Dynamic Transform Method for Ontology DB from Semi-structured Datasets

Gui-hyun Baek, Kee-hong Ahn, Su-kyoung Kim
Hanbat University, Republic of Korea

Abstract

This study is a framework to transform optimized ontology database from semi-structured datasets (such as spreadsheet, JSON, XML, CSV, TSV etc.) based RDM (Relational Data Model). The most important thing in big data analysis ability, is a classification and association between domain according to a context or meaning of sentence. In IoT environment, to increase a semantic interaction between virtual or physical objects, between an objects in different domain has an ability to associate by context or meaning. This ability is impossible using ontology technology based ontology schema model. Most semi-structured dataset has a problem to make a connection or interaction between objects (data) because homonym or synonym of linguistic feature. Therefore, a studies to implement an ontology from existing RDB is in progress, as a result, typically, R2RML was published by W3C RDB2RDF Working Group. But, existing a mapping language can be limited, when RDB is not normalized (such as, 1NF), or when a semi-structured data provided (e.g. spreadsheet, text of key-value type (such as JSON), text of XML type etc.). Therefore, we need a transform method that construct an ontology database from semi-structured datasets for more accurate and speedy. So, we suggest a method that consist of three function block, first a Cell-Value Importer (CVI) and second, a Transformation Table Generator (TTG), and last a Property Expression (PropertyExp) for dynamically importing value at semi-structured data and describing mapping information by intuitive and concise sentence. Lastly, we tested that our design satisfied 10 of all 14 possible features of mapping languages [1] through testing by that transform ontology database from research equipment semi-structured dataset, after we implement prototype of a framework according to JDK Environment.

1. Introduction

Many platforms of Web 2.0 supply an Open API to use a data in those platforms. But, most platform has been designing data model that dependent on specific service through silo domain, and then they has been providing disconnected information based on RDM for which it has been specified relationship between individuals by indirectly connecting primary key and foreign key of entity. Therefore, these information has deficient connectivity. Also, complexity of relationship between objects increase as growing complexity of information, and then query of RDB has been performing complicated task because of this has been performing lot of JOIN operation, in the domain. This being so, development of service required query that is implicit enhanced function has been difficult, but it had been generating various data within various domain such as social media, IoT environment and big data. Additional, exiting platform environment demand lot of cost for reconstructing normalized schema, when service or data model had changed (e.g. change of many-to-many relationship of between entities.)

In order to solve this problems, it continually had been progressing studies about reconstruction existing RDB into metadata of ontologies. If concept hierarchy and relation hierarchy in ontologies exploit to design data model, then reconstructed metadata will redefine standardized naming based on integrated domain, and sharing and reusing of data will facilitate through Linked Data paradigm.

We propose methods that consist of a TTG and CVI, for which the method dynamic transform optimized ontology database from a non-normalized RDB or a semi-structured dataset, and more studied that describe mapping information by intuitive and concise sentence named a Property Expression (PropertyExp). And to prove of a accuracy in our method and algorithm, tested it and we can get a accurate result. Finally, in JDK Environment, we implement prototype of this framework using our method.

2. Related Work

All Many researcher suggested method of converting normalized RDB to RDF recommended R2RML [2] and Direct Mapping [3] about RDB-to-RDF translation in W3C RDB2RDF Working Group. Accordingly, Tools based R2RML or based non-R2RML was surveyed by [1] that motivated to

