Journal of Theoretical and Applied Information Technology
15th August 2018. Vol.96. No 15
© 2005 – ongoing JATIT & LLS
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AUTOMATIC DATA MODELING TRANSFORMATION APPROACH OF NOSQL DOCUMENT AND COLUMN STORES TO RDF

[1]ABDELJALIL BOUMLIK, [2]NASSIMA SOUSSI, [3]MOHAMED BAHAJ
Department of Mathematics and Computer Science, Faculty of Sciences and Technology, University Hassan 1, Settat, Morocco
E-mail : [1]boumlik.abdeljalil@gmail.com, [2]nassima.soussi@gmail.com, [3]mohamedbahaj@gmail.com

ABSTRACT

In this paper, we have proposed a mapping system that makes heterogeneous NoSQL data available in common machine-readable format. In fact, we aim to virtualize the data stored in different NoSQL databases with a specific focus on document and column oriented databases types (considered as the most used ones) using Resource Description Framework (RDF) in order to contribute in the interoperability between applications that exchange data and process it as machine-understandable information, especially in the web application domain and offering more opportunities for novel services and applications that have such needs. In addition, our approach is very useful to carry out some operations that are not currently supported by NoSQL database systems, also to unify their heterogeneous data models. The proposed algorithms are based on a set of procedure and methods that we execute at each stage depend on the input file and as per the mapping rules that we already define. The obtained results via our application were encouraged and reflect exactly what has been expected and specified.

Keywords: *NoSQL-to-RDF, Column-Oriented Database, Document-Oriented Database, Unified NoSQL Database, Interoperability*

1. INTRODUCTION

In recent years, the volume of data in the web has a widely expansion with a dizzying speed due to the rapid growth of social media, mobile applications, web technologies, scientific data, economic data and others generating a significant amount of unstructured data every minutes. This problem has led to the emergence of numerous technologies offering more robust database management systems dedicated specially for Big Data such as NoSQL and Semantic web worlds representing the subject of this paper.

The Semantic Web [1] is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation; it is largely recognized by its ability to exchange data over the web relying on RDF format. This Framework allows representing data with a set of RDF triples; each triple contains a subject, predicate and object. It is promoted by Open Data and Linked Open Data thanks to its way of connecting data by linking objects representing by unique identifiers. Besides of Semantic Web, NoSQL [2] have experienced a widely expansion due to its high ability to manage Big Data [3]. It is a no relational database management system dedicated to manage heterogeneous and unstructured data. This systems avoids join operations and supports dynamic schemas design offering to web users a high flexibility and scalability. Since this two systems aim to make the big data processing smarter, therefore establishing a bidirectional connection between them is a very relevant need. In addition, a considerable number of RDF data management problems require the intervention of Big Data infrastructure. All these raisons have motivated us to write the current paper, which is, to the best of our knowledge, the first work proposing a detailed mapping solution for NoSQL-to-RDF direction so as to carry out some operations not supported by NoSQL systems and to unify the heterogeneous NoSQL databases model.

In our proposed approach, we have chosen to use the RDF model for representing the NoSQL data because this model is considered as a standard for exchanging information on the web in addition to its characteristic of being interpretable and exploitable by machines, so as to contribute in

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

interoperability between applications and minimize human intervention. In this regard, we have established a mapping system that convert NoSQL data which is supposed to be exploitable by machines to RDF format, with a specific focus on column and document-oriented databases considered as the most used ones.

Regarding document-oriented databases, we will focus on these handling data encapsulated in a JSON format. In fact, this type of databases takes advantages of JSON flexibility that allow users to manipulate the data without defining its schema in advance, the same is an important plus-value that will facilitate the rapid integration of data from different sources, therefore, the elimination of an enormous part of schema design problems and challenges that requires considerable efforts and knowledge to be addressed properly.

The remainder of this paper is organized as follows. Section II exposes a brief description of the most recent approaches and discuss related works by providding a comparison study. Section III present a theory background of the different types of NoSQL databases and Semantic Web data model (RDF). Section IV presents our main contribution starting with the description of our proposed solution by detailing all procedures used in our model transformation algorithm of document and column oriented databases to RDF, then we give some examples for each database type. Section V, describes the application that we have developed so as to validate and improve our proposed solution. Finally, section VII concludes this work and suggests some future extensions of this approach.

2. RELATED WORKS

Recently, significant and considerable efforts have been invested in the definition of tools which allow transforming several kinds of data sources into RDF format.

Regarding relational databases, the RDB-to-RDF [4,5] mapping is considered recently as a very pertinent research topic. In fact, various mapping methods are defined (Triplify, Virtuoso, eD2R, D2RQ, R3M, etc.…). in addition to the main used one R2RML, and several implementations already exist [6].

Likewise, various solutions exist to map CSV and spreadsheets data to RDF such as XLWrap [7], Mapping Master [8], Tarql [9] and Vertere [10]. The XLWrap mapping language is based on an RDF-centric mapping approach that allows mapping of information stored in different spreadsheets to arbitrary RDF graphs independent from the representation model. Mapping Master or M2 language is based on an extension of the OWL Manchester Syntax; this method converts data from spreadsheets into the Web Ontology Language. Tarql is a command-line tool for converting CSV files to RDF using SPARQL 1.1 syntax. Vertere is a spreadsheet-to-RDF conversion tool based on templating mechanism.

