M2Onto: an Approach and a Tool to Learn OWL Ontology from MongoDB Database

Hanen Abbes, Faiez Gargouri

MIRACL Laboratory, Higher Institute of Computer Science and Multimedia, Sfax University, Sfax, Tunisia

abbes.hanen@gmail.com, faiez.gargouri@isimsf.rnu.tn

Abstract. Ontologies provide shared and reusable pieces of knowledge about a specific domain. Building an ontology by hand is a very hard and prone to errors task. Ontology learning from existing resources provides a good solution to this issue. Databases are widely used to store data. They were often considered as the most reliable sources for knowledge extraction. NOSQL databases are more and more used to store data. MongoDB database is emerging as the fastest growing NOSQL database in the world. It belongs to the document oriented databases variant. This paper proposes an approach to learn OWL ontology from data in MongoDB database and describes a tool implementing transformation rules.

Keywords: NOSQL database, MongoDB, ontology learning, OWL, transformation rules.

1 Introduction

The semantic Web remains one of the most appealing technologies in the world. Nevertheless, there is still a hard bottleneck in the generation of ontologies as the fundamental cornerstone of the semantic web. Ontologies provide shared and reusable pieces of knowledge about a specific domain[1]. They may be defined as a formalized vocabulary of concepts and their relationships or as an explicit assumption of a subject domain, that represent an agreed "universe of discourse" to link information structures [2].

By defining shared and common domain theories, ontologies help both people and machines to communicate concisely, supporting the exchange of not only syntax, but also semantics. Hence, the cheap and fast construction of domain specific ontologies is crucial for the success and the proliferation of the semantic Web [3].

More and more researchers are attracted by ontology research fields and many ontology editors have been developed to assist domain experts in developing and managing ontologies. Since manual creation of ontologies is a laborious task, there has been a great inclination towards semi-automatic as well as automatic generation of ontologies over the past decade. Ontology learning [4] emerged then as a field of ontology engineering to propose a solution to this issue. It remains an active research area with

considerable potential to facilitate the work of ontology engineers [5]. The main goal of ontology learning methods is to derive automatically an ontology from existing data [6]. To provide an extended automated support to facilitate the production of high quality ontologies from databases, adequate ontology learning methods should be elaborated [7].

Databases are often considered as the most reliable sources for knowledge extraction. NOSQL databases[1] are emerging nowadays as a new kind of databases developed in response to the requirements of modern applications.

NOSQL databases permit to store large volumes of structured, semi-structured, and unstructured data. Moreover, They provide high speed access to semi-structured and unstructured data and are very flexible. There are four types of NOSQL databases namely, key/value stores, column family stores, document oriented databases and graph databases [7].We choose to deal with document oriented databases. They correspond to an extension of the well-known key-value concept where the value consists of a structured document. A document contains hierarchically organized data similar to XML and JSON. This allows to represent one-to-one as well as one-to-many relationships in a single document. Therefore a complex document can be retrieved or stored without using joins. Since document oriented databases are aware of stored data, they enable to define document field indexes as well as to propose advanced query features. The most popular document oriented databases are MongoDB[2] and CouchDB[3].

Due to the abundance of NOSQL databases in the storage of data nowadays, recent attempts have been oriented towards automatic extraction of ontologies from these databases. The results obtained by these methods need yet to be improved.

To the best of our knowledge, there is almost no research works that deal with ontology learning from NOSQL databases in the real sense of the word except some research works dealing with NOSQL databases integration. We are particularly interested to MongoDB as a document oriented NOSQL database.

This paper proposes an automatic ontology learning approach, which acquires OWL[4] ontology based on data in MongoDB database. The resulting ontology is to be used as a conceptual view over the data stored in the MongoDB database.

This paper is outlined as follows. Section 2 describes briefly the related work in terms of ontology based NOSQL databases integration. Our approach to learn an ontology from MongoDB database is detailed in section 3. It first gives an overview of the MongoDB database followed by transformation rules and implementation details. Finally section 4 draws conclusions and suggests further research.

