Semi-Automatic Example-Driven Linked Data Mapping Creation*

Pieter Heyvaert, Anastasia Dimou, Ruben Verborgh, and Erik Mannens

IDLab, Department of Electronics and Information Systems, Ghent University – imec
pheyvaer.heyvaert@ugent.be

Abstract. Linked Data can be generated by applying mapping rules on existing (semi-)structured data. The manual creation of these rules involves a costly process for users. Therefore, (semi-)automatic approaches have been developed to assist users. Although, they provide promising results, in use cases where examples of the desired Linked Data are available they do not use the knowledge provided by these examples, resulting in Linked Data that might not be as desired. This in turn requires manual updates of the rules. These examples can in certain cases be easy to create and offer valuable knowledge relevant for the mapping process, such as which data corresponds to entities and attributes, how this data is annotated and modeled, and how different entities are linked to each other. In this paper, we introduce a semi-automatic approach to create rules based on examples for both the existing data and corresponding Linked Data. Furthermore, we made the approach available via the RMLEditor, making it readily accessible for users through a graphical user interface. The proposed approach provides a first attempt to generate a complete Linked Dataset based on user-provided examples, by creating an initial set of rules for the users.

1 Introduction

In most cases, Linked Data is generated by applying mapping rules on existing (semi-)structured data. The mapping rules state how Linked Data is generated from (raw) data, for every graph pattern that is part of the resulting Linked Data. However, before these rules can be applied, they need to be created. This includes providing (i) which data corresponds to entities and attributes, i.e., the subjects and objects of an RDF triple; (ii) how the data is annotated and modeled, i.e., which classes, properties, datatypes, and languages are used and how they are related to each other; and (iii) how different entities are linked to each other. For example, consider the raw data in Listings 1.1 and 1.2 about books and authors, and a Linked Data example in Listing 1.3. The latter includes entities and attributes which are constructed relying on data values which also appear in the raw data and uses certain classes, properties, and datatypes. For every

* The described research activities were funded by Ghent University, imec, Flanders Innovation & Entrepreneurship (AIO), the Research Foundation – Flanders (FWO), and the European Union.

graph pattern, at least three rules need to be defined to generate this Linked Data, namely, for the subject, predicate, and object. For example, to generate a triple with the subject `http://www.example.com/book/0`, a rule has to state that the string `http://www.example.com/book/` has to be combined with the `id` of the book, which is in this case "0". This results in the creation of at least 21 mapping rules, because there are 7 triples, and each one requires at least 3 rules.

```
id,title,author
0,Harry Potter and The Sorcerer's Stone,J.K. Rowling
1,Homo Deus,Yuval Noah Harari
```

Listing 1.1: CSV data about books

```
{
  "authors": [{
    "id": "jkr",
    "name": "J.K. Rowling",
    "country": "UK"
    "birthdate": "1965-07-31"
  },{
    "id": "ynh",
    "name": "Yuval Noah Harari",
    "country": "Israel",
    "birthdate": "1976-04-24"
  }]
}
```

Listing 1.2: JSON data about authors

```
@prefix book:
  <http://www.example.com/book/> .
@prefix author:
  <http://www.example.com/author/> .

book:0 a schema:Book;
  schema:title
    "Harry Potter and The Sorcerer's Stone"@en;
  schema:author author:jkr.

author:jkr a foaf:Person;
  foaf:name "J.K. Rowling";
  foaf:country "UK";
  schema:birthdate "1965-07-21"^^xsd:date.
```

Listing 1.3: Linked Data example

When the rules are created manually, they are prone to errors, especially when dealing with large and complex data sources [1, 2], and/or multiple data sources at the same time [3]. To ease this process, both semi-automatic and automatic approaches have been the topic of research [4]. The former require user interaction during the generation of the rules, while the latter does not. Although, they provide promising results, in use cases where examples of the desired Linked Data are available the generated Linked Data might not be as desired. This in turn requires users to manually update the rules. This is due to the fact that these approaches do not consider the knowledge embedded in the examples when initially creating rules. Nevertheless, these examples offer knowledge relevant for the mapping process.