Concerning XML-to-RDF, several mapping tools have been developed in order to ensure this conversion. The XSPARQL [11] is a mapping language which combines XQuery and SPARQL so as to query XML and RDF data using the same framework and transform data from one format into the other. On the other hand, there are numerous mapping tools based on XSLT technology such as a generic transformation of XML data into RDF named AstroGrid-D [12], XML Scissor-lift [13] solution uses Schematron's instructions to validate the mapping rules tests, and Krextor library [14]. Similarly, different method exist based on XPath, we quote as example the Tripliser [15] and XML2RDF [16],[17].

On the other hand, and in order to deal with the interoperability challenges related to syntax and semantic heterogeneity between NoSQL databases, several approaches have been proposed to address the adaptation challenge of these databases into RDF stores, so as to unify their data model and make it machine-readable, however, all existing approaches have the same and common weakness since they propose limited set of transformation rules without considering the required high level of abstraction to ensure this interoperability.

We expose recent methods: xR2RML in papers [35], [36] and RDF-ization in paper [37] that generally consist of transforming the raw data from different sources to RDF, including NoSQL ones. Although, these approaches do not propose a strong transformation algorithm to ensure this migration, but they illustrate just some features supported by their solution only. In addition, no one of them supports column-oriented databases.

The authors in [18] propose a systematic attempt at characterizing and comparing NoSQL stores for RDF processing; they study just their mapping applicability but they don't trait the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

interoperability between these two heterogeneous systems.

The work described in [19] performs hybrid query processing by integrating both SQL and NoSQL data into a common data format (RDF); during this process, the authors have developed a very basic mapping algorithm for transforming NoSQL data (MongoDB) to RDF.

From the above analysis, it has been apparent that most recent contributions in this field contains limited rules and applied just to NoSQL document-oriented databases, with a specific focus on ones having JSON format as a data model, while they don't define a clear algorithms or correspondence rules to ensure a better semantic preservation during data exchange compared to our work. In this paper, we present a novel solution in terms of scope, methodology and techniques to generate Linked Data based on column-oriented databases in addition to document-oriented ones by establishing and advanced, automatic and well-arranged conceptual correspondence algorithms designed with a global functionalities and components that transform each database model to its equivalent RDF components characterized by a simple and powerful structure of triples (Subject-Predicate-Object) very similar to human language (Subject-Verb-Object), in one single system called NoSQL2RDF.

3. THEORY BACKGROUD

3.1 NoSQL Databases

As we mentioned previously, NoSQL is a no relational database management system dedicated to manage heterogeneous and unstructured data [20]. From our investigation, we can categorize NoSQL database in two major areas that we present below:

- Key/Value Store Databases or 'the big hash table': Amazon S3 (Dynamo) [21] , RIAK [22].
- Schema-Less, which comes in multiple flavors and differenct formats as below: Document-Oriented Databases (CouchDB [23] and MongoDB [24]), Column-Oriented Databases (Big Table [25] and Cassandra [26]) and Graph Databases (Neo4j [27]).

In this paper, we are interested specifically in Document-Oriented and Column-Oriented Databases, that we present in the next paragraphs.

3.1.1 Document-oriented stores

Document-Oriented databases are most popular among other NoSQL types for deeper nesting structures that offers high performance, availability and automatic scaling. This type of database encapsulates "key-value" concept, while key is an ID of the document and the value is the document itself, which can be retrieved by an ID. Data is stored as a collection of documents *D* equivalent to records in relational databases.

There are various formats that can be relies on a structure or metadata for document-oriented databases such as XML, YAML, JSON and BSON, but most of the time, it relies on JSON (JavaScript Object Notation), hence there is no restriction to use same schema format, which means, each document can contain similar or dissimilar data structure. In this work, we made the choice to use document-oriented stores with JSON/BSON data model [28] as illustrated in figure 1, due to their utilization and reputation around the globe such as CouchDB and MongoDB to provide general solutions based on popular tools.

Each document D_i contains a set of Key/Value pairs ($P_i^j = (K_i^j, V_i^j)$), $D_i = \{(K_i^1,V_i^1), (K_i^2,V_i^2), ..., (K_i^m,V_i^m)\}$ with $j \in [1,m]$; such value may have a simple (Number, String) or complex (Array, embedded document) type. Formally, these documents are grouped into collections $C = \{D_1, D_2,..., D_n\}$.

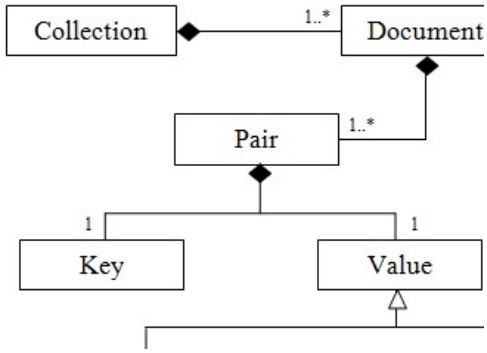

Figure 1: Meta-model Of Document-Oriented Stores Based On JSON/BSON.

3.1.2 Column-oriented stores

The Column-Oriented Database has a special structure dedicated to accommodating many columns (up to several millions) for each line. The main benefit of using columnar databases is that you can quickly access a large amount of data, also it is ease of scaling because data is stored in columns, that's why they are mainly used for keeping non-volatile, long-living information and

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

in scaling use cases. The stored data is based on the sort order of the column family.

