Declarative Data Transformations for Linked Data Generation: the case of DBpedia

Ben De Meester, Wouter Maroy, Anastasia Dimou, Ruben Verborgh, and Erik Mannens

Ghent University – imec – IDLab, Belgium [*]
{firstname.lastname}@ugent.be

Abstract. Mapping languages allow us to define how Linked Data is generated from raw data, but only if the raw data values can be used *as is* to form the desired Linked Data. Since complex data transformations remain out of scope for mapping languages, these steps are often implemented as custom solutions, or with systems separate from the mapping process. The former data transformations remain case-specific, often coupled with the mapping, whereas the latter are not reusable across systems. In this paper, we propose an approach where data transformations (i) are defined declaratively and (ii) are aligned with the mapping languages. We employ an alignment of data transformations described using the Function Ontology (FnO) and mapping of data to Linked Data described using the RDF Mapping Language (RML). We validate that our approach can map and transform DBpedia in a declaratively defined and aligned way. Our approach is not case-specific: data transformations are independent of their implementation and thus interoperable, while the functions are decoupled and reusable. This allows developers to improve the generation framework, whilst contributors can focus on the actual Linked Data, as there are no more dependencies, neither between the transformations and the generation framework nor their implementations.

Keywords: Data Transformations, FnO, Linked Data Generation, RML

1 Introduction

Workflows that generate Linked Data from (semi-)structured data encompass both *schema* and *data* transformations [22]. *Schema* transformations involve (re-)modeling the original data, describing how objects are related, and deciding which vocabularies and ontologies to use [18]. *Data* transformations are needed to support any changes in the structure, representation or content of data [22], for instance performing string transformations or computations.

Schema transformations – also called *mappings* – are defined as a collection of rules that specify correspondences between data in different schemas [13]. Lately,

[*] The described research activities were funded by Ghent University, imec, Flanders Innovation & Entrepreneurship (AIO), the Fund for Scientific Research Flanders (FWO Flanders), and the European Union.

schema transformations are declaratively defined using mapping languages such as the W3C-recommended R2RML [7] or its extension RML [12]. Mapping languages detach rule definitions from the implementation that executes them. This renders the rules *interoperable* between implementations, whilst the systems that process those rules are *use-case independent*.

However, Linked Data generation systems usually assume data transformations are done beforehand. For instance, the R2RML specification explicitly mentions that data transformations or computations should be performed before generating Linked Data by generating a virtual table based on the result-set of an SQL statement (i.e., an SQL view) [7]. Other relevant W3C recommendations and working drafts do not take data transformations into account. More precisely, when discussing the "Convert Data to Linked Data" step, the Linked Data Best Practices [18] recommends using mapping languages – which only implies *schema* transformations – and does not distinguish data transformations elsewhere. Similarly, CSVW [27] specifies how to generate Linked Data from CSV by directly mapping the raw data values *as is*.

Systems that do include data transformations exist, but show one or more of the following limitations: the schema and data transformations are *uncombinable*, the allowed data transformations are *restricted*, the system is *case specific*, or the data transformations are *coupled* with the implementation.

For instance, the DBpedia Extraction Framework [2] (DBpedia EF) that generates Linked Data for one of the most widely known datasets, requires very specific data transformations, which are not available in existing systems. Thus, a case-specific hard-coded framework that depends on an internal set of parsing functions to generate the data values in the correct format was created. These parsing functions are *coupled* with the DBpedia EF, the schema and data transformations are *uncombinable*, and the overall system is *case specific*. Specifically for the DBpedia EF, these parsing functions are of high value. Indeed, they were created to parse manually entered (i.e., ambiguous and error-prone) data and are used for (and thus evaluated on) the entire Wikipedia corpus.

In this paper, we propose an approach that enables (i) *declarative and machine-processable data transformation descriptions* and (ii) the *alignment* of schema and data transformation descriptions. To validate this approach, we employ transformations described using the Function Ontology (FnO) [10], and align them with the RDF Mapping Language (RML) [12].

We apply our approach to the DBpedia EF. In the resulting system:

Schema and data transformations are *combinable*:
No separate systems need to be integrated.

Data transformations are *independent* of the mapping processor:
They are not restricted by the processor's capabilities.

Declarative transformations are *interoperable*:
The implementation can be case-independent.

Data transformations are *reusable*:
Their implementation is no longer dependent on the generation system.

We built and used the extended RMLProcessor with mapping documents to create the same DBpedia dataset, allowing more types of schema transformations, and enabling developers to work separate from contributors. The built Function Processor allows for an easier integration of data transformation libraries with other frameworks, and the DBpedia data parsing functions are made available independently, so other use cases can benefit from these data parsing tasks without needing to re-implement them.