A Linked Data example can be used as a reference point for the creation of rules for generating Linked Data, from some other raw data. This is done by (i) generating entities and attributes in the same way, (ii) applying the same model and semantic annotations, and (iii) providing the same relationships among entities. For example, users may consider existing Linked Data, the complete set or just a sample of it, and want their data to be modeled and annotated the same way. This is reflected in the rules that are created, as mentioned before, based on this existing Linked Data.

To support the aforementioned use cases, we propose a semi-automatic approach for the example-driven creation of Linked Data mapping rules. The ex-

ample Linked Data is used to extract the necessary information to create the corresponding mapping rules. Users provide two elements: a set of data sources and a set of RDF triples. These two elements are used to create the rules through the following steps: (i) the original data sources are aligned with the Linked Data example and (ii) mapping rules are created based on this alignment and the knowledge about the model and semantic annotations extracted from the Linked Data example. In certain cases, manual additions might still be required, such as data transformations on the original data [5].

The remainder of the paper is structured as follows. In Section 2, we discuss the related work. In Section 3, we introduce a running example. In Section 4, we discuss our proposed approach. In Section 5, we elaborate on the implementation of this approach. In Section 6, we discuss both the existing approaches and our approach, and conclude the paper.

2 Related Work

In this section, we discuss the related work regarding mapping rules, and mapping rule creation approaches and solutions.

2.1 Background

A **mapping** consists of one or more mapping rules that state how RDF terms and triples should be generated. A mapping rule denotes how data from an original data source is used in the RDF terms, how these terms are associated to each other, and how these terms form RDF triples. **Mapping languages**, such as R2RML [6] and RML [7], provide a specification of the syntax on how to declaratively construct mappings. This is preferred over the use of custom software and scripts, because mapping languages provide a reusable solution, while custom software and scripts are tied to a specific use case and/or implementation [7].

RML is an extension of R2RML, the W3C-recommended language for mapping relational databases to RDF. While R2RML is limited to only relational databases, RML applies a data-independent approach, allowing the mapping of also, e.g., JSON and CSV to RDF. For a detailed explanation of both mapping languages, we refer to their corresponding specifications [6, 7].

2.2 Mapping Generation Approaches

In previous work [8], we identified four approaches that data owners use when they create mapping rules themselves: the data-driven, schema-driven, model-driven, and result-driven approach. The data-driven approach is based on the data, namely, rules are created for the data fractions of the input data sources. Subsequently, the rules are annotated with classes, properties, datatypes, and languages from schemas (vocabularies and ontologies). The schema-driven approach is based on the schemas, namely, rules are created using the classes, properties, and datatypes. Subsequently, the data fractions are associated with

the correct rules. The model-driven approach is based on the model of the domain. More precisely, the entities, their attributes, and their relationships to other entities are defined, without explicitly indicating neither the schemas nor the data fractions to be used. This leads to abstract rules without data fractions and schema elements. Subsequently, the model is instantiated by applying adequate schema(s) and it is associated with data fractions, by specifying which fractions are associated with which parts of the model. The result-driven approach creates rules based on the data sources and corresponding Linked Data. The rules are created based on the complete Linked Dataset's model and used schemas. Afterwards, the data fractions are associated with the correct rules.

In other research fields, the example-driven approach has been applied successfully [9, 10]. Their application of this approach includes two high-level steps: (i) for a sample of the input, the output, i.e. the example, is given by the user; (ii) similar output is generated for the complete input based on the example, with no or minimal user interaction. For example, users select an example of the data they want on a Webpage and they get all desired data, based on that example, from the page [9]. In our case, for a sample of the existing data, i.e., the input, a Linked Data sample, which is desired to be generated, may act as the example. Linked Data for the complete existing dataset is obtained by (i) creating the mapping rules which define how Linked Data should be generated, and (ii) executing them and generating the Linked Data.

The example-driven approach is related to the result-driven approach. Users either provide the complete Linked Dataset or a sample, i.e., the example. The result-driven approach deals with the case when you have the complete Linked Dataset, while example-driven deals with the case when you have a sample. Furthermore, neither the example-driven approach nor the result-driven approach have been applied so far, to the best of our knowledge.