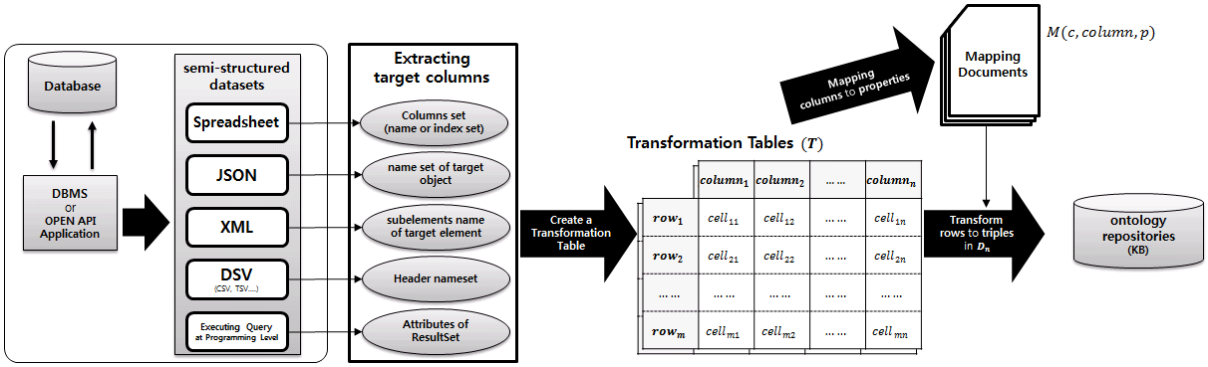

Figure 1. Process for Transforming Ontology DB from Semi-structured Datasets

obtain information in deep web by Linked Data or to integrate heterogeneous information.

On the other hand, method of transforming many kinds of semi-structured dataset to RDF is non-standard, but its tools provide open library, and then, these tools listed "W3C Wiki ConverterToRDF" page. These activities showed that representation RDF from specific semi-structured dataset, yet these tools wasn't been standard level. We research that method transform optimized ontology database according to semantic web from much used operating data among various semi-structured data. This paper use definition of ontology model defined in [4], and use relation algebra for operating TT (Transformation Table) and mapping information. But most method not satisfied above problem, nowadays.

3. Proposed Method

We can see Fig. 1 that process of broad flow which is dynamic transformation ontology database in four part divided. Exceeding semi-structured data -spreadsheet, delimiter-separated values(DSV) - such as, CSV, TSV-, result set of executing SQL from non-normalized RDB as well as normalized RDB - and a type using Open API (such as, JSON, XML) to RDF triple, we aim optimized constructing ontology KB by following process.

1) Extracted target column set and row set which was columns set(name or index set), name set of target object, sub-element name of target element, header name's set and Attributes of ResultSet from semi-structured dataset, in order to construct transformation table.
2) Construct TT (Transformation Table symbol is T) from semi-structured dataset of each other format by Transformation Table Generator.
3) Describe to mapping information in mapping document by PropertyExp, in order to map a extracted column to properties.
4) Transform ontology database from row set of transformation table by using mapping information in Mapping document that describe in Mapping document.

3.1. Modeling of Figure 1's Process

In order to perform the processes in section 3, this section define model and operators used each process. First, transformation dataset D was defined by following (1).

$$D = (T, M) \tag{1}$$

As above (1), D is transformation dataset for constructing ontology KB from semi-structured dataset and construct set of TT and the M (mapping document) about this set. M is set of mapping information. Mapping information contain pair of column and mapping description. T in (1) was defined by following (2)-(4).

$$T(column_1, column_2, \ldots, column_n) = \{t_i\} \quad (0 \le i < |t|) \tag{2}$$

$$t_i = \{row_1, row_2, \ldots\ldots, row_m\} \tag{3}$$

$$row = (value_1, value_2, \ldots, value_n) \tag{4}$$

In (2) and (3), T is set of TT t_i that had element of columns. Also, T is same concept from relation of RDB, use relation algebra. As above (4), rows of t_i corresponds to columns that belong to itself of t_i. Actually, this rows contains cell that had value.

We define following (5)-(8) that cited by [5] about ontology model definition.

$$\alpha_c : \forall_\alpha \forall_c (\alpha_c \in R \wedge c \in C) \text{ and } \iota_R(\alpha_c) \subseteq \iota_C(c) \times c \tag{5}$$

$$p : p \in (R \cup A) \tag{6}$$

In (5), a α_c means relation identifier between instance of class and this class. In the α_c, c is this class of element in set C, C is set of class that defined in ontology, and this called concept identifiers. In (6), p is element within set of union of R and A. R is set of object property, and this called relation identifiers. C is set of class that defined in ontology, and this called attribute identifiers.