This type of NoSQL database offers a high scalability in data storage and flexible schema due to the number of columns that can change from one row to another. We can consider that a column exists if contains a value. At first blush, the Column-Oriented Database looks very like relational database, but the concept is completely different. The model of column-oriented database, as illustrated in figure 2, is composed of a set of tables; each table contains in its turn a set of rows $T = \{R_1, R_2, ..., R_n\}$. Each row can be represented as $R_i = (ID_i, (CF_i^1, CF_i^2 \ldots, CF_i^m))$ with ID_i is a row id and CF_i^j is a column family of the row R_i. A column family can contain a numerous columns $CF_i^j = \{(C_i^{j1}, v_i^{j1}), (C_i^{j2}, v_i^{j2}), \ldots, (C_i^{jp}, v_i^{jp})\}$.

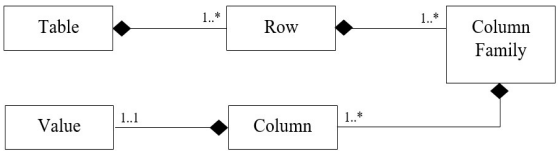

Figure 2: Meta-model Of Column-Oriented Stores

3.2 Semantic Web: RDF Stores

The RDF is a graph model designed to formally describe web resources and metadata to enable automatic processing of such descriptions. Developed by the W3C, RDF is the basic language of the Semantic Web widely used. Several common serialization formats are in use, we quote as example: Turtle [29], N-Triples [30], N-Quads [31], JSON-LD [32], Notation 3 (N3) [33], RDF/XML [34]. In this study, we are only interested about RDF/XML serialization syntax that we described in figure 3.

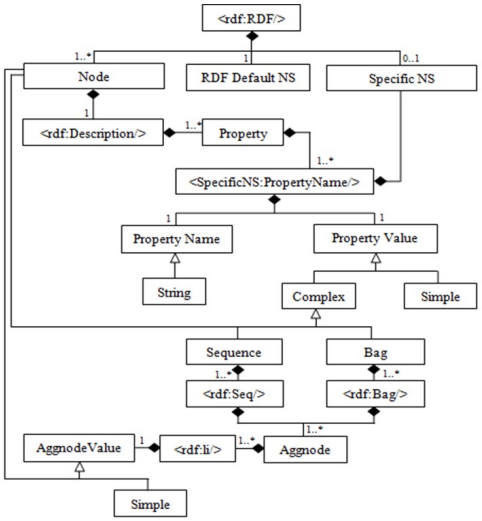

Figure 3: Meta-model of RDF Store's Concept

4. FRAMEWORK DESCRIPTION

This section presents important phases in the proposed framework, that leads to the realization of our goal using the architecture illustrated in Figure 4, it is consisting of layers, which are NoSQL databases as input, Data models, NoSQL2RDF converter, and finally the generation of RDF triple stores. We describe below the details of each phase.

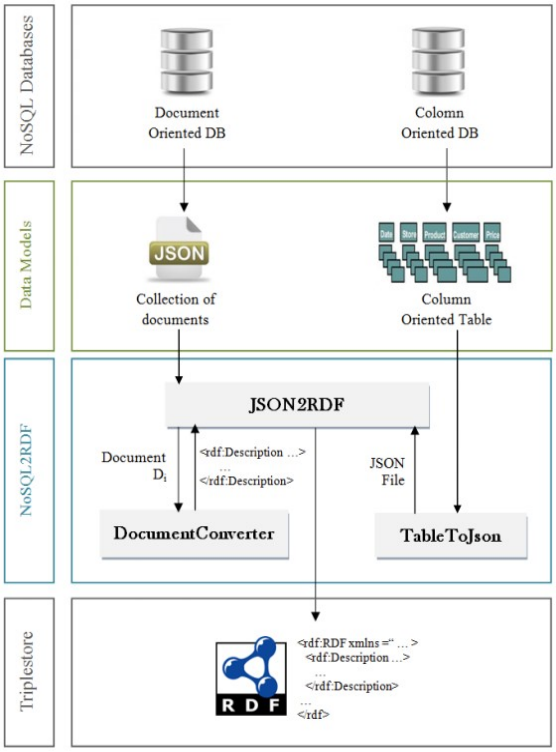

Figure 4: Global Architecture of NoSQL To RDF Mapping System

- **Phase 1: Filtering and Extracting of data**

In this phase from our mapping system, we have started by applying a filtering operation of the targeted NoSQL data that needs to be extracted and exploited by machines. After depth analysis, we chose to use JSON format to represent data for all document-oriented databases like MongoDB. From other hand, we used the column-oriented tables to represent data of oriented column databases.

- **Phase 2: Mapping process**

This phase is referring to the Data model layer, which can distinguish between the type of databases (document or column oriented) that will be processed and moved to *NoSQL2RDF* layer which

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

makes the reel conversion based on the input database type.