---

[1] http://nosql-database.org/

[2] http://www.mongodb.org/

[3] http://couchdb.apache.org/

[4] http://www.w3.org/TR/owl-features/

2 Related Work

In [8], authors propose a data integration framework where the target schema is represented as an OWL ontology and the sources correspond to NOSQL databases namely MongoDB and Cassandra. The first step of this work consists of generating a local schema for each integrated source using an inductive approach. This approach uses non standard description logic reasoning services like Most Specific Concept (MSC) [9] and Least Concept Subsumer (LCS) [10] in order to generate a concept for a group of similar individuals and to define hierarchies for these concepts. The second contribution enables the specification of a global ontology based on the local ontologies generated for each data source. This global ontology results from the correspondences discovered between concept definitions present in each local ontology. For the source ontology generation, authors use containers, i.e. collections in MongoDB and column families in Cassandra to deduce a schema. Authors consider that each container defines a DL concept and that each key label corresponds to a DL property that can be either a dataTypeProperty or an objectProperty and whose domain is the DL concept corresponding to its container. These concepts are hierarchically organized by means of Formal Concept Analysis methods [11]. Finally an incremental schema generation approach is implemented. At the end of this step, a schema for each NOSQL data source is created. To discover alignments between ontologies and to build the global ontology, the first step is to enrich the two ontologies to be aligned using the IDDL reasoner [12], the second step detects the simple correspondences using three classical alignment processes and the last step detects the complex correspondences.

Authors in [13] develop an ontology based semantic integration system for a column-oriented NoSQL data store (HBase). The proposed system is based on four main steps namely schema generation, schema to ontology conversion, ontology alignment-merging, and semantic querying. In the first step, schema has to be generated for the data stores considering the fact that millions of individuals exist in the store and there is no restriction on the number of attributes for each individual. Two methods for schema generation have been proposed in this work: an on-line Schema generation which generates and maintains schema for a data store that is being populated, and an offline schema generation which generates and maintains schema for a data store that is already populated. The proposed system creates a lookup table in HBase for each table whose schema has to be extracted. For this a map-reduce task is launched on the source table. Then, schema extraction is done by finding the fittest individual by means of a genetic algorithm. The schema obtained gives an idea of the column families and columns presenting the greatest common subsumer of the data store which is an individual that can represent every other individual in a given set. This information is converted to suitable OWL primitives and mapped into OWL ontology. The system extracts column details from the persisted schema of the table and applies conversion rules to convert them to ontology entities. In the third step, the local ontologies are

aligned to find semantic similarities. COMA++ 3.0 CE tool[5] was used to find lexical similarities between ontology entities. After generating alignments, a global ontology is formed from local ontologies by adding the new relationships found between entities of local ontologies. Finally, The global ontology serves as T-Box for OWL reasoners that query the SPARQL endpoint. The RDF triples obtained as result of SPARQL query are used for inference and more matching triples are generated as output of query. A custom made SPARQL endpoint with reasoning for HBase is the main contribution in this module.

3 How to learn OWL ontology from MongoDB?

3.1 Overview of MongoDB

MongoDB is the fastest growing new database in the world. It provides a rich document oriented structure with dynamic queries and allows to compartmentalize data into collections to divide data logically. Thus, the speed of queries can increase dramatically by querying on a subset of the data instead of all it. Collections are analogous to tables in a relational database. Each collection contains documents that can be nested in complex hierarchies but still easy to query and index. A document is seen as a set of fields, each one being a key-value pair. A key is a string and the associated value may be a basic type, a document, or an array of values. MongoDB is schemaless. The documents stored in the database can have varying sets of fields, with different types for each field. MongoDB offers a tree-like structure in documents by storing references to "parent" nodes in children nodes to model large hierarchical data relationships. MongoDB does not support joins. However, in some cases it is better to store related information in separate documents, typically in different collections or databases. This is insured by means of DBRef references. DBRefs are references from one document to another using the value of the first document's _id field, collection name, and, optionally, its database name. By including these names, DBRefs allow documents located in multiple collections to be more easily linked with documents from a single collection. MongoDB can manage data of any structure without expensive data warehouse loads, no matter how often it changes. So, we can cheap new functionalities without redesigning the database.