2.3 (Semi-)Automatic Solutions

As the process of creating mappings can become costly process when done manually [1, 2, 3], both semi-automatic and automatic solutions have been proposed. The former combine automatic steps with user interaction, where the latter completely rely on automatic steps. For example, Jiménez-Ruiz et al. [11] developed BootOX that creates mappings for relational databases based on the data schema, and applies user feedback to improve the mappings. Taheriyan et al. [12] propose an automatic solution that creates a new mapping based on previous mappings, the raw data, and the preferred ontologies.

Although these solutions show promising results, during the use cases where a Linked Data example is available, users still need to adjust the rules if the resulting Linked Data does not match the example. This is due to the fact that these approaches do not take into account such use cases and, thus, they do not consider examples when creating mapping rules. Nevertheless, these examples offer knowledge relevant for the mapping process, such as (i) which data corresponds to entities and attributes, i.e., the subjects and objects of an RDF triple; (ii) how the data is annotated and modeled, i.e., which classes, properties,

datatypes, and languages are used and how they are related to each other; and (iii) how different entities are linked to each other. To the best of our knowledge, no research has been conducted in applying the example-driven approach in (semi-)automatic solutions for Linked Data generation [4]. However, the application of this approach in other fields has shown a decrease in the cost of the process [9].

Kranzdorf et al. [9] developed an example-driven system that aids in the creation of path expressions to extract the required text from Webpages. The system works as follows: (i) users select an example text on the page, (ii) the system creates an initial expression, (iii) all the text that is selected by the expression is presented to the users, and (iv) all that text is extracted. Iteratively, users can update or provide additional examples to improve the expressions. Atzori and Zaniolo [10] introduce a method to query DBpedia based on an example Wikipedia infobox: (i) users edit the information of an infobox, which acts as an example of how the infobox of a desired Wikipedia page should look like; (ii) the infobox is used to construct a SPARQL query; (iii) the query is executed on DBpedia; and (iv) the resulting Wikipedia pages are presented. In both systems, the example-driven approach results in minimal user interaction required with regard to the construction of the expressions and the queries. This leads to a less costly process.

As the generation of the mapping rules is similar to the aforementioned construction, the (semi-)automatic example-driven approach is also applicable to mapping generation. Mapping rules are created by the system based on both an existing data and Linked Data example, reducing the required user interaction. This, in turn, also can lead to a less costly process.

3 Running Example

We explain the different elements of the remainder of the paper through a running example. The example contains two data sources in two different formats: CSV (see Listing 1.1) and JSON (see Listing 1.2). The data source in the CSV format provides records for books, including the id, title, and name of the author. The data source in the JSON format provides records for authors, including the id, name, country, and birth date. The Linked Data example is based on the first record of each data source (see Listing 1.3). The data sources are interlinked through the name of the author. This is also reflected in the reuse of the author's IRI in the Linked Data example (see line 9).

4 Approach

In this section, we discuss our example-driven approach. It is based on the two steps executed by users when manually creating mapping rules based on an example: (i) the subjects, predicates, and objects of the RDF triples are aligned with the data sources; and (ii) rules are created based on the alignment, and the

Algorithm 1 Alignment between RDF example and data sources

```
for entity ∈ example do
    objects ← example.getTripleObjects(entity)
    for object ∈ objects do
        for dataSource ∈ dataSources do
            align(object, dataSource)
        end for
    end for
    if isIRI(entity) then
        identifier ← getIdentifierIRI(entity)
        align(identifier, dataSource)
    end if
    selectBestDataSource(dataSources)
end for
for (entity1, entity2) ∈ example do
    if isTripleWith(entity1, entity2) then
        findCondition(entity1, entity2)
    end if
end for
```

model, semantic extractions, and relationships between entities extracted from the RDF triples.

