RDF Mapper – easy conversion of relational databases to RDF

Eliot Bytyçi, Lule Ahmedi and Granit Gashi
University of Prishtina "Hasan Prishtina", 10000, Prishtinë, Kosovo
{eliot.bytyci, lule.ahmedi}@uni-pr.edu, grannit.g@gmail.com

Keywords: Ontology engineering; relational databases; RDF; mapping

Abstract: Nowadays with the raised necessity to serve data through the Web in a rather Linked Data model similar to the way DBpedia extracts structural information from Wikipedia, it is becoming usual to require existing data provided as tables to get mapped into RDF. In this paper, a straightforward alternative to mapping of data from relational database model to RDF is introduced. The mapper has not yet reached its maturity but nevertheless, reduces greatly the amount of time needed for conversion. Furthermore, it allows the user to select parts that are needed for conversion, thus preventing from the unnecessary conversion.

1 INTRODUCTION

It is becoming conventional to require existing data to be provided as tables in order to get mapped into Resource Description Framework (RDF). That is similar to the approach that DBpedia[1] extracts structural information from Wikipedia[2]. Main reason for that lies in the raised necessity to serve data through the Web in a rather Linked Data model. Of course, the reason for that is the model that enables things and people be described arbitrarily, as opposed to the presentation of information on the actual web via HTML pages. Semantic Web and its Linked Data is capable of the representation of information with their description on the web via metadata models such as RDF, a W3C[3] recommendation (Caroll & Klyne, 2004). Hence, there lies the struggle of relating and mapping relational database data with RDF conceptual descriptions. That, and the usage of other tools during our work, served as motivation for creating another tool that would overcome barriers presented.

The RDF Mapper introduced in this paper is responsible for the smooth transition between the two models, namely the relational database and the ontology described in RDF. Although, currently the mapper supports mapping only of databases created in MySQL, in the future it is planned to extend its support to other database systems like Oracle, DB2, MS SQL, and PostgreSQL by keeping an abstraction between the mapping code and the database layer.

In the RDF Mapper, the user first specifies the database from which the mapper shall get the data. After that, the user specifies the ontology that is expected to be populated with those data. RDF Mapper is a desktop system developed in JAVA including the Apache Jena (Caroll, et al., 2004)

[1] www.dbpedia.org
[3] www.w3.org
[2] www.wikipedia.org

extension to support working with ontologies, as well as the Swing toolkit (Eckstein, et al., 1998) for the user interface components. Its architecture is presented in Figure 1.
When a user connects to the database, it is initially the schema of the database that will be read prior to further processing.

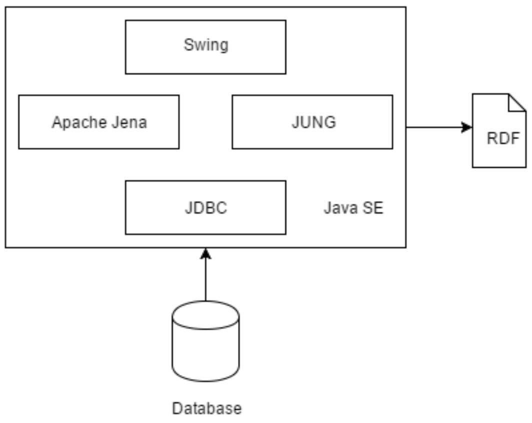


Fig. 1. Implementation architecture of the RDF Mapper

Even though this step, i.e., the schema extraction is done automatically, the user has the possibility to intervene by choosing himself / herself which relevant data to be mapped or left out. By default, every table of the database will be mapped to a Class in ontology, while every database attribute will be mapped to a Property in ontology. Further mapping relations between the database schema and the ontology are presented in Table 1.

Table 1. Mapping rules

Relational DB	**Ontology**
Table	Class | *rdfs:Class*
Attribute	Property | *rdf:Property*
Tuple of X table	Class X instance
Attributes of a tuple	Instance literals
SQL datatypes	RDFS datatypes | *rdfs:Datatype*

The remainder of this article is organized as follows. Section 2 will cover the related work. Section 3 summarizes the mapping schema, while section 4 covers a mapping example. Finally, in section 5, conclusions and future work will be discussed.

