JsonToOnto: Building Owl2 Ontologies from Json Documents

Sara Sbai[1], Mohammed Reda Chbihi Louhdi[2], Hicham Behja[3], Rabab Chakhmoune[4]
LRI – Laboratory, ENSEM, Hassan II University, Casablanca, Morocco[1, 3, 4]
Research Laboratory on computer science innovation, Faculty of Sciences Ain Chock
Hassan II University, Casablanca, Morocco[2]

***Abstract*—The amount of data circulating through the web has grown rapidly recently. This data is available as semi-structured or unstructured documents, especially JSON documents. However, these documents lack semantic description. In this paper, we present a method to automatically extract an OWL2 ontology from a JSON document. We propose a set of transformation rules to transform JSON elements to ontology components. Our approach also allows analyzing the content of JSON documents to discover categorization in order to generate class hierarchy. Finally, we evaluate our approach by conducting experiments on several JSON documents. The results show that the obtained ontologies are rich in terms of taxonomic relationships.**

***Keywords*—JSON documents; OWL2 ontologies; ontology generation; transformation rules; information theory; classification; decision trees**

I. Introduction

A tremendous amount of documents exists on the web, especially semi-structured and unstructured documents, and it is continuously increasing which makes analyzing and retrieving these documents difficult. To overcome these difficulties, we need to consider their semantics.

Semi-structured documents on the web are available in different formats, such as XML, HTML and JSON.

JSON (JavaScript Object Notation) [1] is a lightweight data interchange format that was first specified and popularized by Douglas Crockford. It is based on a subset of the JavaScript Programming Language.

JSON has been widely used due to its simplicity and ability to be processed by both humans and machines easily. However, it lacks semantics due to the fact that it is schema less.

This work is supported by OCP group, Morocco.

Ontologies are essentially used to express semantics and integrate them in web applications.

Tom Gruber [2] defined an ontology as "an explicit specification of a conceptualization of a domain of interest", as for Swartout and colleagues [3], they defined an ontology as "a hierarchically structured set of terms for describing a domain that can be used as a skeletal foundation for a knowledge base". Most existing methods in ontology extraction from semi-structured data use XML documents as an information source.

In this work, we propose an automatic approach to build OWL2 ontology from a JSON document. We propose a set of transformation rules to translate JSON elements to ontology constructs. We also use data mining techniques to analyze the documents 'content in order to discover class hierarchy.

The remainder of this paper is organized as follows. Section II discusses related works in ontology extraction from semi-structured documents. Section III describes the proposed method for extracting OWL2 ontology from JSON document. Section IV presents the experimentations and the results. And finally, Section V concludes this paper, and discusses the perspectives of this work.

II. Related Works

For semi-structured data, we find different formats such as XML and JSON. In this section we will present a few existing methods in ontology extraction from JSON documents.

In [4], the authors propose an automatic approach to convert web data into OWL ontology. This method takes as input related JSON data objects transmitted from web services to applications. It builds semantic models for data instances.

The process of extracting and constructing semantics is divided into four steps: (1) JSON parsing: The authors parse the data according to key-value pairs in JSON objects and transform them into sets of triplets, (2) Semantic mapping: The data is stored as triplets similar to the description of RDF turtle [5]. During this step, triplet sets are analyzed to construct ontologies and their instances, (3) Semantic enrichment: The authors deploy automatic learning methods to improve the use of semantic data, they also take advantage of ontology reasoning to provide additional information on ontology (axioms definition, constraints definition, comments and labels addition) and finally, (4) Ontology merging: During this step, the authors align several ontologies according to the relations and concepts between them and refine the descriptions to build a unified ontology. The authors compare ontology constructs by using domain dictionaries and thesaurus and then merge ontologies according to semantic correspondences between them.

In [6], the authors proposed a protégé plugin named OWLET to assist the experts during the refinement phase of the ontology construction process. This plugin offers an approach to transform real world (image) objects to instances in order to import them to the existing ontology model for semi-automated classification. The image objects are first

transformed to GeoJSON files and then converted to instances to be imported to the existing ontology.