4.1 Data Source Alignment

The triples in a Linked Data example are contain data values that stem from the existing data. To create mapping rules that refer to the correct data values, our approach needs to determine which data values in the example align with which data values in the data sources. The steps of this alignment can be found in Algorithm 1. For each entity, triples with the entity as subject are grouped together. Per triple in a group, for each object the correct reference to a data fraction in each data source is determined, if possible. The example value (as found in the RDF example) is compared with the data values of every data fraction of the data source. For every unique entity there is a unique IRI. A common practice is to construct IRIs by using a base IRI [13] to which a entity-specific value is appended. For example, `http://www.example.com/book/0` and `http://www.example.com/book/1` are IRIs for books. Both IRIs start with `http://www.example.com/book/` and the `id` of the book, i.e., "0" and "1", is appended. This knowledge needs to added to the mapping rules to generate the correct subjects for the triples. Therefore, we analyze the IRIs and extract the document or fragment identifier, i.e., the string after the last `#` or `/`. Next, the identifier is aligned with the data sources.

For each group the data source is selected for which the most references could be found. In case the same number of references are found, an arbitrary choice is made. If two entities are connected, the conditions under which the entities are related to each other can be determined. In our approach, we consider every value

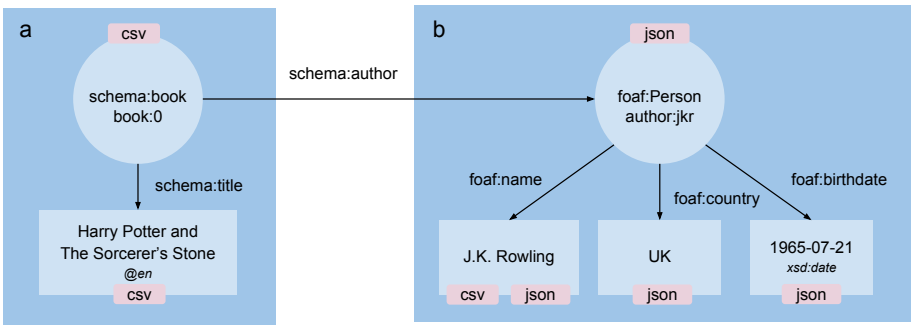

Fig. 1: The RDF example is aligned with the data sources.

from the selected data source for each entity and search for matches between these values.

We apply these steps to the Linked Data example (see Listing 1.3) and data values extracted from the two data sources (see Listings 1.1 and 1.2):

1. There are two groups, because there are two entities (see Figure 1). One group includes the triples regarding the book `book:0` (a) and the other includes the triples regarding the author `author:jkr` (b). In Figure 1 a graph visualization is used to represent the RDF triples from the example. IRIs and blank nodes, i.e., entities, are represented as circular nodes and include the class of an entity. Literals are represented as rectangular nodes and include the datatype or the language of a literal value. Edges are used to represent the relationships connecting the subjects and objects, and include the used predicates.
2. Within each group, we perform the alignment between the example data and the data sources. For group (a), the literal values aligne with the CSV data source, because "Harry Potter and The Sorcerer's Stone" can be found in `title`. For group (b), the literal values "UK" and "1965-07-31" are aligned with the JSON data source via `country` and `birthdate`. The literal value "J.K. Rowling" is aligned with both the CSV and JSON data source, because "J.K. Rowling" can be found in the column `author` and attribute `name`, respectively.
3. For the IRIs of the two entities, we consider the part after the `/` to determine the corresponding reference in the data sources. The value for `book:0` is "0", which aligns with `id` of the CSV source, and for `author:jkr` it is "jkr", which aligns with `id` of the JSON source.
4. We select the appropriate data source (and references) for each group. For group (a) only data values from the CSV data source is used, so we choose this data source. For group (b), both data sources are used; however, only the JSON data source was able to align with four nodes, while the CSV data source was only able to align with one node. Therefore, we choose the JSON data source over the CSV data source for group (b).

Algorithm 2 Creation of mapping rules
based on alignment and extracted knowledge

```
for entity ∈ example do
    triples ← example.getTriples(entity)
    triplesMap ← genererateTriplesMap()
    triplesMap.generateLogicalSource(entity)
    triplesMap.generateSubjectMap(entity)
    for triple ∈ triples do
        predicateObjectMap ← triplesMap.generatePredicateObjectMap(triple.predicate)
        predicateObjectMap.generateObjectmap(triple.object)
    end for
end for
for (entity1, entity2) ∈ example do
    if isTripleBetween(entity1, entity2) then
        triple ← example.getTriple(entity1, entity2)
        pom ← triplesMap.generatePredicateObjectMap(triple.predicate)
        pom.generateReferencingObjectMap(entity1, entity2)
    end if
end for
```

5. The book and the author of the RDF example have the same value for the column `author` and the attribute `name`. These references can be used to determine whether a pair of authors and books are related.