For constructing optimized ontology KB, following (7), (8) extends definition of ontology KB in [4]. The (7) defines graph of class that called class graph. Class graph is unit that stored instances of specific class and relations of these instance. This regard single space as KB defined in [4]. And then the (8) redefines the pairs of the classes in ontology and the class graphs defined (7). Notations of (7) was described by following.

$$cg = (I,\ \iota_R, \iota_A) \tag{7}$$

$$KB(c,\ cg) = \{(c_i,\ cg_i)\}, 0 \le i \le |C| \tag{8}$$

- I : set of instance of class of specific class graph
- a function ι_R : for all r∈R, this function get a set of ordered pair of instance within subset of set I and other instance, which these can connect r, and call a relation instantiation.
- a function ι_A : for all a∈A, this function get a set of ordered pair of instance within subset of set I and specific datatype, which these can connect a, and call an attribute instantiation.

3.2. Transforming Algorithm

The Figure 2 describes algorithm of function TRANSFORM for which construct ontology DB using those model and definition defined in section 4.

$TRANSFORM: D \rightarrow KB$

function $TRANSFORM(D_n)$
  $KB := $ ***create*** *empty* KB;
  $T := D.T$;
  $M := D.M$;

  for (**int** i: = 0; i < $SIZE(T)$; i + +)
  {
    $t := T_i$;
    for (int j : = 0; j < $SIZE(t)$; j + +)
    {
      $row := t$[j];
      for (int $k :=$ 0; $k <$ $SIZE(M)$; $k + +$)
      {
        $m :=$ $M[k]$;
        $column :=$ $m.column$;
        $c :=$ $m.c$;
        $p :=$ $m.p$;
        $cg := \sigma_{c=c}\ (\pi_{cg}\ ((KB))$

        if (cg = null) {
          cg := ***create*** *empty* cg;
          add cg to $KB(c,\ cg)$
        }

        if ($p = \alpha_f$) {
          add $\pi^T_{column}(row)$ **to** $cg.\iota_C(c)$;
        }

      else {
          $a = \pi^T_{TYPE(m,\ c)}(row)$;
          if ($p \in A$) {
            add $(a, \pi^T_{column}(row))$ **to** $cg.\iota_A(p)$;
          }
          else {
            add $(a, \pi^T_{column}(row))$ **to** $cg.\iota_R(p)$;
          }
        }
      }
    }
  }
  output KB
end

Figure 1. Definition of Transform Algorithm

In order to perform the processes in section 3, this section define function TYPE in Figure 2 is algorithm for which get column of TT that corresponded relation between class of ontology and generated instance in specific set of mapping information, show following Figure 3.

$TYPE : (M,\ c) \rightarrow column$ about α_c

function $\boldsymbol{TYPE(M,\ c_p)}$
  output $\boldsymbol{\pi^T_{column}\ (\sigma^T_{c=c_p\ AND\ p=\alpha_{c_p}}(M))}$
end

Figure 2. Definition of *TYPE* Algorithm

4. Design and Implement of Method

4.1. TTG (Transformation Table Generator)

The TTG generates TT for constructing ontology DB from semi-structured datasets of each other format. As Fig. 1, transformation table construct column set and row set. Each row of row set construct cell that corresponds to each column of column set. Cell exist value that corresponds to specified column from the row. The following described the methods that are exploited for generating TT to semi-structured datasets of each other format.

- Exploiting XPath can generate transformation table to XML format.
- Exploiting JSON Path can generate transformation table to JSON format.
- Exploiting names or index of header can generate separated value format, such as CSV, TSV.
- Exploiting name or index of column can generate transformation table to s**preadsheet format**.