In [7], the authors proposed a method to build ontology knowledge base from semi-structured datasets (e.g. spreadsheet, JSON, XML…). The first step is extracting target columns from the semi-structured datasets. Then, the authors proposed a transformation table generator (TTG) and a cell value importer (CVI) to import values from semi-structured data sets. Next, the authors defined a Property expression (ProperyExp) to describe mapping information to map the extracted columns to properties. And finally, the ontology knowledge base is constructed.

In another approach [8], the authors propose KESeDa to extract knowledge from heterogeneous semi-structured data sources. The approach has several processing steps. Before the processing the authors detect the file format first. If the file is an XML or HTML document, the authors use existing tools to extract knowledge. However, if the file is a JSON document, the authors apply their own approach. The first step is preparing the source file for later annotations. Therefore, all values contained in a JSON object are encapsulated in a separate object. This object also contains a table structure as a placeholder to store all identified properties that can be assigned to predicates during the following processes. Then, the values are analyzed using a set of dictionaries. The collected results are stored in the reserved table structure. The approach also offers the possibility to combine several dictionaries to map compound predicates. The next step is analyzing the values according to their data type and format. Then, the keys of the JSON objects are analyzed. If the name of a key exactly matches a predicate, it will be stored in the table. Otherwise, synonyms for the key are searched in a dictionary and evaluated based on a possible mapping. Another step is to transform the extended JSON object source into a JSON-LD [9] representation by selecting an appropriate RDF predicate for each property. Finally, the authors try to find an appropriate RDF class for each object according to its set of predicates.

We tried to find other approaches that link ontologies to JSON documents. We found three existing research works that use document oriented databases for ontology learning. Document oriented databases store documents in JSON format.

NoSQL (Not Only SQL) [10] are databases that are not built on tables and do not use SQL to manipulate data. They are used to manage large amounts of data or big data. NoSQL databases do not support ACID transactions across multiple data partitions for scalability reasons. The NoSQL databases also respond to the CAP theorem which is more suitable for distributed systems.

NoSQL databases are generally classified into four categories:

- Key / Value: The data is simply represented by a key / value pair. The value can be a simple string of characters or a serialized object.

  Key / value databases are simple and allow quick retrieval of values required for application tasks such as managing user profiles or sessions or retrieving product names.

  Example: Dynamo (Amazon), Voldemort (LinkedIn), Redis, BerkleyDB, Riak.

- Column Oriented: Employ a distributed, column-oriented data structure that hosts multiple attributes per key. They are useful for distributed data storage, large scale and batch data processing, and exploratory and predictive analysis by statisticians and programmers.

  Example: Bigtable (Google), Cassandra (Facebook), HBase (Apache).

- Document oriented: They were designed to manage and store documents. These documents are in XML, JSON or BSON format. Document-oriented databases are useful for managing Big Data-sized document collections such as text documents, emails and XML documents.

  Example: CouchDB (JSON), MongoDB (BSON).

- Graph oriented: They are based on graph theory. It is based on the notion of nodes, relationships and properties attached to them. They are useful when one is interested in the relations between the data.

  Example: Neo4j, InfoGrid, GraphDB, AllegroGraph, InfiniteGraph.

In the first approach [11], the authors propose a framework for data integration. They use two NoSQL databases, namely MongoDB as document-oriented database and Cassandra as column-oriented database as a source of information an OWL ontology as a target. The approach is divided into three steps. First, the authors create a local ontology that matches each data source. They consider that each container defines a DL concept and each key label defines an object property or a data property. To organize the concepts in a hierarchy, methods of formal concept analysis (FCA) [12] were used.

In the second step, the authors align the local ontologies to create a global ontology. First they enrich each ontology using the IDDL reasoner [13], then they detect simple and complex correspondences between the two ontologies.

Finally, the authors propose a query language to translate SPARQL to the query language of each source.

In the second approach [14], the authors use MongoDB as a data source and an OWL ontology as a result. The authors define a set of transformation rules to create the ontology concepts and properties. This approach is divided into four stages: (1) Creating the ontology skeleton, (2) Identifying object properties and data type properties, (3) Identifying individuals and finally (4) Deducting axioms and constraints.

In the next section, we propose an automatic approach to build ontology from JSON documents.

III. Proposed Method

The process of building ontology from scratch is tedious and error prone, therefore, we propose an automatic approach to extract an OWL2 ontology from a single JSON document.