- For NoSQL document-oriented databases, the generated JSON file is retrieved using the *JSON2RDF* component that decompose this collection of documents and transform them to a list of data in order to facilitate access to each one and convert it to its equivalent clause in RDF model via *DocumentConverter* component. Finally, the main component *JSON2RDF* concatenates the previous results and generates the equivalent RDF file.
- For NoSQL column-oriented databases, and in order to avoid reinventing the wheel, we challenged ourselves to process this type of NoSQL databases, by adding to our mapping layer NoSQL2RDF the new component *Table2Json* which is in charge of converting the column-oriented database model, represented as an oriented column table to a JSON format based on a set of matching rules that express the semantic correspondence between these two models. Thereafter, we continue the same previous conversion process using the core component *JsonToRDF* to generate the equivalent RDF file.

5. PROPOSED FRAMEWORK

In this section, we present briefly our contribution and the solution by detailing all procedures used in our proposed model of transformation algorithms for both document and column oriented databases to RDF stores (global schema in figure 4), and then we give some examples for each database type to simplify the logic.

4.1 Document-Oriented Stores Model to RDF Transformation

At this stage, we will detail our conversion algorithm for document-oriented database model (JSON/BSON) to RDF by describing the principals procedures (*JsonToRdf* and *DocumentConverter*) used to achieve this aim.

4.1.1 Procedure JsonToRdf

The first procedure "JsonToRdf" takes a JSON File as input and generate at the end an equivalent RDF file. Firstly, it decomposes the JSON file to extract all documents encapsulated in this input file, and stores them in a list for further use, then, it glances through this list of documents and convert each one to its equivalent in RDF via *DocumentConverter* sub procedure that will be presented in the next paragraph.

```
Input : jsonFile
Output : RDFfile
Begin
  RDFfile = ‘’
  nsLabel = ‘ns’; nsValue = jsonFile.getPath()
  ReadJsonFile(jsonfile)
  List Documents = jsonFile.getDocuments()
  If (Documents.isEmpty() = False)
  Then
      RDFfile += '<rdf:RDF xmlns
="http://www.w3.org/1999/02/22-rdf-syntax-ns#" '
      RDFfile += nsLabel+'="'+nsValue+'">'
      For (i =0; i< Documents.getLenght(); i++ )
          RDFfile +=
DocumentConverter(Documents[i], nsLabel)
      End for
  End if
  RDFfile += '</rdf:RDF>'
  Return RDFfile
End
```

4.1.2 Sub procedure DocumentConverter

This sub procedure concists to glances through document's children and convert them to an RDF equivalent components by adopting the rules defined in table 1 which represents the semantic correspondance of each component between JSON/BSON and RDF.

Table 1: Correspondence Rules Between JSON and RDF.

JSON component	RDF component
JSON Path	Xmlns:NS (Namespace)
Document	<rdf:Description/>
Id	IdAttribute (rdf:about)
Key (type = Simple)	<NS:KeyName/>
Key (type = List) For Disordered List	<NS:KeyName > <NS:Bag> <rdf:li>Data</rdf:li> </NS:Bag> </NS:KeyName >
Key (type = List) For Ordered List	<NS:KeyName ><NS:Seq> <rdf:li>Data</rdf:li></NS:Seq> </NS:KeyName >
Key (type = Embedded Document)	<NS:KeyName><NS:Bag> <rdf:li><rdf:Description/></rdf:li> </NS:Bag></NS:KeyName>

```
Input : JsonDocument, nsLabel
Output : TextFile
Begin
TextFile = ' '
ReadJsonFile(Jsonfile)
If (JsonDocument.isEmpty() = False) Then
  TextFile += '<rdf:description'
  List Children = JsonDocument.getChildren()
  For (i =0; i< Children.getLenght(); i++)
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
    Key   = Children[i].getKey()
    Value = Children[i].getValue()
If (Key = '_id') Then
      TextFile += 'rdf:about = "'+Value+'" >'
Else
      TextFile += '>'
End if
If (Value.getType() = simpleValue) Then
 TxtFile += '<'+nsLabel+':'Key+'>'+Value+'</'+
nsLabel+ ':'+Key+'>'
ElseIf (Value.getType() = Array()) Then
      TextFile+ ='<'+nsLabel+':'Key+'>'
      TextFile+ ='<rdf:Bag>'
For ( j=0; j< Value.getLenght(); j++ )
TextFile+ ='<rdf:li>'+Value[j]+'</rdf:li>'
End For
TextFile +='</rdf:Bag>'
TextFile += '</'+nsLabel+':'+Key+'>'
ElseIf(Value.getType() = ComplexeType)Then
If (Value.getLenght() = 1) Then
TextFile += DocumentConverter(Value, nsLabel);
Else
TextFile += '<rdf:Bag>'
          For(k =0; k<Value.getLenght(); k++)
               TextFile += '<rdf:li>'
               TextFile +=
DocumentConverter(Value[k], nsLabel)
               TextFile += '<rdf:li>'
          End For
          TextFile += '</rdf:Bag>'
End If
End If
End For
TextFile += '</rdf:description>'
End If
Return TextFile
End
```