2 RELATED WORK

In (Spanos, et al., 2012), authors have conducted a survey on the creation of an ontology from an existing database instance and the discovery of mappings between an existing database instance and an existing ontology. Their classification of the approaches falls into two categories based on database reverse engineering. Approaches that apply reverse engineering try to recover the initial conceptual schema from the relational schema and translate it to an ontology expressed in a target language (Aumueller, et al., 2005). On the other hand, there are methods that, instead of reverse engineering techniques, apply few basic translation rules from the relational to the RDF model and/or rely on the human expert for the definition of complex mappings and the enrichment of the generated ontology model. The best representatives of the former approach are: D2RQ, SPYDER, COMA++ and Virtuoso.
D2RQ (Bizer & Seaborne, 2004) supports both automatic and manual modes. In the automatic mode, the ontology is created according to the rules common among reverse engineering techniques. While in the user assisted mode, the mapping is completely specified by the user. It is similar to RDF Mapper when used in semi-automatic mode, where the user builds on the automatically generated mapping in order to modify it at will.
Spyder (Miller & McNeil, 2010) creates an RDFS view of a relational database, supporting the entire range of automation levels. The automatic mode is based on the basic approach and its own mapping representation languages. Spyder’s mapping language is rich, while the tool also supports a fair amount of R2RML features.
COMA++ (Aumueller, et al., 2005) in contrast to other systems from the same era, COMA++ is built explicitly also for inter-model matching.
Virtuoso (Blakeley, 2007) offers an RDF view over a relational database with its RDF Views feature with similar functionality to D2RQ. It supports both automatic and manual operation modes. In the former, an RDFS ontology is created following the basic approach, while in the latter, a mapping expressed in the proprietary Virtuoso Meta-Schema language is manually defined.
Another more recent paper (Sequeda, 2013) discusses standards of W3C proposed to bridge the gap between Relational Databases and Semantic Web. It presents two specifications: mapping of relational data to RDF and R2RML: RDB to RDF mapping language. The R2RML subset called R2RML$_{core}$ which has a simpler structure but could be similar to Direct Mapping

(Sequeda, et al., 2012) if views are allowed as input. Direct Mapping.

3 MAPPING SCHEMA

RDF Mapper relies on the Apache Jena (Caroll, et al., 2004) open source library for working with ontologies. That means reading an ontology as well as creating RDF instances based on that ontology. For interacting with SQL databases, the corresponding JDBC driver is used, while for visualizing RDF graphs it utilizes another open source library, namely Java Universal Network/Graph Framework (JUNG)(O'Madadhain, et al., 2003).
Compared to the challenges of other mappers, which are summarized in (Pinkel, et al., 2015), RDF Mapper supports the following:

- naming conflicts are resolved by letting the user manually intervene during the mapping through renaming of certain database schema constructs towards unique naming conventions between database and ontology,
- some of the structural heterogeneity conflicts, which involve:
- type conflicts – and in our case the normalization and denormalization are supported, albeit with a little work around from user, by letting the user "relate" those attributes which belong to several tables to a single class in case of normalization, and the other way around for denormalization.
- key conflicts as well are supported and that both primary and foreign keys,

In the other hand, RDF Mapper at this stage does not yet support covering class hierarchies, dependency conflicts and semantic heterogeneity while mapping, which will be in the future considered for implementation.

Every resulting instance in the RDF will have a rdf:Description tag, which allows grouping of one or more statements into a single container. Furthermore, the linkage between a given instance and the class to which it belongs, in RDF Mapper the rdf:Type relation will be used.
In order to evaluate methods for ontology matching, and initiative has been created called Ontology Alignment Evaluation Initiative (OAEI, 2018) to mandate consensus for evaluation of the methods. In order to achieve goals such as assessing strengths and weaknesses of alignment / matching systems or compare the techniques or even improve evaluation, the initiative will organize yearly events and publicize them for further analyses.

4 MAPPING EXAMPLE

In order to emphasize the features and the characteristics of the RDF Mapper, a walk through examples will be presented. Initially schema reading will be discussed, followed by the generation of RDF.

4.1 Connection and schema reading

Initially on the start of the RDF mapper, the user should select the database to connect to. The database can be a local one or stored in any remote server. In the latter case, besides the username, password and database name, the user should also specify the address where the database is stored. Comparing with other approaches presented in related work, our approach against the database is different from most of database approaches where the schema is typically not predefined. In our case, the schema is first read from the database, and as a result, metadata becomes data. As illustration, one of built-in metadata (system) relations in MySQL *information_schema.columns* is read to extract necessary information regarding columns such as name, type, etc. of a certain table like students using the following command:

```
SELECT * FROM
information_schema.columns
WHERE table_schema = 'students'
```

Further, for instance, to allow finding out all foreign keys in the database and their relationship to corresponding tables and columns, another metadata relation should be read called *information_schema.key_column_usage*, as pre-sented in example below:

