Journal of Theoretical and Applied Information Technology
29th February 2016. Vol.84. No.3

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

QUERY SYSTEM OF GENERATED ONTOLOGICAL MODELS FROM TRADITIONAL DATA SOURCES

[1]WIDAD JAKJOUD, [2]MOHAMED BAHAJ
[1,2]Univ Hassan 1, Laboratory LITEN, 26000 Settat, Morocco
E-mail: [1]jakjoudw@gmail.com, [2]mohamedbahaj@gmail.com

ABSTRACT

The cooperation of ontologies, as sources of knowledge, and traditional data sources are used to defeat the heterogeneity of information systems in terms of integration of information. Also, this cooperation allows the exploitation of classic web resources by inference agents. In this paper, we focus on ontologies generated from traditional data sources. Our goal is to query an ontological model without having to populate ontologies with instances from data source. The proposed system adds an intermediate level of abstraction between the ontological model and the data source schemas, this level can generate partially and temporarily the data in XML format. The system also provides a SPARQL-XQUERY mapping which rewrites any SPARQL query at XQUERY query in order to be executed on already generated data.

Keywords: *SPARQL, XQUERY, mapping, model, ontology, data source*

1. INTRODUCTION

Data sources, being the physical implementations of information systems are heterogeneous because they are designed by different communities and for different circumstances. This heterogeneity is manifested in terms of data storage formats (XML, Relational, Object Oriented,), languages of queries (XQUERY, SQL, OQL ...), access protocols (HTTP ...) and schema's formalisms of data.

The interest of creating ontologies from data sources is particularly due to the need for integration of information from multiple heterogeneous information systems and other hand to the main goal of the Semantic Web to extend the resources of classic web by establishing semantic links between data to make it handled by agents of inferences.

To use a data source with an ontology, we can create instances and relationships using data from the data source and store them in a suitable format for the ontology (Semantic Database [1] Text Files , ...) which requires a change of data structure and storage formalism.

This solution is intuitive which affects the two levels (schema and data), but it does not represent any optimization at the persistence level because the data will be stored in their source of origin and in the proper format to ontology. Thus, for each data schema and storage format (data source) the specific algorithms for generating instances and relations of the ontology must be defined. Another drawback is that the instances must be regenerated after each update at the level of data source. We cite the work of [2] and the three transitional systems from RDB to OWL: DataMaster1 [3], and RDBToOnto3 Kaon2 with these formalisms for mapping D2RQ [4] and R 2 O [5].

Another solution would be keeping the data in the data source and define, for each data source, the mapping rules between its query language and SPARQL the language of the ontological model [6]: Each ontology query is implicitly mapped in the appropriate query language of data source. Already at this stage, the redundancy in data storage is avoided [7]. This solution seems intuitive but is not simple to implement or optimize because of several types of mapping which have to be defined. In addition, it is difficult to exploit all the data models.

2. THE STUDY CONTEXT

This study concerns the ontologies generated from data sources. In [8] and [9] we have proposed a method based on model engineering for the automatic generation of ontological model from a data source (RDB, OODB, XML): The reverse engineering of a data source generates a software model (schema of data) that will undergo transformations to generate an ontological model. In order to interrogate the ontological resulting model, we propose, through this paper, a system which maps queries of generated ontological models to queries of data sources schemas.

The ontology model is generated from one of the data sources (RDB, OODB, or XML), the idea is to interrogate the ontology without populate this one in order to eliminate the double storage of the same data but in different formalisms. This solution

Journal of Theoretical and Applied Information Technology
29th February 2016. Vol.84. No.3

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

defines one mapping between SPARQL and XQUERY.

At the time of execution of XQUERY request [10], we generate partially the data of the data source in XML format. This solution facilitates the optimization of the system because it doesn't need to define multiple mapping between multiple query languages of data: a unique mapping is defined since data will be loaded into a unique format.

