reduces the access time to a block. If the size of the file is smaller than the size of a block, the file will not occupy the total size of this block but just the size necessary for its storage. This figure shows the difference between HDFS blocks and classical operating systems blocks.

Figure 2: Difference between HDFS blocks and OS blocks

4.3 HDFS – Replication

Replica placement is critical to HDFS in order to ensure reliability and performance. HDFS differs from most other distributed file systems by placing replicas. This characteristic requires adjustment and experience. The purpose of this placement policy is to increase data reliability and availability and to reduce network bandwidth usage. HDFS provides a block replication system with a configurable number of replications [13]. During the writing phase, each block corresponding to the file is replicated on several nodes. As for the read phase, if a block is unavailable on a node, copies of this block will be available on other nodes.

Large instances of HDFS work on clusters that are spread across multiple arrays. Communication between two nodes in different racks must go through switches. In most cases, the bandwidth between machines in the same rack is larger than that of machines in a different rack. There is a simple but not an optimal policy, which consists of placing replicas in a unique rack. This, of course, avoids losing data if an array fails and allows the use of bandwidth from multiple racks in reading data. This policy evenly distributes replicas in the cluster, which makes load balancing easy if a component fails. However, this policy increases the cost of writing because writing requires the transfer of blocks to several Rack. The figure below describes the replicas in HDFS.

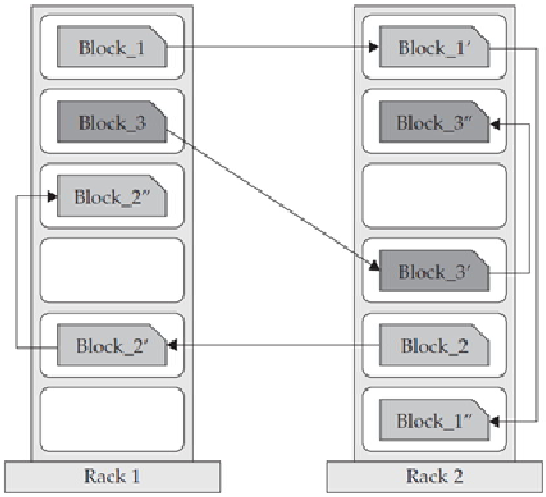

Figure 3: Replicas in HDFS

4.4 Meta-model of Hadoop distributed file system

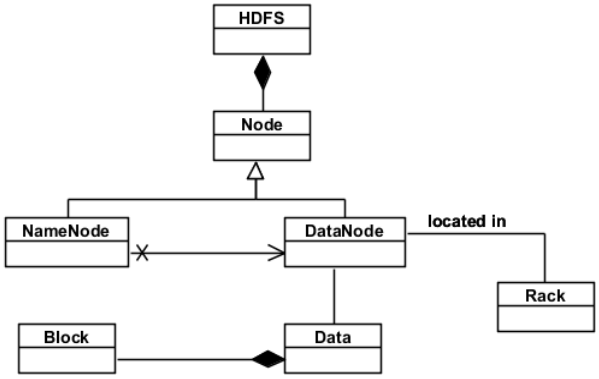

Figure 4: Meta-model of HDFS

The meta-model we proposed for the Hadoop distributed file system has seven meta-classes. These meta-classes define the HDFS Master/Slave architecture, the notion of blocks, which are small units of split data, and finally, the replication represented in this meta-model by Rack to ensure reliability and the performance of Hadoop.

5. META-MODELING OF NOSQL DATABASES

The term NoSQL refers to a type of database management system that goes beyond the relational systems associated with the SQL language. It has the ability to accept more complex data structures. According to their physical models, the DBs managed by these systems fall into four categories: columns, documents, graphs and key-value [15]. Each category offers specific features. For example, in a document-oriented DB such as MongoDB, data is stored in tables whose lines can be nested. This organization of data is coupled to operators that provide access to nesting data [16].

The choice of the most suitable DBMS category for a given application is related to the nature of the processing (queries) applied to the data. However, this choice is not exclusive since, in each category, the DBMS can provide all types of treatments, sometimes at the cost of some heaviness or more extensive programming. In what follows, we shall first present the data models adopted by each category of NoSQL DBMS, and then propose a meta-model for these four types.