Consequentially, after defining the schema of TT for semi-structured dataset, this algorithm is defined in Figure 2, which can perform when describe mapping information by PropertyExp that will be described in section 5.

4.2. CVI (Cell-Value Importer)

Some values in specific row within TT for non-normalized RDB and semi-structured dataset are non-atomic data or raw data that required processing data through to find rule of value, immediately TT was generated by TTG, this TT can't is impossible to construct an ontology KB. Therefore, in order to solve above it, we design that CVI is in charge of importing some value of cell in specified row of transformation table from programing level, and it can customize a procedure for processing non-atomic data or raw data that come up above. Thus, application based this framework enable to convert

Figure 3. Realization of CVI by Java Interface

RDF triple from this data that had finite rule, by extending CVI. Following Figure 4 and Table 1 is shown to implement a Java interface CVImporter for CVI.

```
{
  'datasetPath' : 'path'
  'numberingColumn' : 'column of ID about   TT'
  'mapping' : [
        {
            'column' : 'Column name',
            'required' : '(true)|(false)'
            'property' : 'propertyExp'
        },
        ....
    ]
}
```

Figure 4. Prototype of M using JSON Format

The Java interface CVImporter basically implement abstract method for function of importing value - such as, *string*, *integer*, *long*, *double*, *Boolean* , *date string*, *time string*, *date-time string*, etc. - is contained by specific cell within TT or for function of determining what value in cell is blank. In particular, the abstract method *customizedImport* enable that it is stored to the ontology KB to directly implement procedure of converting a RDF triples from various data by developer according to the need.

4.3. Mapping CVI (Cell-Value Importer)

The *M* (Mapping document) describes a column of TT generated by TTG and Mapping information, as (1). TT generated by TTG has common columns. In (5), we detailed prototype of M that used JSON format. The 'datasetPath' item is defined location of semi-structured dataset. The 'numberingColumn' item is assigned column can be ID of TT. Also, this item is in charge of importing value while value in cell within sequent row that corresponded to assigned column is null, in order to generate TT from semi-structured dataset, "mapping" item is defined these mapping information has "column", "required", and "property" item, "column" item is column of TT. The "required" item determine that can be allowed null in cell within rows corresponding to column. If a "required" item in specific this mapping information has "true", cell within rows corresponding to column doesn't allowed to contain null. The "property" item is defined PropertyExp that can define element of ontology corresponding to cell.

Table I. Description of Abstract Method in CVI

Abstract Method	Description
getNS	get a namespace of Class or a base URI
getDataModel	get the named graph of class or the default graph
customImporter	implement customized process that transformed RDF triples from variable data
get{datatype}Value	get value about {datatype} in cell of row correspond this column
isBlank	get that value in cell of row correspond this column is blank.
getDateFormat	get format for date
setDateTimeFormat	set format for date-time
getColumnIndex	get index of this column
setColumnIndex	set index of this column
getColumnName	get name of this column
setColumnName	set name of this column

({*class_of_subject*}, "&id"): convert values in cell within row correspond to column to a instances correspond to class of ontology
({*class_of_subject*}, {*property*}, {*class_of_object*}): generate triples that connected {property} between converted instance from ({class_of_subject}, "&id") and other converted instance from ({class_of_object}, "&id"}
({*class_of_subject*}, {*property*}, {*datatype*}): generate triples that connected {property} between converted instance from ({class_of_subject}, "&id") and generated literal of {datatype} from values in cell within row correspond to column
({*class_of_subject*}, {*property*}, {*language_tag*}): generate triples that connected {property} between converted instance from ({class_of_subject}, "&id") and generated plain literal with {language_tag} from values in cell within row correspond to column

Figure 6. Definition of Syntax of PropertyExp

▸ ***propertyExp*** : (*unitExp*,){*n* − 1}*unitExp*

▸ ***unitExp*** : *{qname_of_s_class}* (("&id"~?) | (("#"*{property}*) (("#"*{qname_of_o_class}*) | "~" | ("^^"*{datatype}*) | "@"*{language_tag}*?))))