3. THE PROPOSITION

Each SPARQL request would be interpreted and mapped at a XQUERY request whatever the format of the data source. The interpretation determine, even partially, the sets of data involved by the request, only these data will be loaded as XML collections on which the XQUERY request will be executed:

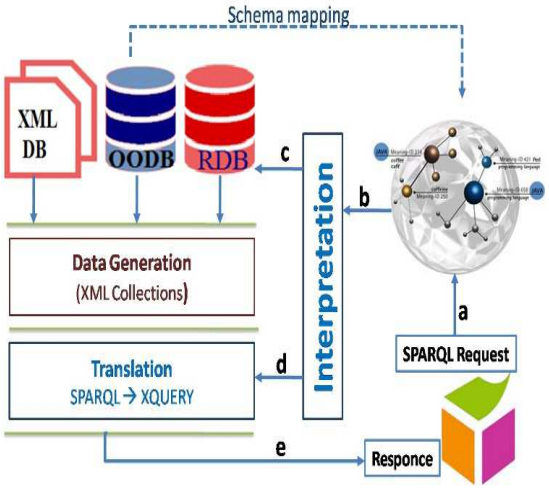

Figure 1 Schematic Illustration Of The Approach

Every query on the ontological model (a) is a SPARQL request which will be interpreted (b) to determinate partially the concerned set of data (c). At the translation level (d), the SPARQL request will be translate to a XQUERY request.

The step (c) is not performed if the original data source is an XML-DB database: Data in data generation level will be generated as XML collections just time to execute query. This will prevent the duplication of data in data sources and ontology.

The result of the query XQUERY will be deployed in XML format.

2.1 Translation Level

2.1.1 SPARQL request

The SPARQL Language (Simple Protocol and RDF Query Language) is a W3C standard for querying triplets of a given ontology. Its grammar can be summarized as:

```
Request  :: = PREFIX [name spaces definitions]
              SELECT [Selection Variables]
              WHERE {Group Graph Pattern}
              OREDER BY [variables]
```

The WHERE clause is composed of Group Graph Patterns (GGP):

```
GGP      ::= '{' TriplesBloc ?
             [(GraphPatternNotTriples|Filter)'.'?Triples
             Bloc]* '}'
TriplesBloc::=TriplesSameSubject('.'TriplesBloc?)
?
Triples ::=IRIs(International Resources Identifiers)
                | litterals (strings in quotes)
                | selection variables prefixed by ?
                | white nodes.
GraphPatternNotTriple    ::=OptionalGraphPattern
          |GroupOrUnionGraphPattern
          |GraphGraphPattern.
OptionalGraphPattern::='OPTIONAL'GroupGraph
Pattern
GraphGraphPattern::= 'GRAPH' varOr|R|ref GGP
GroupOrUnionGraphPattern::=GGP('UNION'GGP
)*
FILTER ::='FILTER' Constraint
```

Initially, we will be limited to two kind of SPARQL queries: simple query and query with FILTER.

2.1.2 Interpretation of the SPARQL query:

The interpretation of the SPARQL query identifies the classes involved in the request. This will enable the Data Generation Module to generate partial data as XML collections.

2.1.3 Query Translation:SPARQLtoXQUERY Module

Several studies have already discussed the mapping of SPARQL to XQUERY. We quote the SPARQL2XQUERY framework [11] a W3C standard which offers: processing modules RDF to XML (RDF-XML transformation), XMLSchema to OWL (XS2OWL). Translation module (Query Translation) which transforms each SPARQL query in XQUERY query that will be resolved from the XML data. We also quote XSPARQL [12] a hybrid scripting language that combines multiple languages including XQUERY, SPARQL and SQL.

Journal of Theoretical and Applied Information Technology
29th February 2016. Vol.84. No.3

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

The problem with these frameworks is that they adopt their own mapping of data schemas in order to translate the SPARQL queries by XQUERY.

In our case, the ontology model is automatically generated from the schema of the data source. To use the existing solutions, schema of data and the model of ontology must be mapped to the formats of these Frameworks!

The module that we propose (SPARQLtoXQUERY) can handles at the moment, two types of queries:

Simple queries:

SELECT [Selection Variables]
WHERE '{' GGP '}'

Queries with filter:

SELECT [Selection Variables]
WHERE '{' GGP
FILTER constraint'}'

We define the latter as follows:

SELECT [Selection Variables]
WHERE '{' GGPF '}'

The proposed module is composed of three sub modules:

