Semi-automatic RDFization using automatically generated mappings

Noorani Bakerally[1,2],

Cyrille Bareau[3], Fabrice Blache[3], Sébastien Bolle[3], Christelle Ecrepont[2], Pauline Folz[3], Nathalie Hernandez[1], Thierry Monteil[2], Gilles Privat[3], and Fano Ramparany[3]

[1] IRIT, Toulouse, France {firstname.lastname}@irit.fr
[2] LAAS-CNRS, National
Institut of applied Sciences of Toulouse, Toulouse 31400, {firstname.lastname}@laas.fr
http://www.springer.com/gp/computer-science/lncs
[3] Orange Labs, Meylan, France
{firstname.lastname}@orange.com

Abstract. Most data available on the Web do not conform to the RDF data model. A number of tools/approaches have been developed to encourage the transition to RDF. Manual and automatic tools/approaches tend to be complex and rigid. On the other hand, semi-automatic tools can hide and automate complex tasks while enhancing flexibility by solicitating human experts for decision making purposes. In this paper, we describe a semi-automatic approach to facilitate the transformation of heterogeneous semi-structured data to RDF. The originality of our approach is its ability to generate exhaustive descriptions using entities from several ontologies without requiring end-users to have a knowledge of ontologies. We provide an implementation of our approach and demonstrate its use using a real dataset from an open data portal.

Keywords: RDF, Data transformation, Semi-automatic approach

1 Introduction

To realize the vision of the Semantic Web, the conformance of existing data to the RDF model is a necessary condition. Yet, it is a fact that most of the data available on the Web do not satisfy this requirement. A number of tools have been developed to facilitate the transition to RDF. Much of them are founded on well-defined mapping languages (R2RML [1], RML [2], SPARQL-Generate [3], etc.). Using mapping languages directly is complex. This is because they have a steep learning curve and require knowing the syntax and semantics of the languages in addition to the Semantic Web stack and ontologies that can be used.

Besides mapping languages, there are automatic and semi-automatic RDFizers. We ignore automatic RDFizers (e.g. Direct Mapping [4], Docker2RDF [5], etc.) as their transformation cannot be customized or they are restricted for specific domain models. The minor category of works (RMLEditor [6], OpenRefine [7], etc.) around semi-automatic RDFizers is our main interest. We focus on this category due to their ability in aiding end-users by automating complex tasks without hindering flexibility by incorporating them for decision making and validation. The main problem with the

latter tools is that they mostly only provide a graphical interface with some facilities for searching through ontologies. By doing so, they still rely on end-users with respect to their knowledge about ontologies and data modeling using them.

In this work, our aim is to provide an approach to further facilitate semi-automatic RDFizers by automatically generating mappings without prior knowledge about ontologies, that may then be customized by end-users. The originality of our contribution is that it automatically generates several holistic mappings and try best to provide an exhaustive description for a given type of objects. To ensure exhaustivity, the type of objects can be described with entities defined in several ontologies as long as semantic coherence is maintained. Our approach is not an alternative but complementary to existing tools. In the rest of this paper, we describe our approach and its implementation in Section 2 and Section 3 respectively. Then, we demonstrate our implementation using a real dataset from open data portal. Finally in Section 5, we conclude with limitations of our approach and future works.

2 Our Approach

We use a divide-and-conquer strategy to RDFize non-RDF data. The base case of this strategy occurs when the non-RDF data describes only one type of object. In this paper, our approach is focused on this base case. Our approach to generate final mappings consists of four main steps: **i)** `Generate Schema Descriptions` **ii)** `Generate candidates` **iii)** `Generate candidate mappings` **iv)** `Refine candidate mappings`, as shown in Figure 2. The refined mapping selected by the user is then automatically represented in a mapping language and used to generetaed the RDF representation of the data.

For illustration purposes, we consider a parking dataset[4] from Grenoble open data portal[5]. Figure 1 is part of a preview of that dataset taken directly from the data portal. Moreover, our approach uses an `Ontology repository`, as despicted in Figure 2.