The paper is organized as follows: after investigating the state of the art in Section 2, we detail why aligning declarative schema and data transformations is needed in Section 3. In Section 4, we introduce our approach, which we apply to RML and FnO, and provide a corresponding implementation. In Section 5, we explain how the DBpedia EF currently works, and prove how applying the proposed approach enables a fully declarative system with the same functionality as the existing DBpedia EF. Finally, we summarize our conclusions in Section 6.

2 State of the Art

Linked Data generation workflows require both *schema* and *data* transformations to generate the desired Linked Data [22]. Nevertheless, even though data transformations are often required [16], recommendations or best practices were not established so far, leading to a broad range of diverse approaches.

The simplest approaches rely on *custom solutions* which try to address both schema and data transformations in a coupled and hard-coded way, such as the DBpedia EF [2]. However, those approaches require new development cycles every time a modification or extension is desired. For instance, any change on the data transformations performed to generate the DBpedia dataset requires extending the DBpedia EF. There are cases where such approaches do allow certain configurations, yet those configurations are limited and, at least for the DBpedia EF, they focus on schema transformations rather than data transformations.

Similarly, *case-specific solutions* were established, which also couple the schema and data transformations. For instance, XSLT- or XPath-based approaches were established for generating Linked Data from data originally in XML format, such as by Lange [20]. In these cases, the range of possible transformations is limited by the respective language or syntax potential, while they can be performed prior or while the mapping is performed. Similarly, even mapping languages, such as HIL [14], D2RQ [6], or R2RML [7] can be considered, as their range of possible data transformations is determined by the range of transformations that can be defined when the data is retrieved from the data source, e.g., data transformations supported by SQL, performed before the mapping.

Other solutions first perform a direct mapping [1] to Linked Data, and then perform schema and data transformations on that generated Linked Data. The range of possible data transformations then often depends on SPARQL, as is the case of Datalift [25]. More customization is enabled by solutions that allow embedded scripts inside mapping documents such as R2RML-F [11]. However, they require existing libraries (and their dependencies) to be embedded (or possibly rewritten) within the mapping document, and are inherently limited to the standard libraries provided by the runtime environment (e.g., runtime environments often – for safety reasons – disallow file Input/Output operations).

There are also *query-oriented* languages that combine SPARQL with other languages or custom alignments to the underlying data structure. For instance, XSPARQL [5] maps data in XML format, R3M [15] data in relational databases and Tarql[1] data in CSV. Query-oriented languages are restricted to data transformations which can be translated when the query translation is performed, such as R3M that requires bidirectional transformations to retain read-write access [16].

Besides the aforementioned custom solutions, there are Linked Data generation workflows which rely on distinct systems to perform the schema and data transformations. These types of transformations cannot always be distinguished, as data transformations may affect the original schema. Such *data transformation tools* typically couple the transformation rules with their implementation, being either format specific (e.g., XSLT for data in XML format), or generic (e.g., Open Refine[2]). As the latter are targeted to contributors, they are often interactive. Thus, most data transformation systems can be configured and this happens often using a User Interface (UI), of which one of the most widely known is OpenRefine. Other systems – specifically for generating Linked Data – include the Linked Data Integration Framework (LDIF) [26], Linked Pipes [19], and DataGraft [24]. Their support for data transformations range from a fixed predefined set of transformations (e.g., LDIF and Linked Pipes) to an embedded scripting environment (e.g., OpenRefine and DataGraft).

Lately, different approaches emerged that define data transformations declaratively, such as Hydra [21] for Web Services, VOLT [23] for SPARQL, or FnO [10] as technology-independent abstraction. Hydra or VOLT depend on the underlying system (Web Services and SPARQL, respectively), thus their use is inherently limited to that system. On the one hand, using Hydra descriptions for executing transformations only works online, and requires all data to be transferred over HTTP, which is not always possible due to size or privacy concerns. On the other hand, VOLT only works for data already existing in a SPARQL endpoint. Describing the transformations using FnO does not include this dependency, thus allows for reuse in other use cases and technologies.

3 Limitations and Requirements

In this section, we discuss current schema and data transformation systems limitations (Section 3.1) and requirements (Section 3.2).

3.1 Limitations of current systems

Considering the current Linked Data generation systems discussed above, we come across data transformations which are (i) *uncombinable*, (ii) *restricted*, (iii) part of a *case specific* system, or (iv) *coupled*, as we discuss below.

[1] `https://tarql.github.io/`

[2] `http://openrefine.org/`

Uncombinable. When schema and data transformations are executed in successive steps (e.g., in the DBpedia EF, R2RML, or Datalift), additional integration is needed between them. However, schema and data transformations often depend on each other. Data transformations could influence the attributes of objects and vice versa. For instance, the calculated population of a settlement decides whether it is called a "town" or "city". This integration thus becomes complex, hurts interoperability, or limits the allowed transformations.