First, we analyze the data in these documents to discover categorization patterns in order to identify class hierarchy. This will eventually enable us to generate ontology with a deep taxonomy. Then we propose a set of transformation rules to convert JSON elements to OWL2 components.

A. Class Hierarchy Identification

1) Inheritance identification using key labels: In this step, we analyze key labels in nested JSON object to identify class hierarchy. First we extract all keys from every object, then we compare them. If we find keys that exist in an object and don't exist in another, we create a super class corresponding to the JSON array of objects. A dataProperty corresponding to the common keys is then extracted where the domain is the super class and the range is the type of the JSON value (i.e. String, Integer…). Then sub classes are created where the label is a concatenation of the word "SubClassOf" plus the label of the super class plus a number, this number ranges from 1 to the number of the obtained subclasses. In the example presented in Fig. 1, we have two common keys "ExternalID" and "Type". We will have a super class "Party", which will be the domain of two Data Properties "hasExternalID" and "hasType". Then we will create two sub classes, "SubClassOfParty1" and "SubClassOfParty2". We then extract four Data Properties, "hasFirstName" and "hasLastName" where the domain is "SubClassOfParty1", and "hasOrganizationName" and "hasListingName" where the domain is "SubClassOfParty2".

2) Inheritance identification using Data Mining techniques: Data mining techniques look for patterns in large data. One of the techniques that are widely used is classification. Classification is used to gather data instances with similar traits in categories or classes.

Classification methods include decision trees, Bayesian networks, and k-nearest neighbor. Decision trees aim to split a dataset into homogenous classes.

Our decision tree induction is a recursive algorithm. It is based on C4.5 algorithm (see Fig. 2).

C4.5 algorithm [15] was proposed by Ross Quinlan in 1993. It is the successor to ID3 (Iterative Dichotomiser 3), it takes into account continuous attributes. Decision trees have a leaf which indicates a class, or a decision node that specifies the test to be carried out. The outcome of the test can either be a leaf or a subtree. The nodes and leafs are connected with branches.

The decision node is chosen by using information theory [16]. Entropy and information gain are calculated. Shannon's entropy is a measure of uncertainty of a random variable. Entropy is defined by:

$$H(X) = -\sum_{i=1}^{n} p_i \log_b p_i \quad (1)$$

Where:

X : The set of examples

n: the number of values

b: The number of distinct values

p_i where i Є [1, n]: the probability of occurrence of an element

The information gain of a set of examples X with respect to a given attribute a_j is the entropy reduction caused by the partition of X according to a_j. It is defined by:

$$Gain(X, a) = H(X) - \sum_{v \in valeur(a)} \frac{|X_{a=v}|}{|X|} H(X_{a=v}) \quad (2)$$

```
{
"customer": {
"details": {
"party": [{
                    "type": "individual",
                  "externalID": "ABC123",
                    "firstname": "John",
                    "lastname": "Smith"
},
        {
                     "type": "organization",
                     "externalID": "Apple",
                         "organizationName": "AppleInc",
                   "listingName": "APPLE"
        }]
}
}
}
```

```
<owl:Class rdf:about="http://www.JsonToOnto.com#SubClassparty1">
<rdfs:subClassOf  rdf:resource="http://www.JsonToOnto.com#party"/>
</owl:Class>
<owl:Class  rdf:about="http://www.JsonToOnto.com#SubClassparty2">
<rdfs:subClassOf rdf:about="http://www.JsonToOnto.com#party"/>
</owl:Class>
```

Fig. 1. An Example of a JSON Object with the Proposed Transformation.

DecisionTreeConstruction {Decision Tree Construction Algorithm}
Input:
- A class C
- Attributes {A1, ..., An}
- A set of data N

Output:
- The decision tree

IF all the examples of N are in the same class C**THEN**
  Create a leaf and assign the the current value of C to it
ELSE
  Select the attribute A the largest information gain as the best attribute
  Assign the label of the attribute A to the current node
  Split the data set N according to the values of the attribute A v1...vn to sub data sets N1, ..., Nn
  FORi = 1 to n
    DecisionTreeConstruction (C, Ai, Ni)
  END FOR
END IF
Return the decision tree
END

Fig. 2. Decision Tree Construction Algorithm.

Where:

$X_{aj = v} \subset X$ is the set of examples where the attribute a_j takes the value v and

$|X|$ indicates the cardinal of X .