3.2 Transformation rules

Our approach to learn an ontology from MongoDB database is based on transformation rules to map MongoDB constructs into OWL ontology [14]. Our choice is oriented to OWL as the ontology representation language since it is the standard rec-

---

5 http://dbs.uni-leipzig.de/en/research/projects/schema_and_ontology_matching/coma_3_0/coma_3_0_community_edition

ommended by the World Wide Web Consortium (W3C)[6] to represent ontologies. OWL facilitates greater machine interoperability of web content than that supported by other ontology representation languages such as XML and RDF by providing additional vocabulary along with formal semantics. We are interested particularly to OWL-DL since it supports the maximum expressiveness while retaining computational completeness and decidability. Our approach is based on five main steps.

The first step is the creation of the ontology skeleton. It consists of defining ontology classes and detecting subsumption relationships between them. To do this, every collection in the database must be transformed into an ontology class. Then, A subsumption relationship is extracted from the field "parent" in every document. It must be transformed into a "subClassOf" relationship in the ontology.

The second step is to learn concepts properties (dataTypeProperties and objectProperties). Each basic field in a document is transformed into a dataTypeProperty in the class corresponding to the collection containing this document into the ontology (Fig. 1).

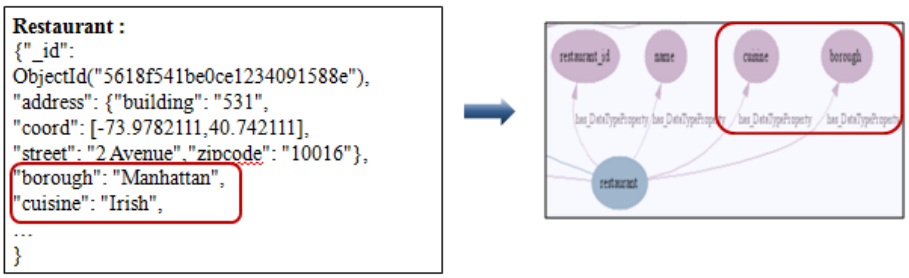

Fig. 1. Learning dataTypeProperties

ObjectProperties are extracted either from embedded documents or from references with DBRef (Fig. 2). An embedded document gives rise to an objectProperty whose "domain" is the class corresponding to the embedding document and the range is the class corresponding to the embedded document. The name of the objectProperty relationship is the concatenation of the word 'has' with the name of the document B (Fig. 2-a). A DBref reference is transformed into objectProperty for which the domain is the class corresponding to the referencing document and the range is the referenced class (corresponding to the referenced document). The name of the objectProperty relationship is the concatenation of the word 'has' with the name of the referenced document (Fig. 2-b).

[6] https://www.w3.org/

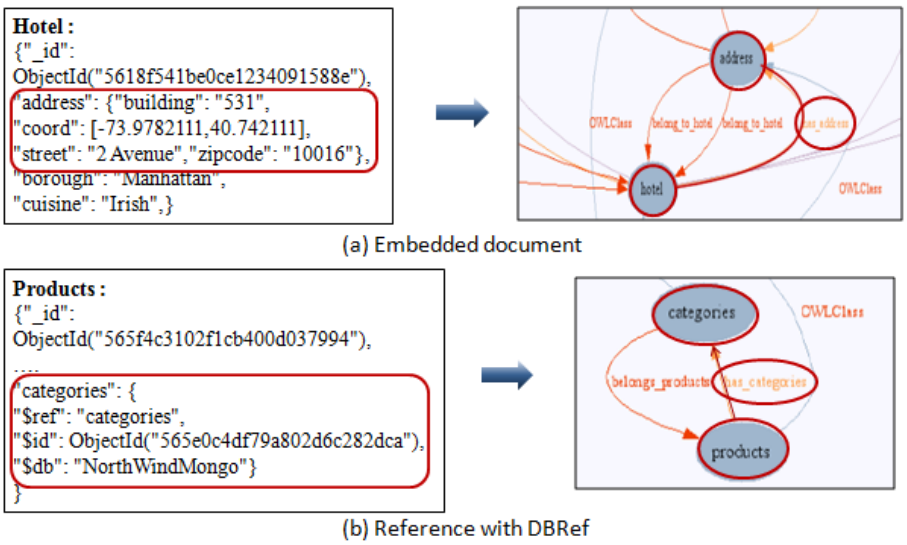

Fig. 2. Learning objectProperties

Individuals are identified in the third step. The values of fields in the database documents are transformed to individuals into the ontology.