_id	CODE	LIBELLE	ADRESSE	TYPE	TOTAL	type	id	lon	lat
11	SPR_PKG...	CATANE	RUE AMP...	PKG	490	PKG	SPR_PKG...	5.70503	45.181035
9	QPA_PKG...	CHAVANT	17, BD M...	PKG	394	PKG	QPA_PKG...	5.731463	45.185612
15	PVP_PKG...	ENCLOS...	PLACE V...	PKG	130	PKG	PVP_PKG_...	5.728358	45.188401

Fig. 1. Parking data from Grenoble Open Data Portal

Suppose that it contains the vocabularies MobiVoc[6], Schema.org[7], WGS84[8] and Dublin Core Metadata Terms[9].

[4] http://data.metropolegrenoble.fr/ckan/dataset/parkings-de-grenoble/resource/a6919f90-4c38-4ee0-a4ec-403db77f5a4b, last accessed on 7 December 2019
[5] http://data.metropolegrenoble.fr/, last accessed on 7 December 2019
[6] https://www.mobivoc.org/, last accessed 10 February 2020
[7] https://schema.org/, last accessed 10 February 2020
[8] https://www.w3.org/2003/01/geo/, last accessed 10 February 2020
[9] https://www.dublincore.org/specifications/dublin-core/dcmi-terms/, last accessed 10 February 2020

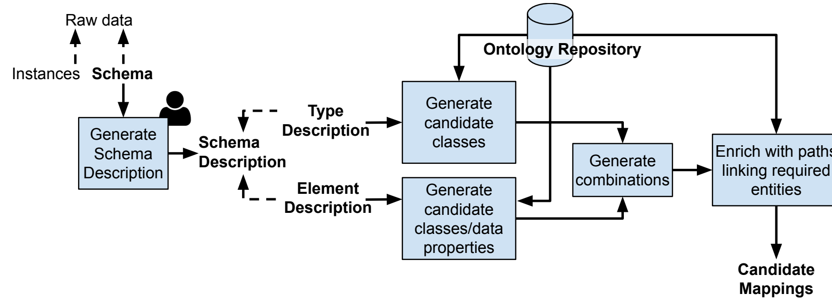

Fig. 2. Mappings Generation Process

Below, we proceed with the descriptions of the steps in our approach.

Generate Schema Description In our approach, we suppose that the file to be transformed contains only one type of object. We make the difference between the type of object (`Type element`) and its properties (`Schema element`). To capture the background knowledge about the schema used in the raw data, we generate a `Schema description` consisting of a `Type description` and `Elements description`, via a user interface (*cf.* Section 3) with the involvement of the end-user.

`Type Description` characterizes the type of objects described by the schema and `Elements descriptions` characterizes the schema elements (e.g. `lon` in Figure 1). For both, `Type Description` and `Elements Description`, the description can be enriched by `keywords` added by the end-user. The `schema description` may not contain a description for all columns. For example, the schema description of the data in Figure 1 may omit the description of the column `CODE` as it contains the same information as the column `id`. In this way, uninterested columns may be ignored.

Generate candidates Using an `Ontology repository`, and the `Schema description`, a set of candidate classes are generated for typing objects, thanks to `Type description` and a set of candidate data properties or classes are generated for modeling the schema element, thanks to `Elements description`. The `Schema description` is then converted into a pseudo-ontology in OWL and simple ontology matching approaches presented in [8] are used to generate the candidate for the `Type element` and `Schema elements`. Table 1 shows a schema description for the schema of Figure 1 and generated ontology entities for its elements. For example, the objects' type in Figure 1 can be described by the keyword 'parking facility'. Using the latter description, the classes `mv:ParkingFacility` and `sc:Park` are generated to type the objects. Similarly, the schema element `TOTAL` is described using the keywords 'capacity' and 'total' using which the class `mv:Capacity` and data property `sc:totalTime` are candidates generated to model it. The candidate proposal `sc:totalTime` is not appropriate to model `TOTAL` as its semantics is not compatible with the latter. To determine the appropriateness of an entity, we also generated a confidence. For the sake of simplicity, we omit this information from Table 1.