The attribute with the highest information gain is used as a decision node.

In our approach, the predefined class is unknown, therefore we consider every attribute as a predefined class and we construct a decision tree for each one. Next, we determine the depth of each tree and choose the tree with the least depth since it leads to homogenous categories the fastest. Finally, we consider its leafs as our categories. The next figure describes our algorithm (Fig. 3).

To illustrate our algorithm, we use the JSON object presented in Fig. 4 as example.

First we construct our decision tree. We obtain the result presented in Fig. 5.

We consider the leafs of our trees as our sub classes. We have the presented in Fig. 6.

As presented in our results, the names of our sub classes are a concatenation of "SubClass" and the name of the super class followed by a number.

B. Transformation Rules

In this paragraph, we present the proposed transformation rules. We illustrate these rules through the example presented in Fig. 7.

Rule 1: Every JSON object is transformed to a simple class in the ontology. Example:

<owl:Class rdf:ID="Class1"/>

<owl:Class rdf:ID="director"/>

"Class1" corresponds to the main JSON object.

Inheritance detection {Inheritance detection Algorithm}
Input:
- Attributes {A1, ..., An}
- A set of data N

Output:
- List V designating the categories

Let d: Positive integer designating the tree's depth
Let C: A class
FOR each attribute Ai
C <- Ai
DecisionTreeConstruction(C, Ai, Ni)
d <- The tree's depth
END FOR
Choose the tree with the least depth
Select the leafs of the tree and add them to the list V
Remove duplicates from the list V
Return V
END

Fig. 3. Inheritance Detection Algorithm.

Rule 2: We analyze the key-value pairs. If the value is a simple type (string, number (integer, double) or boolean (true, false)), then we have a dataProperty where the domain is the class corresponding to the object containing the key and the range is the type of the value into the ontology. The dataProperty name is the concatenation of the "has", the key label and the label of the domain class. For example, see Fig. 8.

Rule 3: If an object B is integrated into an object A, we transform this integration into an ObjectProperty where the domain is the class corresponding to object A and the range is the class corresponding to object B. The name of the object property relationship is the concatenation of the word "has" with the name of the object B. For example, see Fig. 9.


```
[ {
"id": "0001",
   "type": "donut",
   "name": "Cake",
   "ppu": 0.55,
   "batters":  {
        "batter": [
             { "id": "1001", "type": "Regular" },
             { "id": "1002", "type": "Chocolate" },
             { "id": "1003", "type": "Blueberry" },
             { "id": "1004", "type": "Devil's Food" } ]
},
   "topping": [
             { "id": "5001", "type": "None" },
             { "id": "5002", "type": "Glazed" },
             { "id": "5005", "type": "Sugar" },
             { "id": "5007", "type": "Powdered Sugar" },
             { "id": "5006", "type": "Chocolate with Sprinkles" },
             { "id": "5003", "type": "Chocolate" },
             { "id": "5004", "type": "Maple" }   ]
    },
    {
"id": "0002",
"type": "donut",
"name": "Raised",
"ppu": 0.55,
    "batters": {
         "batter": [ { "id": "1001", "type": "Regular" }]
             },
    "topping": [
             { "id": "5001", "type": "None" },
             { "id": "5002", "type": "Glazed" },
             { "id": "5005", "type": "Sugar" },
             { "id": "5003", "type": "Chocolate" },
             { "id": "5004", "type": "Maple" }]
   },
   ……..
]
```


Fig. 4. Inheritance Detection Algorithm.