Restricted. Data transformations are embedded, defined and coupled within the system that executes them. Both in dedicated data transformation systems as when data transformations are embedded in mapping languages, the range and type of transformations used is limited to the ones implemented by the underlying system. Either a fixed set of data transformations is provided (e.g., LDIF, Unified Views), thus no other transformations can be defined, or a restricted scripting environment allows the definition of data transformations (e.g., OpenRefine, R2RML-F). In both cases limitations exist, e.g., using additional libraries, file manipulations, or external services are often disallowed. As such, existing tools cannot be applied for every use case. Supporting specialized use cases then usually requires providing separate systems (e.g., GeoSPARQL [4] for the geospatial domain). For instance, Blake et. al. [23] unveiled quality issues in DBpedia as the current extraction framework does not support basic geographic calculations, such as calculating the population density.

Case specific. Hard-coded systems couple the reference to a certain transformation with its implementation, and also mapping languages and dedicated systems support an opinionated set of transformations. As such, they can only be used for certain cases, and they require changes to the source code to apply any modifications or extensions, i.e., new developments cycles.

Coupled. So far, data transformations definitions are coupled with the implementation that executes them. For instance, data transformations specified by OpenRefine cannot be reused in other systems, and data transformations implemented in hard-coded systems are only available for that system and not reusable by others. Similarly, the coverage of data transformations differs across data sources, e.g., it is different between different SQL dialects for relational databases, XQuery for XML documents, and JSONPath for JSON documents. Chances of discrepancies between different systems (and the Linked Data they produce) are thus very high.

3.2 Requirements for future systems

In this paper, we argue that data transformations should be (i) *declaratively* specified, and (ii) *aligned* with declarative mapping languages. By specifying data transformations declaratively, just as for mapping languages, we decouple the transformations from the implementation that executes them. By aligning them with mapping languages instead of embedding them within the mapping languages, we remove the burden of the mapping processor to provide all needed functionality, allowing the implementations of the data transformations to exist separately from the generation system. This way, we achieve data transformations which are *reusable*, *interoperable*, *independent*, and *combinable*, as detailed below.

Reusable. Data transformations implementations should be reusable across use cases and systems, not necessarily only for Linked Data generation.

Interoperable. The declarations for data transformations should remain independent of the underlying implementation, i.e., be interoperable. This strictly separates the concerns of developers with those of contributors: developers can implement and improve the tools without being required to obtain domain knowledge, whilst contributors can focus on data modeling without being needed to get acquainted with the systems' source code. The generation of these declarations can be facilitated using a (graphical) editor [17].

Independent. Schema and data transformation declarations should be independent from each other. As such, their corresponding implementations also remain independent of each other, without enforcing mutual limitations. As such, (custom) data transformations can be integrated in the mapping process, but it is not required, i.e., they can still be executed in advance, and the mapping languages can still be used without data transformations.

Combinable. Data transformations should be usable not only in separate steps, but be combinable, e.g., with schema transformations. This enables, e.g., joining and meanwhile transforming multiple input values, or conditionally change the schema depending on the data transformations and vice versa.

4 Declarative Data Transformations

We provide a solution that implements the aforementioned declarative, machine processable data transformations which are aligned with schema transformations to Linked Data. Its main components are (i) the FnO ontology (Section 4.1), which enables describing functions in a declarative and machine processable way without making assumptions of their implementation; and (ii) the RML language (Section 4.2) that allows defining schema transformations (i.e., mappings) for generating Linked Data, independent of the input format. The Function Map is introduced, as an extension of RML, to facilitate the alignment of the two as explained in Section 4.3. Details regarding our proof-of-concept implementations are summarized in Section 4.4. For the remainder of this paper, we will use the following prefixes:

```
PREFIX fno:  <http://w3id.org/function/ontology#>
PREFIX grel: <http://semweb.datasciencelab.be/ns/grel#>
PREFIX rr:   <http://www.w3.org/ns/r2rml#>
PREFIX rml:  <http://semweb.datasciencelab.be/ns/rml#>
PREFIX fnml: <http://semweb.datasciencelab.be/ns/fnml#>
```

```
1  grel:toTitleCase a fno:Function ;
2    fno:name    "title case" ;
3    dcterms:description "return the input string in title case" ;
4    fno:expects ( [ fno:predicate grel:stringInput  ] ) ;
5    fno:output  ( [ fno:predicate grel:stringOutput ] ) .
6  :exe a fno:Execution ;
7    fno:executes grel:toTitleCase ;
8    grel:stringInput  "This is an input STRING." ;
9    grel:stringOutput "This Is An Input String." .
```

Listing 1: Function descriptions and Executions using FnO

4.1 The Function Ontology (FnO)

The Function Ontology (FnO) [8,10] allows agents to declare and describe functions uniformly, unambiguously, and independently of the technology that implements them. As mentioned in Section 2, we choose FnO over other declarative languages as it does not depend on the underlying system or implementation. A *function* (`fno:Function`) is an activity which has input parameters, output, and implements certain algorithm(s). A *parameter* (`fno:Parameter`) is the description of a function's input value. An *output* (`fno:Output`) is the description of the function's output value. An *execution* (`fno:Execution`) assigns values to the parameters of a function for a certain execution.