Generate candidate mappings Candidate mappings are build in two steps. First, candidate entities are combined with a cartesian product, producing a set of `combination`

Schema Description			Generated Entities	
		Keyword	Classes	Data Properties
Type Description		'parking facility'	mv:ParkingFacility, sc:Park	
Elements Description	id	'identifier'		dc:identifier
	LIBELLE	'description'		sc:description
	TOTAL	'capacity','total'	mv:Capacity	sc:totalTime
	lat	'latitude'		wgs84:lat
	lon	'longitude'		wgs84:long
	ADDRESSE	'address'		sc:address

Table 1. Candidate entities for typing and schema elements

	Type Class	id	LIBELLE	TOTAL	lat	lon	ADDRESSE
1.	mv:ParkingFacility	dc:identifier	sc:description	mv:Capacity	wgs84:lat	wgs84:lon	sc:address
2.	mv:ParkingFacility	dc:identifier	sc:description	sc:totalTime	wgs84:lat	wgs84:lon	sc:address
3.	sc:Park	dc:identifier	sc:description	mv:Capacity	wgs84:lat	wgs84:lon	sc:address
4.	sc:Park	dc:identifier	sc:description	sc:totalTime	wgs84:lat	wgs84:lon	sc:address

Table 2. Combinations of generated entities for type class and schema elements

`of mappings` where a `combination of mappings` consists of a candidate class for typing the object, that we refer as the `type class`, and a candidate data property or class for each schema elements. Table 2 shows all combinations generated from the candidate entities in Table 1. As we can see, in each combination, there is one candidate entity for the type class and one for each schema element. In a second step, we keep `combination of mappings` where we can assess the existence of a path between the `type class` and candidate entities for the `schema element`. These paths are identified using patterns that we have defined. They exploit the graph structure of ontologies. For example, Figure 3 shows the first combination from Table 2 and the required paths, illustrated as dotted lines, that will be generated at this step. It is possible that more than one path or no path exist between some entities. We then obtain a set of `candidate mappings`.

Generate final mapping A user interface is provided to allow choosing and refining a `candidate mapping` to obtain the `final mapping`. There are cases where a `schema element` may be modeled by a class. In these cases, data properties containing the latter class in their domains may be used to specify the values. Refining consists in choosing the appropriate data property.

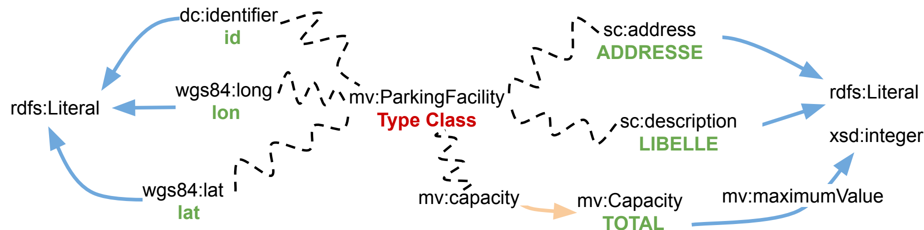

Fig. 3. Candidate mappings for first combination without generated paths

3 Implementation

An overview of our implementation, SAURON, is shown in Figure 4. Core to SAURON is the `RDFizer` that generate final mappings using the approach described in the previous section. To facilitate human intervention, we provide a graphical `User Interface`. Using the interface, users can upload the raw data in the CSV format and may enrich it with keywords to generate the `Schema description`. The `User Interface` interacts with the `RDFizer` via a `Web Service`. Eventually, on obtaining the candidate mappings, one of them is chosen and refined and validated by the end-user and sent to the web service together with the raw data for transformation to RDF. This is done with SPARQL-Generate in the current implementation.