4.1.3 Technical implementation

To test and validate our solution, we used the below case study in which we consider a JSON file illustrated in the figure 5 containing two documents encapsulating a set of PhD students information with different types such as name, age, status, experiences and diplomas having simple type (Number or String), List and Embedded document type respectively. Starting from this JSON file, we have established the equivalent RDF file (represented in figure 6) based on the conversion algorithm defined previously.

```
{ _id : "http://fsts.com/phdStudent /100",
name : "Abdeljalil Boumlik",
age : 28,
status: "phd student",
university : "FST",
experiences : [ "Senior Programmer", "Support
Engineer", "System Analyst" ],
diplomas : [
{diploma : "Bac" , mention : "pretty good"},
{diploma : "Licence" , mention : "good"},
{diploma : "Master" , mention : "good"} ]
}
{ _id : "http://fsts.com/phdStudent/101",
name : "Nassima Soussi",
age : 26,
status: "phd student",
university : "FST",
experiences : [ "DBA", "Computer Engineer", "Web
Designer"],
diplomas : [ { diploma : "Bac" , mention : "good" } ,
{ diploma : "Engineering", mention : "good" } ] }
```

Figure 5: Collection of Documents

```
<rdf:RDF xmlns ="http://www.w3.org/1999/02/22-
rdf-syntax-ns#"xmlns:s="http:://fsts.com/doctorate"
>
<rdf:Description rdf:about=
"http:://fsts.com/phdStudentId/100" >
<s:name> Abdeljalil Boumlik </s:name>
<s:age> 28 </s:age>
<s:status> phd student </s:status>
<s:university> FST </s:university>
<s:experiences> <rdf:Bag>
<rdf:li> Senior Programmer </rdf:li>
<rdf:li> Support Engineer </rdf:li>
<rdf:li> System Analyst </rdf:li>
</rdf:Bag> </s:experiences>
<s:diplomas> <rdf:Bag>
<rdf:li><rdf:Description>
<s:diploma> Bac </diploma>
<s:mention> pretty good </mention>
</rdf:Description></rdf:li>
<rdf:li><rdf:Description>
<s:diploma> Licence </diploma>
<s:mention> good </mention>
</rdf:Description></rdf:li>
<rdf:li><rdf:Description>
<s:diploma> Master </diploma>
<s:mention> good </mention>
</rdf:Description></rdf:li>
</rdf:Bag></s:diplomas></rdf:Description>
<rdf:Description rdf:about=
"http:://fsts.com/phdStudentId/101" >
<s:name> Nassima Soussi </s:name>
<s:age> 26 </s:age>
<s:status> phd student </s:status>
<s:university> FST </s:university>
<s:experiences>
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
<rdf:Bag><rdf:li> DBA </rdf:li>
<rdf:li> Computer Engineer </rdf:li>
<rdf:li> Web Designer </rdf:li></rdf:Bag>
</s:experiences>
<s:diplomas><rdf:Bag>
<rdf: li><rdf:Description>
<s:diploma> Bac </diploma>
<s:mention> good </mention>
</rdf:Description></rdf:li>
<rdf:li>
<rdf:Description>
<s:diploma> Engineering </diploma>
<s:mention> good </mention>
</rdf:Description>
</rdf:li></rdf:Bag>
</s:diplomas>
</rdf:Description>
</rdf:RDF>
```

Figure 6: Generated Equivalent RDF File

4.2 Column-Oriented Stores Model to RDF Transformation

This paragraph describes our conversion algorithm of Column oriented database model to RDF based on the previous algorithm. Firstly, we convert the column-oriented database model to JSON model via the sub procedure *TableToJson* by respecting the correspondence rules defined in table 2, and then we continue the conversion process using the previous algorithm for transforming JSON files to RDF.

Table 2: Correspondence Rules Between Column-Oriented DB And JSON.

Column-oriented DB Component	JSON Component
Record	Document
ID	_id
Column Family	Key
Column Family's Columns	Value of the type Document or Embedded Document

We will detail our conversion algorithm for column-oriented database to RDF by describing the principals procedures (*ColumnToRDF* and *TableToJson*) used to achieve this aim.

4.2.1 Procedure ColumnToRdf

This procedure takes as input the column-oriented table in order to convert it to a JSON format using the sub-procedure *TableToJson* and continue the same previous process to obtain the RDF equivalent file.

```
Input : Column-Oriented Table (T[N,M])
Output : RDF file (rdfFile)
Begin
    jsonFile = TableToJson(T)
    rdfFile = JsonToRdf(jsonFile)
    Return rdfFile
End
```

4.2.2 Sub procedure TableToJson

This sub procedure aim to convert the column-oriented table to a json file by adopting the following algorithm:

```
Input : Column Oriented Table (T[N,M])
Output : JsonFile
Begin
JsonFile= '', ColumnLabel = ''
List SubColumns = Null
For( i=0; i<N; i++ )
  If (T[i,M].isEmpty() = False) then  //if the record isn't empty
JsonFile += '{';   //the beginning of the document
    For( j=0; j<M; j++ )
      If (T[i,j].getType().isId() = True) then
JsonFile += '_id :' + T[i,j] + ','
      Else if (T[i,j].isEmpty() = False) then
ColumnLabel = T[i,j].getColumnLabel()
SubColumns = T[i,j].getSubColumns()
JsonFile += ColumnLabel + ':' + '{'
        For( k=0; k< SubColumns.size(); k++ )
SubColLabel = SubColumns[k].getLabel()
SubColValue = SubColumns[k].getValue()
JsonFile += SubColLabel + ':'
      If (SubColValue.isString() = True) then
JsonFile += ' " ' + SubColValue + ' " ' + ','
      Else if (SubColValue.isNumber() = True) then
JsonFile +=  SubColValue + ','
          Else if (SubColValue.isList() = True) then
JsonFile +=  '['
          For ( l=0; l<SubColValue.size(); l++)
JsonFile += SubColValue[l]
          If (l< SubColValue.size()-1) then
JsonFile += ','
          End if
        End for
JsonFile +=  ']'
      End if
    End for
JsonFile += '}' + ','
    End if
  End for
JsonFile += '}'  //end of document
  End if
 End for
Return JsonFile
End
```