5.1 Key/Value

Key/Value [17]: This database model is based on the principle of storing a value associated with a unique key. Nevertheless, unlike other NoSQL databases, the value associated with a key can be varied. It can be either a simple string like a document or a much more complex object that can contain a multitude of information. However, these databases are mainly made for temporary storage and it only allows four operations: Create Read, Update, and Delete (or CRUD) operations. At this stage, it is important to note that the best-known solutions are Riak [18], Redis [19], and Voldemort [20] which were created by LinkedIn. For example, a Redis database offers a very good performance by its simplicity and rapidity. It can even be used to store user sessions or the cache of your site. This figure presents the meta-model proposed for the Key/Value databases:

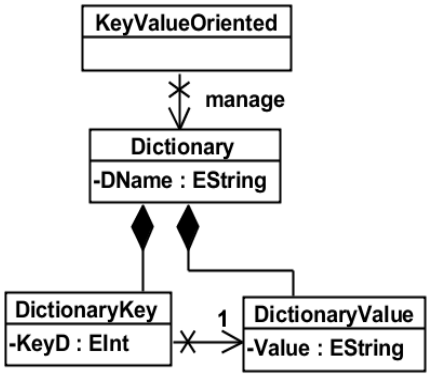

Figure 5: Meta-model for Key/Value database

5.2 Column-oriented

Column-oriented [21]: This model resembles relational databases because the data is saved as a row with columns. It stores data by column and not by row. Column-oriented databases have two main characteristics:

- The columns are dynamic. Within the same table, two individuals may not have the same number of columns because the null values are not stored (which is the case in relational RDBMS).
- Data logging is done in value and not in line as in RDBMS. This, of course, prevents the storage of duplicate information and thus significantly reduces the database and computing time.

As for the solutions, we mainly find HBase [9] (Open Source implementation of the Big Table model [22] published by Google) as well as Cassandra [23] (Apache project that respects the distributed architecture of Amazon Dynamo [24] and Google's Big Table model).
Figure 6 presents the meta-model that we have proposed for the column-oriented database:

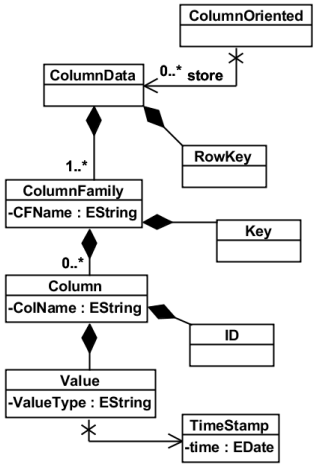

Figure 6: Meta-model for Column-oriented database

5.3 Document-oriented

Document-oriented [25]: This model is based on the paradigm [key, value]. The value, in this case, is a document, which has a tree structure. This tree structure contains a list of fields and each field is associated with a value that can even be a list. These documents are mainly JSON or XML type. The advantage of a document-oriented model is its ability to recover, via a single key, a hierarchically structured set of information. The most popular implementations for this model are RavenDB [26] (intended for .NET / Windows platforms with the possibility of querying via LINQ), CouchDB from Apache [27], and MongoDB [28].

Figure 7 presents the meta-model that we have proposed for the document-oriented database:

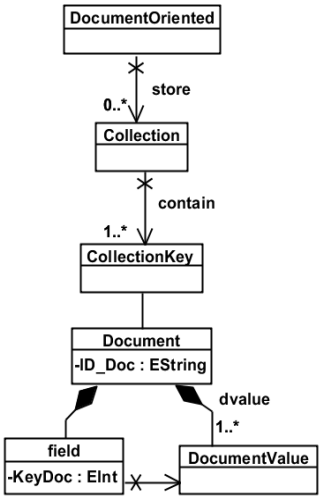

Figure 7: Meta-model for Document-oriented database

5.4 Graph-oriented

Graph-oriented [29]: This model of data representation is based on graph theory. It relies mainly on the notion of:

- Nodes that each have their own structure
- Relations between the nodes
- Properties (of nodes or relations)

Indeed, this storage model facilitates the representation of the real world, which makes it particularly well suited to the processing of data from social and geographical networks for example, and all data strongly connected in a general way. The main solution to this model is Neo4J [30].