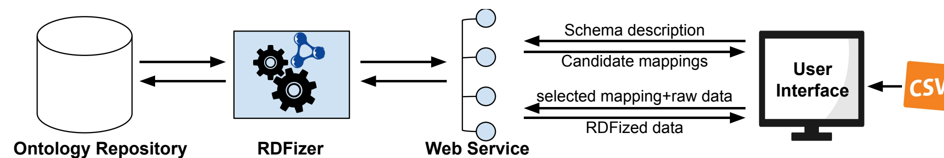

Fig. 4. Overview of SAURON

The user interface is a web application implemented using the JavaScript library *React*[10]. The video available online[11] shows the use of the interface to generate mappings for the CSV parking dataset in Section 2. As it can be seen, the user interface has three main parts. The top left part is focused on the raw data that is imported using the `import CSV` menu item. On clicking on a column, keywords can be entered. The bottom left part shows the candidate mappings and on selecting one of them, its corresponding description graph is rendered on the right part. The end-user can interact with different part of the latter graph and select and validate the paths.

4 Demonstration

During this demonstration, we intend to RDFize Grenoble parking dataset partly shown in Figure 1. We perform two experiments: `NK` and `WK`. In `NK`, we only specify the `type description` with keywords. In `WK`, we also specify keywords for interested columns. These keywords are shown in Table 1. Results are despicted in Table 3. As we can see in Table 3 and the video, adding keywords greatly improve the quality of mappings that are generated.

5 Conclusion

We have tested our approach on real datasets from open data portals and the results were promising. However, there are three main limitations. Firstly, the success of the

[10] https://reactjs.org/
[11] https://youtu.be/LKZH4gs7sNQ

approach depends much on the selection of keywords. It may not be easy for the user to define the keywords that will correspond to labels of ontologies entities. An extension of the approach that will suggest keywords according to these labels is currently being implemented. Secondly, as mentioned in Section 2, our approach can consider raw data containing only one type of object described by several data properties. However in some cases, the object can be link in its description to other objets. Approaches dealing with entity resolution and entity linking could be used. Thirdly, as of now, there are no alignments between the ontologies in the ontology repository. The existence of these alignments can improve the quality of the generated mappings.

	LIBELLE	ADRESSE	TOTAL	id	lon	lat
NK	–	schema:adress	–	mobivoc:id	–	–
WK	schema:label	schema:adress	mv:Capacity	dc:id	geo:long	geo:lat

Table 3. Initial mappings without keywords (`NK`) and with keywords (`WK`)

References

1. Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF Mapping Language, W3C Recommendation 27 September 2012. Technical report.
2. A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. Van de Walle. RML: A generic language for integrated RDF mappings of heterogeneous data. In *LDOW*, 2014.
3. M. Lefrançois, A. Zimmermann, and N. Bakerally. Flexible rdf generation from rdf and heterogeneous data sources with sparql-generate. In *EKAW*. Springer, 2016.
4. Marcelo Arenas, Alexandre Bertails, Eric Prud'hommeaux, and Juan Sequeda. A Direct Mapping of Relational Data to RDF, W3C Recommendation 27 September 2012. W3C Recommendation, World Wide Web Consortium (W3C), September 27 2012.
5. Ahmed Ben Ayed, Julien Subercaze, Frederique Laforest, Tarak Chaari, Wajdi Louati, and Ahmed Hadj Kacem. Docker2rdf: Lifting the docker registry hub into rdf. In *2017 IEEE World Congress on Services (SERVICES)*, pages 36–39. IEEE, 2017.
6. Pieter Heyvaert, Anastasia Dimou, Aron-Levi Herregodts, Ruben Verborgh, Dimitri Schuurman, Erik Mannens, and Rik Van de Walle. Rmleditor: a graph-based mapping editor for linked data mappings. In *European Semantic Web Conference*, pages 709–723. Springer, 2016.
7. Ruben Verborgh and Max De Wilde. *Using OpenRefine*. Packt Publishing Ltd, 2013.
8. Elodie Thiéblin, Ollivier Haemmerlé, Nathalie Hernandez, and Cassia Trojahn. Survey on complex ontology matching. *Semantic Web*, (Preprint):1–39, 2019.