4.2.3 Technical implementation

The example described below presents a column-oriented database table (table 3) named Person

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

(T^{Person}) containing four columns families CF_i^{Name}, $CF_i^{Address}$ and CF_i^{Job} defined as follow:
CF_{100}^{Name}= {(LastName, "Boumlik")}
CF_{100}^{Job}= {(Place, "Casablanca"), (Company, "CompX")}
CF_{101}^{Name}= {(FirstName, "Nassima"), (LastName, "Soussi")}
$CF_{101}^{Address}$ = {(City, "Khouribga")}
CF_{101}^{Job} = {(Profil, "Engineer")}

Starting from the table T^{Person}, we have generated the equivalent JSON file represented in figure 7 based on the table 2 that contains the correspondence rules between column-oriented database table and JSON file. Then, we have constructed easily the equivalent RDF file represented in figure 8 from this JSON file based on our previous algorithms.

Table 3: Example of Column-Oriented Database Table (T^{Person}).

ID	Name	Address	Job
…	…	…	…
100	**LastName** Boumlik		**Place** Casablanca **Company** CompX
101	**FirstName** Nassima **LastName** Soussi	**City** Khouribga	**Profil** Engineer
…	…	…	…

```
{ _id : 100,
Name : {  LastName: "Boumlik"},
Job : { Place: "Casablanca", Company: "CompX"},
}
{ _id : 101,
Name : {  FirstName: "Nassima", LastName: "Soussi"},
Address : { City: "Khouribga" },
Job : { Profil: "Engineer"},
}
```

Figure 7: The Equivalent JSON File of T^{Person} Table

```
<rdf:RDF xmlns="http://www.w3.org/1999/02/22-rdf-
syntax-ns#"s="http:://fsts.com/Person" >
…
<rdf:Description rdf:about= 100 >
<s:Name><rdf:Description>
<s:LastName> Boumlik </s:LastName>
</rdf:Description></s:Name>
<s:Job><rdf:Description>
<s:Place> Casablanca </s:Place>
<s:Company> CompX </s:Company>
</rdf:Description></s:Job>
</rdf:Description>
<rdf:Description rdf:about= 101>
<s:Name><rdf:Description>
<s:FirstName> Nassima </s:FirstName>
<s:LastName> Soussi </s:LastName>
</rdf:Description></s:Name>
<s:Address><rdf:Description>
<s:City> Khouribga </s:City>
</rdf:Description></s:Address>
<s:Job><rdf:Description>
<s:Profil> Engineer </s:Profil>
</rdf:Description></s:Job>
</rdf:Description>
...
</rdf:RDF>
```

Figure 8: RDF Equivalent File Of Table Person

6. IMPLIMNETATION

To validate our approach, we have developed an application, as illustrated in figure 9, figure 10 and figure 11, with java programming language aiming to establish a system ensuring a mapping from NoSQL databases models (Document and Column oriented databases) to RDF format. The experiments were carried out on the PC with 2.4 GHz Core i5 CPU and MS Windows Seven Titan.

During our development, we decide to make a single application that manage the mapping for both databases models to RDF, for that we tried to find a common points between these models to base our application on it, then, we found out that both database models can be exported to JSON format directly or by using some open source tools which allow us to generate the JSON files from existing databases, like 'sstable2json' that was provided by apache community to users.

This tool was giving us the possibility to have a structured JSON file that respond to our needs from usability and formats perspectives. The same processing rules and stages will be applied on JSON file generated by this tool, for column-oriented database or the one that we create for document-oriented database, below we present the steps that we follow before generating the RDF equivalent format via the application:

Journal of Theoretical and Applied Information Technology
15th August 2018. Vol.96. No 15
© 2005 – ongoing JATIT & LLS
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Step 1: consists to upload the JSON file generated from a MongoDB database or sstable2json tool available in Cassandra distribution. This JSON file contains a collection of documents supposed to be represented as a Liked Data via RDF model (figure 9).

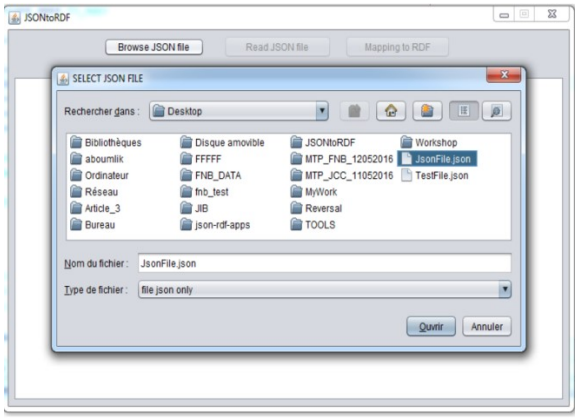

Figure 9: Upload JSON File To The Application

Step 2: at this level, our system takes as input the generated JSON file (figure 10) in order to transform each document in this collection, or even each pair key/value, to its semantically equivalent RDF triples by adopting a set of matching rules (Table 1) to facilitate the mapping between these two heterogeneous models (JSON and RDF). The Concatenation of previous intermediate results leads to generate the equivalent RDF model (Figure 11).