The actual implementation of the function can be retrieved separately from its description. Depending on the system, different implementations can be retrieved/used, e.g., a system implemented in Java can retrieve the implementation as a Java archive (JAR), whilst a browser-based system might rely on external APIs. Via content negotiation, different systems can request and discover different implementations of the same described function [9], given that these implementations exist. This allows a mapping processor to parse any function description, and retrieve and trigger the corresponding implementation for executing it.

For instance, `grel:toTitleCase`[3] (Listing 1, line 1) is a function that renders a given string into its corresponding title cased value. It expects a string, indicated by the `grel:stringInput` property (line 4) as input. An Execution (line 6) can be instantiated to bind a value to the parameter. The result is then bound to that Execution via the `grel:stringOutput` property (line 9).

4.2 The RDF Mapping Language (RML)

R2RML [7] is the W3C-recommended mapping language for defining mappings of data in relational databases to the RDF data model. Its extension RML [12] broadens its scope and covers also schema transformations from sources in different (semi-)structured formats, such as CSV, XML, and JSON. RML documents [12] contain rules defining how the input data will be represented in RDF. The main

[3] Specified from the description as provided by OpenRefine on `https://github.com/OpenRefine/OpenRefine/wiki/GREL-String-Functions#totitlecasestring-s`

```
1 <#Mapping> rml:logicalSource <#InputX> ;
2   rr:subjectMap [ rr:template "http://ex.com/{ID}"; rr:class foaf:Person ];
3   rr:predicateObjectMap [ rr:predicate foaf:knows;
4     rr:objectMap [ rr:parentTriplesMap <#Acquaintance> ]].
5 <#Acquaintance> rml:logicalSource <#InputY> ;
6   rr:subjectMap [ rml:reference "acquaintance"; rr:termType rr:IRI; rr:class ex:Person]].
```

Listing 2: RML mapping definitions

building blocks of RML documents are Triples Maps (Listing 2: line 1). A Triples Map defines how triples of the form (subject, predicate, object) will be generated.

A Triples Map consists of three main parts: the Logical Source, the Subject Map and zero or more Predicate-Object Maps. The Subject Map (line 2, 6) defines how unique identifiers (URIs) are generated for the mapped resources and is used as the subject of all RDF triples generated from this Triples Map. A Predicate-Object Map (line 3) consists of Predicate Maps, which define the rule that generates the triple's predicate (line 3) and Object Maps or Referencing Object Maps (line 4), which define how the triple's object is generated. The Subject Map, the Predicate Map and the Object Map are Term Maps, namely rules that generate an RDF term (an IRI, a blank node or a literal). A Term Map can be a *constant-valued term map* (line 3) that always generates the same RDF term, or a *reference-valued term map* (line 6) that uses the data value of a referenced data fragment in a given Logical Source, or a *template-valued term map* (line 2) that uses a valid string template that can contain referenced data fragments of a given Logical Source.

Other languages used for mapping (such as CSVW, XPath, and SPARQL) are dependent on the input format (CSV, XML, and SPARQL, respectively). RML abstracts the input source format, making it applicable in more use cases. Moreover, as the schema transformations are declared in RDF, the integration with external vocabularies or data sources is inherently available.

4.3 Model integration

Typically, mapping languages refer to raw data values. Therefore, aligning them with declarative data transformations requires a way to refer to terms which are derived from raw data, but after applying certain transformations, i.e., functions.

In the case of [R2]RML, Term Maps determine how to generate an RDF term relying on references to raw data. Therefore, a new type of Term Map was introduced, the Function Map (`fnml:FunctionMap`, Listing 3: line 10). A Function Map is a Term Map generated by executing a function, instead of using a constant or a reference to the raw data values. In contrast to an RDF Term Map that uses values referenced from a Logical Source to generate an RDF term, a Function Map uses values referenced from a Logical Source to execute a function (line 12). Once the function is executed, its output value is the term generated by this Function Map. To this end, the `fnml:functionValue` property was introduced to indicate which instance of a function needs to be executed to generate an output and considering which values (line 11). Such a function is described using FnO.

```
<#Person_Mapping>
  rml:logicalSource     <#LogicalSource> ; # Specify the data source
  rr:subjectMap         <#SubjectMap>    ; # Specify the subject
  rr:predicateObjectMap <#NameMapping>   . # Specify the predicate-object-map

<#NameMapping>
  rr:predicate dbo:title                 ; # Specify the predicate
  rr:objectMap <#FunctionMap>            . # Specify the object-map

<#FunctionMap>
  fnml:functionValue [                     # The object is the result of the function
    rml:logicalSource <#LogicalSource>   ; # Use the same data source for input
    rr:predicateObjectMap [
      rr:predicate fno:executes          ; # Execute `grel:titleCase`
      rr:objectMap [ rr:constant  grel:titleCase ] ] ;
    rr:predicateObjectMap [
      rr:predicate grel:inputString      ;
      rr:objectMap [ rr:reference "name" ] ] # Use as input the "name" reference
  ] .
```

Listing 3: Alignment RML and FnO

This extension of one class and one property allows us to align RML and FnO, without creating additional dependencies between the two. This is possible as they are both declarative and described in RDF.