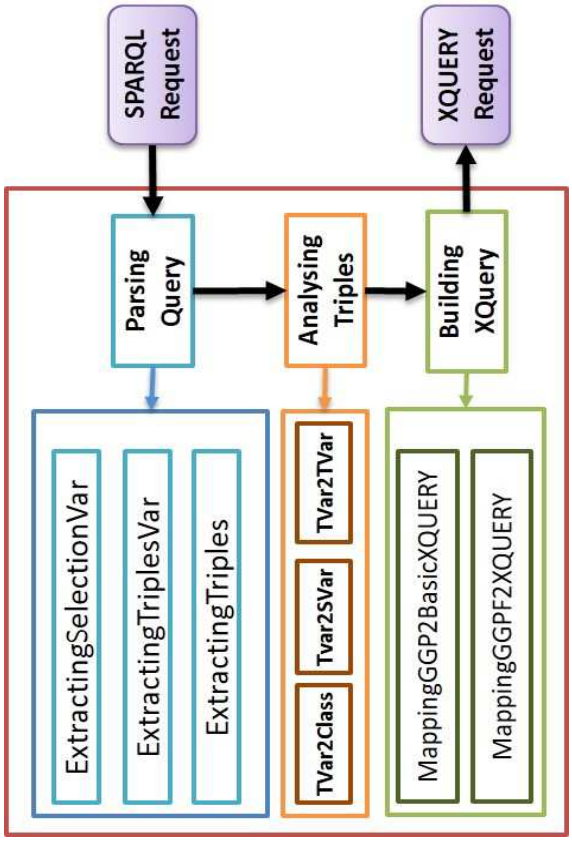

Figure 1: The Mapping Process Of A SPARQL Query

- *SPARQL Query Analysis*: the analysis of the SPARQL query retrieves three lists: a list of selected variables, a list of Triples's variables and a list of triples of block "WHERE".
- *Triples Analysis:* We consider that a triple SPARQL can be in one of three forms: Tvar rdf :type class, TVar dataTypeProperty SVar, and TVar dataTypeProperty TVar
- *Building the query XQUERY:* the building of the final XQUERY query which can be either a simple query or a query with the WHERE clause.

Simple query

For [variable] in doc(….)//RootOfGraph
Let [Selection Variables] :=path
Return [selection variable]

Query with WHERE clause

For [variable] in doc(….)//RootOfGraph
Let [Selection Variables] :=path
Where [constraints]
Return [selection variables]

MappingGGP2BasicXQUERY{

//Mapping Group Graph Pattern to simpleXQUERY
List TRIPLES ← ExtractingTriples()
HashTable Hash←NULL
String XQUERY←NULL
For i ϵ TRIPLES { // Building the paths of the LET clause
If (TVar2Class(i) == true)
Hash←(TVar,Class)
Else if(TVar2SVar(i) == true)
if (TVar ∈ Hash)
Hash←(SVar,TVar+"/"+DataProperty)
Else
Hash←(SVar,Hash(TVar)+"/"+DataProperty)
Endif
Else if(TVar2TVar(i) ==true)
If($TVar_1$ ∈ Hash)
Hash←($TVar_2$,$TVar_1$+"/"+DataProperty)
Else
Hash←($TVar_2$,Hash($TVar_1$)+"/"+DataProperty)
EndIf
EndIf
EndIf
}
//Building the FOR and LET clauses
String varFor=NULL ;
String var=NULL ;
String Path =NULL;
//Get the node that is the root graph
XQUERY← "FOR" + varFor + "/"+root
For(i ϵ ExtractingSelectionVar()) {
var←"?" + ExtractingSelectionVar(i)
Path←varFor+"/"

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

```
   For(j ϵ Hash){
      If(Hash(j) == var)
            Path← Path + Hash(j)
      EndIf
   }
   var ← var.Replace("?","$")
 XQUERY←XQUERY+"\n"+"LET"+var+":="+Path
  }
}
MappingGGPF2XQUERY{
String XQUERY←MappingGGP2BasicXQUERY()
String FilterClause ←
ExtractFilterClause(SPARQLQUERY)
FilterClause ← FilterClause.Replace("?","$")
XQUERY ←XQUERY + "\n" + FilterClause
}
```

2.2 Data Generation Level:

Interpreting the SPARQL query identifies the classes of the ontological model involved in the interrogation. As the ontological model is automatically generated from the UML model [8] [9], it can be inferred entities of the data source (classes or tables) necessary to the execution of the XQUERY query.