- ***{ qname_of_s_class }*** : a namespace of class graph that corresponded subject of triple
- **"&id"** : meaning that a value of this column correspond to instance of class *{qname_of_s_class}* (rdf:type property)
- **{property}** : a name of property that can correspond to instance of class *{qname_of_s_class}* from value of this column
  - ✓ ***{ qname_of_o_class }*** : a namespace of class graph of class *{qname_of_o_class}* that can connected this column from anther column corresponded to *{ qname_of_s_class}* through *{property}*
  - ✓ **"~"** : namespace of default graph deal with base URL that can connected this column from anther column corresponded to *{qname_of_s_class}* through *{property}*
  - ✓ ***{datatype}*** : meaning literal deal with XMLSchema datatype that can connected this column from anther column corresponded to *{qname_of_s_class}* through *{property}*
  - ✓ ***{language_tag}*** : meaning plain literal deal with language tag that can connected this column from anther column corresponded to *{qname_of_s_class}* through *{property}*

Figure 7. ProeprtyExp description form

Mapper construct ontology KB using M. Mapper request to generate TT about these file from TTG, after this mapper is loaded files within path of semi-structured dataset in "datasetPath" item in M as Figure 5. Subsequently, this mapper imports value of cell within row in TT generated by TTG, corresponding to column in mapping information in M. And then, mapper interprets expression within "property" item in M, according ProeprtyExp description forms, such as Figure 7. Finally, this mapper converts RDF triple from TT, and then constructs ontology KB. This constructed ontology KB combined other ontology KB that was constructed by other mapper.

4.4. Syntax of a PropertyExp

PropertyExp is Non-R2RML, is intuitive and concise expression for mapping description that correspond column of transformation table to properties of ontology schema. This is design for which one of column correspond to many properties of ontology. Following Figure 6 describe syntax and its elements for PropertyExp. Among elements of PorpertyExp, *{qname_of_s_class}*, *{property}*, *{qname_of_o_class}* and *{datatype}* is described QName (Qualified Name).

Items of Figure 7 are ProeprtyExp description forms using converted method that convert a value in cell within row in TT to RDF triples.

5. Optimization of Ontology DB

5.1. Configuration of Ontology DB Environment

This section describes method for distributing graph that is storing a converted triples and is contained by ontology KB. Ontology KB construct instance that is materialized class defined on ontology model. At this moment, a named graph of triples about instance that is materialized class is graph of selection by ontology engineer. This method exists drawback that selecting named graph when create a SPARQL statement by developer is ambiguous and non-unitive. In order to solve above it, triples of instance must be regularly and unitivestore specified a named graph when construct an ontology KB. We were able to solve this problem by using that following simple rules of domains defined by Time Berners-Lee and a PropertyExp:

> row-to-resource : each row of a table is a resource, i.e. an individual which class is represented by the table. The resource URI is formed using the primary key: "namespace/database/table/primaryKey"
> or "namespace/database/table#primaryKey"

Above this rule is that all resources of RDF concatenate in front of namespace when convert data contained RDB to RDF .i.e.:

"*namespace/database/table*"
"*namespace/database/table#*"

First, we define namespace of class which a string is concatenated *{qname_of_s_class}* or *{qname _of _o_class}* in PropertyExp described in all of M and "/" or "#". And then, if the namespace of class concatenate all instance of class, all instance have prefix as namespace of class. Also, such instances can be concatenated prefix corresponding to this namespace of class. We define prefix of class such as this prefix, and define graph of class which IRI of named graph assigned this prefix. Consequentially, when TT convert RDF triples corresponding to M, it can be distinguished a local name of all instance from other classes.

5.2. Ontology DB Rule Processing and Searching

This section describes two algorithms, one is for optimization structure of Ontology DB repository, another is searching instance quickly. The Figure 8 and Figure 9 introduce the algorithm.