In the fourth step, we deduce class axioms (equivalence and disjoining), property axioms (inverseOf) and constraints (cardinality constraints, value constraints). To deal with class axioms, we consider that two classes C1 and C2 are equivalent if both class extensions contain exactly the same set of individuals, else C1 is disjoint with C2. As for the property axiom inverseOf, for each objectProperty corresponds an inverse objectProperty with exchanged domain and range. To learn cardinality constraints, a maxcardinality constraint is calculated from the size of DBList fields. Mincardinality = 0 if the document's fields are empty, else mincardinality = 1. The owl:cardinality constraint is defined if mincardinality = maxcardinality. The value constraint stipulates that all individuals related to an objectProperty must belong to its range.

Finally, we enrich the ontology with classes definition operators (union, intersection, complement).

Classes definition operators (union, intersection, complement) are borrowed from Description Logic. They can be viewed as representing the AND, OR and NOT operators on classes. The intersection class constructor describes a class whose set of individuals contains precisely those ones that are simultaneously individuals of all class descriptions in the list. It is assimilated to logical conjunction. The union class constructor describes an anonymous class whose individuals are those ones that occur in at least one of the class extensions of the class descriptions in the list. It is assimilated to logical disjunction. The complement class constructor describes a class whose individuals are exactly those ones that do not belong to the class extension of the class description that is the object of the statement. It is assimilated to logical negation.

3.3 Implementation

We implemented a tool (M2Onto: MongoDB to ontology) to perform these tasks. This work is directed by means of the JAVA programming language. To access MongoDB database we used the mongo-java-driver-2.13.0 API and to perform the ontology creation we used the OWL-API version 4.0.1. Our tool allows to select an existing database. By clicking on the "Create Ontology" button (Fig. 3), the M2Onto tool executes transformation rules and generates the corresponding OWL ontology. Our tool provides an OWL file as output as well as a graphical representation of the resulting ontology. The following figure shows an OWL file as output to the ontology learning process.

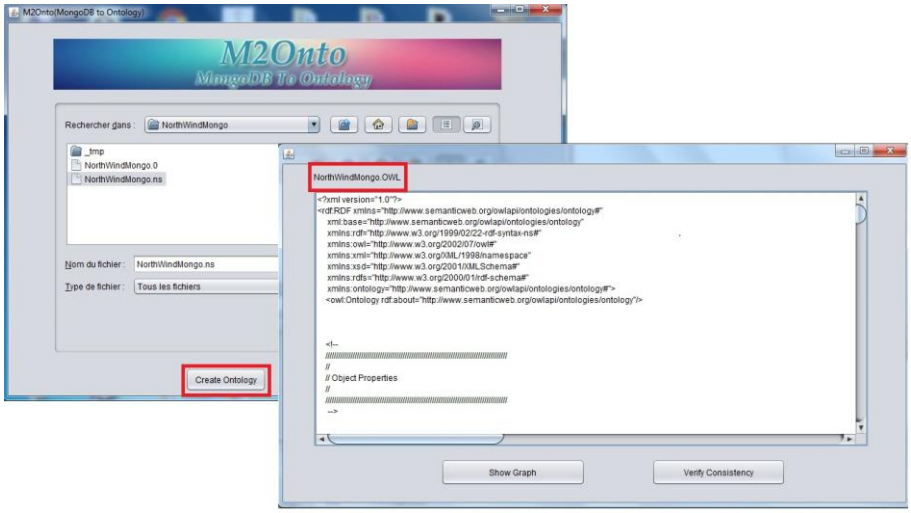

Fig. 3. Visualization of an OWL file with the M2Onto tool

By clicking the "Show Graph" button, a graphical representation of the resulting ontology appears. The graphical representation is available thanks to the GraphViz API. It is an API that allows to generate graphs from a text file describing nodes and arrows of the graph. GraphViz places the nodes in a way that it minimizes arrows intersection. It allows to visualize concepts and relationships of the ontology. We used GraphViz version 2.24.