In the case of a relational data source, several studies have focused on the migration of RDB to XML, we quote [13] [14] [15] [16] and [17]. Most of the proposals were based on hypotheses, simplify the migration process, but with gaps [18]. These proposals focus on the mapping schema level rather than data level (RDB to XML Schema or RDB to DTD), and ignore the data migration process.

We opt for an RDB data migration algorithm in XML that is based on the following rules:

Rule 1: the relations, records and relational attribute are converted to three levels of XML elements: An element to represent the relation, an element for the record and an element for the attribute.

Rule 2: For associations (1, n), we use an embedded data representation.

Rule 3: For associations (n, n), it generates the corresponding documents to the relations (not yet generated) that constitute the association and the algorithm will be responsible for calculating the data.

The following example illustrate this generation:

- ***Customer***

idCustomer	Name	Adress
12	C1	A1
15	C3	A5

- ***Order***

idOr	Date	idCust
250	12/02/2015	12
286	25/03/2015	15
301	28/03/2015	12

- ***Product***

IdP	Description	Price
Prd1	Screen	500.0
Prd2	Printer	1200.0
Prd3	HardDisk	250.0

- ***OrderProduct***

IdOr	IdP	Quantity
250	Prd1	2
250	Prd3	1
286	Prd1	3
301	Prd2	2

Figure 2 : RDB example

The generated XML collection of this RDB:

```
<Costumers>      //level 1 : the relation
      <Customer>      //level 2 :the records
            <idCustomer> 12 </idCustomer>
             //livel 3 :the attributs
            <Name> C1 </Name>
            <Adress>  A1 </Adress>
            <Order>
                  <idOr>250</idOr>
                  <Date>12/02/2015</Date>
            </Order>
            <Order>
                  <idOr>301</idOr>
                  <Date>28/03/2015</Date>
            </Order>
      </Customer>
      <Customer>
            <idCustomer>  15 </idCustomer>
            <Name> C3 </Name>
            <Adress>  A5 </Adress>
            <Order>
                  <idOr>286</idOr>
                  <Date>25/03/2015</Date>
            </Order>
      </Customer>
</Customers>
<Products>
      <Product>
            <idP>Prd1</idP>
            <Description>Screen</Description>
            <Price>500.0</Price>
```

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

```
        </product>
        <Product>
                <idP>Prd2</idP>
                <Description>Printer</Description>
                <Price>1200.0</Price>
        </product>
        <Product>
                <idP>Prd3</idP>
            <Description>HardDisk</Description>
                <Price>250.0</Price>
        </product>
</Products>
<OrderProducts>
        <OrderProduct>
                <idOr>250</idP>
                <idP>Prd1</Description>
                <Quantity>2</Quantity>
        </OrderProduct>
        <OrderProduct>
                <idOr>250</idOr>
                <idP>Prd3</idP>
                <Quantity>1</Quantity>
        </OrderProduct>
        <OrderProduct>
                <idOr>286</idOr>
                <idP>Prd1</idP>
                <Quantity>3</Quantity>
        </OrderProduct>
        <OrderProduct>
                <idOr>301</idOr>
                <idP>Prd2</idP>
                <Quantity>2</Quantity>
        </OrderProduct>
</OrderProducts>
```

For an Oriented Object Data source, we keep the same approach:

Rule 1: the classes, objects and properties of the data source will be converted into three levels of XML elements as in the case of relational.

Rule 2: The methods are not mapped.

Rule 3: For associations (1, n) and compositions, we use an embedded data representation.

Rule 4: for associations (n, n) and aggregations, we proceed as in the relational case.