Figures 10 and 11 are algorithms that composed RDF repository to satisfy sub-section 3.1. This algorithm loads the schema graph, and then composes the datasets, and then loading instance data files, or resort instance data.

(*PI-1*) : The number of D that composing R, if n=1, it defines SR(Simple Repository) having single data set.
(*PI-2*) : The number of D that composing R, if n?1, it defines CR (Composite Repository)
(PI-3) : CR searches schema graph one times, and searches selecting D. Or it can search Ds as linear, therefore it can store datasets distributing in same schema

Figure 8. Define PR-I Step

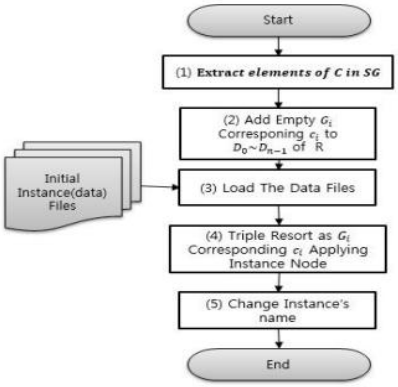

Figure 9. Preprocessing Rule Algorithm

(*PII-1*) : Classes(ci) of class set {c} of SG correspond graph of entire datasets. At this time, graph nodes that represent graph are value of concat having URI. (concat, str is inner function of SPARQL specification. Sep-str is namespace or distinguishing text (normally use "#", "/") and prefix)
(*PII-2*) : Graph of entire dataset is the corresponding class that of triple set of instances. That triple set of istance node is {t|(in,pn,on)}.

Figure 10. Define PR-II Step

(1-2) is procedure satisfying PR-I step of figure 9. (4) of Figure 9 is procedure satisfying PR-II step. Finally (5) of Figure 9 is procedure satisfying PR-II. If the algorithm of Figure 9 completed, the algorithm of Figure 10 could be searched instance nodes and related triple is quicker than before.

6. Test and Conclusion

We defined model, algorithm and major component in order to construct ontology KB from semi-structured dataset. And then, we described how construct optimized ontology KB.

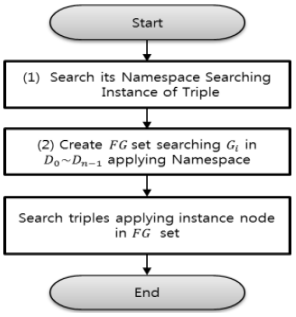

Figure 11. Searching Instance Quickly Algorithm in Optimized Ontology DB Structure

In order to proof our approaches, we implemented prototype of this framework using Java Environment, Apache Jena, and Apache POI etc. And then, this test to constructing Jena TDB from spreadsheet of a research equipment. As a result to construct total 54,868 instance (810,592 triples) in ontology KB from domain of research equipment at Korea NFEC institute, shown flowing table 2. Also, this ontology DB satisfied 10 of 14 possible features of mapping languages**.** PropertyExp can describe more concise than existing mapping language.

Table II. Result for Test Using the Semi-Structured Dataset within Equipment Domain at NFEC

Mapping Property	Count of TT	Count of Original Cell in Row	Rate of Integrity
rdf:type	54,868	54,868	100.0%
rdfs:label	54,868	54,868	100.0%
hasImageURL	53,868	53,868	100.0%
hasName	109,590	109,590	100.0%
hasTakeDate	54,868	54,868	100.0%
hasTakePrice	54,868	54,868	100.0%
hasWGS_X	54,707	54,707	100.0%
hasWGS_Y	54,707	54,707	100.0%
relatedForm	54,868	54,868	100.0%
relatedModel	54,861	54,861	100.0%
relatedProject	43,952	43,952	100.0%
relatedStatus	54,868	54,868	100.0%
relatedUseScope	54,852	54,852	100.0%
relatedUseType	54,847	54,847	100.0%
TOTAL	810,592	810,592	100.0%

Finally, in order to construct ontology KB from semi-structured dataset, we propose structural method more than existing method. This framework becomes flexible because CVI support implementation of method for converts various data.