4.4 Implementation

As a proof of concept, we extended the RMLProcessor to support the Function Map, available at `github.com/RMLio/RML-Mapper/tree/extension-fno`. In addition, we implemented a generic Function Processor in Java which can be found at `github.com/FnOio/function-processor-java` that uses the function declarations described in FnO to retrieve and execute their relevant implementations. When the RMLProcessor encounters a Function Map[4], it extracts the function identifier (i.e., its URI) and the parameter values as described in the mapping document or from the data sources[5], and sends those to the Function Processor. When receiving an unknown function identifier, the Function Processor discovers the relevant implementations online [9], and obtains an implementation to be executed locally if available[6]. Based on the function description using FnO, the Function Processor automatically detects how to execute the needed function and returns the resulting value back to the RMLProcessor.

We extracted GREL functions and the DBpedia parsing Functions (see Section 5) as independent libraries at `github.com/FnOio/grel-functions-java` and `github.com/FnOio/dbpedia-parsing-functions-scala`, respectively. Their descriptions using FnO are available at `semweb.datasciencelab.be/ns/grel` and `semweb.datasciencelab.be/ns/dbpedia-functions` respectively. Thus, we can

[4] `https://git.io/vyI0k`
[5] `https://git.io/vyIRh`
[6] Currently, Java snippets and JARs are supported, as the latter allows using additional dependencies in the implemented functions.

(re-)use these functions separately from their original systems (i.e., OpenRefine and the DBpedia EF), but we can also – when using their descriptions in mapping documents – require them as data transformations within the RMLProcessor.

Our resulting extension of the RMLProcessor overcomes the limitations as stated in Section 3. It is capable of combining schema and data transformations. It could already process [R2]RML statements, and now, it can also extract the Function Map and allows the Function Processor to perform the data transformations. Next, the Function Processor is independent of the RMLProcessor, thus no limitations are enforced between them, and the system does not depend on the use case, as all schema and data transformations are specified in the mapping document and the implementations of the needed data transformations are obtained on the fly. Finally, all data transformations are available as stand-alone libraries, independent of the use case, the Function Processor, or the RMLProcessor.

We also extended the RMLEditor [17] to support the definition of Function Maps so users can easily edit mapping documents with declarative data transformations, without needing prior knowledge about RML or FnO. The default version of the RMLEditor considers the GREL functions, but any other function may be available. A screencast showcasing how the RMLEditor was extended can be found at `www.youtube.com/watch?v=-38pkkTxQ1s`. In total, users from 16 companies and research institutes profit from this RMLEditor extension in addition to the DBpedia community.

5 Application to DBpedia

In this section, we show the current DBpedia generation workflow (Section 5.1), the changes we implemented (Section 5.2), and validate our approach (Section 5.3).

5.1 Current generation workflow with the DBpedia EF

DBpedia is a crowd-sourced community effort to extract structured information from Wikipedia and make this information available on the Web [2]. Data from DBpedia is generated in two parts: The first maps data from the relationships already stored in the underlying relational database tables and the second directly extracts data from the article texts and infobox templates within the articles [3]. Figure 1 shows the current DBpedia EF, specifically focused on the RDF generation from infobox templates (i.e., the second part). The grey area denotes the DBpedia-specific implementation, and the cogs denote the successive processing steps.

Infobox templates are text fragments inside the article text with specific syntax to denote certain visualizations (e.g., '`{{`' and '`}}`' denote the beginning and ending of an infobox table, respectively). The DBpedia EF consists of the following steps: step *a*, all Wikipedia pages containing infobox templates for the relational database are selected. Then, step *b*, only the significant templates which are contained in these pages are selected and extracted. Step *c*, each template is then parsed to generate the desired triples (i.e., the subject and predicate-object pairs). Afterwards, step *d*, object values are further post-processed, i.e., i) when

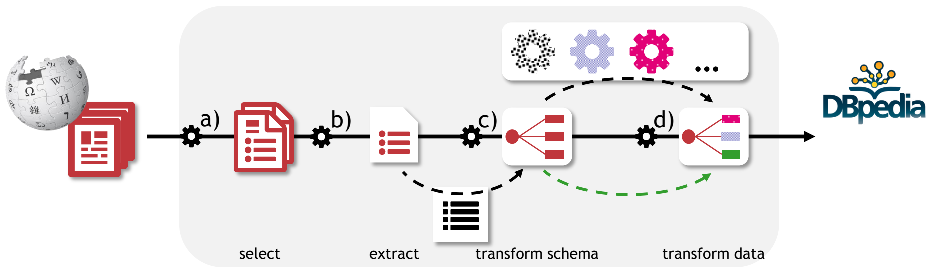

Fig. 1: The current generation workflow: successive hard-coded processes a) select pages, b) extract infoboxes, c) transform the schema, and d) transform the data, either by generating URIs (bottom arrow) or by using hard-coded parsing functions (top arrow).