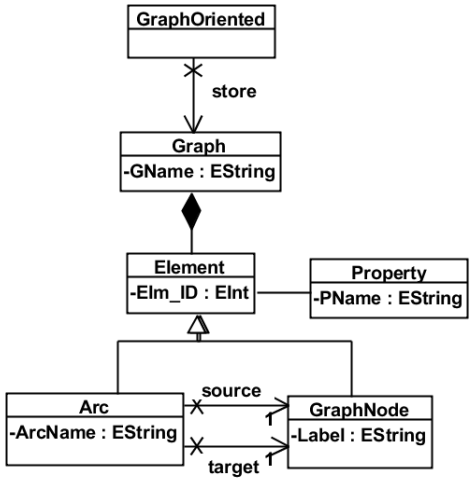

Figure 8: Meta-model for Graph-oriented database

5.5 The generic meta-model for NoSQL Databases

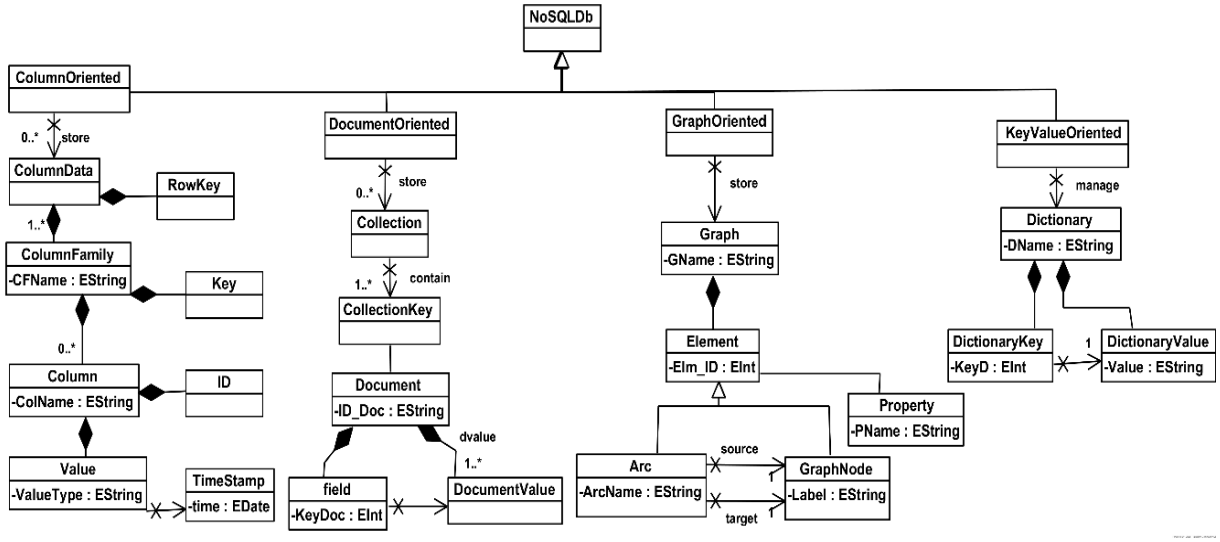

Figure 9: Meta-model of NoSQL Databases

Our proposed meta-model includes the four types of NoSQL databases we have cited above. Firstly, we talked about The Key/Value databases. These are used to manage dictionaries that consist of a key/value pair. Then, we present Graph-oriented databases, which can store graphs based on the notion of nodes, relationships, and properties attached to them. After that, we meta-modeled the Document-oriented databases with seven meta-classes that store collections containing documents. Finally, we have column-oriented databases that store data as rows with columns. However, this storage is distinguished by the fact that the number of columns can vary from one row to another.

6. GENERIC META-MODELING

In our previous work, we have already proposed a universal meta-modeling for the two layers Data Sources and Ingestion [3]. Still based on the global architecture of a Big Data system, which we have already introduced in our previous work [2], we have worked in this article on the storage layer which has two essential components which are HDFS and NoSQL databases. To express the relationship between storage layer and the other two layers that are Data Sources and Ingestion, we used the following meta-package diagram:

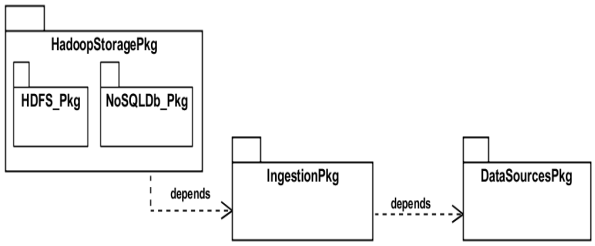

Figure 10: Meta-package of Storage, Ingestion, and Data Sources layers

Our meta-package diagram shows the relationship between the three layers: Hadoop Storage, Ingestion, and Data Sources. The HadoopStoragePkg meta-package contains two meta-packages, which are: HDFS_Pkg and NoSQLDb_Pkg, which define the meta-models of the HDFS and NoSQL databases that we presented in the previous paragraphs. The meta-package "HadoopStoragePkg" has a direct relationship with the meta-packages "IngestionPkg" and "DataSourcesPkg"; that is why we used the dependency relationships between our three meta-packages.