Also, Optimization for ontology KB grows inference performance, because of possible to finding triples of instance in a named graph.

In the Future, our framework required research for method of inserting real-time data generated by IoT and big data environment from ontology KB, and required what is achieved growing performance for architecture of existing Triple store using our method.

7. References

[1] Franck Michel, Johan Montagnat and Catherine Faron-Zucker. “A survey of RDB to RDF translation approaches and tools”, hal-00903568v1, 2013

[2] Souripriya Das, Seema Sundara and Richard Cyganiak. W3C Recommendation. “R2RML: RDB to RDF Mapping Language”. http://www.w3.org/TR /r2rml/ (Access Date: 1 August, 2015).

[3] Marcelo Arenas, Alexandre Bertails, Eric Prud'hommeaux and Juan Sequeda. W3C Recommendation. “A Direct Mapping of Relational Data to RDF”. http://www.w3.org/ TR/rdb-direct-mapping/ (Access Date: 1 August, 2015).

[4] W3C Wiki ConverterToRDF, https://www.w3.org/wiki/ ConverterToRdf (Access Date: 1 August, 2015).

[5] Cimiano and Philipp, “Ontology Learning and Population from Text : Algorithms, Evaluation and Applications”, 2007, ISBN 0387306323.

[6] Jeffrey D.Ullman and Jennifer Widom, “A First Course in Dataset Systems : Third Edition”, Pearson International Edition, 2008, ISBN 0135021766.

[7] Tim Berners-Lee, “Relational Databases on the Semantic Web”, 2009. http://www.w3.org/DesignIssues/RDB-RDF.html (Access Date: 1 August, 2015).

[8] Maksym Korotkiy and Jan L. Top, “From Relational Data to RDFS Models”, Lecture Notes in Computer Science Volume 3140, pp 430-434, 2004.

[9] Su-Kyoung Kim, Gui-Hyun Baek and Gi-Tae Lee. “Lecture Notes in Computer Science: Light-weight Ontology Reasoning Engine Model Research for Real-time Context-aware based on Smart Phone”. SUComS 2013, ASTL Vol. 26, pp. 31 - 42, 2013.

[10] Graham Klyne, Jeremy J. Carroll and Brian McBride. W3C Recommendation. “RDF 1.1 Concepts and Abstract Syntax”. Access Date: 1 August, 2015. http://www.w3.org/TR/201 4/REC-rdf11-concepts-20140225/ (Access Date: 1 August, 2015).

[11] Deborah L. McGuinness and Frank van Harmelen. W3C Recommendation, “OWL Web Ontology Language Overview” http://www.w3.org/TR/2004/ RE C-owl-features-20040210/ (Access Date: 1 August, 2015).

[12] W3C, “SPARQL 1.1 Query Language”, 2013, http://www w.w3.org/TR/2013/REC-sparql11-query-20130321/ (Access Date: 1 August, 2015).

[13] Apache Jena project team, “Jena documentation overview”. Access Date: 1 August, 2015. https://jena.apache.org/ documentation/index.html (Access Date: 1 August, 2015).

[14] Apache POI project team, “POI-HSSF and POI-XSSF - Java API To Access Microsoft Excel Format Files”, https://poi.apache.org/spreadsheet/index.html (Access Date: 1 August, 2015).

[15] Hajung Sung, Jangwon Gim, Sukhoon Lee and Doo-Kwon Baik, “An RDB to RDF Mapping System Considering Semantic Relations of RDB Components”, KIPS Tr. Software and Data Eng. Vol.3, No.1 pp.19~30, 2014.

[16] Hoyoung Jeong, Jungmin Kim, Junwon Jung. et al. “Design and Implementation of RDF Storage and RDQL Query Processor”, KIISE Database, Vol. 41, No.3, pp.162-167.