4.2 Mapping Creation

Once the Linked Data example is aligned with data sources, a mapping is created. During the creation of the rules, additional knowledge is required that can be extracted from the Linked Data example. More specific, this knowledge includes (i) how the data is annotated and modeled, i.e., which classes, properties, datatypes, and languages are used and how they are related to each other; and (ii) how different entities are linked to each other.

The previous step in our approach was mapping language independent. However, to create an actual mapping that can be used to generate Linked Data, we need to rely on a specific mapping language. In our approach we use RML, as it allows extracting data values from multiple, heterogeneous data sources and semantically annotating them (see Section 2).

Again, triples are grouped by entity. The steps that are performed for each group can be found in Algorithm 2. For every group, a Triples Maps is created. Each Triples Map contains the details about how subjects, predicates, and objects are generated for a certain type of entity. Each Triples Map requires a Logical Source. This explains which data source is used to create the different subjects, predicates, and objects. In the case of RML, an iterator is required if multiple entities need to be mapped to Linked Data. In our approach, the iterator can be determined by taking the common path of the references for each group. Each Triples Map needs a Subject Map that explains how subjects of triples are

generated. A Subject Map includes the classes and the template to generate the correct IRIs when IRIs are required instead of blank nodes. The classes can be extracted from the triples and the template is available from the alignment. For every combination of predicates and objects a Predicate Object Map is needed. The required information is available in a triple's predicate and object. The predicate of the triple is added to Predicate Object Map. If an object of a triple is a literal, it determines the details of a Object Map of the Predicate Object Map: (i) the reference found via the alignment is added via `rml:reference`; (ii) if the literal has a datatype, it is added via `rr:datatype`; and (iii) if the literal has a language, it is added via `rr:language`. If the object refers to an entity, instead of an Object Map, a Referencing Object Map is used. This map refers to the Triples Map of the other entity. Additionally, join conditions are added, if found during the alignment. A join condition states the condition under which there is a relationship between two entities.

```
1  <#TM_B> rml:logicalSource <#LS_B>;
2          rr:subjectMap <#SM_B>;
3          rr:predicateObjectMap <#POM_B1>, <#POM_B2>.
4
5  <#LS_B> rml:source "books.csv"; rml:referenceFormulation ql:CSV.
6
7  <#SM_B> rr:class schema:Book; rr:template "http://www.example.com/book/{id}".
8
9  <#POM_B1> rr:predicate schema:title;
10           rr:objectMap [ rml:reference "title"; rr:language "en" ].
11
12 <#POM_B2> rr:predicate schema:author;
13           rr:objectMap [
14             rr:parentTriplesMap <#TM_A>;
15             rr:joinCondition [rr:child "author"; rr:parent "name"]
16           ].
17
18 <#TM_A> rml:logicalSource <#LS_A>;
19         rr:subjectMap <#SM_A>;
20         rr:predicateObjectMap <#POM_A1>, <#POM_A2>, <#POM_A3>.
21
22 <#LS_A> rml:source "authors.json";
23         rml:iterator "$.authors[*]";
24         rml:referenceFormulation ql:JSONPath.
25
26 <#SM_A> rr:class schema:Person;
27         rr:template "http://www.example.com/author/{id}".
28
29 <#POM_A1> rr:predicate foaf:name; rr:objectMap [ rml:reference "name" ].
30
31 <#POM_A2> rr:predicate foaf:country; rr:objectMap [ rml:reference "country" ].
32
33 <#POM_A3> rr:predicate foaf:birthdate;
34           rr:objectMap [ rml:reference "birthdate"; rr:datatype xsd:date ].
```