In order to test the validity of our proposal, it was compulsory to use a sample MongoDB database. Unfortunately, we didn't find a sample database that covers all MongoDB constructs. We finally resorted to the "Northwind" database which is a sample database provided with the Microsoft Access DataBase Management System. It describes all commercial transactions of the "Northwind Traders" company which is specialized in the import-export business of aliments. The relational modeling of the "NorthWind" database is provided in Fig. 4. The database describes all sale transactions that take place between the company and its customers and all purchase transactions between the company and its suppliers. The "Northwind" database is available in the JSON format for MongoDB with the name "NorthwindMongo". It may be

downloaded from https://github.com/raynaldmo/northwind-mongodb. It contains 13 collections namely: “categories”, “companies”, “customers”, “employees”, “northwind”, “order_details”, “orders”, “products”, “regions”, “shippers”, “suppliers” and “territories”.

The “NorthwindMongo” database is a simple data dump from the “Northwind” database to a data set in the JSON format. It consists on a “flat” transformation that ignores both notions of embedded documents and references with DBRef. It simply transforms relational tables to collections and ignores joins and integrity constraints which have to be transformed into references and embedded documents. Moreover, this data dump fails to implement the “parent/child” relationship between collections. Thus, it is not adequate to test the totality of transformation rules that we proposed. To this end, we modified an excerpt of the database that carries on the collections highlighted in Fig. 4. Moreover, to represent the «parent/child» relationship, we created a collection “companies“ and added the field “parent” in the collections “customers“ and “shippers“. As an example, a document containing the field "parent" in the collection "customers" becomes as follows : {"_id" : ObjectId("566602bd442e0ce009000029"), "parent" : "companies"}. We also eliminated null valued fields in order to preserve the shemalessness aspect of the MongoDB database.

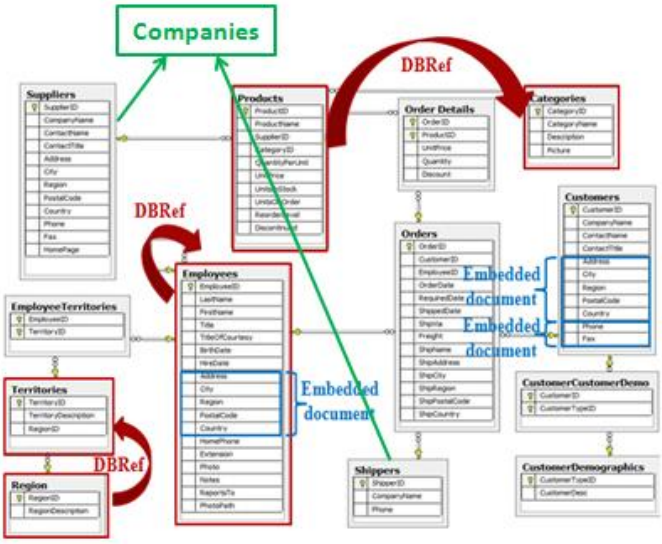

Fig. 4. Summary of the changes performed against the “NorthWindMongo” database schematized over the relational model of the “NorthWind” database

The execution of the M2Onto tool over the “NorthWindMongo” database gives a consistent Owl ontology as output. Fig. 3. provides the resulting OWL file and Fig. 5. shows the graphical representation of the “NorthWindMongo” ontology.