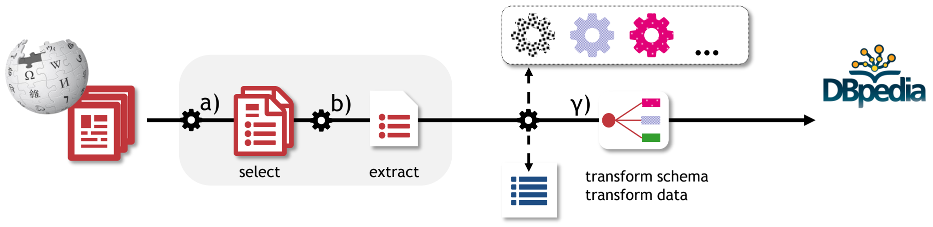

Fig. 2: The new generation workflow: after a) selecting pages and b) extracting the infoboxes using the original framework, γ) both schema and data transformations are combined using an interoperable mapping processor and reusable parsing functions, specified by a fully declarative mapping document.

these object values contain Wiki links, suitable URI references are generated (the bottom arrow of step *d* in Figure 1), otherwise, ii) uniform typed literals are generated by parsing the strings and numeric values (the top arrow of step *d*). The data of DBpedia is structured using the dedicated DBpedia Ontology[7]: a cross-domain ontology, which has been manually created based on the most commonly used infoboxes within Wikipedia.

The extracted infobox contains a textual representation of a list of key-value pairs, e.g., the item '`established = 4 October 1830`'. After assigning per key a fixed predicate from the DBpedia Ontology and a fixed data type to the value [3], each value is processed individually according to that datatype. Wiki links are converted to meaningful URIs, but other values need to be parsed. However, since there are not many restrictions on the design of Wikipedia templates, the format of these manually entered values can be very diverse. For instance, when revisiting the previous example, the same date can be written down as '`04-10-1830`', '`1830, 4 10`', '`October 4th 1830`', etc. Many other types

[7] `http://dbpedia.org/ontology`

of discrepancies occur, for example, using different numbering formats (e.g., '`1 000 000`' vs '`1E6`' vs '`1 million`'), or using different units than specified in the template (e.g., '`area_km2 = 11,787 sqmi`'). This situation is aggravated because information in Wikipedia is crowd-sourced, thus these differences in cultures and countries – coming from different contributors of Wikipedia – can occur within one page, together with already existing inaccuracies inherent to manual entries, such as typos and misspellings.

To accommodate to this situation, the DBpedia EF consists of a large amount of parsing functions that fruitfully handle most edge cases. Each of these parsing functions were tested against thousands of values coming from Wikipedia. They are thus very robust and essential to the generation framework. However, they form an internal set of functions, hard-coded in the framework. Each change in these parsing functions requires another development cycle for the entire framework, but, moreover, they cannot be reused for other use cases. As valuable as these parsing functions are, they are hidden deep within the DBpedia EF.

Hence, the following limitations arise. First, the DBpedia EF successively performs the schema and then the data transformations, which limits its capabilities, e.g., it is currently not possible to join multiple values from the infobox templates to form one output value, nor is it possible to connect with external data sources. Second, all transformations are hard-coded. Changes require knowledge of the source code and involve new development cycles. Third, all parsing functions are embedded in the framework, making them non-reusable and use-case specific.

5.2 New generation workflow with RML and FnO

We apply our system that enables declarative data transformations which can be aligned with schema transformations to the the DBpedia EF as can be seen in Figure 2. In step a, Wikipedia pages containing infobox templates are selected. Then, in step b, the significant templates are selected and extracted from those pages. Finally, in step γ, on these templates, schema and data transformations are performed together to achieve the resulting RDF.

Steps a and b provide the input data and have not changed. Step γ however is performed using the RMLProcessor, the transformations are declared using a DBpedia mapping document, and the DBpedia parsing functions are used as standalone library. The generation of the DBpedia mapping document in RML based on the existing mappings in the DBpedia EF has been done in previous work[8]. This work has been extended to include the data transformation descriptions.