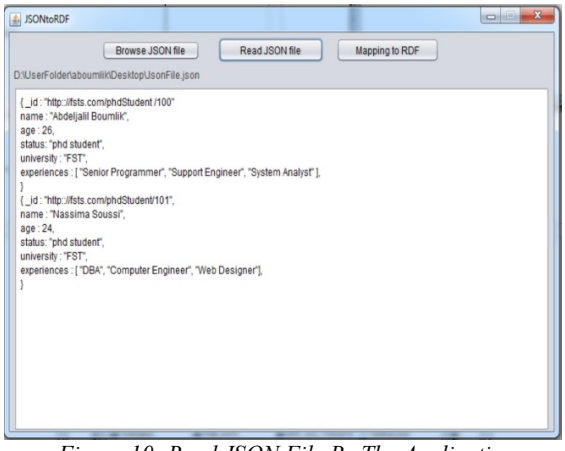

Figure 10: Read JSON File By The Application

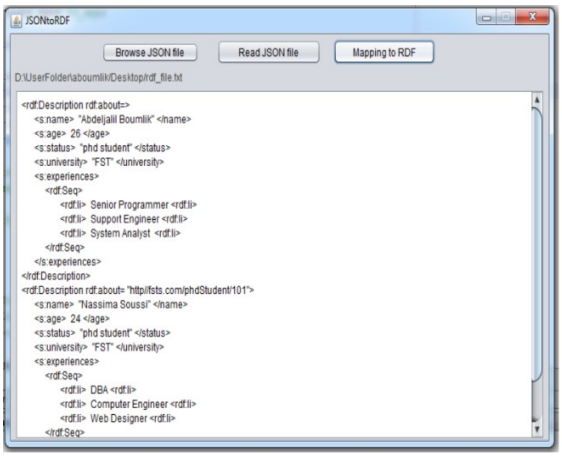

Figure 11: Result Of Processing JSON File To RDF

7. RESULTS AND ANALYSIS

Our approach contributes in the interoperability between the two major pillars of managing a large amount of data: NoSQL and Semantic web by proposing an efficient mapping system that makes NoSQL data (with a specific focus on column and document-oriented database types) available in RDF stores in order to be exploitable semantically.

Regarding the NoSQL document-oriented database's model, we have interested specially to JSON as a data tree that provide support for a simple writing in text format, and natively interpretable unlike XML which needs parsing and sometimes DOM/XSLT to access its structure and content; in addition the NoSQL databases having JSON as a data model is considered as the most used such as MongoDB and CouchDB. In fact, we have started with the decomposition of JSON file in order to extract the different documents encapsulated in this collection, and then we have converted all pairs for each document to their semantic equivalent RDF components based on the correspondence rules defined previously in table 1.

Concerning the NoSQL column-oriented database, we have converted its data model to JSON format by respecting the correspondence rules defined in table 2, and then we continue the conversion process using the previous algorithm for transforming JSON file to RDF.

Our solution is very helpful for organizations working entirely with NoSQL databases and aiming to expose some data to be exploitable semantically without spending so much in the training of their users in the semantic technologies new for them, in addition, this solution

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

is beneficial for unifying the heterogeneous NoSQL databases model.

8. CONCLUSION

The principal aim of this paper is to conceive an interoperability context between the two main pillars of managing large amount of data: NoSQL and Semantic web that offer to users a better profit and maximum exploitation of web resources. Through this work, we have contributed in realizing this goal by exposing NoSQL data which is supposed to be exploitable by machines into RDF format in order to facilitate and improve the interoperability between web applications without human intervention. In fact, we have elaborating an efficient conversion model algorithm which transform each model of NoSQL databases treated in our work (Document and Column oriented database) to its equivalent RDF format. In addition, we have developed a portable java application that consume JSON file and makes the transformation to RDF model.

Our subsequent work will be focused on enhancing and reinforcing our approach by automating the first step of filtering and extracting data before starting the mapping process. We aim also to support more NoSQL database types (Key-Value and Graph Databases).

REFRENCES:

[1] Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. Scientific american, 284(5), 28-37.

[2] Strauch, C., Sites, U. L. S., & Kriha, W. (2011). NoSQL databases. Lecture Notes, Stuttgart Media University, 20.

[3] Sri, P. A., & Anusha, M. (2016). Big Data-Survey. Indonesian Journal of Electrical Engineering and Informatics (IJEEI), 4(1), 74-80.

[4] Michel, F., Montagnat, J., & Faron-Zucker, C. (2014). A survey of RDB to RDF translation approaches and tools (Doctoral dissertation, I3S).

[5] Hert, M., Reif, G., & Gall, H. C. (2011, September). A comparison of RDB-to-RDF mapping languages. In Proceedings of the 7th International Conference on Semantic Systems (pp. 25-32). ACM.

[6] Villazón-Terrazas, B. & Hausenblas, M. (2012). RDB2RDF Implementation Report. https://www.w3.org/TR/rdb2rdf-implementations/. Accessed October 8, 2016.

[7] Langegger, A., & Wöß, W. (2009, October). XLWrap–querying and integrating arbitrary spreadsheets with SPARQL. In International Semantic Web Conference (pp. 359-374). Springer Berlin Heidelberg.