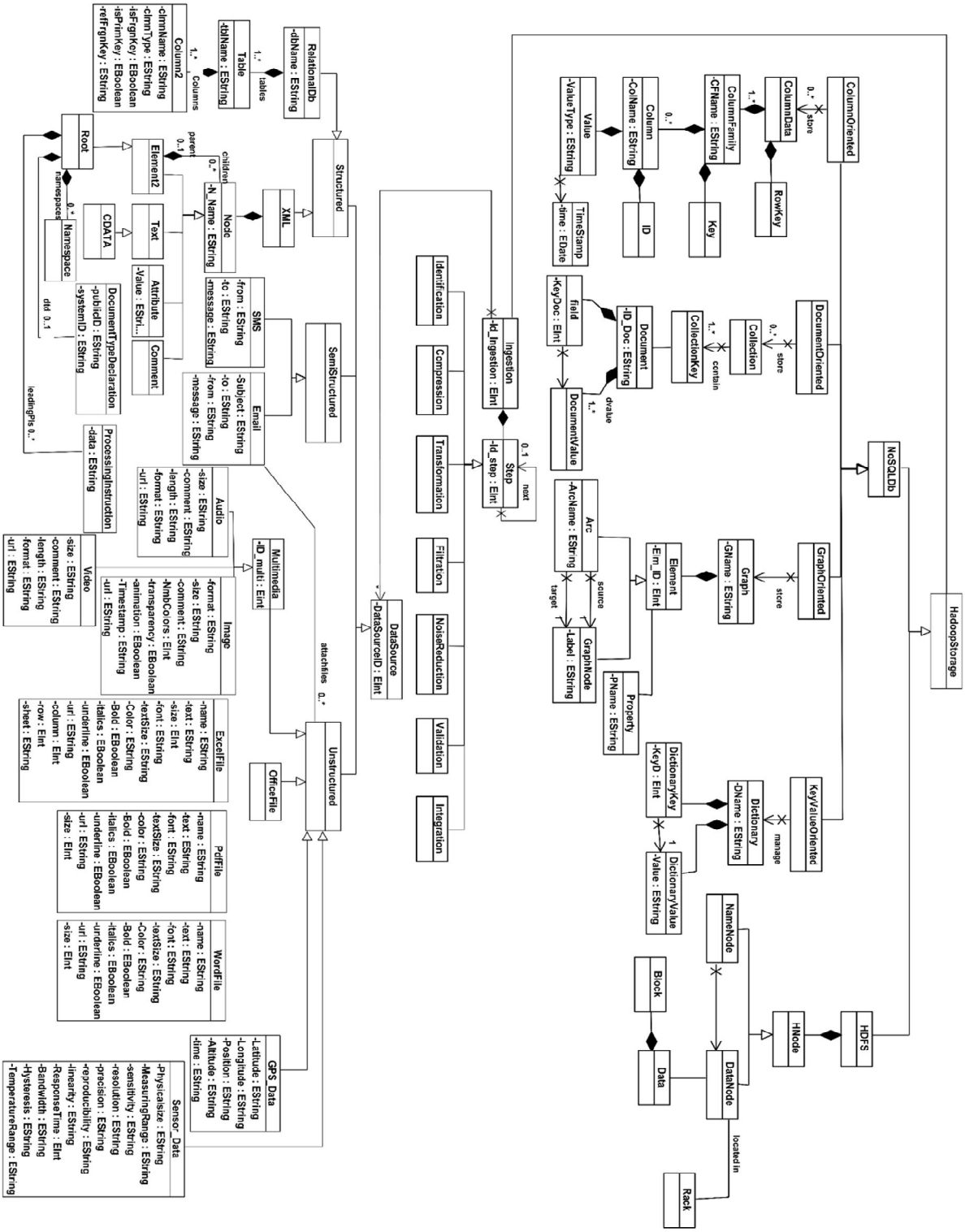

Figure 11: Meta-model of Storage, Ingestion, and Data Sources layers

Figure 11 presents the generic meta-model for the three layers: Data Sources, Ingestion, and Storage. This meta-model contains all the meta-classes needed to standardize concepts at a Big Data level since the majority of big data solution providers do not use meta-models to develop their solutions, which causes problems with regard to the diversity of the proposed solutions.