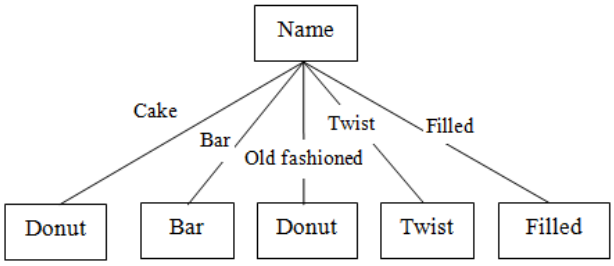

Fig. 5. Obtained Decision Tree with the Least Depth.

```
<owl:Class
rdf:about="http://www.JsonToOnto.com#SubClassClass11">
<rdfs:subClassOf
rdf:resource="http://www.JsonToOnto.com#Class1"/>
</owl:Class>
<owl:Class
rdf:about="http://www.JsonToOnto.com#SubClassClass12">
<rdfs:subClassOf
rdf:about="http://www.JsonToOnto.com#Class1"/>
</owl:Class>
<owl:Class
rdf:about="http://www.JsonToOnto.com#SubClassClass13">
<rdfs:subClassOf
rdf:about="http://www.JsonToOnto.com#Class1"/>
</owl:Class>
<owl:Class
rdf:about="http://www.JsonToOnto.com#SubClassClass14">
      <rdfs:subClassOf
rdf:about="http://www.JsonToOnto.com#Class1"/>
</owl:Class>
```

Fig. 6. Theresult OWL from the JSON Object in Fig. 4.

```
{
     "title": "The Social network",
     "summary": "On a fall night in 2003, Harvard undergrad
and
programming genius Mark Zuckerberg sits
                    down at his computer and heatedly begins
working (...)",
     "year": 2010,
     "director":   {    "last_name": "Fincher",
                         "first_name": "David"}
}
```

Fig. 7. Anexample of a JSON Object.

IV. Experiments and Results

To evaluate the efficiency of our approach, we implemented it with java and jena api. Our application allows building an OWL2 ontology from a JSON document. We use as an illustrative example the JSON object presented in Fig. 4. We obtained as a result the ontology presented in Fig. 10.

First, we used the inheritance identification using key labels to extract sub classes but we didn't find any results. Next, we applied our inheritance detection algorithm using decision trees. We were able to identify four sub classes. Finally, we applied our transformation rules to generate the final ontology. In total, we obtained seven data properties and three object properties.

```
<owl:DatatypeProperty rdf:about=http://www.JsonToOnto.com#hasfirst_name_director>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
<rdfs:domain rdf:resource="http://www.JsonToOnto.com#director"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://www.JsonToOnto.com#haslast_name_ director">
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
<rdfs:domain rdf:resource="http://www.JsonToOnto.com#director"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://www.JsonToOnto.com#hasyear_Class1">
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
<rdfs:domain rdf:resource="http://www.JsonToOnto.com#Class1"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://www.JsonToOnto.com#hassummary_Classe1">
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
<rdfs:domain rdf:resource="http://www.JsonToOnto.com#Class1"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://www.JsonToOnto.com#hastitle_Class1">
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
<rdfs:domain rdf:resource="http://www.JsonToOnto.com#Class1"/>
</owl:DatatypeProperty>
```

Fig. 8. The Result OWL from the JSON Object in Fig. 7 by Applying Rule 2.

```
    <owl:ObjectProperty rdf:about="http://www.JsonToOnto.com#hasdirector">
    <rdfs:range rdf:resource="http://www.JsonToOnto.com#director"/>
    <rdfs:domain rdf:resource="http://www.JsonToOnto.com#Class1"/>
</owl:ObjectProperty>
```

Fig. 9. Theresult OWL from the JSON Object in Fig. 7 by Applying Rule 3.

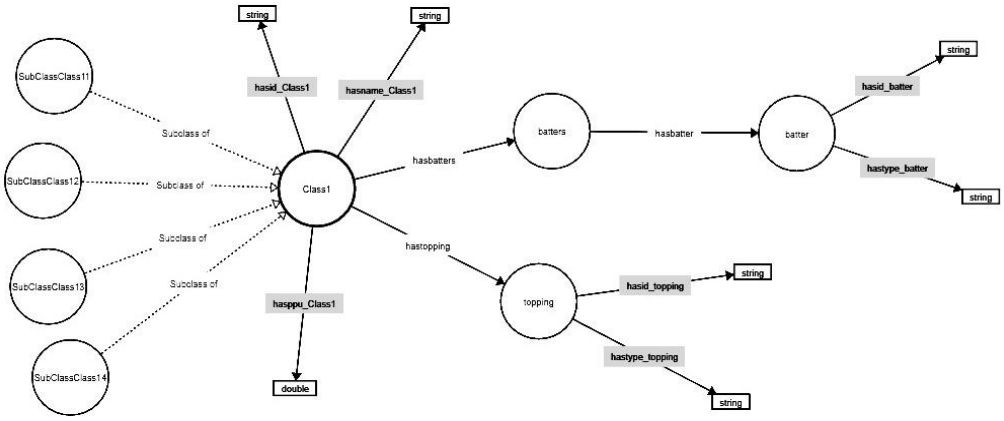

Fig. 10. The Result OWL 2 Ontology.