4. IMPLEMENTATION

We consider the schema of the following XML-DB data source

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element id="0" name="dbperson">
<xs:complextype>
<xs:sequence>
        <xs:element id="1" name="person">
        <xs:complextype>
        <xs:sequence>
        <xs:element id="2" name="LName"
type="xs:string">  </xs:element>
        <xs:element id="3" name="FName" type=
"xs:string" maxOccurs="unbounded">
</xs:element>
        <xs:element id="4" name="adress"
maxOccurs="unbounded">
         <xs:complextype>
        <xs:sequence>
                <xs:element id="5"
name="number" type="xs:int"></xs:element>
                <xs:element id="6"
name="street" type="xs:string"></xs:element>
         </xs:sequence>
         </xs:complextype>
         </xs:element>
</xs:sequence>
</xs:complextype>
</xs:element>
</xs:sequence>
</xs:complextype>
</xs:element>
</xs:schema>
```

The generated ontology after the conversion process at the model level expressed in OWL (ontology Web Language) is as follow:

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
ns#"
   xmlns:genOnto="file://D:/genOnto.owl#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
>
 <rdf:Description rdf:about="genOnto:number">
   <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#an
yURI"/>
   <rdfs:domain rdf:resource="genOnto:owl_adress"/>
   <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#Datatype
Property"/>
 </rdf:Description>
 <rdf:Description rdf:about="genOnto:owl_adress">
   <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
 </rdf:Description>
 <rdf:Description rdf:about="genOnto:street">
   <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#an
yURI"/>
   <rdfs:domain rdf:resource="genOnto:owl_adress"/>
```

Journal of Theoretical and Applied Information Technology
29th February 2016. Vol.84. No.3

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

```
  <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#Datatype
Property"/>
 </rdf:Description>
 <rdf:Description rdf:about="genOnto:FName">
  <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#an
yURI"/>
  <rdfs:domain rdf:resource="genOnto:owl_person"/>
  <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#Datatype
Property"/>
 </rdf:Description>
 <rdf:Description rdf:about="genOnto:LName">
  <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#an
yURI"/>
  <rdfs:domain rdf:resource="genOnto:owl_person"/>
  <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#Datatype
Property"/>
 </rdf:Description>
 <rdf:Description rdf:about="genOnto:owl_person">
  <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
 </rdf:Description>
 <rdf:Description rdf:about="file://D:/genOnto.owl#">
  <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#Ontology
"/>
 </rdf:Description>
</rdf:RDF>
```

The system take in input the SPARQL query, with the SPARQL2XQUERY button, this query will be mapped at XQUERY query. The RDB2XMLCOLLECTION button generates the data as XML Collection in an XML-DB data source (Exist-db), the query will be executed by the EXECUTION button.

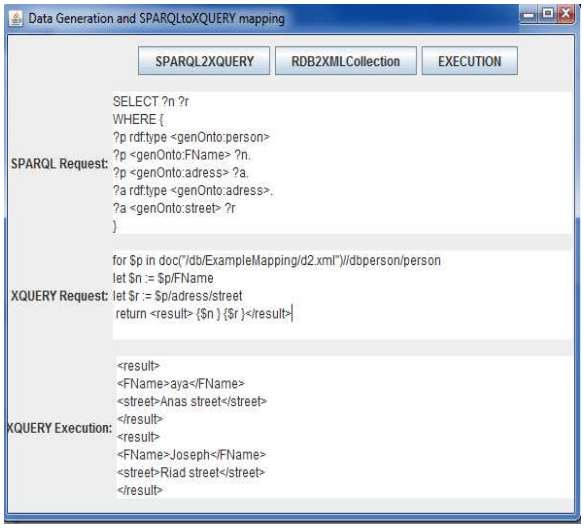

Figure 3 : Data Generation And SPARQL2XQUERY Mapping Prototype

We consider the schema of the following RDB named dbPerson :

person(**idPerson**, FName, LName, age, #idAdd)

idPerson	FName	LName	age	idAdd
1	Aya	Durond	9	A1
2	Joseph	Zalanado	37	A2

address(**idAdd**,street,number)

idAdd	street	number
A1	Anas street	15
A2	Riad street	27

Figure 4 : dbPerson database

The Data Generation Module generates a data source (dbPerson) in XML-DB (db-exist): Then an XML document will be generated in this data source: xml_gen.xml.

```
<dbPerson>
        <person>
                <idPerson>1</idPerson>
                <FName>aya</FName>
                <LNAME>Durond</LNAME>
                <age>9</age>
                <adress>
                        <street>Anas
                street</street>
                        <number>15</number>
                </adress>
        </person>
        <person>
                <idPerson>2</idPerson>
                <FName>joseph</FName>
                <LNAME>Zalanado</LNAME>
                <age>37</age>
                <adress>
                        <street>Riad
                street</street>
                        <number>27</number>
                </adress>
        </person>
</dbPerson>
```

5. CONCLUSION AND PERSPECTIVES:

Our goal is querying an ontological model generated from a traditional data source (RDB, OODB, XML-DB) without having to populate the ontology; the data will remain in the data source.