7. DISCUSSION

We live in a digital world, where our actions on the Internet generate digital traces closely related to our personal lives. The volume of these traces generated daily increases exponentially, creating massive loads of information, called Big Data. Such a large amount of information can not be stored or processed by using standard Database Management System (DBMS) tools. Hence, new tools have emerged to help us to meet Big Data challenges. Accordingly, Big Data is an important subject that enables us to identify and extract valuable and relevant knowledge. Many researchers have worked on big data, particularly on its value chain and on its processing tools [33]. Therefore, this work relies more particularly on our three research studies that we have already done in the world of Big Data and its different solutions. We have found that there are several distributions that can handle Big Data (HortonWorks, Pivotal HD, IBM Big Insights, etc.). Each distribution provider designs a solution in its own way without respecting standard references such as meta-models; the fact that caused the diversity of solutions and the non-interoperability between the different solutions. In our research project, we apply techniques related to the engineering of the models to propose a universal meta-modeling including all the layers of the architecture of a Big Data system. After the creation of these meta-models, in the next step, we shall work on the creation of models respecting these meta-models. Then we shall define the transformation rules between these meta-models using the transformation language ATL (Atlas Transformation Language) [31,32]. These meta-models are platform independent according to Model Driven Architecture pattern, which describes the structures of Data Sources, Ingestion, and Hadoop Storage independently from any specific platform.

8. CONCLUSION AND PERSPECTIVE

The evolution of the size of the databases has increased in a considerable way (from a few KB to To). For this, the "data scientists" have adapted by extending different methods to analyze and store Big Data. In this paper, we continued the application of techniques related to the engineering of models "MDE" in order to propose a meta-modeling for NoSQL databases and HDFS. These methods are independent of the model-driven structure, which describes the structures of independent storage of any specific platform. In our next work, we will link the proposed meta-models for storage layer with the others proposed for the other layers, and we will create transformations between these meta-models using the ATL transformation language.

REFERENCES

1. Richards, Ken. **Machine Learning: For Beginners - Your Starter Guide For Data Management, Model Training, Neural Networks, Machine Learning Algorithms.** CreateSpace Independent Publishing Platform, 2018.
2. Abdessamad B, and Abderrahim Tragha, "**Digging into Hadoop-based Big Data Architectures,**" *Int. J. Comput. Sci. Issues IJCSI*, vol. 14, no. 6, pp. 52–59, Nov. 2017. https://doi.org/10.20943/01201706.5259
3. Erraissi Allae, et Abdessamad Belangour. « **Data Sources and Ingestion Big Data Layers: Meta-Modeling of Key Concepts and Features** ». *International Journal of Engineering*, s. d., 7.
4. Alapati, Sam R. **Expert Hadoop Administration: Managing, Tuning, and Securing Spark, YARN, and HDFS**. Boston, MA: Addison Wesley, 2016. https://doi.org/10.1007/978-1-4842-3126-5_12
5. Raj, Pethuru, and Ganesh Chandra Deka. **A Deep Dive into NoSQL Databases: The Use Cases and Applications**. S.l.: Academic Press, 2018.
6. Erraissi, A., et A. Belangour. « **Meta-modeling of Zookeeper and MapReduce processing** ». In 2018 *International Conference on Electronics, Control, Optimization and Computer Science (ICECOCS)*, 1–5, 2018. https://doi.org/10.1109/ICECOCS.2018.8610630
7. Erraissi A., Belangour A. (2019) **Capturing Hadoop Storage Big Data Layer Meta-Concepts**. In: Ezziyyani M. (eds) *Advanced Intelligent Systems for Sustainable Development (AI2SD'2018). AI2SD 2018*. Advances in Intelligent Systems and Computing, vol 915. Springer, Cham https://doi.org/10.1007/978-3-030-11928-7_37
8. B.Manoj, K.V.K.Sasikanth, M.V.Subbarao, et V Jyothi Prakash. « **Analysis of Data Science with the Use of Big Data** ». *International Journal of Advanced Trends in Computer Science and Engineering 7*, nº 6 (15 December 018): 87–90. https://doi.org/10.30534/ijatcse/2018/02762018.
9. George, Lars. **Hbase: The Definitive Guide: Random Access to Your Planet-size Data**. 2nd Revised edition. O'Reilly Media, Inc, USA, 2018.
10. **Apache Hive Essentials: Essential techniques to help you process, and get unique insights from, big data**, 2nd Edition eBook: Dayong Du: Gateway.
11. Daniel, Dai, et Gates Alan. « **Programming Pig: Dataflow Scripting with Hadoop**: **9781491937099: Computer Science Books** », s. d.
12. N. Sawant and H. (Software engineer) Shah, **Big data application architecture Q & A a problem-solution approach.** Apress, 2013. https://doi.org/10.1007/978-1-4302-6293-0