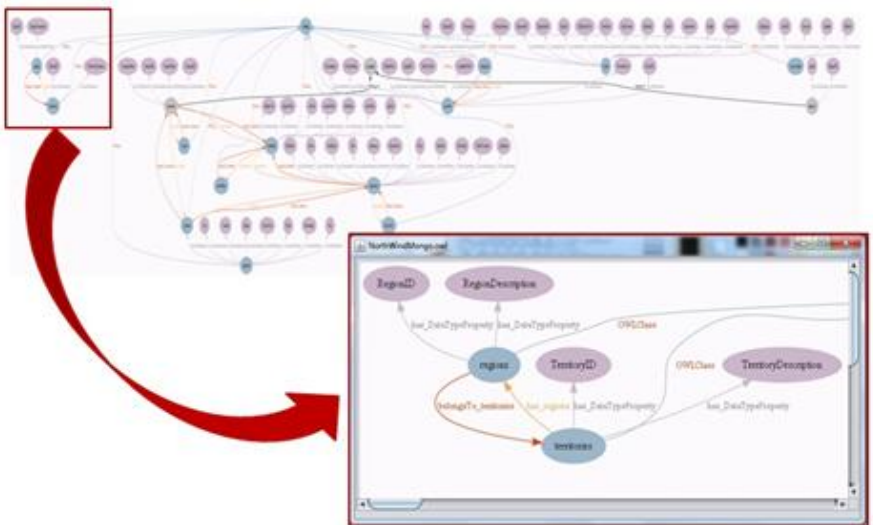

Fig. 5. The "NorthWindMongo" ontology graph visualized by the M2Onto tool

Classes are represented by blue nodes. Sub-classes are represented by grey nodes. DataTypeProperties are represented by purple nodes. ObjectProperties are represented by red edges and their inverses by orange edges.
The evaluation of the resulting ontology was performed by checking the consistency of the generated ontology. To do this, we integrated the Pellet[7] reasoner to the M2Onto tool.

4 Conclusions and Future Work

Ontologies play an important role for knowledge intensive applications as a set of formally defined terms to allow concise communication. Research on ontology is becoming increasingly widespread in the computer science community. Ontology learning is a good solution to minimize mass of handwork met by the ontology engineer. This paper dealt with ontology learning from MongoDB database to address the semantic aspect of the stored data. We discussed state of art regarding NOSQL databases integration. An ontology learning approach is proposed to automatically construct OWL ontology based on data stored in MongoDB database. The related learning rules are discussed. It can be seen that the approach is practical and helpful to automate the task of ontology building from MongoDB database. A tool implementing the proposal is also described.

[7] https://www.w3.org/2001/sw/wiki/Pellet

As future work, we envisage to integrate other ontology learning techniques into our approach to obtain better learning result and to propose an approach to update the resulting ontology taking into account the updates of the MongoDB database.

References

1. Gruber, T.R.: A translation approach to portable ontology specifications. Knowledge Acquisition. 5(2), 199--220 (1993)
2. George, D: Understanding Structural and Semantic Heterogeneity in the Context of Database Shema Integration. Proceedings of the SIXTH Conference in the Dept. of Computing, (Journal of the Dept. of Computing, UCLan, Issue Number 4, ISSN 1476-9069, pp. 29--44 (2005)
3. Maedche, A., Staab, S.: Ontology Learning for the Semantic Web. In: IEEE Intelligent Systems 16(2), pp.72--79 (2001)
4. Maedche, A., Staab, S.: Ontology Learning. Handbook on Ontologies, 173--190 (2004)
5. Cimiano, P., Mädche, A., Staab, S., Völker, J.: Ontology Learning. Handbook on Ontologies, 245--267 (2009)
6. Bontcheva, K., Sabou, M.: Learning Ontologies from Software Artifacts: Exploring and Combining Multiple Sources. Proceedings of the 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE) (2006)
7. Nayak, A., Poriya, A., Poriya, A.: Type of NOSQL Databases and its Comparison with Relational Databases. International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868 Foundation of Computer Science FCS, New York, USA Volume 5– No.4 (2013)
8. Curé, O., Lamolle, M., Le Duc, C.: Ontology Based Data Integration Over Document and Column Family Oriented NOSQL. The Computing Research Repository (Corr) (2013)
9. Baader, F., Calvanese, D., McGuiness, D.L., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook: Theory, Implementation, Applications. Cambridge University Press, Cambridge, UK (2003)
10. Baader, F., Sertkaya, B., Turhan, A.Y.: Computing the least common subsumer w.r.t. a background terminology. Journal of Applied Logic Springer, pp. 400--412 (2004)
11. Obitko, M., Snasel, V., Smid, J.: Ontology design with formal concept analysis. CLA. vol. 110 (2004)
12. Zimmermann A., Le Duc, C.: Reasoning with a network of aligned ontologies. In: Proceedings of the 2nd International Conference on Web Reasoning and Rule Systems (ICWRRS), pp. 43--57 (2008)
13. Kiran, V.K., Vijayakumar, R.: Ontology Based Data Integration of NoSQL Datastores. 9th International Conference on Industrial and Information Systems (ICIIS) (2014)
14. Abbes, H., Boukettaya, S., Gargouri, F.: Learning Ontology from Big Data through MongoDB Database. 2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA) (2015)