Listing 1.4: RML mapping for the example

In Listing 1.4 the resulting mapping rules for our example can be found. Two Triples Maps with each a Logical Source, Subject Map and one or more Predicate Object Maps were created (lines 1-16 and 18-34). The Logical Source for the books (line 5) refers to the CSV data source (`rml:source`) and states that we are using the reference formulation for CSV (`rml:referenceFormulation`). The Subject Map (line 7) denotes that every book is of the class `schema:Book`

and that the IRI for each book is constructed based on the `id`, as found during the alignment. The only attribute for the books is related to the entity via `schema:title` and uses the data from the column `title`, which is in English. This is reflected in a Predicate Object Map (line 9), with a connected `rr:objectMap` pointing to the correct column and language. A second Predicate Object Map (line 12) is used to state the relationship between the books and the authors and is only valid when the author of the book and the name of the author are the same (line 15). Similar maps are created for the authors. However, the Logical Source points to a different data source and a different reference formulation is used, and an iterator is added, as we are dealing with the JSON data source (lines 22-24). Additionally, the datatype of the birth date is set to `xsd:date` (line 34).

```
1  book:0 a schema:Book;
2         schema:title "Harry Potter and The Sorcerer's Stone"@en;
3         schema:author author:jkr.
4
5  book:1 a schema:Book;
6         schema:title "Homo Deus"@en;
7         schema:author author:ynh.
8
9  author:jkr a foaf:Person;
10            foaf:name "J.K. Rowling";
11            foaf:country "UK";
12            schema:birthdate "1965-07-21"^^xsd:date.
13
14 author:ynh a foaf:Person;
15            foaf:name "Yuval Noah Harari";
16            foaf:country "Israel";
17            schema:birthdate "1976-04-24"^^xsd:date.
```

Listing 1.5: Generated Linked Data based on the two data sources and the RML mapping

The generated Linked Data based on the two data sources and the RML mapping can be found in Listing 1.5. The Linked Data contains the RDF triples that were provided as example (lines 1-3 and 9-12), and the RDF triples for the second book and author that are present in the data sources (lines 5-7 and 14-17).

5 Implementation

The approach is available via a JavaScript library[1]. This library is available for Node.js and the browser. The library supports both the CSV and JSON format, showcasing the support for tabular and hierarchical data. Furthermore, it is accessible through a command line interface and a graphical user interface via the RMLEditor [14]. The RMLEditor provides a graphical user interface for the creation and editing of mappings, with RML as its underlying mapping language. To apply the example-driven approach, users need to perform two steps: load the different data sources and provide a Linked Data example through a set of

[1] `https://github.com/RMLio/example2rml`

RDF triples. Subsequently, the mapping is created as described and visualized in the interface. This process is shown in the screencast at `https://www.youtube.com/watch?v=IQVwlYQXwAo`.

6 Discussion and Conclusion

Although existing approaches, such as data-driven and schema-driven, and their corresponding (semi-)automatic solutions, have been the topic of multiple research efforts, they offer limited benefits when dealing with use cases that provide Linked Data examples. These approaches consider one or more different elements as input, such as data, data schemas, ontologies, and existing mappings. The example-driven approach considers as input the data and a Linked Data example. The advantage of the example-driven approach is the use of knowledge that can be extracted from the Linked Data example: (i) which data corresponds to entities and attributes, i.e., the subjects and objects of an RDF triple; (ii) how the data is annotated and modeled, i.e., which classes, properties, datatypes, and languages are used and how they are related to each other; and (iii) how different entities are linked to each other. By using this knowledge, the example-driven approach creates mapping rules that generate Linked Data as desired by the users. The other approaches do not consider this example, and, thus, the created mapping rules might not generate the desired Linked Data. Consequently, users need to manually update the mapping rules. Nonetheless, these approaches are better suited when an example cannot be provided. In these use cases, the example-driven approach cannot be applied due to lack of an example. Therefore, the use case at hand drives the choice for the appropriate approach. Furthermore, the example-driven approach can be extended by applying techniques introduced by the other approaches, such as the use of data schemas, other ontologies, and existing mappings. This results in a hybrid approach that uses an increased amount of knowledge when creating the mapping rules, compared to the individual approaches. This increase of used knowledge leads to an improvement of the created mapping rules, which reduces the cost of the mapping process to generate the desired Linked Data.