13. Balasubramanian, Sriram. **Big Data Hadoop The Premier Interview Guide**, 2017.
14. D. Borthakur, "**HDFS architecture guide**," Hadoop Apache Proj. http//hadoop apache …, pp. 1–13, 2008.
15. Angadi, A. B., Angadi, A. B., & Gull, K. C. (2013). **Growth of New Databases & Analysis of NOSQL Datastores**. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3, 1307-1319.
16. Kumar, R., Charu, S., & Bansal, S. (2015). **Effective way to handling big data problems using NoSQL Database (MongoDB)**. *Journal of Advanced Database Management & Systems*, 2(2), 42-48.
17. M. Seeger and S. Ultra-Large-Sites, "**Key-value stores: a practical overview,**" … Sci. Media, pp. 1–21, 2009.
18. M. Meyer, "**Riak Handbook**," 2011.
19. Carlson, Josiah L. **Redis in Action**. Pap/Psc. Shelter Island, NY: Manning Publications, 2013.
20. B. Akboka, N. Filipchuk, and E. Zimanyi, "**Advance database : Voldemort**," 2015.
21. D. Abadi, "**The Design and Implementation of Modern Column-Oriented Database Systems**," Found. Trends® Databases, vol. 5, no. 3, pp. 197–280, 2012. https://doi.org/10.1561/1900000024
22. F. Chang et al., "**Bigtable: A distributed storage system for structured data,**" *7th Symp. Oper. Syst. Des. Implement. (OSDI '06)*, Novemb. 6-8, Seattle, WA, USA, pp. 205–218, 2006.
23. Carpenter, Jeff, and Eben Hewitt. **Cassandra – The Definitive Guide** 2e. 2nd ed. Sebastopol, CA: O′Reilly, 2015.
24. Amazon Web Services, "Amazon DynamoDB Developer Guide API Version 2012-08-10." 2012.
25. A. Issa and F. Schiltz, "**Document oriented Databases**," 2015.
26. Syn-Hershko, Itamar. **RavenDB in Action**. Manning Publications, 2016.
27. Team, CouchDB. **CouchDB 2.0 Reference Manual**. Samurai Media Limited, 2015.
28. Bradshaw, Shannon, and Kristina Chodorow. Mongodb: **The Definitive Guide: Powerful and Scalable Data Storage.** 3rd ed. Place of publication not identified: O'Reilly Media, Inc, USA, 2018.
29. Robinson, Ian, Jim Webber, and Emil Elfrem. **Graph Databases 2e.** 2nd ed. Beijing: O′Reilly, 2015.
30. Baton, Jerome, and Rik Van Bruggen. **Learning Neo4j 3.x - Second Edition: Effective data modeling, performance tuning and data visualization techniques in Neo4j.** 2nd Revised edition. Packt Publishing Limited, 2017.
31. "**ATL: Atlas Transformation Language Specification of the ATL Virtual Machine.**"
32. Banane, Mouad, et Abdessamad Belangour. **« New Approach Based on Model Driven Engineering for Processing Complex SPARQL Queries on Hive** ». *International Journal of Advanced Computer Science and Applications* 10, nº 4 (2019). https://doi.org/10.14569/IJACSA.2019.0100474.
33. Danish Ahamad, Shabi Alam Hameed, and MD Mobin Akhtar. « **A Review and Analysis of Big Data and MapReduce** ». *International Journal of Advanced Trends in Computer Science and Engineering* 8, nº 1 (15 February 2019): 1‒3. https://doi.org/10.30534/ijatcse/2019/01812019.