We also conducted our approach on five different data sets. The sets are all various JSON objects with different sizes. The results of our experiments are presented in the technical report available at http://apps.ensam-umi.ac.ma/jsontoonto/.

V. Conclusion and Perspectives

In this paper, we have proposed an automatic approach to extract an OWL2 ontology from a JSON document. We were able to extract deep taxonomies from JSON documents using the key labels and data mining techniques. We also were able to transform the JSON document elements to OWL components by proposing a set of transformation rules. In order to give the obtained ontology classes a meaningful name, we add the intervention of a domain expert to semantically validate the generated schema and rename the ontology classes.

We developed a prototype that implements our proposal and we tested it using various JSON documents. The obtained results were satisfactory. However, this is still the first version of our prototype and there is still plenty of room for improvement. We are currently working on converting larger JSON documents.

As we mentioned before, we used a single JSON document to extract an OWL2 ontology. We intend to improve our approach to handle multiple documents as an information source.

Acknowledgment

This work is supported by the project “Knowledge Management for Development in the Context of OCP Group (KM4Dev – OCP Group)” granted by OCP Group, Morocco.

References

[1] Crockford, D. JSON: The fat-free alternative to XML. Proc. OfXML. Vol. 2006.

[2] T.Gruber, “Ontology”, in the Encyclopedia of Database Systems, Ling Liu and M. Tamer Özsu (Eds), Springer-Verlag, 2009.

[3] B.Swartout, R. Patil, K. Knight and T. Russ, "Towards Distributed Use of Large-Scale Ontologies" Spring Symposium Series on Ontological Engineering. Stanford University, CA. 1997, Pages:138-148.

[4] Y. Yao, H. Liu, J. Yi, H. Chen, X. Zhao & X. Ma, “An automatic semantic extraction method for web data interchange”,The 6th International Conference on CSIT. ISBN: 987-1-4799-3999-2, 2014.

[5] RDF 1.1 Turle, W3C Recommendation–25 February 2014: https://www.w3.org/TR/2014/REC-turtle-20140225/.

[6] T.J. Lampoltshammer and T. Heinstracher, “Ontology Evaluation with Protégé using OWLET”, Infocommunications Journal, 2014.

[7] G.H. Baek, S.K. Kim & K.H. Ahn, “Framework for automatically construct ontology knowledge base from semi-structured datasets”, the 10th International Conference for Internet Technology and Secured Transactions (ICITST), 2015.

[8] M. Seidel, M. Krug, F. Burian & M. Gaedke,“KESeDa: Knowledge extraction from heterogeneous semi-structured data sources”, In Proceedings of the 12th International Conference on Semantic Systems, pages 129-136, 2016.

[9] JSON-LD 1.0, A JSON-based Serialization for Linked Data. W3C Recommendation – 16 January 2014: https://www.w3.org/TR/json-ld/.

[10] A. Nayak, A. Poriya & D. Poojary, “Types of NoSQL Databses and its comparison with Relational Databases”, International Journal of Applied Information Systems (IJAIS)–ISSN: 2249-0868 Volume 5– No.4, March 2013.

[11] O. Curé, M Lamolle & C Le Duc,”Ontology Based Data Integration Over Document and Column Family Oriented NOSQL”. SSWS 2011.

[12] O. Curé and R. Jeansoulin,“An fca-based solution for ontology mediation”. JCSE, 3(2):90-108, 2009.

[13] A. Zimmermann and C. Le Duc,“Reasoning with a network of aligned ontologies”.In Proceedings of the 2nd International Conference on Web Reasoning and RuleSystems (ICWRRS), pages 43-57, 2008.

[14] H. Abbes, S. Boukettaya & F. Gargouri, “Learning Ontology from Big Data through MongoDB Database”. In: the 12th ACS/IEEE InternationalConference on Computer Systems and Applications (AICCSA); 2015.

[15] J. Ross Quinlan, “C4.5. Programs for Machine Learning”, Elsevier Inc, 1993.

[16] T.M. Cover & J.A. Thomas, “Elements of Information Theory”, Second Edition, 2006.