`fnoio.github.io/dbpedia-demo/` allows users to try out the possible customizations of the new DBpedia EF. Changes can be made to a mapping document – used for the country-infobox template – both for schema and data transformations. Both the DBpedia parsing functions as the GREL functions are loaded. It is thus possible to, e.g., change string values using GREL functions, or use a different parsing function, whilst also changing the schema transformations, without needing prior knowledge of the DBpedia EF.

[8] `www.mail-archive.com/dbpedia-discussion@lists.sourceforge.net/msg07837.html`

5.3 Validation

By applying our approach to DBpedia, we have not only created a fully declarative system that is capable of extracting the same RDF data from the Wikipedia infoboxes as the current DBpedia EF, we also achieve the following:

***Combinable* schema and data transformations.** *Before*, schema and data transformations were executed in successive steps in the DBpedia EF. Consequently, the data transformations were executed based on the data type as assigned by the schema transformations, and transformations applying to both schema and data were not supported. *Now*, data transformations can be specified within the structure, not just the data type, and joining multiple input values, or conditionally assigning types based on the data values becomes possible.

***Independent* schema and data transformations.** *Before*, all data parsing functions needed to be hard-coded inside the DBpedia EF, as existing tools did not provide the required data transformation capabilities. *Now*, all data parsing functions are separate libraries, and no dependencies exist between these data parsing functions and the DBpedia EF.

An *interoperable* system. *Before*, the DBpedia EF was a hard-coded system depending on a custom mapping document that mapped keys to predicates of the DBpedia Ontology, after which hard-coded data transformations were performed. Every change in the generation process required a new development cycle. This explains why the DBpedia EF has been developed by only forty-two developers[9]. *Now*, no dependencies exist between the implementation and the specification of the generated Linked Data, as schema transformations, data transformations, and their alignment are all specified declaratively. The adjusted RMLProcessor remains a use-case independent system, and the declarations do not depend on any implementation, separating the concerns of the contributors with those of the developers.

***Reusable* data transformations.** *Before*, all data parsing functions were embedded in the DBpedia EF, making it even harder for developers to improve its code. The core team that improved the DBpedia EF parsing functions consisted of barely six out of the forty-two people. *Now*, all parsing functions exist as a stand-alone library, without dependencies to the original DBpedia EF, RML or FnO. They can be used and improved or extended by anyone, for any use case. The common problem of parsing manually entered data has just become more easy as this set of functions can now freely be used: it has been tested on the Wikipedia corpus, is capable of resolving many typos and ambiguities, and now no longer depends on the use case or data source type. Its usage has been made user-friendly by including data transformations in the RMLEditor.

[9] See `github.com/dbpedia/extraction-framework`.

6 Conclusion and Future Work

Linked Data generation encompasses both schema and data transformations. However, in this paper, we identified that data transformations in current Linked Data generation processes are uncombinable with the schema transformations, restricted by the mapping language, part of a case-specific system, or non-reusable.

Our proposed approach specifies data transformations declaratively and aligns them with declarative schema transformations. We employed this approach by aligning FnO with RML and provided an implementation by extending the RMLProcessor and building the Function Processor. As validated on the DBpedia EF, schema and data transformations remain independent but are combinable. The created system is interoperable and data transformations are reusable across systems and data sources. The DBpedia EF now supports more schema and data transformations, separates the concerns between contributors and developers, and the DBpedia parsing functions are available as independent libraries.

In the future, we aim to reuse well-tested descriptive data transformations, such as the DBpedia parsing functions to facilitate different use cases.

References

1. Arenas, M., Bertails, A., Prudhommeaux, E., Sequeda, J.: A Direct Mapping of Relational Data to RDF. W3C Recommendation (2012), http://www.w3.org/TR/rdb-direct-mapping/
2. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Ives, Z.: DBpedia: A Nucleus for a Web of Open Data. In: The Semantic Web: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007. Proceedings. pp. 722–735. Springer (2007)
3. Auer, S., Lehmann, J.: What have Innsbruck and Leipzig in common? Extracting semantics from wiki content. In: The Semantic Web: Research and Applications: 4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, June 3-7, 2007. Lecture Notes in Computer Science, vol. 4519. Springer (2007)
4. Battle, R., Kolas, D.: GeoSPARQL: enabling a geospatial Semantic Web. Semantic Web Journal 3(4), 355–370 (2011)
5. Bischof, S., Decker, S., Krennwallner, T., Lopes, N., Polleres, A.: Mapping between RDF and XML with XSPARQL. Journal on Data Semantics 1(3), 147–185 (2012)
6. Cyganiak, R., Bizer, C., Garbers, J., Maresch, O., Becker, C.: The D2RQ Mapping Language. Tech. rep. (2012), http://d2rq.org/d2rq-language
7. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF Mapping Language. Working group recommendation, W3C (sep 2012), http://www.w3.org/TR/r2rml/
8. De Meester, B., Dimou, A.: The Function Ontology. Unofficial Draft (2016), https://w3id.org/function/spec
9. De Meester, B., Dimou, A., Verborgh, R., Mannens, E.: Discovering and Using Functions via Content Negotiation. In: 15th International Semantic Web Conference: Posters & Demonstrations Track. CEUR Workshop Proceedings, vol. 1690 (2016)
10. De Meester, B., Dimou, A., Verborgh, R., Mannens, E., Van de Walle, R.: An Ontology to Semantically Declare and Describe Functions. In: The Semantic Web: ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29 – June 2, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9989, pp. 46–49. Springer (2016)