[8] O'connor, M. J., Halaschek-Wiener, C., & Musen, M. A. (2010, November). Mapping master: a flexible approach for mapping spreadsheets to OWL. InInternational Semantic Web Conference (pp. 194-208). Springer Berlin Heidelberg.

[9] Cyganiak, R. (2015). Tarql. https://github.com/cygri/tarql. Accessed August 15, 2016.

[10] Möller, K. (2012). Vertere-RDF. https://github.com/knudmoeller/Vertere-RDF. Accessed August 15, 2016.

[11] Bischof, S., Decker, S., Krennwallner, T., Lopes, N., & Polleres, A. (2012). Mapping between RDF and XML with XSPARQL. Journal on Data Semantics,1(3), 147-185.

[12] Breitling, F. (2009). A standard transformation from XML to RDF via XSLT.Astronomische Nachrichten, 330(7), 755-760.

[13] Fennell, P. (2014). Schematron-more useful than you'd thought. XML LONDON 2014.

[14] Lange, C. (2011, July). Krextor-an extensible framework for contributing content math to the Web of Data. In International Conference on Intelligent Computer Mathematics (pp. 304-306). Springer Berlin Heidelberg.

[15] Rogers, D. (2011). Tripliser. http://daverog.github.io/tripliser/. Accessed August 15, 2016.

[16] Huang, J. Y., Lange, C., & Auer, S. (2015, September). Streaming Transformation of XML to RDF using XPath-based Mappings. In Proceedings of the 11th International Conference on Semantic Systems (pp. 129-136). ACM.

[17] Huang, J. Y. (2016). XML2RDF. https://github.com/allen501pc/XML2RDF. Accessed October 8, 2016.

[18] Cudré-Mauroux, P., Enchev, I., Fundatureanu, S., Groth, P., Haque, A., Harth, A., ... & Wylot, M. (2013, October). Nosql databases for rdf: an empirical evaluation. In International Semantic Web Conference (pp. 310-325). Springer Berlin Heidelberg.

[19] Michel, F., Djimenou, L., Faron-Zucker, C., & Montagnat, J. (2015, October). Translation of relational and non-relational databases into RDF with xR2RML. In 11th International Confenrence on Web Information Systems and Technologies (WEBIST'15) (pp. 443-454).

[20] Pratiba, D., Deepak, D., & Shwetha, S. (2016). Comparative Analysis of NOSQL Databases. Indonesian Journal of Electrical Engineering and Computer Science, 3(3), 601-606.

[21] Voegels, W. (2012). Amazon DynamoDB—a fast and scalable NoSQL database service designed for Internet-scale applications.

[22] Docs, R. (2015). Riak documentation.

[23] Anderson, J. C., Lehnardt, J., & Slater, N. (2010). CouchDB: the definitive guide. " O'Reilly Media, Inc."

[24] Chodorow, K. (2013). MongoDB: the definitive guide. " O'Reilly Media, Inc.".

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[25] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., ... & Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS), 26(2), 4.

[26] Hewitt, E. (2010). Cassandra: the definitive guide. " O'Reilly Media, Inc.".

[27] N. Technologies, (2013). The Neo4j Manual v1.9.M04.

[28] Yusof, M. K. (2017). Efficiency of JSON for Data Retrieval in Big Data. Indonesian Journal of Electrical Engineering and Computer Science, 7(1), 250-262.

[29] Beckett, D., Berners-Lee, T., Prud'hommeaux, E., & Carothers, G. (2014). RDF 1.1 Turtle–Terse RDF Triple Language. W3C Recommendation. World Wide Web Consortium (Feb 2014), available at http://www. w3. org/TR/turtle.

[30] Beckett, D. (2014). RDF 1.1 N-Triples: A line-based syntax for an RDF graph.W3C Recommendation, http://www. w3. org/TR/n-triples, 25.

[31] Carothers, G. (2014). RDF 1.1 N-Quads: A line-based syntax for RDF datasets.W3C Recommendation.

[32] Davis, I., Steiner, T., & Hors, A. L. (2013). RDF 1.1 JSON Alternate Serialization (RDF/JSON). W3C Recommendation.

[33] Berners-Lee, T., & Connolly, D. (2011). Notation3 (N3): a readable RDF syntax. W3C Team Submission. World Wide Web Consortium. Beschikbaar op http://www. w3. org/TeamSubmission/n3.

[34] Gandon, F., & Schreiber, G. (2014). RDF 1.1 XML Syntax: W3C Recommendation 25 February 2014. World Wide Web Consortium. http://www. w3. org/TR/rdf-syntax-grammar. Accessed October 8, 2016.

[35] Michel, F., Djimenou, L., Faron-Zucker, C., & Montagnat, J. (2015, October). Translation of relational and non-relational databases into RDF with xR2RML. In 11th International Confenrence on Web Information Systems and Technologies (WEBIST'15).

[36] Michel, F., Djimenou, L., Zucker, C. F., & Montagnat, J. (2017). xR2RML: Relational and non-relational databases to RDF mapping language (Doctoral dissertation, CNRS).

[37] Gualán, R., Freire, R., Tello, A., Espinoza, M., & Saquicela, V. (2017). Automatic RDF-ization of big data semi-structured datasets. Maskana, 7(Supl.), 117-127.