As future work, we envision the addition of an extra step to our approach to create rules that apply data transformations on the raw data during the Linked Data generation. This is needed for use cases where the data in the RDF triples is not just a copy of the raw data, but instead the raw data needs to transformed before it can be used as (part of) of a subject, predicate, or object. Furthermore, we plan to explore how to combine the different approaches to achieve the aforementioned hybrid approach.

References

[1] Evgeny Kharlamov, Dag Hovland, Ernesto Jiménez-Ruiz, Davide Lanti, Hallstein Lie, Christoph Pinkel, Martin Rezk, Martin G. Skjæveland, Evgenij Thorstensen, Guohui Xiao, Dmitriy Zheleznyakov, and Ian Horrocks. Ontology Based Access

to Exploration Data at Statoil. In *Proceedings of the 14th International Semantic Web Conference*, pages 93–112. Springer, 2015.

[2] Evgeny Kharlamov, Nina Solomakhina, Özgür Lütfü Özçep, Dmitriy Zheleznyakov, Thomas Hubauer, Steffen Lamparter, Mikhail Roshchin, Ahmet Soylu, and Stuart Watson. How Semantic Technologies Can Enhance Data Access at Siemens Energy. In *Proceedings of the 13th International Semantic Web Conference*, pages 601–619. Springer, 2014.

[3] Bin He, Mitesh Patel, Zhen Zhang, and Kevin Chen-Chuan Chang. Accessing the Deep Web. *Communications of the ACM*, 50(5):94–101, 2007.

[4] Pieter Heyvaert. Ontology-based data access mapping generation using data, schema, query, and mapping knowledge. In *Proceedings of the 14th Extended Semantic Web Conference: PhD Symposium*, May 2017.

[5] De Meester, Maroy, Dimou, Verborgh, and Mannens. Declarative Data Transformations for Linked Data Generation: the case of DBpedia . In Eva Blomqvist, D. Maynard, Aldo Gangemi, R. Hoekstra, Pascal Hitzler, and Olaf Hartig, editors, *Proceedings of the 14th ESWC*, LNCS, pages 33–48. Springer, Cham, may 2017. ISBN 978-3-319-58450-8, 978-3-319-58451-5. doi: 10.1007/978-3-319-58451-5_3. URL `https://link.springer.com/chapter/10.1007/978-3-319-58451-5_3`.

[6] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF Mapping Language. Working group recommendation, W3C, September 2012. URL `http://www.w3.org/TR/r2rml/`.

[7] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In *Proceedings of the 7th Workshop on Linked Data on the Web*, 2014.

[8] Pieter Heyvaert, Anastasia Dimou, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. Towards Approaches for Generating RDF Mapping Definitions. In *Proceedings of the 14th International Semantic Web Conference: Posters and Demos*, 2015.

[9] Jochen Kranzdorf, Andrew Sellers, Giovanni Grasso, Christian Schallhart, and Tim Furche. Visual OXPath: robust wrapping by example. In *Proceedings of the 21st International Conference on World Wide Web*, pages 369–372. ACM, 2012.

[10] Maurizio Atzori and Carlo Zaniolo. SWIPE: Searching Wikipedia by Example. In *Proceedings of the 21st International Conference on World Wide Web*, pages 309–312. ACM, 2012.

[11] Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Dmitriy Zheleznyakov, Ian Horrocks, Christoph Pinkel, Martin G. Skjæveland, Evgenij Thorstensen, and Jose Mora. BootOX: Practical Mapping of RDBs to OWL 2. In *Proceedings of the 14th International Semantic Web Conference (Part II)*, pages 113–132. Springer, 2015.

[12] Mohsen Taheriyan, Craig A Knoblock, Pedro Szekely, and José Luis Ambite. Learning The Semantics of Structured Data Sources. *Web Semantics: Science, Services and Agents on the World Wide Web*, 37:152–169, 2016.

[13] Larry Masinter, Tim Berners-Lee, and Roy T Fielding. Uniform resource identifier (URI): Generic syntax. Technical report, 2005.

[14] 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 *The Semantic Web – Latest Advances and New Domains (ESWC 2016)*, pages 709–723. Springer, 2016.