11. Debruyne, C., O'Sullivan, D.: R2RML-F: Towards Sharing and Executing Domain Logic in R2RML Mappings. In: Workshop on Linked Data on the Web. CEUR Workshop Proceedings, vol. 1593 (2016)
12. Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R.: RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In: Proceedings of the 7th Workshop on Linked Data on the Web. CEUR Workshop Proceedings, vol. 1184 (2014)
13. Euzenat, J., Shvaiko, P.: Ontology Matching. Springer (2013)
14. Hernández, M., Koutrika, G., Krishnamurthy, R., Popa, L., Wisnesky, R.: HIL a high-level scripting language for entity integration. In: Proceedings of the 16th International Conference on Extending Database Technology. ACM (2013)
15. Hert, M., Reif, G., Gall, H.C.: 'Semantic Web 2.0' - write-enabling the Web of Data. In: 6th Workshop on Semantic Web Applications and Perspectives (2010)
16. Hert, M., Reif, G., Gall, H.C.: A Comparison of RDB-to-RDF Mapping Languages. In: Proc. of the 7th International Conference on Semantic Systems. ACM (2011)
17. Heyvaert, P., Dimou, A., Herregodts, A.L., Verborgh, R., Schuurman, D., Mannens, E., Van de Walle, R.: RMLEditor: A Graph-based Mapping Editor for Linked Data Mappings. In: The Semantic Web – Latest Advances and New Domains (ESWC 2016). Lecture Notes in Computer Science, vol. 9678, pp. 709–723. Springer (2016)
18. Hyland, B., Atemezing, G., Villazón-Terrazas, B.: Best Practices for Publishing Linked Data. WG Note, W3C (jan 2014), `http://www.w3.org/TR/ld-bp/`
19. Klímek, J., Škoda, P., Nečaskỳ, M.: LinkedPipes ETL: Evolved Linked Data preparation. In: The Semantic Web: ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29 – June 2, 2016, Revised Selected Papers. Springer (2016)
20. Lange, C.: Krextor -An Extensible Framework for Contributing Content Math to the Web of Data. In: Intelligent Computer Mathematics: 18th Symposium, Calculemus 2011, and 10th International Conference, MKM 2011, Bertinoro, Italy, July 18-23, 2011. Proceedings. pp. 304–306. Springer (2011)
21. Lanthaler, M.: Hydra Core Vocabulary. Unofficial Draft (Jun 2014), `http://www.hydra-cg.com/spec/latest/core/`
22. Rahm, E., Do, H.H.: Data cleaning: Problems and current approaches. IEEE Data Engineering Bulletin 23(4), 3–13 (2000)
23. Regalia, B., Janowicz, K., Gao, S.: VOLT: A Provenance-Producing, Transparent SPARQL Proxy for the On-Demand Computation of Linked Data and its Application to Spatiotemporally Dependent Data. In: Proceedings of the 13th International Conference on The Semantic Web. Latest Advances and New Domains. pp. 523–538. Springer International Publishing, Springer (2016)
24. Roman, D., Nikolov, N., Putlier, A., Sukhobok, D., Elvesaeter, B., Berre, A., Ye, Xianglinand Dimitrov, M., Simov, A., Zarev, M., Moynihan, R., Roberts, B., Berlocher, I., Kin, K.S., Tony Lee andSmith, A., Heath, T.: DataGraft: One-Stop-Shop for Open Data Management. Semantic Web Journal (2016)
25. Scharffe, F., Atemezing, G., Troncy, R., Gandon, F., Villata, S., Bucher, B., Hamdi, F., Bihanic, L., Képéklian, G., Cotton, F., Euzenat, J., Fan, Z., Vandenbussche, P.Y., Vatant, B.: Enabling Linked Data publication with the Datalift platform. In: Proceedings AAAI workshop on semantic cities (2012)
26. Schultz, A., Matteini, A., Isele, R., Bizer, C., Becker, C.: LDIF – Linked Data Integration Framework. In: Proc. of the Second International Conference on Consuming Linked Data. CEUR Workshop Proceedings, vol. 782, pp. 125–130 (2011)
27. Tennison, J., Kellogg, G., Herman, I.: Generating RDF from Tabular Data on the Web. W3C Recommendation (dec 2015), `https://www.w3.org/TR/csv2rdf/`