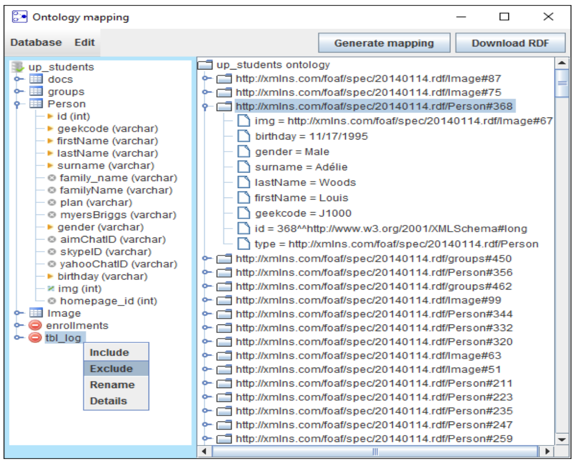
```
SELECT * FROM
information_schema.key_column_usage
WHERE table_schema = 'students'
AND referenced_table_name
IS NOT NULL
```

4.2 Mapping through our approach

After the above mentioned successful connection to a given database, the application will read the schema of that database and show its corresponding tables and attributes. Those will be shown in tree view, in order to have a clearer and easier way for the user to eventually change their names, to show their details and choose what to include. Including means everything from the database – all by default, or what to exclude from mapping. The process can be tried several times in order to have a different mapping each time – depending on user selections for inclusion and exclusion of mapping parameters. Figure 2 represents one view of a certain database ready for mapping (left-hand side), with a preview of the generated mapping result (right-hand side), while its graph representation is shown in Figure 3.
Initially, the user defines the base URI of the target ontology, and then it is connected to a table through its name, chosen as well by the user, since the user can modify the given name. In addition, classes will have their rdf:type as decided by W3C rdfs:Class.
An example of an RDF/XML output as generated by the mapper is provided next.

```
<rdf:Description
rdf:about="http://xmlns.com/foaf/sp
ec/20140114.rdf/Person#368">

<foaf:birthday>11/17/1995</foaf:bir
thday>
    <foaf:gender>Male</foaf:gender>

<foaf:surname>Adélie</foaf:surname>

<foaf:lastName>Woods</foaf:lastName
>

<foaf:firstName>Louis</foaf:firstNa
me>

<foaf:geekcode>J1000</foaf:geekcode
>
    <foaf:id
rdf:datatype="http://www.w3.org/200
1/XMLSchema#long"
    >368</foaf:id>
  <foaf:img
rdf:resource="http://xmlns.com/foaf
/spec/20140114.rdf/Image#67"/>
<rdf:type
rdf:resource="http://xmlns.com/foaf
/spec/20140114.rdf/Person"/>

</rdf:Description>
```

The generation of this RDF by the mapper is illustrated in Figure 2. From that we can present few of the examples mentioned in the beginning of this section such as renaming example, usage of keys, foreign keys and exclusion of unneeded features.
By consulting Figure 2, we observed class Person in the FOAF ontology, which is actually mapped by the user from the table students. Renaming is performed manually within the mapper so that table student becomes Person class, and hence correspond to the target Person ontology.
Again from the Figure 2 as an example of usage of Keys we observed that the URIs of the instances are composed of the base URI of the ontology, followed by class name and then the primary key value of the given tuple (high-lighted in the Figure 2 case of key 368) of the table.
Similar to the primary keys, the foreign Keys example can be viewed in Figure 2. The user has mapped the table *profile_pics* to the FOAF class Image and then the many-to-one relationship between table students and *profile_pics* to the FOAF object property *img*.
Of course, with the tables, which do not have any meaning in the ontology such as enrolments, user can choose to exclude them from the mapping.



```
up_students
  docs
  groups
  Person
    id (int)
    geekcode (varchar)
    firstName (varchar)
    lastName (varchar)
    surname (varchar)
    family_name (varchar)
    familyName (varchar)
    plan (varchar)
    myersBriggs (varchar)
    gender (varchar)
    aimChatID (varchar)
    skypeID (varchar)
    yahooChatID (varchar)
    birthday (varchar)
    img (int)
    homepage_id (int)
  Image
  enrollments
  tbl_log
```

```
up_students ontology
  http://xmlns.com/foaf/spec/20140114.rdf/Image#87
  http://xmlns.com/foaf/spec/20140114.rdf/Image#75
  http://xmlns.com/foaf/spec/20140114.rdf/Person#368
    img = http://xmlns.com/foaf/spec/20140114.rdf/Image#67
    birthday = 11/17/1995
    gender = Male
    surname = Adélie
    lastName = Woods
    firstName = Louis
    geekcode = J1000
    id = 368^^http://www.w3.org/2001/XMLSchema#long
    type = http://xmlns.com/foaf/spec/20140114.rdf/Person
  http://xmlns.com/foaf/spec/20140114.rdf/groups#450
  http://xmlns.com/foaf/spec/20140114.rdf/Person#356
  http://xmlns.com/foaf/spec/20140114.rdf/groups#462
  http://xmlns.com/foaf/spec/20140114.rdf/Image#99
  http://xmlns.com/foaf/spec/20140114.rdf/Person#344
  http://xmlns.com/foaf/spec/20140114.rdf/Person#332
  http://xmlns.com/foaf/spec/20140114.rdf/Person#320
  http://xmlns.com/foaf/spec/20140114.rdf/Image#63
  http://xmlns.com/foaf/spec/20140114.rdf/Image#51
  http://xmlns.com/foaf/spec/20140114.rdf/Person#211
  http://xmlns.com/foaf/spec/20140114.rdf/Person#223
  http://xmlns.com/foaf/spec/20140114.rdf/Person#235
  http://xmlns.com/foaf/spec/20140114.rdf/Person#247
  http://xmlns.com/foaf/spec/20140114.rdf/Person#259
```