The system we propose in this paper allows the mapping of any query that interrogates the ontology to interrogate the data in their original storage format. The SPARQL query on the model of ontology will be interpreted to determine the concerned classes, which will be, generate temporarily in XML format, then the query will be

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

mapped at a XQUERY query and executed on already generated data.

The system provides the optimization of data persistence level: data will not be restructured and re-stored in the ontology; they will be temporarily generated in XML format.

The system also provides the optimization of the mapping between the languages of interrogation of data and SPARQL: indeed, the mapping SPARQL to XQUERY rewrite each SPARQL query at XQUERY that will be executed on the data already generated in XML.

In perspective, we aim to extend the mapping algorithm SPARQLtoXQUERY to include other cases of SPARQL queries. We also aim to optimize the data generation algorithm.

RÉFÉRENCES:

[1] Mbaiossoum, B., Bellatreche, L., Jean, S., & Baron, M. (2013). Comparaison théorique et empirique de systèmes de bases de données sémantiques. Ingénierie des Systèmes d'Information, 18(3), 39-63.

[2] Krivine, S., Nobécourt, J., Soualmia, L. F., Cerbah, F., & Duclos, C. (2009). Construction automatique d'ontologies à partir d'une base de données relationnelles: application au médicament dans le domaine de la pharmacovigilance. In Actes d'IC (pp. 73-84)

[3] de Laborda, C. P., & Conrad, S. (2005, January). Relational. OWL: a data and schema representation format based on OWL. In Proceedings of the 2nd Asia-Pacific conference on Conceptual modelling-Volume 43 (pp. 89-96). Australian Computer Society, Inc.

[4] Bizer, C. (2003). D2r map-a database to rdf mapping language.

[5] Barrasa Rodríguez, J., Corcho, Ó., & Gómez-Pérez, A. (2004). R2O, an extensible and semantically based database-to-ontology mapping language.

[6] http://www.w3.org/TR/rdf-sparql-query/

[7] Diallo, G. (2006). Une Architecture à base d'Ontologies pour la Gestion Unifiées des Données Structurées et non Structurées (Doctoral dissertation, Université Joseph-Fourier-Grenoble I).

[8] Jakjoud, W., Bahaj, M., & Bakkas, J. (2015, March). Generic transposition from data source toward ontology. In Electrical and Information Technologies (ICEIT), 2015 International Conference on (pp. 205-211). IEEE.

[9] Jakjoud, W., Bahaj, M., & Bakkas, J. Automatic Generation of Ontology from Data Source Directed by Meta Models.

[10] http://www.w3.org/TR/xquery/

[11] Bikakis, N., Tsinaraki, C., Stavrakantonakis, I., Gioldasis, N., & Christodoulakis, S. (2013). The SPARQL2XQuery interoperability framework.World Wide Web, 18(2), 403-490.

[12] Lopes, N., Bischof, S., Decker, S., & Polleres, A. (2011, October). On the semantics of heterogeneous querying of relational, XML and RDF data with XSPARQL. In Proceedings of the 15th Portuguese Conference on Artificial Intelligence (EPIA 2011), Lisbon, Portugal.

[13] Fong, J., & Cheung, S. K. (2005). Translating relational schema into XML schema definition with data semantic preservation and XSD graph. Information and software technology, 47(7), 437-462.

[14] Kleiner, C., & Lipeck, U. W. (2001, September). Automatic Generation of XML DTDs from Conceptual Database Schemas. In GI Jahrestagung (1) (pp. 396-405).

[15] Vela, B., & Marcos, E. (2003, June). Extending UML to Represent XML Schemas. In CAiSE Short Paper Proceedings (Vol. 74).

[16] Du, W., Lee, M. L., & Ling, T. W. (2001, December). XML structures for relational data. In Web Information Systems Engineering, 2001. Proceedings of the Second International Conference on (Vol. 1, pp. 151-160). IEEE

[17] Lee, D., Mani, M., Chiu, F., & Chu, W. W. (2002, November). Net & Cot: Translating relational schemas to XML schemas. In ACM CIKM.

[18] Maatuk, A., Ali, A., & Rossiter, N. (2008, January). Relational database migration: A perspective. In Database and Expert Systems Applications (pp. 676-683). Springer Berlin Heidelberg.