Fig. 2. A ready-for-mapping database example

4.3 RDF generator

After the user has performed necessary edit on the database parameters prior to mapping, the system is ready for ontology generation. The view of the mapping result will be in a tree like form, but as well as a graph for better visualization, as depicted in Figure 3. The triplets generated by the graph represent subject, predicate and object.

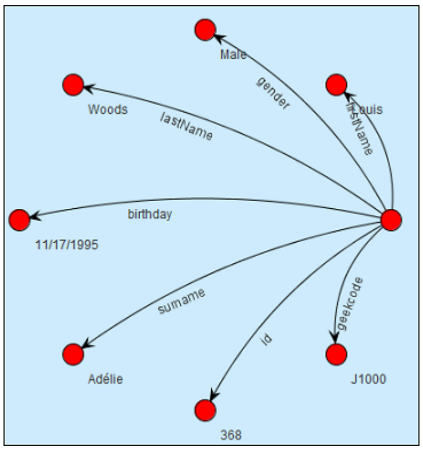


Fig. 3. The graph of the ontology resulting from mapping

In the end, a user can choose among several representation formats of RDF data for exporting the file like to RDF/XML, N-triples, Turtle and N3.

5 CONCLUSION

The RDF mapper even though in its first stage, has proven to be of a value when dealing with MySQL databases. It offers a straightforward and practical system for relational database conversion into RDF. Some of the contributions to mention are the drag and drop possibilities (add or remove constructs) or even the renaming of construct names such as classes, prefixes or properties).
In the future, the system will be further extended with other new features, making it possible to deal with other database formats and with more complex relational databases. Furthermore, the system will be extended to deal with conflicts not supported at this stage such as automatic process of normalized database schema where sometimes we may have a case that some of the attributes of a record that need to be mapped to a single RDF instance, are stored in another table and related with a foreign key. There is also the case of a denormalized database schema where we may want to map a record of a table into more than one RDF instances. Moreover, covering class hierarchies, dependency conflicts and semantic heterogeneity while mapping, will also be challenges to be addressed in the future.

REFERENCES

Aumueller, D., Do, H.-H., Massmann, S. & Rahm, E., 2005. *Schema and ontology matching with COMA++*. Baltimore, Maryland, ACM.

Bizer, C. & Seaborne, A., 2004. *D2RQ-treating non-RDF databases as virtual RDF graphs.* s.l., ISWC.

Blakeley, C., 2007. *Mapping relational data to RDF with Virtuoso's RDF Views,* s.l.: OpenLink Software.

Caroll, J. J. et al., 2004. *Jena: implementing the semantic web recommendations.* New York, ACM.

Caroll, J. J. & Klyne, G., 2004. Resource Description Framework (RDF). *Concepts and Abstract Syntax.*

Eckstein, R., Marc, L. & Wood, D., 1998. *Java swing.* Sebastopol, CA: O'Reilly & Associates, Inc..

Miller, A. & McNeil, D., 2010. *Revelytix RDB Mapping Language Specification,* s.l.: Revelytix.

O'Madadhain, J., Fisher, D., White, S. & Boey, Y., 2003. *The jung (java universal network/graph) framework,* s.l.: University of California.

OAEI, 2018. *Ontology Alignment Evaluation Initiative.* [Online] Available at: http://oaei.ontologymatching.org/ [Accessed 20 07 2018].

Pinkel, C. et al., 2015. *RODI: A benchmark for automatic mapping generation in relational-to-ontology data integration,* s.l.: ESWC.

Sequeda, J., 2013. *On the Semantics of R2RML and its Relationship with the Direct Mapping.* s.l., ISWC.

Sequeda, J. F., Arenas, M. & Miranker, D. P., 2012. *On directly mapping relational databases to RDF and OWL.* s.l., ACM.

Spanos, D.-E., Stavrou, P. & Mitrou, N., 2012. Bringing relational databases into the semantic web: A survey. *Semantic Web,* 3(2), pp. 169-209.